

Кэвин П. Мэрфи



Вероятностное машинное обучение

Введение

Кэвин П. Мэрфи

Вероятностное машинное обучение

Введение

Kevin P. Murphy

Probabilistic Machine Learning

An Introduction



Cambridge, Massachusetts
London, England

Кэвин П. Мэрфи

Вероятностное машинное обучение

Введение



Москва, 2022

УДК 004.048
ББК 32.972
М97

Мэрфи К. П.

М97 Вероятностное машинное обучение: введение / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2022. – 940 с.: ил.

ISBN 978-5-93700-119-1

Данный классический труд содержит современное введение в машинное обучение, рассматриваемое сквозь призму вероятностного моделирования и байесовской теории принятия решений. Включен базовый математический аппарат (в том числе элементы линейной алгебры и теории оптимизации), основы обучения с учителем (включая линейную и логистическую регрессию и глубокие нейронные сети), а также более глубокие темы (в частности, перенос обучения и обучение без учителя).

Упражнения в конце глав помогут читателям применить полученные знания. В приложении приводится сводка используемых обозначений.

Книга будет полезна специалистам в области машинного обучения и студентам профильных специальностей.

УДК 004.048
ББК 32.972

Copyright Original English language edition published by The MIT Press Cambridge, MA. Copyright © 2021 Kevin P. Murphy. Russian-language edition copyright © 2022 by DMK Press. All rights reserved. The rights to the Russian-language edition obtained through Alexander Korzhenevski Agency (Moscow). Права на издание получены при помощи агентства Александра Корженевского (Москва).

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN 978-0-2620468-2-4 (англ.)
ISBN 978-5-93700-119-1 (рус.)

© Kevin P. Murphy, 2021
© Перевод, оформление, издание,
ДМК Пресс, 2022

Содержание

От издательства	30
Предисловие	31
Глава 1. Введение	34
1.1. Что такое машинное обучение?	34
1.2. Обучение с учителем	35
1.2.1. Классификация	35
1.2.1.1. Пример: классификация ирисов	35
1.2.1.2. Разведочный анализ данных	37
1.2.1.3. Обучение классификатора	38
1.2.1.4. Минимизация эмпирического риска	39
1.2.1.5. Неопределенность	41
1.2.1.6. Оценка максимального правдоподобия	42
1.2.2. Регрессия	43
1.2.2.1. Линейная регрессия	44
1.2.2.2. Полиномиальная регрессия	45
1.2.2.3. Глубокие нейронные сети	46
1.2.3. Переобучение и обобщаемость	47
1.2.4. Теорема об отсутствии бесплатных завтраков	48
1.3. Обучение без учителя	48
1.3.1. Кластеризация	49
1.3.2. Обнаружение латентных «факторов изменчивости»	50
1.3.3. Самостоятельное обучение	51
1.3.4. Оценка обучения без учителя	52
1.4. Обучение с подкреплением	53
1.5. Данные	55
1.5.1. Некоторые широко известные наборы изображений	55
1.5.1.1. Небольшие наборы изображений	55
1.5.1.2. ImageNet	56
1.5.2. Некоторые широко известные наборы текстовых данных	57
1.5.2.1. Классификация текста	58
1.5.2.2. Машинный перевод	59
1.5.2.3. Другие задачи типа seq2seq	59
1.5.2.4. Языковое моделирование	59
1.5.3. Предобработка дискретных входных данных	60
1.5.3.1. Унитарное кодирование	60
1.5.3.2. Перекрестные произведения признаков	60
1.5.4. Предобработка текстовых данных	61
1.5.4.1. Модель мешка слов	61
1.5.4.2. TF-IDF	62

1.5.4.3. Погружения слов.....	63
1.5.4.4. Обработка новых слов	63
1.5.5. Обработка отсутствующих данных.....	64
1.6. Обсуждение.....	65
1.6.1. Связь МО с другими дисциплинами.....	65
1.6.2. Структура книги.....	66
1.6.3. Подводные камни.....	66

Часть I. ОСНОВАНИЯ68

Глава 2. Вероятность: одномерные модели69

2.1. Введение.....	69
2.1.1. Что такое вероятность?.....	69
2.1.2. Типы неопределенности.....	70
2.1.3. Вероятность как обобщение логики	70
2.1.3.1. Вероятность события	70
2.1.3.2. Вероятность конъюнкции двух событий	71
2.1.3.3. Вероятность объединения двух событий.....	71
2.1.3.4. Условная вероятность одного события при условии другого	71
2.1.3.5. Независимость событий.....	72
2.1.3.6. Условная независимость событий	72
2.2. Случайные величины	72
2.2.1. Дискретные случайные величины	72
2.2.2. Непрерывные случайные величины.....	73
2.2.2.1. Функция распределения.....	73
2.2.2.2. Функция плотности распределения	74
2.2.2.3. Квантили.....	75
2.2.3. Множества связанных случайных величин	75
2.2.4. Независимость и условная независимость.....	76
2.2.5. Моменты распределения.....	77
2.2.5.1. Среднее распределения.....	78
2.2.5.2. Дисперсия распределения	78
2.2.5.3. Мода распределения	79
2.2.5.4. Условные моменты.....	80
2.2.6. Ограничения сводных статистик*	81
2.3. Формула Байеса.....	83
2.3.1. Пример: тестирование на COVID-19.....	84
2.3.2. Пример: парадокс Монти Холла.....	86
2.3.3. Обратные задачи*	88
2.4. Распределение Бернулли и биномиальное распределение.....	89
2.4.1. Определение.....	89
2.4.2. Сигмоидная (логистическая) функция.....	90
2.4.3. Бинарная логистическая регрессия	92
2.5. Категориальное и мультиномиальное распределение	93
2.5.1. Определение.....	93
2.5.2. Функция softmax	94

2.5.3. Многоклассовая логистическая регрессия.....	95
2.5.4. Логарифмирование, суммирование, потенцирование.....	96
2.6. Одномерное гауссово (нормальное) распределение	97
2.6.1. Функция распределения	98
2.6.2. Функция плотности вероятности.....	99
2.6.3. Регрессия.....	100
2.6.4. Почему гауссово распределение так широко используется?	101
2.6.5. Дельта-функция Дирака как предельный случай	102
2.7. Другие часто встречающиеся одномерные распределения*	102
2.7.1. Распределение Стюдента	102
2.7.2. Распределение Коши	104
2.7.3. Распределение Лапласа	105
2.7.4. Бета-распределение.....	105
2.7.5. Гамма-распределение	106
2.7.6. Эмпирическое распределение	107
2.8. Преобразования случайных величин*	108
2.8.1. Дискретный случай	109
2.8.2. Непрерывный случай	109
2.8.3. Обратимые преобразования (биекции)	109
2.8.3.1. Замена переменных: скалярный случай.....	109
2.8.3.2. Замена переменных: многомерный случай.....	110
2.8.4. Моменты линейного преобразования.....	112
2.8.5. Теорема о свертке	113
2.8.6. Центральная предельная теорема.....	115
2.8.7. Аппроксимация Монте-Карло.....	115
2.9. Упражнения.....	116
Глава 3. Вероятность: многомерные модели.....	120
3.1. Совместные распределения нескольких случайных величин.....	120
3.1.1. Ковариация	120
3.1.2. Корреляция	121
3.1.3. Некоррелированные не значит независимые	122
3.1.4. Из коррелированности не следует наличие причинно-следственной связи.....	122
3.1.5. Парадокс Симпсона.....	123
3.2. Многомерное гауссово (нормальное) распределение	126
3.2.1. Определение	126
3.2.2. Расстояние Махаланобиса	127
3.2.3. Маргинальные и условные распределения для многомерного нормального распределения*	129
3.2.4. Пример: обуславливание двумерного гауссова распределения.....	130
3.2.5. Пример: подстановка отсутствующих значений*	131
3.3. Линейные гауссовы системы*	132
3.3.1. Формула Байеса для гауссовых распределений	132
3.3.2. Вывод*	133
3.3.3. Пример: вывод неизвестного скаляра.....	134
3.3.4. Пример: вывод неизвестного вектора.....	136

3.3.5. Пример: слияние показаний датчиков.....	137
3.4. Экспоненциальное семейство распределений*	139
3.4.1. Определение.....	139
3.4.2. Пример	140
3.4.3. Логарифмическая функция разбиения является производящей функцией полуинвариантов	141
3.4.4. Вывод максимальной энтропии экспоненциального семейства	141
3.5. Смесовые модели	142
3.5.1. Модель гауссовой смеси.....	143
3.5.2. Модели бернуллиевой смеси	145
3.6. Графовые вероятностные модели*	146
3.6.1. Представление.....	146
3.6.1.1. Пример: оросительная система	147
3.6.1.2. Пример: марковская цепь	148
3.6.2. Вывод	149
3.6.3. Обучение	149
3.6.3.1. Блочная нотация.....	150
3.7. Упражнения	151
Глава 4. Статистика.....	153
4.1. Введение.....	153
4.2. Оценка максимального правдоподобия (MLE).....	153
4.2.1. Определение.....	154
4.2.2. Обоснование MLE	155
4.2.3. Пример: MLE для распределения Бернулли	156
4.2.4. Пример: MLE для категориального распределения	157
4.2.5. Пример: MLE для одномерного гауссова распределения	158
4.2.6. Пример: MLE для многомерного гауссова распределения.....	159
4.2.6.1. MLE среднего.....	159
4.2.6.2. MLE ковариационной матрицы	160
4.2.7. Пример: MLE для линейной регрессии	161
4.3. Минимизация эмпирического риска (ERM)	162
4.3.1. Пример: минимизации частоты неправильной классификации.....	163
4.3.2. Суррогатная потеря	163
4.4. Другие методы оценивания*	165
4.4.1. Метод моментов	165
4.4.1.1. Пример: MOM для одномерного гауссова распределения	165
4.4.1.2. Пример: MOM для равномерного распределения.....	166
4.4.2. Онлайн-овое (рекурсивное) оценивание	167
4.4.2.1. Пример: рекурсивная MLE среднего гауссова распределения ...	167
4.4.2.2. Экспоненциально взвешенное скользящее среднее	167
4.5. Регуляризация	169
4.5.1. Пример: оценка MAP для распределения Бернулли	170
4.5.2. Пример: оценка MAP для многомерного гауссова распределения* ...	171
4.5.2.1. Оценка усадки.....	171
4.5.3. Пример: уменьшение весов	172
4.5.4. Подбор регуляризатора с помощью контрольного набора	173

4.5.5. Перекрестная проверка	174
4.5.5.1. Правило одной стандартной ошибки	175
4.5.5.2. Пример: гребневая регрессия	176
4.5.6. Ранняя остановка	176
4.5.7. Больше данных	177
4.6. Байесовские статистики*	178
4.6.1. Сопряженные априорные распределения	179
4.6.2. Бета-биномиальная модель	180
4.6.2.1. Правдоподобие Бернулли	180
4.6.2.2. Биномиальное правдоподобие	180
4.6.2.3. Априорное распределение	181
4.6.2.4. Апостериорное распределение	181
4.6.2.5. Пример	181
4.6.2.6. Апостериорная мода (оценка MAP)	182
4.6.2.7. Апостериорное среднее	183
4.6.2.8. Апостериорная дисперсия	183
4.6.2.9. Апостериорное прогнозное распределение	184
4.6.2.10. Маргинальное правдоподобие	187
4.6.2.11. Смеси сопряженных априорных распределений	187
4.6.3. Дирихле-мультиномиальная модель	189
4.6.3.1. Правдоподобие	189
4.6.3.2. Априорное распределение	189
4.6.3.3. Апостериорное распределение	191
4.6.3.4. Апостериорное прогнозное распределение	192
4.6.3.5. Маргинальное правдоподобие	192
4.6.4. Гауссова-гауссова модель	193
4.6.4.1. Одномерный случай	193
4.6.4.2. Многомерный случай	195
4.6.5. За пределами сопряженных априорных распределений	196
4.6.5.1. Неинформативные априорные распределения	197
4.6.5.2. Иерархические априорные распределения	197
4.6.5.3. Эмпирические априорные распределения	197
4.6.6. Байесовские доверительные интервалы	198
4.6.7. Байесовское машинное обучение	200
4.6.7.1. Подстановочная аппроксимация	201
4.6.7.2. Пример: скалярный вход, бинарный выход	201
4.6.7.3. Пример: бинарный вход, скалярный выход	203
4.6.7.4. Вертикальное масштабирование	205
4.6.8. Вычислительные трудности	205
4.6.8.1. Сеточная аппроксимация	206
4.6.8.2. Квадратичная аппроксимация (Лапласа)	206
4.6.8.3. Вариационная аппроксимация	207
4.6.8.4. Аппроксимация методом Монте-Карло по схеме марковских цепей	208
4.7. Частотная статистика*	208
4.7.1. Выборочное распределение	209
4.7.2. Гауссова аппроксимация выборочного распределения MLE	210

4.7.3. Бутстрэпная аппроксимация выборочного распределения любого оценщика	211
4.7.3.1. Бутстрэп – апостериорное распределение «для бедных»	211
4.7.4. Доверительные интервалы	212
4.7.5. Предостережения: доверительные интервалы и байесовские доверительные интервалы не одно и то же	214
4.7.6. Компромисс между смещением и дисперсией	215
4.7.6.1. Смещение оценки	215
4.7.6.2. Дисперсия оценки	216
4.7.6.3. Компромисс между смещением и дисперсией	216
4.7.6.4. Пример: оценка МАР среднего гауссова распределения	217
4.7.6.5. Пример: оценка МАР для линейной регрессии	218
4.7.6.6. Применение компромисса между смещением и дисперсией для классификации	220
4.8. Упражнения	220
Глава 5. Теория принятия решений	225
5.1. Байесовская теория принятия решений	225
5.1.1. Основы	225
5.1.2. Проблемы классификации	227
5.1.2.1. Бинарная потеря	228
5.1.2.2. Классификация с учетом стоимости	228
5.1.2.3. Классификация с возможностью отклонения примера	229
5.1.3. ROC-кривые	230
5.1.3.1. Матрицы неточностей классификации	230
5.1.3.2. Обобщение ROC-кривой в виде скаляра	233
5.1.3.3. Несбалансированность классов	233
5.1.4. Кривые точность–полнота	233
5.1.4.1. Вычисление точности и полноты	234
5.1.4.2. Обобщение кривых точность–полнота в виде скаляра	234
5.1.4.3. F-мера	235
5.1.4.4. Несбалансированность классов	235
5.1.5. Задачи регрессии	236
5.1.5.1. ℓ_2 -потеря	236
5.1.5.2. ℓ_1 -потеря	237
5.1.5.3. Функция потерь Хьюбера	237
5.1.6. Задачи вероятностного предсказания	238
5.1.6.1. Расхождение КЛ, перекрестная энтропия и логарифмическая потеря	238
5.1.6.2. Правила верной оценки	239
5.2. Байесовская проверка гипотез	240
5.2.1. Пример: проверка симметричности монеты	241
5.2.2. Байесовский выбор модели	242
5.2.2.1. Пример: полиномиальная регрессия	243
5.2.3. Бритва Оккама	244
5.2.4. Связь между перекрестной проверкой и маргинальным правдоподобием	246

5.2.5. Информационные критерии.....	246
5.2.5.1. Байесовский информационный критерий (BIC)	247
5.2.5.2. Информационный критерий Акаике	247
5.2.5.3. Минимальная длина описания (MDL)	248
5.3. Частотная теория принятий решений	248
5.3.1. Вычисление риска оценки.....	248
5.3.1.1. Пример	249
5.3.1.2. Байесовский риск	250
5.3.1.3. Максимальный риск	251
5.3.2. Состоятельные оценки.....	251
5.3.3. Допустимые оценки	252
5.4. Минимизация эмпирического риска	253
5.4.1. Эмпирический риск.....	253
5.4.1.1. Ошибка аппроксимации и ошибка оценивания	254
5.4.1.2. Регуляризованный риск	255
5.4.2. Структурный риск.....	255
5.4.3. Перекрестная проверка	256
5.4.4. Статистическая теория обучения*	257
5.4.4.1. Нахождение границы ошибки обобщения	257
5.4.4.2. VC-размерность	258
5.5. Частотная проверка гипотез*	258
5.5.1. Критерий отношения правдоподобия.....	259
5.5.1.1. Пример: сравнение гауссовых средних	259
5.5.1.2. Простые и сложные гипотезы.....	260
5.5.2. Проверка значимости нулевой гипотезы	260
5.5.3. р-значения	261
5.5.4. О вреде р-значений	261
5.5.5. Почему же не все исповедуют байесовский подход?	264
5.6. Упражнения.....	266
Глава 6. Теория информации	268
6.1. Энтропия	268
6.1.1. Энтропия дискретных случайных величин	268
6.1.2. Перекрестная энтропия	271
6.1.3. Совместная энтропия.....	271
6.1.4. Условная энтропия.....	272
6.1.5. Перплексия	273
6.1.6. Дифференциальная энтропия непрерывных случайных величин*	274
6.1.6.1. Пример: энтропия гауссова распределения.....	274
6.1.6.2. Связь с дисперсией.....	275
6.1.6.3. Дискретизация	275
6.2. Относительная энтропия (расхождение KL)*	275
6.2.1. Определение	276
6.2.2. Интерпретация.....	276
6.2.3. Пример: расхождение КЛ между двумя гауссовыми распределениями.....	276

12 ❖ Содержание

6.2.4. Неотрицательность расхождения КЛ.....	277
6.2.5. Расхождение КЛ и оценка максимального правдоподобия.....	278
6.2.6. Прямое и обратное расхождение КЛ.....	279
6.3. Взаимная информация*.....	280
6.3.1. Определение.....	280
6.3.2. Интерпретация.....	280
6.3.3. Пример.....	282
6.3.4. Условная взаимная информация.....	282
6.3.5. Взаимная информация как «обобщенный коэффициент корреляции».....	283
6.3.6. Нормированная взаимная информация.....	284
6.3.7. Максимальный коэффициент информации.....	285
6.3.8. Неравенство обработки данных.....	287
6.3.9. Достаточные статистики.....	288
6.3.10. Неравенство Фано*.....	288
6.4. Упражнения.....	289
Глава 7. Линейная алгебра.....	292
7.1. Введение.....	292
7.1.1. Обозначения.....	292
7.1.1.1. Векторы.....	292
7.1.1.2. Матрицы.....	293
7.1.1.3. Тензоры.....	294
7.1.2. Векторные пространства.....	295
7.1.2.1. Сложение векторов и умножение вектора на скаляр.....	295
7.1.2.2. Линейная независимость, линейная оболочка и базисы.....	296
7.1.2.3. Линейные отображения и матрицы.....	296
7.1.2.4. Образ и ядро матрицы.....	297
7.1.2.5. Линейная проекция.....	297
7.1.3. Нормы вектора и матрицы.....	298
7.1.3.1. Нормы вектора.....	298
7.1.3.2. Нормы матрицы.....	299
7.1.4. Свойства матриц.....	300
7.1.4.1. След квадратной матрицы.....	300
7.1.4.2. Определитель квадратной матрицы.....	300
7.1.4.3. Ранг матрицы.....	301
7.1.4.4. Числа обусловленности.....	301
7.1.5. Специальные типы матриц.....	303
7.1.5.1. Диагональная матрица.....	303
7.1.5.2. Треугольные матрицы.....	304
7.1.5.3. Положительно определенные матрицы.....	304
7.1.5.4. Ортогональные матрицы.....	305
7.2. Умножение матриц.....	306
7.2.1. Умножение векторов.....	307
7.2.2. Произведение матрицы на вектор.....	307
7.2.3. Произведение матриц.....	308
7.2.4. Приложение: манипулирование матрицами данных.....	310

7.2.4.1. Суммирование срезов матрицы	310
7.2.4.2. Масштабирование строк и столбцов матрицы.....	311
7.2.4.3. Матрица сумм квадратов и матрица рассеяния	311
7.2.4.4. Матрица Грама	312
7.2.4.5. Матрица расстояний	313
7.2.5. Произведения Кронекера*	313
7.2.6. Суммирование Эйнштейна*	314
7.3. Обращение матриц	315
7.3.1. Обращение квадратной матрицы.....	315
7.3.2. Дополнения Шура*	316
7.3.3. Лемма об обращении матрицы*	317
7.3.4. Лемма об определителе матрицы*	318
7.3.5. Приложение: вывод условных распределений для многомерного гауссова распределения	319
7.4. Спектральное разложение	320
7.4.1. Основные сведения.....	320
7.4.2. Диагонализация	321
7.4.3. Собственные значения и собственные векторы симметричных матриц	322
7.4.3.1. Проверка на положительную определенность.....	322
7.4.4. Геометрия квадратичных форм.....	323
7.4.5. Стандартизация и отбеливание данных.....	323
7.4.6. Степенной метод.....	324
7.4.7. Понижение порядка	326
7.4.8. Собственные векторы оптимизируют квадратичные формы.....	326
7.5. Сингулярное разложение (SVD)	327
7.5.1. Основные сведения.....	327
7.5.2. Связь между сингулярным и спектральным разложением.....	328
7.5.3. Псевдообратная матрица.....	329
7.5.4. SVD и образ и ядро матрицы*	330
7.5.5. Усеченное сингулярное разложение	331
7.6. Другие матричные разложения*	332
7.6.1. LU-разложение	332
7.6.2. QR-разложение	333
7.6.3. Разложение Холецки	334
7.6.3.1. Приложение: выборка из многомерного гауссова распределения	334
7.7. Решение систем линейных уравнений*	335
7.7.1. Решение квадратных систем	336
7.7.2. Решение недоопределенных систем (оценка по наименьшей норме).....	336
7.7.3. Решение переопределенных систем (оценка по методу наименьших квадратов)	338
7.8. Матричное исчисление.....	339
7.8.1. Производные	339
7.8.2. Градиенты	340
7.8.3. Производная по направлению	340

7.8.4. Полная производная*	341
7.8.5. Якобиан	341
7.8.5.1. Умножение якобиана на вектор	342
7.8.5.2. Якобиан композиции	342
7.8.6. Гессиан	342
7.8.7. Градиенты часто встречающихся функций	343
7.8.7.1. Функции, отображающие скаляры в скаляры	343
7.8.7.2. Функции, отображающие векторы в скаляры	343
7.8.7.3. Функции, отображающие матрицы в скаляры	344
7.9. Упражнения	345

Глава 8. Оптимизация 346

8.1. Введение	346
8.1.1. Локальная и глобальная оптимизация	346
8.1.1.1. Условия оптимальности для локальных и глобальных оптимумов	347
8.1.2. Условная и безусловная оптимизация	348
8.1.3. Выпуклая и невыпуклая оптимизация	349
8.1.3.1. Выпуклые множества	349
8.1.3.2. Выпуклые функции	350
8.1.3.3. Характеристика выпуклых функций	351
8.1.3.4. Сильно выпуклые функции	352
8.1.4. Гладкая и негладкая оптимизация	353
8.1.4.1. Субградиенты	354
8.2. Методы первого порядка	355
8.2.1. Направление спуска	356
8.2.2. Размер шага (скорость обучения)	356
8.2.2.1. Постоянный размер шага	356
8.2.2.2. Линейный поиск	358
8.2.3. Скорость сходимости	359
8.2.4. Метод импульса	360
8.2.4.1. Импульс	360
8.2.4.2. Момент Нестерова	361
8.3. Методы второго порядка	362
8.3.1. Метод Ньютона	362
8.3.2. BFGS и другие квазиньютоновские методы	364
8.3.3. Методы на основе доверительных областей	365
8.4. Стохастический градиентный спуск	366
8.4.1. Приложение к задачам с конечной суммой	367
8.4.2. Пример: СГС для обучения модели линейной регрессии	368
8.4.3. Выбор размера шага (скорости обучения)	369
8.4.4. Итеративное усреднение	371
8.4.5. Уменьшение дисперсии*	372
8.4.5.1. SVRG	372
8.4.5.2. SAGA	373
8.4.5.3. Применение в глубоком обучении	373
8.4.6. Предобусловленный СГС	374

8.4.6.1. AdaGrad	374
8.4.6.2. RMSProp и AdaDelta.....	375
8.4.6.3. Adam	376
8.4.6.4. Проблемы, связанные с адаптивной скоростью обучения	376
8.4.6.5. Недиagonальные матрицы предобуславливания	377
8.5. Условная оптимизация.....	377
8.5.1. Множители Лагранжа.....	378
8.5.1.1. Пример: двумерная квадратичная целевая функция с одним линейным ограничением в виде равенства.....	379
8.5.2. Условия Каруша–Куна–Таккера	380
8.5.3. Линейное программирование	381
8.5.3.1. Симплекс-метод	382
8.5.3.2. Приложения.....	382
8.5.4. Квадратичное программирование.....	382
8.5.4.1. Пример: квадратичная целевая функция в двумерном случае с линейными ограничениями в виде равенств	383
8.5.4.2. Приложения.....	384
8.5.5. Смешанно-целочисленное программирование*	384
8.6. Проксимальный градиентный метод*	384
8.6.1. Спроецированный градиентный спуск.....	385
8.6.2. Проксимальный оператор для регуляризатора по норме ℓ_1	387
8.6.3. Применение проксимального оператора в случае квантования	388
8.6.4. Инкрементные (онлайновые) проксимальные методы	389
8.7. Граничная оптимизация*	389
8.7.1. Общий алгоритм	389
8.7.2. ЕМ-алгоритм.....	391
8.7.2.1. Нижняя граница.....	392
8.7.2.2. Е-шаг	392
8.7.2.3. М-шаг	393
8.7.3. Пример: ЕМ-алгоритм для смеси гауссовых распределений	394
8.7.3.1. Е-шаг	394
8.7.3.2. М-шаг	394
8.7.3.3. Пример	395
8.7.3.4. Оценка MAP	395
8.7.3.5. Невыпуклость NLL	398
8.8. Оптимизация черного ящика и оптимизация без использования производных.....	399
8.9. Упражнения.....	399

Часть II. ЛИНЕЙНЫЕ МОДЕЛИ.....400

Глава 9. Линейный дискриминантный анализ.....401

9.1. Введение	401
9.2. Гауссов дискриминантный анализ	401
9.2.1. Квадратичные решающие границы.....	402
9.2.2. Линейные решающие границы	403

9.2.3. Связь между ЛДА и логистической регрессией.....	403
9.2.4. Обучение модели	405
9.2.4.1. Связанные ковариационные матрицы	406
9.2.4.2. Диагональные ковариационные матрицы	406
9.2.4.3. Оценка MAP.....	406
9.2.5. Классификатор по ближайшему центроиду	407
9.2.6. Линейный дискриминантный анализ Фишера*	407
9.2.6.1. Нахождение оптимального одномерного направления	409
9.2.6.2. Обобщение на большую размерность и несколько классов	411
9.3. Наивные байесовские классификаторы	412
9.3.1. Примеры моделей.....	413
9.3.2. Обучение модели	413
9.3.3. Байесовская интерпретация наивной байесовской модели	415
9.3.4. Связь между наивной байесовской моделью и логистической регрессией.....	416
9.4. Порождающие и дискриминантные классификаторы.....	417
9.4.1. Преимущества дискриминантных классификаторов	417
9.4.2. Преимущества порождающих классификаторов.....	418
9.4.3. Обработка отсутствующих признаков.....	419
9.5. Упражнения.....	419
Глава 10. Логистическая регрессия	420
10.1. Введение	420
10.2. Бинарная логистическая регрессия.....	420
10.2.1. Линейные классификаторы	421
10.2.2. Нелинейные классификаторы	422
10.2.3. Оценка максимального правдоподобия	423
10.2.3.1. Целевая функция	423
10.2.3.2. Оптимизация целевой функции	424
10.2.3.3. Вывод градиента.....	425
10.2.3.4. Вывод гессиана	426
10.2.4. Стохастический градиентный спуск.....	427
10.2.5. Алгоритм перцептрона.....	427
10.2.6. Метод наименьших квадратов с итеративным пересчетом весов.....	428
10.2.7. Оценка MAP	430
10.2.8. Стандартизация	431
10.3. Мультиномиальная логистическая регрессия	432
10.3.1. Линейные и нелинейные классификаторы	433
10.3.2. Оценка максимального правдоподобия	433
10.3.2.1. Целевая функция	434
10.3.2.2. Оптимизация целевой функции	434
10.3.2.3. Вывод градиента.....	434
10.3.2.4. Вывод гессиана	435
10.3.3. Градиентная оптимизация	436
10.3.4. Граничная оптимизация.....	436
10.3.5. Оценка MAP	438

10.3.6. Классификаторы максимальной энтропии	439
10.3.7. Иерархическая классификация.....	440
10.3.8. Работа с большим числом классов	440
10.3.8.1. Иерархическая softmax-модель.....	441
10.3.8.2. Несбалансированность классов и длинный хвост	441
10.4. Робастная логистическая регрессия*	443
10.4.1. Смесовая модель правдоподобия.....	443
10.4.2. Дважды смягченная потеря	444
10.5. Байесовская логистическая регрессия*	447
10.5.1. Аппроксимация Лапласа	447
10.5.2. Аппроксимация апостериорного прогнозного распределения	449
10.5.2.1. Аппроксимация Монте-Карло.....	451
10.5.2.2. Пробит-аппроксимация	451
10.6. Упражнения.....	452
Глава 11. Линейная регрессия	455
11.1. Введение	455
11.2. Линейная регрессия по методу наименьших квадратов	455
11.2.1. Терминология.....	455
11.2.2. Оценивание по методу наименьших квадратов.....	457
11.2.2.1. Обыкновенный метод наименьших квадратов	457
11.2.2.2. Геометрическая интерпретация метода наименьших квадратов	458
11.2.2.3. Алгоритмические проблемы	460
11.2.2.4. Метод взвешенных наименьших квадратов	461
11.2.3. Другие подходы к вычислению MLE	461
11.2.3.1. Нахождение смещения и углового коэффициента по отдельности.....	461
11.2.3.2. Простая линейная регрессия (одномерные входные данные)	462
11.2.3.3. Частная регрессия	462
11.2.3.4. Рекурсивное вычисление MLE.....	462
11.2.3.5. Вывод MLE с порождающей точки зрения	464
11.2.3.6. Вывод MLE для σ^2	465
11.2.4. Измерение степени согласия оценки	465
11.2.4.1. Графики невязок.....	465
11.2.4.2. Точность предсказания и R^2	466
11.3. Гребневая регрессия	467
11.3.1. Вычисление оценки MAP.....	467
11.3.1.1. Решение с использованием QR-разложения.....	468
11.3.1.2. Решение с использованием сингулярного разложения	469
11.3.2. Связь между гребневой регрессией и PCA.....	469
11.3.3. Выбор силы регуляризатора	471
11.4. Регрессия lasso	471
11.4.1. Оценка MAP с априорным распределением Лапласа (ℓ_1 -регуляризация).....	472
11.4.2. Почему ℓ_1 -регуляризация дает разреженные решения?	473

11.4.3. Жесткие и мягкие пороги	474
11.4.4. Путь регуляризации	476
11.4.5. Сравнение методов наименьших квадратов, lasso, гребневой регрессии и выбора подмножеств	478
11.4.6. Согласованность выбора переменных	479
11.4.7. Групповое lasso	481
11.4.7.1. Приложения	481
11.4.7.2. Штрафование по норме ℓ_2	482
11.4.7.3. Штрафование по норме ℓ_∞	482
11.4.7.4. Пример	483
11.4.8. Эластичная сеть (комбинация гребневой регрессии и lasso)	484
11.4.9. Алгоритмы оптимизации	485
11.4.9.1. Покоординатный спуск	485
11.4.9.2. Спроецированный градиентный спуск	486
11.4.9.3. Проксимальный градиентный спуск	486
11.4.9.4. LARS	486
11.5. Регрессионные сплайны*	487
11.5.1. В-сплайны в качестве базисных функций	488
11.5.2. Обучение линейно модели с помощью сплайнового базиса	489
11.5.3. Сглаживающие сплайны	490
11.5.4. Обобщенные аддитивные модели	490
11.6. Робастная линейная регрессия*	491
11.6.1. Правдоподобие Лапласа	491
11.6.1.1. Вычисление MLE методами линейного программирования	492
11.6.2. t-правдоподобие Стьюдента	493
11.6.3. Функция потерь Хьюбера	493
11.6.4. RANSAC	494
11.7. Байесовская линейная регрессия*	494
11.7.1. Априорные распределения	494
11.7.2. Апостериорные распределения	495
11.7.3. Пример	495
11.7.4. Вычисление апостериорного прогнозного распределения	497
11.7.5. Преимущество центрирования	498
11.7.6. Мультиколлинеарность	499
11.7.7. Автоматическое определение релевантности (ARD)*	501
11.8. Упражнения	502
Глава 12. Обобщенные линейные модели*	505
12.1. Введение	505
12.2. Примеры	506
12.2.1. Линейная регрессия	506
12.2.2. Биномиальная регрессия	506
12.2.3. Регрессия Пуассона	507
12.3. GLM с неканоническими функциями связи	508
12.4. Оценка максимального правдоподобия	509
12.5. Рабочий пример: предсказание обращений за страховыми выплатами	510

Часть III. ГЛУБОКИЕ НЕЙРОННЫЕ СЕТИ	513
Глава 13. Нейронные сети для структурированных данных	514
13.1. Введение	514
13.2. Многослойные перцептроны (МСП)	516
13.2.1. Задача XOR	516
13.2.2. Дифференцируемые МСП	517
13.2.3. Функции активации	518
13.2.4. Примеры моделей	519
13.2.4.1. МСП для классификации двумерных данных по двум категориям	519
13.2.4.2. МСП для классификации изображений	520
13.2.4.3. МСП для классификации текстов	522
13.2.4.4. МСП для гетероскедастической регрессии	523
13.2.5. Важность глубины	524
13.2.6. Революция глубокого обучения	525
13.2.7. Связи с биологией	526
13.3. Обратное распространение	529
13.3.1. Прямой и обратный режим дифференцирования	530
13.3.2. Дифференцирование в обратном режиме для многослойных перцептронов	531
13.3.3. Произведение вектора на якобиан для типичных слоев	533
13.3.3.1. Слой перекрестной энтропии	533
13.3.3.2. Поэлементная нелинейность	534
13.3.3.3. Линейный слой	535
13.3.3.4. Соберем все вместе	536
13.3.4. Графы вычислений	536
13.4. Обучение нейронных сетей	538
13.4.1. Настройка скорости обучения	539
13.4.2. Исчезающие и взрывные градиенты	539
13.4.3. Функции активации без насыщения	540
13.4.3.1. ReLU	542
13.4.3.2. ReLU без насыщения	542
13.4.3.3. Другие варианты	543
13.4.4. Остаточные связи	544
13.4.5. Инициализация параметров	545
13.4.5.1. Эвристические схемы инициализации	545
13.4.5.2. Инициализации, управляемые данными	546
13.4.6. Параллельное обучение	546
13.5. Регуляризация	548
13.5.1. Ранняя остановка	548
13.5.2. Уменьшение весов	548
13.5.3. Разреженные ГНС	548
13.5.4. Прореживание	549
13.5.5. Байесовские нейронные сети	551
13.5.6. Эффекты регуляризации, порождаемые стохастическим градиентным спуском*	551

13.6. Другие виды сетей прямого распространения*	553
13.6.1. Сети радиально-базисных функций	553
13.6.1.1. RBF-сеть для регрессии	554
13.6.1.2. RBF-сеть для классификации	554
13.6.2. Смесь экспертов	555
13.6.2.1. Смесь линейных экспертов	558
13.6.2.2. Глубокие сети экспертов	558
13.6.2.3. Иерархические смеси экспертов	559
13.7. Упражнения	559
Глава 14. Нейронные сети для изображений	561
14.1. Введение	561
14.2. Наиболее употребительные слои	563
14.2.1. Сверточные слои	563
14.2.1.1. Свертка в одномерном случае	563
14.2.1.2. Свертка в двумерном случае	564
14.2.1.3. Свертка как умножение матрицы на вектор	565
14.2.1.4. Граничные условия и дополнение	566
14.2.1.5. Свертка с шагом	568
14.2.1.6. Несколько входных и выходных каналов	568
14.2.1.7. Свертка 1×1 (поточечная)	569
14.2.2. Пулинговые слои	569
14.2.3. Соберем все вместе	571
14.2.4. Слои нормировки	571
14.2.4.1. Пакетная нормировка	572
14.2.4.2. Другие виды слоя нормировки	573
14.2.4.3. Сети без нормировки	575
14.3. Распространенные архитектуры классификации изображений	575
14.3.1. LeNet	575
14.3.2. AlexNet	577
14.3.3. GoogLeNet	578
14.3.4. ResNet	579
14.3.5. DenseNet	581
14.3.6. Поиск архитектуры нейронной сети	581
14.4. Другие формы свертки*	582
14.4.1. Дырявая свертка	582
14.4.2. Транспонированная свертка	583
14.4.3. Пространственная раздельная свертка	584
14.5. Решение других дискриминантных задач компьютерного зрения с помощью СНС*	585
14.5.1. Аннотирование изображений	586
14.5.2. Определение объектов	586
14.5.3. Сегментация экземпляров	588
14.5.4. Семантическая сегментация	589
14.5.5. Оценивание позы человека	590
14.6. Генерирование изображений посредством инвертирования СНС*	591

14.6.1. Преобразование обученного классификатора в порождающую модель.....	592
14.6.2. Априорные распределения изображений.....	592
14.6.2.1. Гауссово априорное распределения	593
14.6.2.2. Априорное распределение на основе полной вариации	594
14.6.3. Визуализация признаков, обученных с помощью СНС	595
14.6.4. Deep Dream.....	595
14.6.5. Нейронный перенос стиля	597
14.6.5.1. Как это работает	598
14.6.5.2. Ускорение метода	600
Глава 15. Нейронные сети для последовательностей.....	602
15.1. Введение	602
15.2. Рекуррентные нейронные сети (РНС).....	602
15.2.1. Vec2Seq (генерирование последовательностей)	602
15.2.1.1. Модели	603
15.2.1.2. Приложения.....	604
15.2.2. Seq2Vec (классификация последовательностей).....	606
15.2.3. Seq2Seq (трансляция последовательностей)	607
15.2.3.1. Выровненный случай.....	607
15.2.3.2. Невыровненный случай	608
15.2.4. Принуждение со стороны учителя	609
15.2.5. Обратное распространение во времени	610
15.2.6. Исчезающие и взрывные градиенты	612
15.2.7. Вентильная и долгосрочная память.....	612
15.2.7.1. Управляемые рекуррентные блоки (GRU).....	612
15.2.7.2. Долгая краткосрочная память (LSTM)	613
15.2.8. Лучевой поиск	615
15.3. Одномерные СНС	617
15.3.1. Применение одномерных СНС для классификации последовательностей	618
15.3.2. Применение каузальных одномерных СНС для генерирования последовательностей	618
15.4. Модель внимания.....	620
15.4.1. Механизм внимания как мягкий поиск в словаре	620
15.4.2. Ядерная регрессия как непараметрическое внимание	622
15.4.3. Параметрическое внимание	623
15.4.4. Модель Seq2Seq с вниманием	624
15.4.5. Модель Seq2Vec с вниманием (классификация текста).....	626
15.4.6. Модель Seq+Seq2Vec с вниманием (классификация пар предложений).....	627
15.4.7. Мягкое и жесткое внимание.....	629
15.5. Трансформеры.....	629
15.5.1. Самовнимание	630
15.5.2. Многопутевое внимание	632
15.5.3. Позиционное кодирование	632
15.5.4. Соберем все вместе.....	634

15.5.5. Сравнение трансформеров, СНС и НУС.....	636
15.5.6. Применение трансформеров для изображений*	636
15.5.7. Другие варианты трансформеров*	638
15.6. Эффективные трансформеры*	639
15.6.1. Фиксированные необучаемые локализованные паттерны внимания	639
15.6.2. Обучаемые паттерны разреженного внимания.....	640
15.6.3. Методы с добавлением памяти и рекуррентные методы	640
15.6.4. Низкоранговые и ядерные методы.....	640
15.7. Языковые модели и обучение представлений без учителя	643
15.7.1. ELMo	643
15.7.2. BERT.....	644
15.7.2.1. Замаскированная языковая модель	645
15.7.2.2. Задача предсказания следующего предложения	645
15.7.2.3. Дообучение BERT для приложений NLP	647
15.7.3. GPT	649
15.7.3.1. Приложения GPT	649
15.7.4. T5	649
15.7.5. Обсуждение	650

Часть IV. НЕПАРАМЕТРИЧЕСКИЕ МОДЕЛИ 652

Глава 16. Методы на основе эталонов 653

16.1. Классификация методом К ближайших соседей (KNN)	653
16.1.1. Пример	654
16.1.2. Проклятие размерности	655
16.1.3. Снижение требований к скорости и памяти	656
16.1.4. Распознавание открытого множества	657
16.1.4.1. Онлайнное обучение, обнаружение посторонних и распознавание открытого множества.....	657
16.1.4.2. Другие задачи открытого мира	658
16.2. Обучение метрик	658
16.2.1. Линейные и выпуклые методы.....	659
16.2.1.1. Метод ближайших соседей с большим зазором	659
16.2.1.2. Анализ компонентов соседства.....	660
16.2.1.3. Анализ латентных совпадений	660
16.2.2. Глубокое обучение метрики.....	661
16.2.3. Потери классификации.....	662
16.2.4. Потери ранжирования	662
16.2.4.1. Попарная (сопоставительная) потеря и сиамские сети.....	663
16.2.4.2. Триплетная потеря.....	663
16.2.4.3. N-парная потеря	664
16.2.5. Ускорение оптимизации потери ранжирования	665
16.2.5.1. Добычные методы	665
16.2.5.2. Методы на основе представителей.....	665
16.2.5.3. Оптимизация верхней границы.....	666

16.2.6. Другие приемы глубокого обучения метрики.....	668
16.3. Ядерные оценки плотности.....	669
16.3.1. Ядра плотности	669
16.3.2. Оконная оценка плотности Парцена	670
16.3.3. Как выбирать полосу пропускания	672
16.3.4. От KDE к KNN-классификации	672
16.3.5. Ядерная регрессия	673
16.3.5.1. Оценка среднего Надарая–Ватсона.....	673
16.3.5.2. Оценка дисперсии.....	675
16.3.5.3. Локально взвешенная регрессия.....	675
Глава 17. Ядерные методы*	676
17.1. Ядра Мерсера.....	676
17.1.1. Теорема Мерсера	678
17.1.2. Некоторые популярные ядра Мерсера.....	678
17.1.2.1. Стационарные ядра для вещественных векторов	678
17.1.2.2. Создание новых ядер из существующих.....	681
17.1.2.3. Комбинирование ядер с помощью сложения и умножения	682
17.1.2.4. Ядра для структурированных входов	683
17.2. Гауссовы процессы	683
17.2.1. Незашумленные наблюдения.....	684
17.2.2. Зашумленные наблюдения.....	685
17.2.3. Сравнение с ядерной регрессией	686
17.2.4. Пространство весов и пространство функций.....	687
17.2.5. Численные проблемы	688
17.2.6. Оценивание параметров ядра.....	688
17.2.6.1. Эмпирическая байесовская оценка	689
17.2.6.2. Байесовский вывод.....	691
17.2.7. Применение гауссовых процессов для классификации	692
17.2.8. Связи с глубоким обучением.....	694
17.2.9. Масштабирование ГП на большие наборы данных	694
17.2.9.1. Разреженные аппроксимации	694
17.2.9.2. Распараллеливание с использованием структуры ядерной матрицы.....	694
17.2.9.3. Аппроксимация случайными признаками.....	695
17.3. Метод опорных векторов	696
17.3.1. Классификаторы с широким зазором.....	697
17.3.2. Двойственная задача	699
17.3.3. Классификаторы с мягким зазором	701
17.3.4. Ядерный трюк.....	702
17.3.5. Преобразование выходов SVM в вероятности.....	703
17.3.6. Связь с логистической регрессией	704
17.3.7. Многоклассовая классификация с применением SVM.....	705
17.3.8. Как выбирать регуляризатор C	706
17.3.9. Ядерная гребневая регрессия.....	707
17.3.10. Применение SVM для регрессии.....	708
17.4. Метод разреженных векторов	711

17.4.1. Метод релевантных векторов.....	711
17.4.2. Сравнение разреженных и плотных ядерных методов	711
17.5. Упражнения	715

Глава 18. Деревья, леса, бэггинг и бустинг

18.1. Деревья классификации и регрессии.....	716
18.1.1. Определение модели.....	716
18.1.2. Обучение модели	717
18.1.3. Регуляризация	719
18.1.4. Обработка отсутствующих входных признаков	720
18.1.5. Плюсы и минусы	720
18.2. Ансамблевое обучение	721
18.2.1. Стековое обобщение	722
18.2.2. Ансамблевое обучение не то же, что байесовское усреднение моделей	722
18.3. Бэггинг	723
18.4. Случайные леса.....	724
18.5. Бустинг.....	725
18.5.1. Прямое поэтапное аддитивное моделирование.....	726
18.5.2. Квадратичная потеря и бустинг наименьших квадратов.....	727
18.5.3. Экспоненциальная потеря и AdaBoost	727
18.5.4. LogitBoost	731
18.5.5. Градиентный бустинг	732
18.5.5.1. Градиентный бустинг деревьев	734
18.5.5.2. XGBoost	734
18.6. Интерпретация ансамблей деревьев	736
18.6.1. Важность признаков.....	736
18.6.2. Графики частичной зависимости.....	738

Часть V. ЗА ПРЕДЕЛАМИ ОБУЧЕНИЯ С УЧИТЕЛЕМ

Глава 19. Обучение при меньшем числе помеченных примеров

19.1. Приращение данных.....	740
19.1.1. Примеры.....	740
19.1.2. Теоретическое обоснование.....	741
19.2. Перенос обучения	742
19.2.1. Дообучение	742
19.2.2. Адаптеры.....	744
19.2.3. Предобучение с учителем.....	745
19.2.4. Предобучение без учителя (самостоятельное обучение)	746
19.2.4.1. Задачи подстановки.....	747
19.2.4.2. Замещающие задачи.....	748
19.2.4.3. Сопоставительные задачи.....	748
19.2.4.4. SimCLR.....	748
19.2.4.5. CLIP	751

19.2.5. Адаптация домена	752
19.3. Обучение с частичным привлечением учителя	753
19.3.1. Самообучение и псевдопометка	754
19.3.2. Минимизация энтропии	755
19.3.2.1. Кластерное допущение	756
19.3.2.2. Взаимная информация между входом и выходом	757
19.3.3. Совместное обучение	758
19.3.4. Распространение меток на графах	759
19.3.5. Регуляризация по согласованности	760
19.3.6. Глубокие порождающие модели*	762
19.3.6.1. Вариационные автокодировщики	763
19.3.6.2. Порождающие состязательные сети	765
19.3.6.3. Нормализующие потоки	766
19.3.7. Сочетание самостоятельного обучения и обучения с частичным привлечением учителя	767
19.4. Активное обучение	768
19.4.1. Подход на основе теории принятия решений	769
19.4.2. Теоретико-информационный подход	769
19.4.3. Пакетное активное обучение	770
19.5. Метаобучение	770
19.5.1. Метаобучение, не зависящее от модели (MAML)	771
19.6. Обучение на малом числе примеров	772
19.6.1. Сопоставляющие сети	773
19.7. Обучение со слабым учителем	774
19.8. Упражнения	775
Глава 20. Понижение размерности	776
20.1. Метод главных компонент	776
20.1.1. Примеры	777
20.1.2. Вывод алгоритма	779
20.1.2.1. Базовый случай	779
20.1.2.2. Оптимальный вектор весов максимизирует дисперсию спроецированных данных	780
20.1.2.3. Шаг индукции	781
20.1.3. Вычислительные трудности	782
20.1.3.1. Ковариационная матрица и корреляционная матрица	782
20.1.3.2. Работа с данными высокой размерности	783
20.1.3.3. Вычисление PCA с использованием SVD	783
20.1.4. Выбор числа латентных измерений	784
20.1.4.1. Ошибка реконструкции	784
20.1.4.2. Графики каменистой осыпи	785
20.1.4.3. Правдоподобие профиля	785
20.2. Факторный анализ*	787
20.2.1. Порождающая модель	787
20.2.2. Вероятностный PCA	789
20.2.3. EM-алгоритм для ФА/PPCA	790
20.2.3.1. EM-алгоритм для ФА	791

20.2.3.2. EM-алгоритм для (P)PCA	791
20.2.3.3. Преимущества	792
20.2.4. Неидентифицируемость параметров	794
20.2.5. Нелинейный факторный анализ	795
20.2.6. Смеси факторных анализаторов	795
20.2.7. Факторный анализ экспоненциального семейства	797
20.2.7.1. Пример: бинарный PCA	798
20.2.7.2. Пример: категориальный PCA	798
20.2.8. Модели факторного анализа для парных данных	799
20.2.8.1. PCA с учителем	799
20.2.8.2. Метод частичных наименьших квадратов	800
20.2.8.3. Канонический корреляционный анализ	801
20.3. Автокодировщики	802
20.3.1. Автокодировщики с сужением	802
20.3.2. Шумоподавляющие автокодировщики	804
20.3.3. Сжимающие автокодировщики	806
20.3.4. Разреженные автокодировщики	806
20.3.5. Вариационные автокодировщики	808
20.3.5.1. Обучение VAE	809
20.3.5.2. Перепараметризация	809
20.3.5.3. Сравнение VAE с автокодировщиками	811
20.4. Обучение многообразий*	813
20.4.1. Что такое многообразие?	813
20.4.2. Гипотеза многообразия	814
20.4.3. Подходы к обучению многообразий	815
20.4.4. Многомерное шкалирование	816
20.4.4.1. Классическое ММШ	816
20.4.4.2. Метрическое ММШ	817
20.4.4.3. Неметрическое ММШ	818
20.4.4.4. Отображение Саммона	818
20.4.5. Isomap	819
20.4.6. Ядерный PCA	820
20.4.7. Максимальное раскрытие дисперсии	822
20.4.8. Локально линейное погружение	823
20.4.9. Лапласовы собственные отображения	824
20.4.9.1. Использование собственных векторов лапласиана графа для вычисления погружений	824
20.4.9.2. Что такое лапласиан графа?	825
20.4.10. t-SNE	827
20.4.10.1. Стохастическое погружение соседей	827
20.4.10.2. Симметричное SNE	829
20.4.10.3. SNE с t-распределением	829
20.4.10.4. Выбор линейного масштаба	830
20.4.10.5. Вычислительные проблемы	831
20.4.10.6. UMAP	831
20.5. Погружения слов	832
20.5.1. Латентно-семантический анализ и индексирование	832

20.5.1.1. Латентно-семантическое индексирование	832
20.5.1.2. Латентно-семантический анализ	833
20.5.1.3. Поточечная взаимная информация	834
20.5.2. Word2vec	835
20.5.2.1. Модель Word2vec CBOW	835
20.5.2.2. Скипграммная модель Word2vec	835
20.5.2.3. Отрицательная выборка	836
20.5.3. GloVE	837
20.5.4. Аналогичные слова	838
20.5.5. Модель погружений слов RAND-WALK	839
20.5.6. Контекстуальные погружения слов	840
20.6. Упражнения	840
Глава 21. Кластеризация	843
21.1. Введение	843
21.1.1. Оценивание выхода методов кластеризации	843
21.1.1.1. Чистота	844
21.1.1.2. Индекс Рэнда	844
21.1.1.3. Взаимная информация	845
21.2. Иерархическая агломеративная кластеризация	846
21.2.1. Алгоритм	847
21.2.1.1. Одиночная связь	848
21.2.1.2. Полная связь	848
21.2.1.3. Средняя связь	849
21.2.2. Пример	849
21.2.3. Расширения	850
21.3. Кластеризация методом К средних	851
21.3.1. Алгоритм	851
21.3.2. Примеры	852
21.3.2.1. Кластеризация точек на плоскости	852
21.3.2.2. Кластеризация временных рядов экспрессии генов дрожжей	852
21.3.3. Векторное квантование	853
21.3.4. Алгоритм K-means++	854
21.3.5. Алгоритм К медоидов	855
21.3.6. Способы ускорения	856
21.3.7. Выбор числа кластеров K	857
21.3.7.1. Минимизация искажения	857
21.3.7.2. Максимизация маргинального правдоподобия	857
21.3.7.3. Силуэтный коэффициент	858
21.3.7.4. Инкрементное увеличение количества компонент смеси	860
21.3.7.5. Методы разреженного оценивания	860
21.4. Кластеризация с помощью смесовых моделей	860
21.4.1. Смеси гауссовых распределений	860
21.4.1.1. Метод К средних – частный случай ЕМ-алгоритма	861
21.4.1.2. Неидентифицируемость и переключение метки	861
21.4.1.3. Байесовский выбор модели	864

21.4.2. Смеси распределений Бернулли.....	865
21.5. Спектральная кластеризация*	865
21.5.1. Нормализованные разрезы.....	866
21.5.2. Собственные векторы лапласиана графа кодируют кластеризацию	866
21.5.3. Пример	867
21.5.4. Связь с другими методами	868
21.5.4.1. Связь с kPCA	868
21.5.4.2. Связь с анализом случайного блуждания	868
21.6. Бикластеризация*	869
21.6.1. Базовая бикластеризация.....	869
21.6.2. Модели вложенного разбиения (Crosscat)	870

Глава 22. Рекомендательные системы

22.1. Явная обратная связь	873
22.1.1. Наборы данных	874
22.1.2. Коллаборативная фильтрация	874
22.1.3. Матричная факторизация	875
22.1.3.1. Вероятностная матричная факторизация	876
22.1.3.2. Пример: Netflix	876
22.1.3.3. Пример: MovieLens.....	877
22.1.4. Автокодировщики	878
22.2. Неявная обратная связь	879
22.2.1. Байесовское персонализированное ранжирование	880
22.2.2. Машины факторизации	881
22.2.3. Нейронная матричная факторизация	882
22.3. Использование побочной информации	882
22.4. Компромисс между исследованием и использованием.....	884

Глава 23. Погружения графов*

23.1. Введение	885
23.2. Погружение графа как задача о кодировщике и декодере	887
23.3. Поверхностные погружения графов	889
23.3.1. Обучение погружений без учителя	889
23.3.2. На основе расстояния: евклидовы методы	890
23.3.3. На основе расстояния: неевклидовы методы.....	890
23.3.4. На основе внешнего произведения: методы матричной факторизации.....	891
23.3.5. На основе внешнего произведения: скипграммные методы	892
23.3.6. Обучение погружений с учителем	894
23.3.6.1. Распространение меток.....	894
23.4. Графовые нейронные сети.....	895
23.4.1. Графовые нейронные сети передачи сообщений.....	895
23.4.2. Спектральные свертки графов.....	897
23.4.3. Пространственные свертки графов	897
23.4.3.1. Выборочные пространственные методы.....	898

23.4.3.2. Пространственные методы на основе механизма внимания	898
23.4.3.3. Геометрические пространственные методы.....	899
23.4.4. Неевклидовы графовые свертки	899
23.5. Глубокие погружения графов	900
23.5.1. Обучение погружений без учителя.	900
23.5.1.1. Структурное погружение с помощью глубокой сети	900
23.5.1.2. Вариационные графовые автокодировщики.....	901
23.5.1.3. Итеративное порождающее моделирование графов (Graphite).....	902
23.5.1.4. Методы на основе сопоставительных потерь	902
23.5.2. Обучение погружений с частичным привлечением учителя	903
23.5.2.1. SemiEmb	903
23.5.2.2. Planetoid.....	903
23.6. Приложения	904
23.6.1. Приложения без учителя	904
23.6.1.1. Реконструкция графа	904
23.6.1.2. Предсказание связей.....	905
23.6.1.3. Кластеризация	906
23.6.1.4. Визуализация	906
23.6.2. Приложения с учителем.....	907
23.6.2.1. Классификация вершин	907
23.6.2.2. Классификация графов.....	907

Приложение А. Обозначения..... 909

A.1. Введение	909
A.2. Общепринятые математические символы.....	909
A.3. Функции	910
A.3.1. Функции с одним аргументом	910
A.3.2. Функции двух аргументов	910
A.3.3. Функции более двух аргументов.....	911
A.4. Линейная алгебра	911
A.4.1. Общие обозначения.....	911
A.4.2. Векторы.....	911
A.4.3. Матрицы	912
A.4.4. Матричное исчисление	912
A.5. Оптимизация	913
A.6. Вероятность.....	913
A.7. Теория информации.....	914
A.8. Статистика и машинное обучение.....	915
A.8.1. Обучение с учителем	915
A.8.2. Обучение без учителя и порождающие модели	915
A.8.3. Байесовский вывод	916
A.9. Аббревиатуры.....	916

Предметный указатель..... 918

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Предисловие

В 2012 году я опубликовал 1200-страничную книгу под названием «Machine Learning: A Probabilistic Perspective», в которой сложившаяся на тот момент дисциплина машинного обучения (МО) достаточно полно рассматривалась сквозь объединяющую призму вероятностного моделирования. Книга была хорошо принята и получила премию де Грута (<https://bayesian.org/project/degroot-prize/>) в 2013 году.

2012-й также принято считать началом «революции глубокого обучения». Термином «глубокое обучение» называют раздел МО, основанный на нейронных сетях с большим количеством слоев (отсюда и слово «глубокий»). Хотя базовая технология к тому времени существовала уже много лет, именно в 2012 году в работе [KSH12] глубокие нейронные сети (ГНС) выиграли конкурс по классификации изображений ImageNet с таким отрывом, что это привлекло внимание профессионального сообщества. Примерно в то же время был достигнут прогресс в таких трудных задачах, как распознавание речи (см., например, [Cir+10; Cir+11; Hin+12]). Эти прорывы стали возможными благодаря достижениям в развитии оборудования (в частности, перепрофилированию быстрых графических процессоров, GPU, с видеоигр на МО), технологиях сбора данных (в частности, применению краудсорсинговых инструментов типа «Механического турка» от Amazon для сбора больших размеченных наборов данных, как в ImageNet), а также различным новым алгоритмическим идеям; некоторые из них мы рассмотрим в этой книге.

Начиная с 2012 года, область глубокого обучения развивалась лавинообразно, новые результаты появлялись со все возрастающей скоростью. Столь же лавинообразно рос и интерес к этой области, подогреваемый коммерческим успехом технологии. Поэтому в 2018 году я решил написать второе издание своей книги и попытаться подвести какой-то итог.

К марту 2020 года черновой вариант второго издания разросся до 1600 страниц, а еще много тем оставались неосвещенными. В результате издательство MIT Press порекомендовало мне разбить книгу на два тома. Но тут разразилась пандемия COVID-19. Я решил на время отвлечься от написания книги и поучаствовать в разработке алгоритма оценки рисков для разрабатываемого Google приложения, уведомляющего о подверженности заражению [MKS21], а также в других проектах, связанных с прогнозированием [Wah+21]. Однако к осени 2020 года я решил вернуться к работе над книгой.

Чтобы наверстать упущенное, я попросил нескольких коллег помочь мне в написании различных разделов (см. благодарности ниже). В результате появились две новые книги «Вероятностное машинное обучение: введение», которую вы сейчас читаете, и «Вероятностное машинное обучение: дополнительные вопросы», которая является ее продолжением [Mur22]. Надеюсь, что в совокупности они дают довольно полное представление о состоянии дел в МО в 2021 году – сквозь ту же объединяющую призму вероятностного

моделирования и байесовской теории принятия решений, которая использовалась в книге 2012 года.

Почти весь материал первого издания сохранен, но теперь он более-менее равномерно распределен между двумя новыми книгами. Кроме того, в каждую книгу включено много дополнительных тем из области глубокого обучения и других разделов теории, например порождающие модели, вариационный вывод и обучение с подкреплением.

Чтобы сделать эту вводную книгу более замкнутой и полезной студентам, я добавил ряд сведений по базовым дисциплинам, в частности оптимизации и линейной алгебре, которых за скудостью места не было в издании 2012 года. Материал повышенной сложности, который можно опустить в курсе вводного уровня, помечен звездочкой * в названии раздела или главы. В конце некоторых глав имеются упражнения. Решения упражнений, помеченных звездочкой, могут быть предоставлены преподавателям при обращении к издательству MIT Press; решения остальных упражнений можно найти в сети. Дополнительные учебные материалы (например, рисунки и слайды) см. на сайте книги [probml.ai](https://probml.github.io/pml-book/) (<https://probml.github.io/pml-book/>).

Еще одно существенное изменение – использование Python вместо Matlab во всех примерах программ. (В будущем мы, возможно, создадим версию кода на Julia.) В новом коде используются такие стандартные Python-библиотеки, как NumPy (<https://numpy.org/>), Scikit-learn (<https://scikit-learn.org/stable/>), JAX (<https://github.com/google/jax>), PyTorch (<https://pytorch.org/>), TensorFlow (<https://www.tensorflow.org/>), PyMC3 (<https://docs.pymc.io/en/v3/>) и др. О том, как использовать код, см. на странице по адресу <https://probml.github.io/pml-book/>.

Благодарности

Я выражаю благодарность следующим людям, помогавшим мне при написании книги:

- Зико Колтеру (Zico Kolter), помогавшему писать части главы 7 («Линейная алгебра»);
- Фредерику Кунстнеру (Frederik Kunstner), Си И Менгу (Si Yi Meng), Аарону Мишкину (Aaron Mishkin), Шарану Васвани (Sharan Vaswani) и Марку Шмидту (Mark Schmidt), помогавшим писать части главы 8 («Оптимизация»);
- Матью Блонделю (Mathieu Blondel), помогавшему писать раздел 13.3 («Обратное распространение»);
- Кшиштофу Хоромански (Krzysztof Choromanski), помогавшему писать раздел 15.6 («Эффективные преобразователи»*);
- Колину Раффелю (Colin Raffel), помогавшему писать раздел 19.2 («Перенос обучения») и раздел 19.3 («Обучение с частичным привлечением учителя»);
- Брайану Пероцци (Bryan Perozzi), Сами Абу Эль Хайджа (Sami Abu-El-Haija) и Инес Чами (Ines Chami), помогавшим писать главу 23 («Погружения графов»*);
- Джону Фэнсу (John Fearn) и Питеру Черно (Peter Cerno) за внимательную вычитку книги;

- многочисленных членов сообщества github за поиск опечаток и других ошибок (см. перечень таковых по адресу <https://github.com/probml/pml-book/issues?q=is:issue>);
- четырем анонимным рецензентам, выбранным MIT Press;
- Махмуду Солиману (Mahmoud Soliman), который написал весь код, связующий latex, colab, github и пр., и поведал мне о GCP и TPU;
- всей когорте студентов, работавших над кодом книги на Google Summer of Code 2021 года; это Алейна Кара (Aleyna Kara), Срикаар Джилугу (Srikar Jilugu), Дришти Патель (Drishti Patel), Минь Лиан Анг (Ming Liang Ang), Жерардо Дюран-Мартен (Gerardo Durán-Martín) (см. перечень их трудов по адресу <https://probml.github.io/pml-book/gsoc2021.html>);
- многочисленным членам сообщества github за предложенный ими код (см. <https://github.com/probml/pyprobml#acknowledgements>);
- авторам работ [Zha+20], [Gér17] и [Mar18], разрешившим мне использовать или модифицировать части открытого исходного кода, включенного в их замечательные книги;
- моему начальнику в Google, Дугу Эку (Doug Eck), который позволил мне тратить принадлежащее компании время на написание этой книги;
- своей жене Маргарет, позволившей мне тратить принадлежащее семье время на эту книгу.

Об обложке

На обложке изображена нейронная сеть (глава 13), которая используется для отнесения рукописной цифры x к одному из десяти классов меток $y \in \{0, 1, \dots, 9\}$. Гистограмма справа – вывод модели, соответствующий условному распределению вероятности $p(y|x)$.

Кэвин Патрик Мэрфи
Паль-Альто, Калифорния
август 2021

Глава 1

Введение

1.1. Что такое машинное обучение?

Популярное определение **машинного обучения**, или **МО**, принадлежащее Тому Митчеллу [Mit97], звучит следующим образом:

Говорят, что компьютерная программа обучается на опыте E относительно некоторого класса задач T и меры качества P , если ее качество на задачах, принадлежащих T , измеренное в соответствии с P , улучшается с увеличением опыта E .

Таким образом, существует много видов машинного обучения, зависящих от природы задачи T , решению которой мы хотим обучить систему, природы меры качества P , используемой для оценки работы системы, и природы обучающего сигнала, или опыта E , на котором обучается система.

В этой книге мы будем рассматривать наиболее распространенные типы МО, но с **вероятностной точки зрения**. Грубо говоря, это означает, что все неизвестные переменные (например, предсказания будущего значения некоторой величины, скажем температуры на завтра или параметров некоторой модели) рассматриваются как **случайные величины**, имеющие **распределение вероятностей**, которое описывает взвешенное множество возможных значений переменной (см. краткое введение в основы теории вероятностей в главе 2).

Есть две основные причины, чтобы отдать предпочтение вероятностному подходу. Во-первых, он оптимален для **принятия решений в условиях неопределенности**, это будет объяснено в разделе 5.1. Во-вторых, вероятностное моделирование – язык, используемый в большинстве других областей науки и техники, так что мы получаем единую систему понятий. Шакир Мохамед (научный сотрудник компании DeepMind)¹ говорил:

Почти все машинное обучение можно представить в вероятностных терминах, так что вероятностный подход является фундаментальным. Такое представление, конечно, не единственное. Но именно оно позволяет

¹ Источник: слайд 2 на странице <https://bit.ly/3pyHyPn>.

связать машинное обучение с другими областями вычислительных наук, будь то стохастическая оптимизация, теория управления, исследование операций, эконометрика, теория информации, статистическая физика или биостатистика. Уже по этой причине умение рассуждать в вероятностных терминах обязательно.

1.2. ОБУЧЕНИЕ С УЧИТЕЛЕМ

Самая распространенная форма МО – **обучение с учителем**. В этом случае задача T заключается в том, чтобы обучиться отображению f множества входов $x \in \mathcal{X}$ в множество выходов $y \in \mathcal{Y}$. Входы x называются также **признаками**, **ковариатами** или **предикторами**; часто это числовой вектор фиксированной размерности, например рост и вес человека или пиксели изображения. В таком случае $\mathcal{X} = \mathbb{R}^D$, где D – размерность вектора (т. е. количество входных признаков). Выход y называется также **меткой**, **целью** или **откликом**¹. Опыт E задается в виде множества N пар вход–выход $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$, которое называется **обучающим набором** (а N – **размером выборки**). Мера качества P зависит от типа предсказываемого выхода, мы обсудим это ниже.

1.2.1. Классификация

В задачах **классификации** пространство выходов \mathcal{S} неупорядочено, а метки называются **классами**, $\mathcal{Y} = \{1, 2, \dots, C\}$. Проблема предсказания метки класса заданного входа называется также **распознаванием образов**. (Если имеется всего два класса, часто обозначаемых $y \in \{0, 1\}$ или $y \in \{-1, +1\}$, то говорят о **бинарной классификации**.)

1.2.1.1. Пример: классификация ирисов

В качестве примера рассмотрим проблему классификации ирисов с отношением к трем видам: щетиный (Setosa), разноцветный (Versicolor) и виргинский (Virginica). На рис. 1.1 показано по одному примеру из каждого класса.

В задаче **классификации изображений** пространством входов \mathcal{X} является множество изображений, имеющее очень высокую размерность: для цветного изображения с $C = 3$ каналами (например, в формате RGB) и $D_1 \times D_2$ пикселями имеем $\mathcal{X} = \mathbb{R}^D$, где $D = C \times D_1 \times D_2$. (На практике яркость пикселя представлена целым числом, обычно в диапазоне $\{0, 1, \dots, 255\}$, но для простоты обозначений мы предполагаем, что входы – вещественные числа.) Обучиться отображению $f: \mathcal{X} \rightarrow \mathcal{Y}$ изображений в метки очень трудно, как легко видеть из рис. 1.2. Однако эту задачу все же можно решить с помощью

¹ Иногда (например, в Python пакете statsmodels [https://www.statsmodels.org/devel/endog_exog.html]) входы x называются **экзогенными переменными**, а выходы y – **эндогенными переменными**.

функций специального вида, например **сверточной нейронной сети** (CHC) (convolutional neural network – CNN), которую мы будем обсуждать в разделе 14.1.

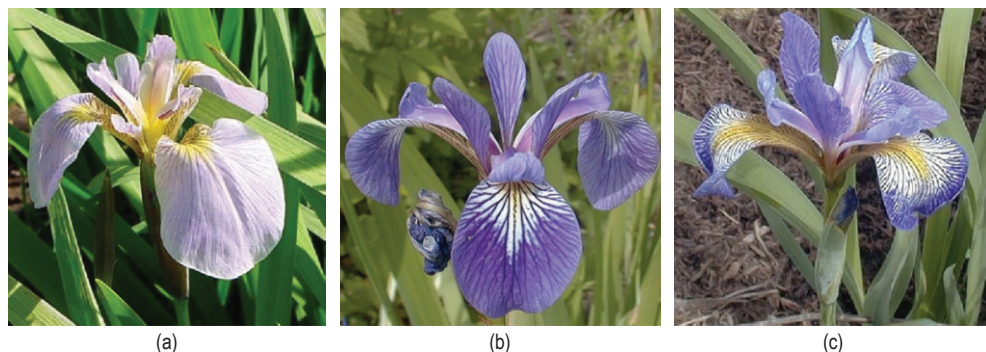


Рис. 1.1 ❖ Три типа ирисов: щетиный (Setosa), разноцветный (Versicolor) и виргинский (Virginica).
Печатается с разрешения Денниса Крэмба и компании SIGNA

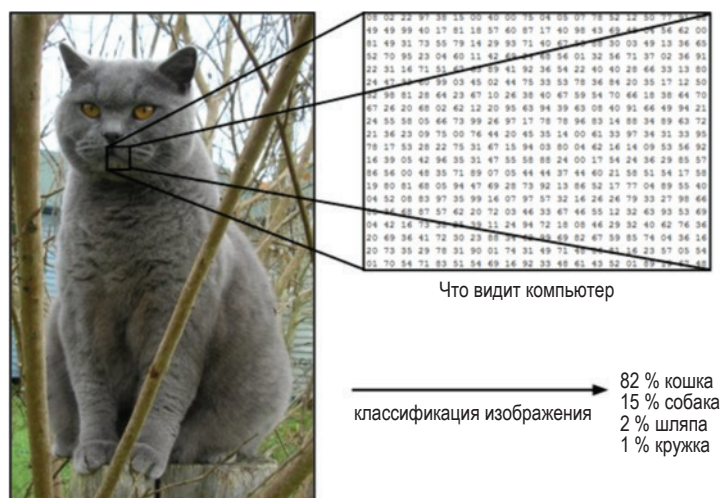


Рис. 1.2 ❖ Иллюстрация к проблеме классификации изображений.
Взято с сайта <https://cs231n.github.io/>.
Печатается с разрешения Андрея Карпатого

К счастью для нас, ботаники уже выделили 4 простых, но весьма информативных числовых признака – длину чашелистика, ширину чашелистика, длину лепестка и ширину лепестка, – с помощью которых можно различить три вида ирисов. В этом разделе мы для простоты будем использовать именно такое пространство входов куда более низкой размерности: $\mathcal{X} = \mathbb{R}^4$. **Набор данных Iris** представляет собой совокупность 150 помеченных примеров ирисов, по 50 каждого вида, описываемых этими четырьмя признаками. Он широко ис-

пользуется в качестве примера, потому что невелик и прост для понимания. (Далее в книге нам встретятся наборы данных побольше и посложнее.)

Небольшие наборы признаков часто хранят в виде **матрицы плана** размера $N \times D$, в которой строки представляют примеры, в столбцы – признаки, см. пример в табл. 1.1¹.

Таблица 1.1. Подмножество матрицы плана для ирисов.

Признаки: длина чашелистика (*sl*), ширина чашелистика (*sw*), длина лепестка (*pl*), ширина лепестка (*pw*). В каждом классе по 50 примеров

Индекс	sl	sw	pl	pw	Метка
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
...
50	7.0	3.2	4.7	1.4	Versicolor
...
149	5.9	3.0	5.1	1.8	Virginica

Набор данных Iris – пример **табличных данных**. Если размер входных данных переменный (например, последовательности слов или социальные сети), в отличие от векторов фиксированной длины, то данные обычно хранятся в формате, отличном от матрицы плана. Однако такие данные часто преобразуются в представление с признаками фиксированного размера (этот процесс называется **фичеризацией**), т. е. неявно создается матрица плана для последующей обработки. Пример такого рода мы приведем в разделе 1.5.4.1, когда будем обсуждать «представление» последовательностей в виде «мешка слов».

1.2.1.2. Разведочный анализ данных

Прежде чем применять к какой-то задаче МО, полезно провести **разведочный анализ данных** и выяснить, есть ли очевидные закономерности (их наличие может подсказать, какой метод выбрать) или очевидные проблемы (например, зашумленные метки или выбросы).

Для табличных данных с небольшим числом признаков часто строят **график парных отношений**, на котором панель (i, j) содержит диаграмму рассеяния величин i и j , а диагональные элементы (i, i) показывают маргинальную плотность величины i ; дополнительно все графики могут быть окрашены цветом, кодирующим метку классу, см. пример на рис. 1.3.

Для данных более высокой размерности зачастую сначала выполняют **понижение размерности**, а затем визуализируют данные в двух или трех измерениях. Методы понижения размерности мы будем обсуждать в главе 20.

¹ В этой конкретной матрице плана $N = 150$ строк и $D = 4$ столбцов, т. е. она высокая и узкая, поскольку $N \gg D$. Встречаются также наборы данных (например, в геномике), в которых признаков больше, чем примеров, т. е. $D \gg N$; для них матрица плана низкая и широкая. Термин **«большие данные»** обычно означает, что N велико, а термин **«широкие данные»** – что D велико (по сравнению с N).

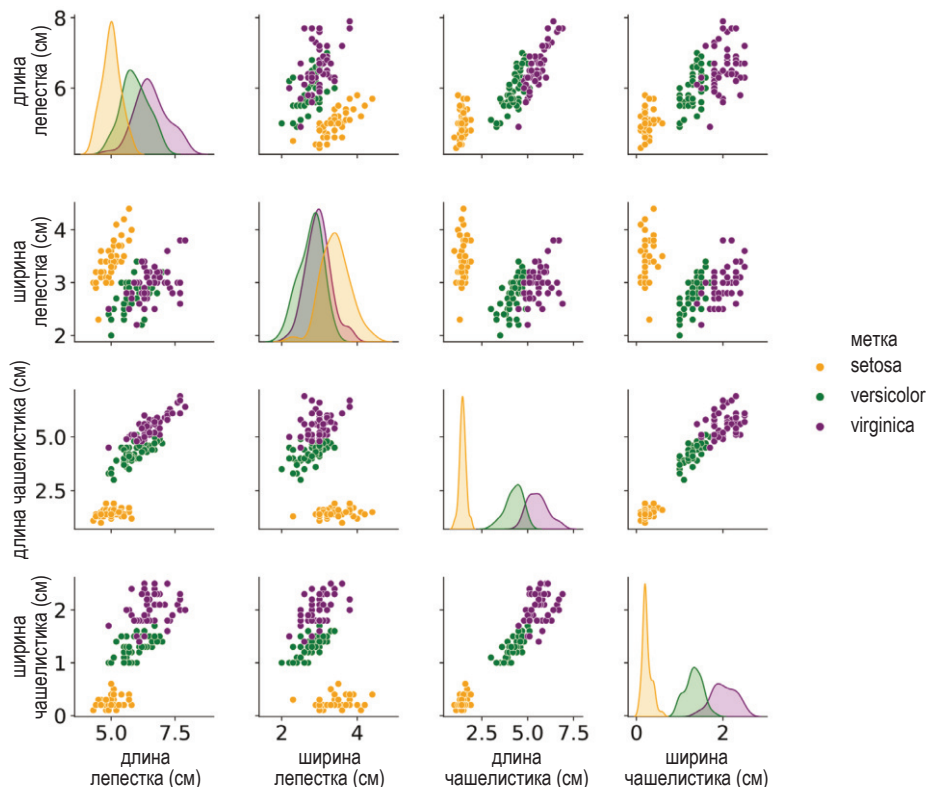


Рис. 1.3 ❖ Визуализация набора данных Iris в виде попарной диаграммы рассеяния. На главной диагонали приведены графики маргинального распределения¹ каждого признака для каждого класса. Элементы вне диагонали содержат диаграммы рассеяния всех возможных пар признаков. Построено программой по адресу figures.problml.ai/book1/1.3

1.2.1.3. Обучение классификатора

Из рис. 1.3 видно, что класс Setosa довольно легко отличить от двух остальных. Например, можно создать такое **решающее правило**:

$$f(x, \theta) = \begin{cases} \text{Setosa, если длина лепестка} < 2.45 \\ \text{Versicolor или Virginica в противном случае} \end{cases} \quad (1.1)$$

Это очень простой пример классификатора; мы разбили пространство входов на две области, разделенные одномерной **решающей границей** $x_{\text{длина лепестка}} = 2.45$. Точки слева от этой границы классифицируются как Setosa, а точки справа – как Versicolor или Virginica.

¹ Ранее в русскоязычной литературе термин *marginal distribution* переводился как «частное распределение», сейчас более употребителен перевод «маргинальное распределение». – Прим. перев.

Мы видим, что это правило абсолютно точно классифицирует примеры Setosa, но не примеры Virginica и Versicolor. Для улучшения качества мы можем рекурсивно разбить те области пространства, в которых классификатор допускает ошибки. Например, можно добавить еще одно решающее правило, применяемое к входным данным, которые не прошли первого теста, и проверить, меньше ли ширина лепестка, чем 1.75 см (в таком случае мы предсказываем класс Versicolor), или больше (и тогда мы предсказываем Virginica). Эти вложенные правила можно организовать в виде древовидной структуры, называемой **решающим деревом**, как показано на рис. 1.4а. Это дерево индуцирует двумерную **решающую поверхность**, показанную на рис. 1.4б.

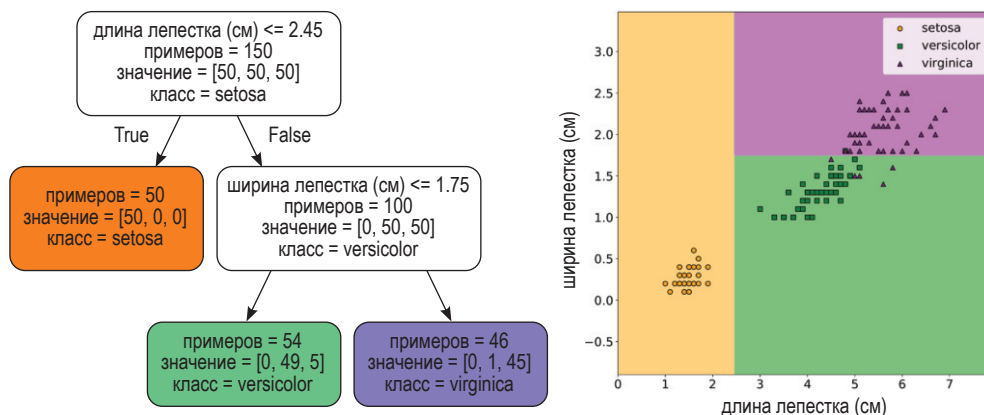


Рис. 1.4 ❖ Пример решающего дерева глубины 2 применительно к набору данных Iris с использованием только признаков «длина лепестка» и «ширина лепестка». Листовые узлы окрашены в соответствии с предсказанным классом. В каждом узле указано количество обучающих примеров, переданных от корня в этот узел; показано, сколько значений из каждого класса попадает в узел. Этот вектор счетчиков можно нормировать и получить распределение меток класса в каждом узле. Затем мы можем выбрать мажоритарный класс. На основе рис. 6.1 и 6.2 из работы [Gér19]. Построено программой по адресу figures.problml.ai/book1/1.4

Мы можем представить это дерево, сохранив для каждого внутреннего узла индекс использованного признака и соответствующее пороговое значение. Все эти **параметры** будем обозначать θ . Как выполнить обучение этих параметров, мы обсудим в разделе 18.1.

1.2.1.4. Минимизация эмпирического риска

Цель обучения с учителем – автоматическое построение моделей классификации типа показанной на рис. 1.4а, так чтобы они надежно предсказывали метки для любых подаваемых на вход данных. Часто для измерения качества модели на такой задаче применяется **частота неправильной классификации** на обучающем наборе:

$$\mathcal{L}(\theta) \triangleq \frac{1}{N} \sum_{n=1}^N \mathbb{I}(y_n \neq f(\mathbf{x}_n; \theta)), \quad (1.2)$$

где $\mathbb{I}(e)$ – бинарная **индикаторная функция**, которая возвращает 1, если условие e истинно, и 0 в противном случае, т. е.:

$$\mathbb{I}(e) = \begin{cases} 1, & \text{если } e \text{ равно true} \\ 0, & \text{если } e \text{ равно false} \end{cases}. \quad (1.3)$$

Здесь предполагается, что все ошибки равноценны. Однако бывают случаи, когда одни ошибки обходятся дороже, чем другие. Предположим, к примеру, что мы ищем съедобные растения в лесу и нашли какие-то ирисы. Еще предположим, что луковицы ирисов *Setosa* и *Versicolor* вкусные, а *Virginica* ядовитые. В таком случае можно было бы использовать асимметричную **функцию потерь** $\ell(y, \hat{y})$, показанную в табл. 1.2.

Таблица 1.2. Гипотетическая асимметричная матрица потерь для классификации набора данных *Iris*

		Оценка		
		Setosa	Versicolor	Virginica
Истина	Setosa	0	1	1
	Versicolor	1	0	1
	Virginica	10	10	0

Тогда можно определить **эмпирический риск** как среднюю потерю предиктора на обучающем наборе:

$$\mathcal{L}(\theta) \triangleq \frac{1}{N} \sum_{n=1}^N \ell(y_n, f(\mathbf{x}_n; \theta)). \quad (1.4)$$

Мы видим, что частота неправильной классификации (1.2) равна эмпирическому риску, если для сравнения истинной метки с предсказанной использовать **бинарную функцию потерь** (zero-one loss):

$$\ell_{01}(y, \hat{y}) = \mathbb{I}(y \neq \hat{y}). \quad (1.5)$$

Детали см. в разделе 5.1.

Один из способов поставить задачу о **подгонке** или **обучении модели** – найти параметры, доставляющие минимум эмпирическому риску на обучающем наборе:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta) = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^N \ell(y_n, f(\mathbf{x}_n; \theta)). \quad (1.6)$$

Такая постановка называется **минимизацией эмпирического риска**.

Однако наша истинная цель состоит в том, чтобы минимизировать ожидаемую потерю на будущих данных, которые не предъявлялись модели. То

есть мы хотим добиться **обобщаемости**, а не просто хорошего качества на обучающем наборе. Мы обсудим этот важный момент в разделе 1.2.3.

1.2.1.5. Неопределенность

[Надлежит избегать] ложной уверенности, проистекающей из неведения вероятностной природы мира, из желания видеть черное и белое там, где правильнее было бы видеть серое.

— Иммануил Кант в пересказе Марии Конниковой [Kon20]

Во многих случаях мы не можем точно предсказать, какой выход воспоследует из известного входа, поскольку не знаем, как устроено отображение входа в выход (это называется **эпистемической неопределенностью** или **неопределенностью модели**), и (или) вследствие внутренне присущей (неустранимой) стохастичности отображения (это называется **алеаторной неопределенностью** или **неопределенностью данных**).

Представление неопределенности в предсказании может играть важную роль в некоторых приложениях. Например, вернемся к примеру с отравленным цветком, матрица потерь которого показана в табл. 1.2. Если мы с высокой вероятностью предсказываем, что это ирис Virginica, то не должны есть такую луковицу. С другой стороны, мы могли бы заняться **сбором информации**, скажем выполнить диагностический тест, чтобы уменьшить неопределенность. Дополнительные сведения о принятии оптимальных решений в условиях неопределенности см. в разделе 5.1.

Мы можем описать неопределенность с помощью следующего **условного распределения вероятностей**:

$$p(y = c | \mathbf{x}; \boldsymbol{\theta}) = f_c(\mathbf{x}; \boldsymbol{\theta}), \quad (1.7)$$

где $f : \mathcal{X} \rightarrow [0, 1]^C$ – отображение распределения вероятностей C возможных выходных меток. Поскольку $f_c(\mathbf{x}; \boldsymbol{\theta})$ возвращает вероятность метки класса c , мы требуем, чтобы для каждого c было $0 \leq f_c \leq 1$ и $\sum_{c=1}^C f_c = 1$. Чтобы избежать этого ограничения, часто требуют вместо этого, чтобы модель возвращала ненормированные логарифмы вероятностей. Затем мы преобразуем их в вероятности с помощью **функции softmax**, определенной следующим образом:

$$\mathcal{S}(\mathbf{a}) \triangleq \left[\frac{e^{a_1}}{\sum_{c'=1}^C e^{a_{c'}}}, \dots, \frac{e^{a_C}}{\sum_{c'=1}^C e^{a_{c'}}} \right]. \quad (1.8)$$

Эта функция отображает \mathbb{R}^C в $[0, 1]^C$ и удовлетворяет ограничениям $0 \leq \mathcal{S}(\mathbf{a})_c \leq 1$ и $\sum_{c=1}^C \mathcal{S}(\mathbf{a})_c = 1$. Входные данные softmax, $\mathbf{a} = f(\mathbf{x}; \boldsymbol{\theta})$, называются **логитами**. Подробнее см. раздел 2.5.2. Таким образом, мы определяем общую модель в виде:

$$p(y = c | \mathbf{x}; \boldsymbol{\theta}) = \mathcal{S}_c(\mathbf{x}; \boldsymbol{\theta}). \quad (1.9)$$

В распространенном частном случае этого определения f – **аффинная функция** вида

$$f(\mathbf{x}; \boldsymbol{\theta}) = b + \mathbf{w}^T \mathbf{x} = b + w_1 x_1 + w_2 x_2 + \dots + w_D x_D, \quad (1.10)$$

где $\boldsymbol{\theta} = (b, \mathbf{w})$ – параметры модели. Такая модель называется **логистической регрессией** и будет подробно обсуждаться в главе 10.

В статистике параметры \mathbf{w} обычно называются **коэффициентами регрессии** (и обозначаются β), а b – **свободным членом**. В МО параметры \mathbf{w} называются **весами**, а b – **смещением**. Эта терминология заимствована из электротехники, где функция f рассматривается как электрическая цепь, которая принимает на входе \mathbf{x} и возвращает $f(\mathbf{x})$. Входы подаются цепи по «проводам», имеющим веса \mathbf{w} . Цепь вычисляет сумму входов и прибавляет постоянное смещение b . (Это употребление термина «смещение» не следует путать со статистическим понятием смещения, обсуждаемым в разделе 4.7.6.1.)

Чтобы не загромождать обозначения, принято включать смещение b в состав весов \mathbf{w} , определив векторы $\tilde{\mathbf{w}} = [b, w_1, \dots, w_D]$ и $\tilde{\mathbf{x}} = [1, x_1, \dots, x_D]$, так что:

$$\tilde{\mathbf{w}}^T \tilde{\mathbf{x}} = b + \mathbf{w}^T \mathbf{x}. \quad (1.11)$$

Тем самым мы преобразуем аффинную функцию в **линейную функцию**. Обычно мы будем предполагать, что это уже сделано, поэтому функцию предсказания можно записать в виде:

$$f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}. \quad (1.12)$$

1.2.1.6. Оценка максимального правдоподобия

При обучении вероятностных моделей часто используют отрицательную логарифмическую вероятность в качестве функции потерь:

$$\ell(y; f(\mathbf{x}; \boldsymbol{\theta})) = -\log p(y|f(\mathbf{x}; \boldsymbol{\theta})). \quad (1.13)$$

Причины этого будут раскрыты в разделе 5.1.6.1, но интуиция подсказывает, что хорошей (с низкими потерями) является модель, которая назначает высокую вероятность истинному выходу y , соответствующему входу \mathbf{x} . Средняя отрицательная логарифмическая вероятность обучающего набора равна

$$\text{NLL}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{n=1}^N \log p(y_n | f(\mathbf{x}_n; \boldsymbol{\theta})). \quad (1.14)$$

Эта величина и называется **отрицательным логарифмическим правдоподобием**. Минимизировав ее, мы найдем **оценку максимального правдоподобия** (maximum likelihood estimate – **MLE**):

$$\hat{\boldsymbol{\theta}}_{\text{mle}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \text{NLL}(\boldsymbol{\theta}). \quad (1.15)$$

Ниже мы увидим, что это очень распространенный способ подгонки моделей к данным.

1.2.2. Регрессия

Теперь предположим, что требуется предсказать вещественную величину $y \in \mathbb{R}$, а не метку класса $y \in \{1, \dots, C\}$; такая задача называется **регрессией**. Так, в случае набора Iris y может быть степенью токсичности луковицы цветка или средней высотой растения.

Регрессия очень похожа на классификацию. Но, поскольку выходом является вещественное значение, нам нужна другая функция потерь. Для регрессии чаще всего используют **квадратичную потерю**, или **ℓ_2 -потерю**:

$$\ell_2(y; \hat{y}) = (y - \hat{y})^2. \quad (1.16)$$

Эта функция штрафует за большие **невязки** $y - \hat{y}$ сильнее, чем за малые¹. Эмпирический риск при использовании квадратичной потери равен **средне-квадратической ошибке (СКО, англ. MSE)**:

$$\text{MSE}(\theta) = \frac{1}{N} \sum_{n=1}^N (y_n - f(\mathbf{x}_n; \theta))^2. \quad (1.17)$$

Вспоминая обсуждение в разделе 1.2.1.5, мы должны также смоделировать неопределенность предсказания. В задачах регрессии принято предполагать, что распределение выходов **гауссово**, или **нормальное**. В разделе 2.6 мы объясним, что это распределение определено формулой

$$\mathcal{N}(y|\mu, \sigma^2) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-\mu)^2}, \quad (1.18)$$

где μ – среднее, σ^2 – дисперсия, а $\sqrt{2\pi\sigma^2}$ – нормировочная постоянная, которая необходима, чтоб интеграл плотности был равен 1. В контексте регрессии среднее может зависеть от входов: $\mu = f(\mathbf{x}_n; \theta)$. Поэтому мы получаем такое условное распределение вероятностей:

$$p(y|\mathbf{x}; \theta) = \mathcal{N}(y|f(\mathbf{x}; \theta); \sigma^2). \quad (1.19)$$

Если предположить, что дисперсия σ^2 фиксирована (для простоты), то отрицательное логарифмическое правдоподобие принимает вид:

$$\text{NLL}(\theta) = -\sum_{n=1}^N \log \left[\left(\frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp \left(-\frac{1}{2\sigma^2} (y_n - f(\mathbf{x}_n; \theta))^2 \right) \right] \quad (1.20)$$

$$= \frac{N}{2\sigma^2} \text{MSE}(\theta) + \text{const}. \quad (1.21)$$

¹ Если в данных имеются выбросы, то квадратичный штраф может оказаться слишком суровым. В таких случаях лучше использовать более **робастную** ℓ_1 -потерю. Подробнее см. раздел 11.6.

Мы видим, что NLL пропорциональна MSE. Поэтому вычисление оценки максимального правдоподобия параметров сводится к минимизации квадратичной ошибки, что представляется разумным подходом к подгонке модели.

1.2.2.1. Линейная регрессия

В качестве примера модели регрессии рассмотрим одномерные данные на рис. 1.5а. Их можно аппроксимировать простой моделью **линейной регрессии** вида

$$f(\mathbf{x}; \boldsymbol{\theta}) = b + wx, \quad (1.22)$$

где w – **угловой коэффициент**, b – **смещение**, а $\boldsymbol{\theta} = (w, b)$ – параметры модели. Варьируя $\boldsymbol{\theta}$, мы можем уменьшать сумму квадратов ошибок, представленных вертикальными отрезками на рис. 1.5b, и в итоге найдем решение с наименьшей СКО:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \operatorname{MSE}(\boldsymbol{\theta}). \quad (1.23)$$

Детали см. в разделе 11.2.2.1.

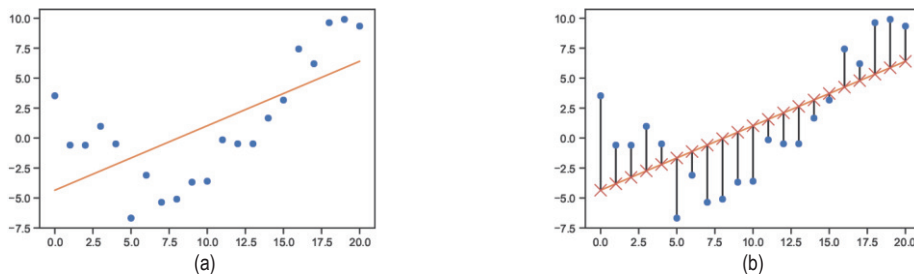


Рис. 1.5 ❖ (а) Линейная регрессия для одномерных данных. (б) Вертикальные отрезки обозначают невязки между наблюдаемым (синий кружочек) и предсказанным (красный крестик) выходным значением. Цель регрессии методом наименьших квадратов – выбрать линию, которая минимизирует сумму квадратов невязок. Сгенерировано программой по адресу figures.problml.ai/book1/1.5

При наличии нескольких входных признаков мы можем написать

$$f(\mathbf{x}; \boldsymbol{\theta}) = b + w_1x_1 + \dots + w_Dx_D = b + \mathbf{w}^T\mathbf{x}, \quad (1.24)$$

где $\boldsymbol{\theta} = (w, b)$. Это называется **многофакторной линейной регрессией**.

Например, рассмотрим задачу предсказания температуры как функции двумерного местоположения в комнате. На рис. 1.6а показаны результаты линейной модели вида

$$f(\mathbf{x}; \boldsymbol{\theta}) = b + w_1x_1 + w_2x_2. \quad (1.25)$$

Мы можем обобщить эту модель на $D > 2$ входных признаков (скажем, добавить время суток), но тогда ее будет труднее визуализировать.

1.2.2.2. Полиномиальная регрессия

Линейная модель на рис. 1.5а, очевидно, не очень хорошо аппроксимирует данные. Мы можем улучшить дело, воспользовавшись **моделью полиномиальной регрессии** степени D . Она имеет вид $f(x; \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(x)$, где $\boldsymbol{\phi}(x)$ – вектор признаков, построенный по входным данным и имеющий вид

$$\boldsymbol{\phi}(x) = [1; x; x^2; \dots; x^D]. \quad (1.26)$$

Это простой пример **предобработки признаков**, или **конструирования признаков**. На рис. 1.7а видно, что при $D = 2$ результаты аппроксимации гораздо лучше. Можно и дальше увеличивать D , а вместе с ним и количество параметров модели, вплоть до $D = N - 1$, когда число параметров сравняется с числом точек и интерполяция будет идеальной. В такой модели СКО будет равно 0, как показано на рис. 1.7с. Однако интуитивно понятно, что результирующая функция вряд ли окажется хорошим предиктором для будущих данных, потому что она слишком «заковыристая». Мы обсудим этот вопрос подробнее в разделе 1.2.3.

Мы можем также применить полиномиальную регрессию к многомерным входным данным. Например, на рис. 1.6b показаны предсказания модели температуры, когда входные данные аппроксимируются квадратичной функцией:

$$f(x; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2. \quad (1.27)$$

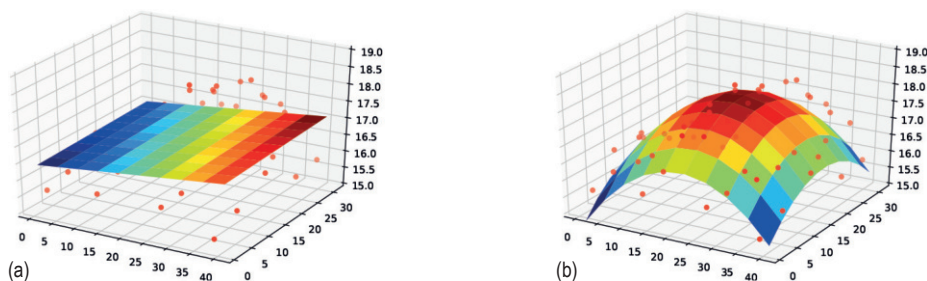


Рис. 1.6 ❖ Линейная и полиномиальная регрессия применительно к двумерным данным. По вертикальной оси отложена температура, а по горизонтальным – местоположение в комнате. Данные были собраны с помощью беспроводных однокристалльных датчиков в лаборатории Intel в Беркли, штат Калифорния (используются с разрешения Ромейна Тибо). (а) Аппроксимирующая плоскость вида $\hat{f}(x) = w_0 + w_1 x_1 + w_2 x_2$. (б) Данные о температуре аппроксимированы квадратичной функцией вида $\hat{f}(x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2$. Построено программой по адресу figures.problml.ai/book1/1.6

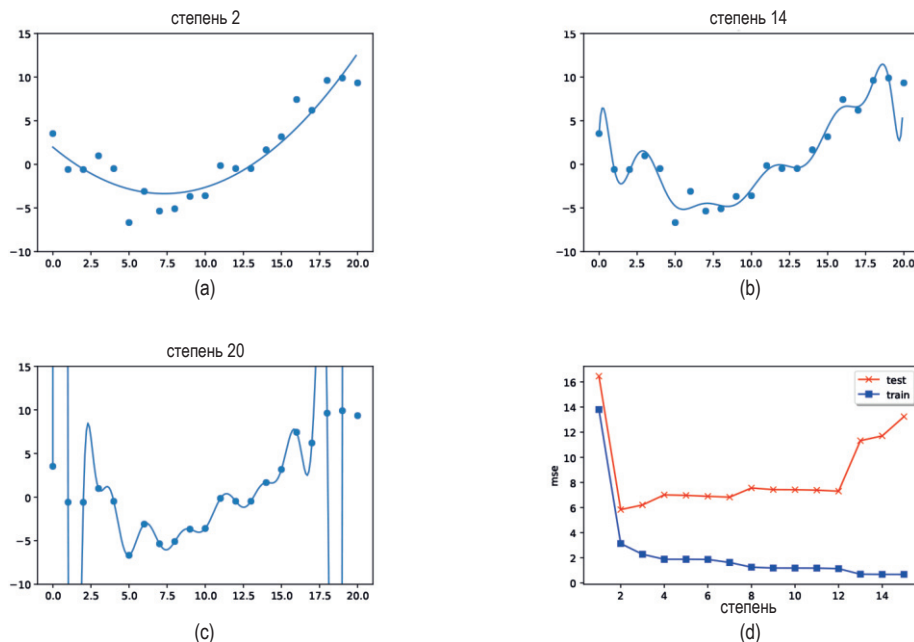


Рис. 1.7 ❖ (а–с) Аппроксимация 21 точки полиномами степени 2, 14 и 20 (данные те же, что на рис. 1.5). (д) Зависимость СКО от степени. Построено программой по адресу figures.problm.ai/book1/1.7

Квадратичная модель лучше аппроксимирует данные, чем линейная, показанная на рис. 1.6а, потому что улавливает тот факт, что в центре комнаты теплее. Можно также добавить перекрестные произведения типа x_1x_2 , чтобы уловить эффекты взаимодействия. Детали см. в разделе 1.5.3.2.

Отметим, что во всех перечисленных выше моделях функция-предиктор линейно зависит от параметров \mathbf{w} , хотя и нелинейно от входных данных \mathbf{x} . Это важно, потому что линейная модель индуцирует среднеквадратическую функцию потерь $MSE(\theta)$, имеющую единственный глобальный оптимум, как будет объяснено в разделе 11.2.2.1.

1.2.2.3. Глубокие нейронные сети

В разделе 1.2.2.2 мы вручную задали преобразование входных признаков – полиномиальное разложение $\phi(\mathbf{x}) = [1, x_1, x_2, x_1^2, x_2^2, \dots]$. Можно создать гораздо более мощные модели, обучившись производить такое нелинейное **выделение признаков** автоматически. Если считать, что $\phi(\mathbf{x})$ имеет собственный набор параметров, скажем \mathbf{V} , то полная модель будет иметь вид:

$$f(\mathbf{x}; \mathbf{w}; \mathbf{V}) = \mathbf{w}^T \phi(\mathbf{x}; \mathbf{V}). \quad (1.28)$$

Мы можем рекурсивно разложить функцию выделения признаков $\phi(\mathbf{x}; \mathbf{V})$ в композицию более простых функций. В результате получается модель, состоящая из L вложенных функций:

$$f(\mathbf{x}; \boldsymbol{\theta}) = f_L(f_{L-1}(\dots(f_1(\mathbf{x}))\dots)), \quad (1.29)$$

где $f_\ell(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\theta}_\ell)$ – функция слоя ℓ . Последний слой линейный и имеет вид $f_L(\mathbf{x}) = \mathbf{w}^\top f_{1:L-1}(\mathbf{x})$, где $f_{1:L-1}(\mathbf{x})$ – обученная функция выделения признаков (экстрактор). Эта идея – ключ к **глубоким нейронным сетям (ГНС)**, у которых есть такие широко известные варианты, как **сверточные нейронные сети (СНС)** для изображений и **рекуррентные нейронные сети (РНС)** для последовательностей. Подробнее см. часть III.

1.2.3. Переобучение и обобщаемость

Мы можем переписать эмпирический риск в формуле (1.4) в эквивалентной, но более удобной форме:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_{\text{train}}} \ell(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta})), \quad (1.30)$$

где $|\mathcal{D}_{\text{train}}|$ – размер обучающего набора $\mathcal{D}_{\text{train}}$. Эта форма полезна, потому что из нее явно видно, на каком наборе данных вычисляется функция потерь.

Если модель достаточно гибкая, то можно свести потери на обучающем наборе к нулю (в предположении, что метки незашумленные), просто запомнив правильный выход для каждого входа. Так, модель на рис. 1.7с идеально интерполирует обучающие данные (не считая одной точки справа). Но нас-то интересует точность предсказания на новых данных, которые могут и не быть частью обучающего набора. Если модель точно аппроксимирует обучающие данные, но при этом слишком сложна, то говорят, что она **переобучена**.

Чтобы выяснить, является ли модель переобученной, предположим (пока), что имеется доступ к истинному (но неизвестному) распределению $p^*(\mathbf{x}, \mathbf{y})$, использованному для порождения обучающего набора. Тогда вместо вычисления эмпирического риска мы вычисляем теоретически ожидаемую потерю, или **риск на генеральной совокупности**:

$$\mathcal{L}(\boldsymbol{\theta}; p^*) \triangleq \mathbb{E}_{p^*(\mathbf{x}, \mathbf{y})}[\ell(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta}))]. \quad (1.31)$$

Разность $\mathcal{L}(\boldsymbol{\theta}; p^*) - \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}})$ называется **разрывом обобщения**. Если разрыв обобщения для модели велик (т. е. эмпирический риск низкий, а риск на генеральной совокупности высокий), то можно заподозрить переобучение. На практике распределение p^* неизвестно. Однако мы можем разбить имеющиеся данные на два подмножества: обучающий набор и **тестовый набор**. И аппроксимировать риск на генеральной совокупности **риском на тестовом наборе**:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{test}}) \triangleq \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_{\text{test}}} \ell(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta})). \quad (1.32)$$

Так, на рис. 1.7d построен график ошибки на обучающем и на тестовом наборе для полиномиальной регрессии в виде функции от степени D . Мы видим, что ошибка на обучающем наборе стремится к 0 при усложнении модели. Однако ошибка на тестовом наборе имеет характерную U-образную форму: в левой части, где $D = 1$, модель недообучена, а справа, где $D \gg 1$, переобучена. А вот при $D = 2$ сложность модели как раз такая, «как надо».

Как правильно выбрать сложность модели? Если для оценивания различных моделей использовать обучающий набор, то мы всегда будем выбирать самую сложную модель, потому что у нее наибольшее число **степеней свободы**, а значит, наименьшая потеря. Поэтому выбирать следует модель с минимальной потерей на тестовом наборе.

На практике данные следует разбивать на три множества: обучающий набор, тестовый и **контрольный набор**; последний используется для выбора модели, а тестовый только для оценки качества на будущих данных (риска на генеральной совокупности). То есть тестовый набор не используется ни для обучения модели, ни для ее выбора. Дальнейшие детали см. в разделе 4.5.4.

1.2.4. Теорема об отсутствии бесплатных завтраков

Все модели неверны, но некоторые из них полезны.

— Джордж Бокс [BD87, стр. 424]¹

В литературе описано множество самых разных моделей, поэтому естественно спросить, какая из них наилучшая. К сожалению, не существует одной модели, которая была бы оптимальна для всех задач, – иногда этот факт называют **теоремой об отсутствии бесплатных завтраков** [Wol96]. Причина в том, что набор допущений (иногда называемый **индуктивным смещением**), который хорошо работает в одной предметной области, может давать никуда не годные результаты в другой. При выборе подходящей модели лучше всего опираться на знание предметной области и (или) на метод проб и ошибок (т. е. применять такие способы выбора модели, как перекрестная проверка (раздел 4.5.4) или байесовские методы (раздел 5.2.2)). Поэтому так важно иметь в своем арсенале много моделей и алгоритмических методов.

1.3. ОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ

В обучении с учителем предполагается, что с каждым входным примером x в обучающем наборе ассоциировано множество выходных меток y , а наша цель – обучиться отображению входа в выход. Хотя это полезно, а иногда трудно, по сути дела, обучение с учителем – всего лишь «напыщенная аппроксимация кривой» [Pea18].

¹ Джордж Бокс – профессор статистики Висконсинского университета, ныне на пенсии.

Гораздо более интересная задача – попытаться «извлечь смысл» из данных, а не просто обучить отображение. То есть у нас есть только наблюдаемые «входы» $\mathcal{D} = \{\mathbf{x}_n : n = 1 : N\}$, но никаких соответствующих им «выходов» \mathbf{y}_n . Такая постановка задачи называется **обучением без учителя**.

С вероятностной точки зрения, задачу обучения без учителя можно рассматривать как обучение безусловной модели вида $p(\mathbf{x})$, которая умеет порождать новые данные \mathbf{x} , тогда как обучение с учителем – это обучение условной модели $p(\mathbf{y}|\mathbf{x})$, задающей распределение выходов при условии входов¹.

В обучении без учителя нет нужды собирать большие помеченные наборы данных для обучения, что может оказаться делом долгим и дорогостоящим (представьте, что вы просите врачей пометить медицинские изображения).

Нет также нужды обучаться разбиению мира на категории, зачастую произвольные. Подумайте, например, о задаче пометить на видео действие, скажем «питье» или «прихлебывание». Когда оно начинается: когда человек поднимает стакан или когда стакан касается губ? Люди зачастую не могут прийти к единой точке зрения по таким вопросам [Idr+17], а значит, задача поставлена некорректно. Поэтому странно было бы ожидать, что машина сможет обучиться такому отображению².

Наконец, обучение без учителя вынуждает модель «объяснять» входы высокой размерности, а не только выходы низкой размерности. Это позволяет обучать более развитые модели «устройства мира».

Джеффри Хинтон, знаменитый профессор МО из Торонтского университета, писал:

Когда мы учимся видеть, никто не подсказывает нам правильные ответы – мы просто смотрим. Бывает, мама говорит «это собака», но информации в этих словах очень мало. Хорошо, если таким способом удастся получить хоть несколько битов информации – пусть даже один бит в секунду. В зрительной системе мозга порядка 10^{14} нейронных связей. А наша жизнь продолжается всего 10^9 с. Так что обучаться со скоростью один бит в секунду бесполезно. Нужно быстрее, порядка 10^5 бит в секунду. И есть только одно место, где можно добыть так много информации: сами входные данные.

— Джеффри Хинтон (цитируется по [Gor06])

1.3.1. Кластеризация

Простой пример обучения без учителя – проблема нахождения **кластеров** в данных. Цель – разбить множество входов на области, содержащие «похо-

¹ У статистиков принято использовать букву \mathbf{x} для обозначения экзогенных переменных, которые не моделируются, а просто являются входными данными. Поэтому безусловная модель обозначалась бы $p(\mathbf{y})$, а не $p(\mathbf{x})$.

² Более разумный подход – попытаться уловить распределение вероятностей меток, поставленных «толпой» аннотаторов (см., например, [Dum+18; Ago+19]). При этом учитывается тот факт, что данному входу может соответствовать несколько «правильных» меток – просто в силу неоднозначности задачи.

жие» точки. Рассмотрим двумерный вариант набора данных Iris. На рис. 1.8a показаны точки без меток класса. Интуитивно понятно, что в данных имеется по меньшей мере два кластера, в левом нижнем и в правом верхнем углу. Кроме того, если предположить, что «хороший» кластер должен быть относительно компактным, то точки в правом верхнем углу хочется разбить на два подкластера (по крайней мере). Результирующее разбиение на три кластера показано на рис. 1.8b. (Отметим, что никто не говорит, сколько должно быть кластеров; мы сами должны выбрать компромисс между сложностью модели и качеством аппроксимации ею данных. Как это сделать, обсуждается в разделе 21.3.7.)

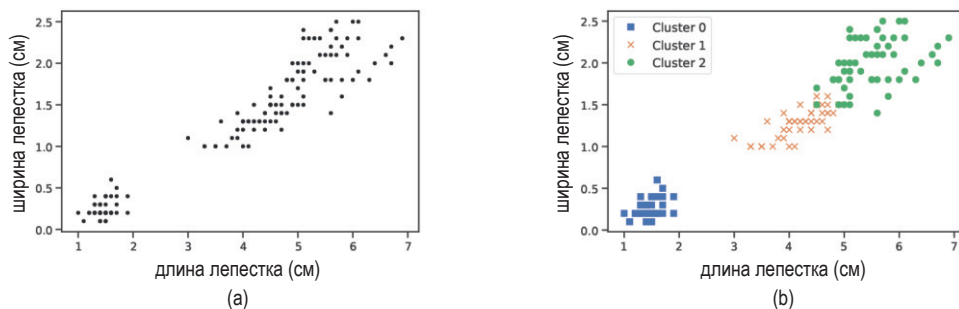


Рис. 1.8 ❖ (a) Диаграмма рассеяния признаков, относящихся к лепесткам, в наборе данных об ирисах. (b) Результат кластеризации без учителя при $K = 3$. Построено программой по адресу figures.problml.ai/book1/1.8

1.3.2. Обнаружение латентных «факторов изменчивости»

При работе с данными высокой размерности часто бывает полезно понизить размерность путем проецирования на подпространство меньшей размерности, которое улавливает основные особенности данных. Один из подходов к этой проблеме – предположить, что все наблюдаемые многомерные выходы $\mathbf{x}_n \in \mathbb{R}^D$ порождены множеством скрытых, т. е. ненаблюдаемых **латентных факторов** низкой размерности $\mathbf{z}_n \in \mathbb{R}^K$. Мы можем наглядно представить модель в виде $\mathbf{z}_n \rightarrow \mathbf{x}_n$, где стрелка обозначает причинно-следственную связь. Поскольку латентные факторы \mathbf{z}_n нам неизвестны, часто предполагается простая модель априорной вероятности для $p(\mathbf{z}_n)$, например гауссова, согласно которой каждый фактор является случайным K -мерным вектором. Если данные вещественные, то можно также использовать гауссово правдоподобие.

Простейший пример – использование линейной модели $p(\mathbf{x}_n | \mathbf{z}_n; \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_n | \mathbf{W}\mathbf{z}_n + \boldsymbol{\mu}, \boldsymbol{\Sigma})$. Результирующая модель называется **факторным анализом** (ФА). Он похож на линейную регрессию с тем отличием, что мы наблюдаем только выходы \mathbf{x}_n , а не входы \mathbf{z}_n . В частном случае, когда $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$, это сводится к модели вероятностного **метода главных компонент** (principal components analysis — **PCA**), который мы будем рассматривать в разделе 20.1.

На рис. 1.9 показано, как этот метод позволяет найти двумерное линейное подпространство, будучи применен к простым трехмерным данным.

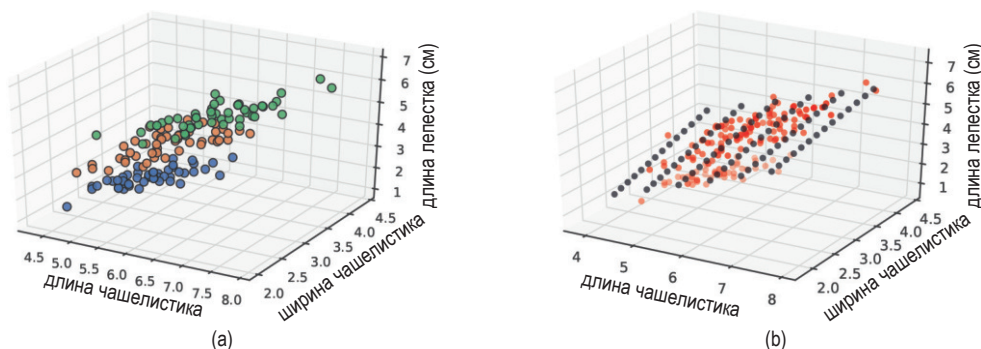


Рис. 1.9 ❖ (а) Диаграмма рассеяния признаков данных об ирисах (первые три признака). Цвет точек соответствует классам. (б) Мы аппроксимируем трехмерные данные двумерным линейным подпространством, применяя метод РСА. Метки классов игнорируются. Красные точки – оригинальные данные, черные точки сгенерированы по модели с использованием $\tilde{\mathbf{x}} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu}$, где \mathbf{z} – латентные точки на выведенном двумерном линейном многообразии. Построено программой по адресу figures.probl.ai/book1/1.9

Конечно, предположение о линейности отображения \mathbf{z}_n в \mathbf{x}_n чрезмерно ограничительно. Однако мы можем создать нелинейные обобщения, определив $p(\mathbf{x}_n|\mathbf{z}_n; \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_n|f(\mathbf{z}_n; \boldsymbol{\theta}), \sigma^2\mathbf{I})$, где $f(\mathbf{z}; \boldsymbol{\theta})$ – нелинейная модель, например глубокая нейронная сеть. Обучить такую модель (т. е. оценить параметры $\boldsymbol{\theta}$) оказывается гораздо труднее, потому что необходимо выводить как входы нейронной сети, так и параметры модели. Но существуют различные приближенные методы, например вариационные автокодировщики (см. раздел 20.3.5).

1.3.3. Самостоятельное обучение

Недавно приобрел популярность новый подход к обучению без учителя – **самостоятельное обучение** (self-supervised learning). В этом случае мы создаем замещающие задачи обучения с учителем по непомеченным данным. Например, можно попробовать обучиться предсказанию цветного изображения по полутоновому или замаскировать некоторые слова в предложении и попытаться предсказать их по окружающему контексту. Мы надеемся, что получившийся в результате предиктор $\hat{\mathbf{x}}_1 = f(\mathbf{x}_2; \boldsymbol{\theta})$, где \mathbf{x}_2 – наблюдаемый вход, а $\hat{\mathbf{x}}_1$ – предсказанный выход, сможет обучиться полезным признакам на основе данных, а затем использовать их в последующих стандартных задачах обучения с учителем. Тем самым мы обходим трудную проблему вывода «истинных латентных факторов» \mathbf{z} , стоящих за наблюдаемыми данными, а вместо этого полагаемся на стандартные методы обучения с учителем. Мы подробнее обсудим этот подход в разделе 19.2.

1.3.4. Оценка обучения без учителя

Хотя идея обучения без учителя кажется весьма соблазнительной, очень трудно оценить качество такого метода обучения, потому что результат не с чем сравнивать [ТОВ16]. Наиболее распространенный способ оценки моделей, обученных без учителя, – измерение вероятности, назначенной моделью не предъявлявшимся ранее тестовым примерам. Это можно сделать, вычислив (безусловное) отрицательное логарифмическое правдоподобие данных:

$$\mathcal{L}(\theta; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}|\theta). \quad (1.33)$$

Тем самым мы сводим проблему обучения без учителя к проблеме **оценивания плотности**. Идея в том, что хорошую модель не должны «удивлять» примеры фактических данных (т. е. она назначает им высокую вероятность). Кроме того, сумма вероятностей должна быть равна 1.0, а значит, если модель назначает высокую вероятность тем областям пространства данных, из которых взяты примеры, то областям, откуда нет данных, неявно назначается низкая вероятность. Таким образом, модуль обучилась улавливать **типичные паттерны** в данных. Это можно использовать в алгоритме **сжатия данных**.

К сожалению, оценить плотность трудно, особенно если размерность велика. Кроме того, модель, которая назначает высокую вероятность данным, может не обучиться полезным паттернам (ведь она могла бы просто запомнить все обучающие примеры).

Альтернативный способ оценивания – использовать представление, полученное в результате обучения без учителя, в качестве признаков или подать на вход следующему далее методу обучения с учителем. Если метод обучения без учителя выявил полезные паттерны, то, возможно, ими удастся воспользоваться для обучения с учителем на гораздо меньшем объеме помеченных данных, чем при работе с оригинальными признаками. Например, в разделе 1.2.1.1 мы видели, что четыре вручную определенных признака ирисов содержат большую часть информации, необходимой для классификации. И таким образом сумели обучить почти идеальный классификатор всего на 150 примерах. Если бы на вход подавались исходные пиксели, то для достижения сравнимого качества потребовалось бы куда больше примеров (см. раздел 14.1). Следовательно, мы сможем повысить **выборочную эффективность** обучения (т. е. уменьшить количество помеченных примеров, необходимых для получения приемлемого качества), если предварительно обучимся хорошему представлению.

Повышенная выборочная эффективность – полезная метрика оценки, но во многих приложениях, особенно научных, цель обучения без учителя – *понять данные*, а не улучшить качество на какой-то задаче предсказания. Для этого нужны модели, **допускающие интерпретацию**, но при этом способные генерировать или «объяснять» большинство наблюдаемых в данных паттернов. Перефразируя Платона, можно сказать, что цель состоит в том, чтобы выяс-

нить, как «природа делится в местах сочленений»¹. Конечно, чтобы оценить, правильно ли мы распознали истинную структуру, стоящую за некоторым набором данных, зачастую необходимо ставить эксперименты и, стало быть, взаимодействовать с миром. Мы продолжим обсуждение этой темы в разделе 1.4.

1.4. ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ

Помимо обучения с учителем и без учителя, есть еще и третий вид МО – **обучение с подкреплением** (ОП, англ. **RL**). В проблемах этого класса система или **агент** должна обучиться взаимодействию со своим окружением. Это можно представить в виде **политики** $a = \pi(x)$, определяющей, какое действие предпринять в ответ на любой возможный вход x (действие выводится из состояния окружающей среды).

Например, рассмотрим агента, который обучается играть в видеоигру, скажем Atari Space Invaders (см. рис. 1.10a). В этом случае входом x является изображение (или последовательность прошлых изображений), а выходом – направление движения (влево или вправо) и признак пуска ракеты. В качестве более сложного примера рассмотрим робота, обучающегося ходить (см. рис. 1.10b). Здесь вход x – множество положений и углов сочленения для всех конечностей, а выход – множество сигналов, подаваемых на приводы или электрические двигатели.

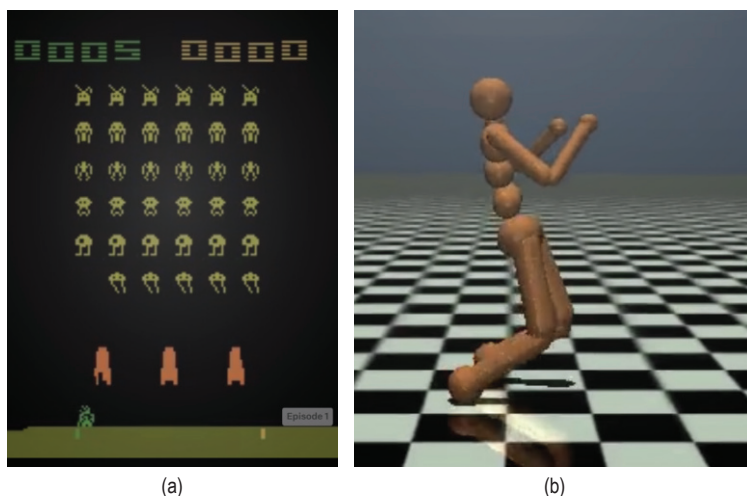


Рис. 1.10 ❖ Пример некоторых задач управления. (a) Игра Space Invaders компании Atari. Взято с <https://gym.openai.com/envs/SpaceInvaders-v0/>. (b) Управление роботом-гуманоидом в эмуляторе MuJoCo; цель – добиться максимальной скорости ходьбы без падений. Взято с <https://gym.openai.com/envs/Humanoid-v2/>

¹ Отсылка к следующему месту из диалога Платона «Федр»: «Другой [род] состоит в уменьи делить предмет на виды – и делить, как водится, почленно, так чтобы, подобно плохому повару, не раздробить ни одной части». – *Прим. перев.*

Отличие от обучения с учителем заключается в том, что системе никто не говорит, какое действие наилучшее (т. е. какой выход следует сгенерировать для данного входа). Вместо этого система время от времени получает сигнал **вознаграждения** (или наказания) в ответ на предпринятые ей действия. Это скорее напоминает **обучение с критиком**, который периодически поднимает или опускает большой палец, а не с учителем, который на каждом шаге говорит, что нужно делать.

В последнее время популярность ОП заметно выросла благодаря широкой применимости (ведь в качестве сигнала вознаграждения, который агент пытается оптимизировать, можно выбирать любую интересующую метрику). Но по разным причинам заставить ОП работать труднее, чем в случае обучения с учителем или без него. Основная трудность заключается в том, что сигнал вознаграждения может подаваться лишь изредка (например, только когда агент в конечном итоге достигает желаемого состояния), и даже в этих случаях может быть неясно, какое из многочисленных действий привело к успеху. (Примером могут служить шахматы, когда единственный сигнал – выигрыш или проигрыш – подается в конце партии.)

Чтобы компенсировать скудость информации, поступающей от сигнала вознаграждения, часто используют другие источники информации, например проводимые экспертами демонстрации, которые можно использовать, как в обучении с учителем, или неразмеченные данные, которые могут использоваться системой обучения без учителя, чтобы выявить структуру окружающей среды. Благодаря таким уловкам становится реально обучиться на ограниченном числе испытаний (раундов взаимодействия с окружающей средой). Как заметил Ян Лекун в лекции на конференции NIPS¹ в 2016 году, «если представить интеллект в виде торта, то обучение без учителя будет шоколадной прослойкой, обучение с учителем – глазурью, а обучение с подкреплением – вишенкой». Эта мысль иллюстрируется на рис. 1.11.

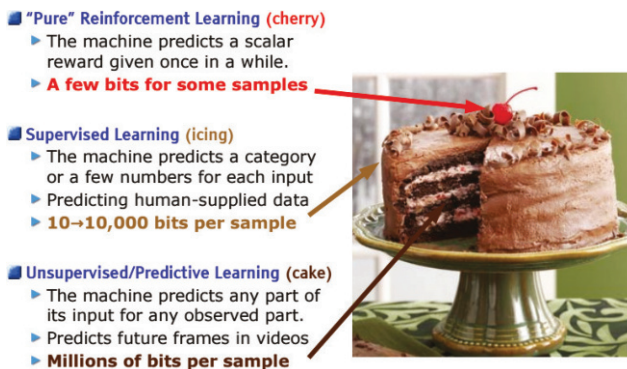


Рис. 1.11 ❖ Три типа машинного обучения, представленные в виде слоев шоколадного торта. Этот рисунок (взятый с <https://bit.ly/2m65Vs1>) был использован в лекции Яна Лекунa на конференции NIPS'16 и печатается с его разрешения

¹ NIPS означает Neural Information Processing Systems (Системы обработки нейронной информации). Это одна из самых авторитетных конференций по машинному обучению. Недавно она была переименована в **NeurIPS**.

Дополнительную информацию об ОП можно найти во втором томе этой книги, [Mur22].

1.5. ДАННЫЕ

Предмет машинного обучения – построение моделей данных с помощью различных алгоритмов. Хотя мы акцентируем внимание на аспектах моделирования и алгоритмов, необходимо отметить, что природа и качество обучающих данных также играют важную роль в успехе любой обученной модели.

В этом разделе мы кратко опишем некоторые используемые в этой книге наборы данных, содержащие изображения и текст. Мы также немного поговорим о предобработке данных.

1.5.1. Некоторые широко известные наборы изображений

В этом разделе приводится краткое обсуждение используемых в книге наборов данных, содержащих изображения.

1.5.1.1. Небольшие наборы изображений

Один из самых простых и широко распространенных наборов данных – **MNIST** [LeC+98; YB19]¹. Этот набор содержит 60 000 обучающих и 10 000 тестовых полутоновых изображений рукописных цифр из 10 категорий, каждое размером 28×28 пикселей. Каждый пиксель представлен целым числом в диапазоне $\{0, 1, \dots, 255\}$; обычно они приводятся к диапазону $[0, 1]$, чтобы представить яркость пикселей. Дополнительно можно преобразовать изображение в бинарное, воспользовавшись методом бинаризации (см. рис. 1.12а).

Набор MNIST настолько широко используется в сообществе МО, что знаменитый ученый Джеффри Хинтон назвал его «дрозофилой машинного обучения», поскольку если некоторый метод не работает на MNIST, то вряд ли он будет хорошо работать на более трудных наборах данных. Однако в наши дни классификация MNIST считается «слишком легкой» задачей, поскольку различить большинство пар цифр можно всего по одному пикселю. Были предложены различные расширения.

В работе [Coh+17] предложен набор **EMNIST** (extended MNIST), включающий также строчные и заглавные буквы (см. рис. 1.12b). Этот набор данных гораздо труднее MNIST, поскольку классов 62 и некоторые из них неоднозначны (например, сравните цифру 1 со строчной буквой l).

¹ Акроним MNIST означает Modified National Institute of Standards. Слово «modified» употребляется, потому что изображения были подвергнуты предварительной обработке с целью поместить цифры примерно в центр.

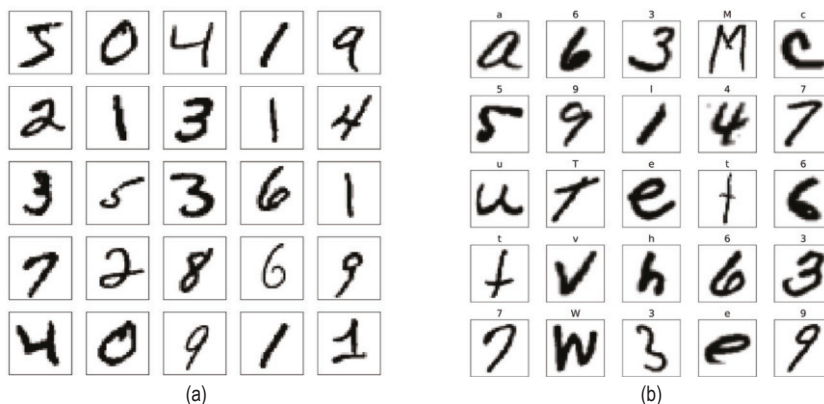


Рис. 1.12 ❖ (a) Визуализация набора данных MNIST. Размер каждого изображения 28×28 . Всего имеется 60к обучающих и 10к тестовых примеров. Показаны первые 25 изображений из обучающего набора. Построено программой по адресу figures.probl.ai/book1/1.12. (b) Визуализация набора данных EMNIST. Имеется 697 932 обучающих и 116 323 тестовых примеров, размер каждого 28×28 . Имеется 62 класса (a–z, A–Z, 0–9). Показаны первые 25 изображений из обучающего набора. Построено программой по адресу figures.probl.ai/book1/1.12

В работе [XRV17] предложен набор **Fashion-MNIST**. Его размер и форма изображений такие же, как в MNIST, но изображены предметы одежды, а не рукописные цифры (см. рис. 1.13a).

Что касается небольших цветных изображений, то наиболее известен набор данных **CIFAR** [KH09]¹. Он содержит 60 000 изображений хорошо знакомых объектов из 10 или 100 классов, примеры см. на рис. 1.13b.

1.5.1.2. ImageNet

Набольшие наборы данных хороши для апробирования идей, но важно также тестировать методы на более крупных наборах – с точки зрения как размера изображений, так и количества помеченных примеров. Самым употребительным из таких наборов является **ImageNet** [Rus+15]. Он содержит порядка 14 млн изображений размера $256 \times 256 \times 3$, иллюстрирующих различные объекты из 20 000 классов, примеры см. на рис. 1.14a.

Набор данных ImageNet был положен в основу конкурса ImageNet Large Scale Visual Recognition Challenge (**ILSVRC**), проводившегося с 2010 по 2018 год. В нем использовалось подмножество, содержащее 1.3 млн изображений, принадлежащих 1000 классам. За эти годы сообщество добилось значительного прогресса, как показывает рис. 1.14b. В частности, в 2015 году СНС впервые превзошла человека (или, по крайней мере, одного человека, Андрея Карпа-

¹ CIFAR означает Canadian Institute For Advanced Research. Эта организация финансировала разметку набора данных, за основу которого взят набор данных TinyImages по адресу <http://groups.csail.mit.edu/vision/TinyImages/>, созданный Антонио Торралба. Детали см. в [KH09].

того) в задаче классификации изображений из набора ImageNet. Заметим, что это не означает, что ЧНС видят лучше человека (см., например, работу [YL21], где описаны некоторые типичные ошибки). Скорее всего, это отражает тот факт, что в наборе данных много нюансов **детальной классификации**, например различий между «тигром» и «тигровой кошкой», которые человеку просто непонятны. С другой стороны, достаточно гибкие ЧНС могут обучиться любым паттернам, в том числе случайным меткам [Zha+17a].



Рис. 1.13 ❖ (a) Визуализация набора данных Fashion-MNIST [XRV17]. Этот набор данных имеет такой же размер, как MNIST, но труднее для классификации. Всего имеется 10 классов: футболка/топ, брюки, пуловер, платье, куртка, сандалии, рубашка, кроссовки, сумка, полусапоги. Показаны первые 25 изображений из обучающего набора. Построено программой по адресу figures.probl.ai/book1/1.13. (b) Несколько изображений из набора данных CIFAR-10 [KH09]. Все изображения имеют размер $32 \times 32 \times 3$, где последнее измерение (3) означает формат RGB. Набор содержит 50k обучающих и 10k тестовых примеров. Имеется 10 классов: самолет, автомобиль, птица, кошка, собака, лягушка, лошадь, судно и грузовик. Показаны первые 25 изображений из обучающего набора. Построено программой по адресу figures.probl.ai/book1/1.13

Хотя набор ImageNet гораздо труднее MNIST и CIFAR в роли эталона классификации, он тоже дошел почти до точки «насыщения» [Beu+20]. Тем не менее качество работы того или иного метода на наборе ImageNet зачастую оказывается на удивление хорошим предиктором качества на других, никак не связанных задачах классификации изображений (см., например, [Res+19]), поэтому он по-прежнему широко используется.

1.5.2. Некоторые широко известные наборы текстовых данных

Машинное обучение часто применяется к текстам для решения самых разных задач. Эта область называется **обработкой естественного языка** (natural language processing – NLP) (детали см., например, в [JM20]). Ниже даны

краткие описания нескольких наборов текстовых данных, используемых в этой книге.

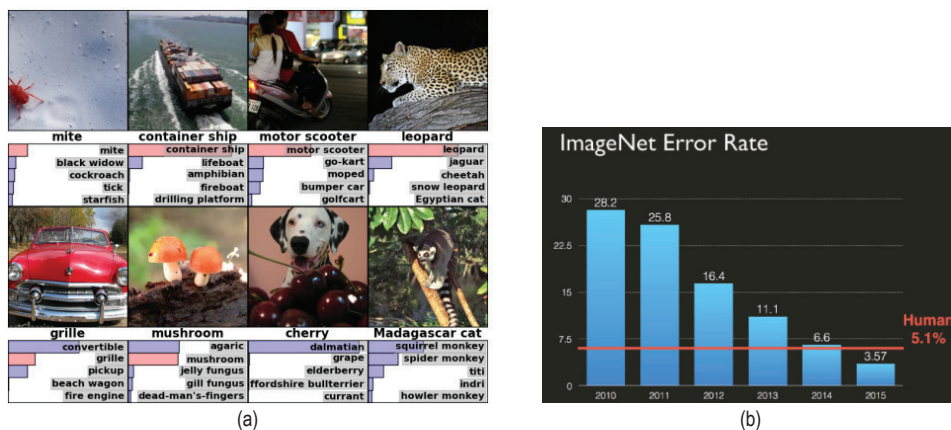


Рис. 1.14 ❖ (а) Примеры изображений из набора данных ImageNet [Rus+15]. Это подмножество состоит из 1.3М цветных обучающих изображений размером 256×256 пикселей. Всего имеется 1000 меток, каждому изображению сопоставляется одна метка, а задача заключается в том, чтобы гарантировать попадание правильной метки в пять предсказаний с наибольшей вероятностью. Под каждым изображением показана истинная метка и распределение вероятностей первых пяти предсказанных меток. Если истинная метка оказывается среди первых пяти, то соответствующий ей столбик окрашен красным. Предсказания сгенерированы сверточной нейронной сетью (СНС) AlexNet (раздел 14.3.2). Заимствовано из работы [KSH12] (рис. 4). Печатается с разрешения Алекса Крижевски. (б) Изменение коэффициента неправильной классификации (непопадание в первые 5) в конкурсах ImageNet со временем. Печатается с разрешения Андрея Карпатого

1.5.2.1. Классификация текста

Простой задачей NLP является классификация текста, ее можно использовать для **классификации спама**, **анализа эмоциональной окраски** (например, является ли обзор фильма или товара положительным или отрицательным) и т. д. Для оценки таких методов часто используется **набор данных IMDB**, содержащий обзоры фильмов (см. [Maa+11]) (IMDB означает Internet Movie Database). В нем 25 000 помеченных примеров для обучения и 25 000 для тестирования. Каждый пример сопровождается бинарной меткой: положительная или отрицательная оценка. Примеры предложений приведены в табл. 1.3.

Таблица 1.3. Примеры двух первых предложений из набора данных IMDB, содержащего обзоры фильмов. Первый пример помечен как положительный, второй как отрицательный (<UNK> означает неизвестную лексему)

- | |
|---|
| 1. this film was just brilliant casting location scenery story direction everyone's really suited the part they played robert <UNK> is an amazing actor ... |
| 2. big hair big boobs bad music and a giant safety pin these are the words to best describe this terrible movie i love cheesy horror movies and i've seen hundreds... |

1.5.2.2. Машинный перевод

Более трудная задача NLP – обучиться отображать предложение x на одном языке в «семантически эквивалентное» предложение y на другом языке; это называется **машинным переводом**. Для обучения таких моделей нужны соответствующие пары (x, y) . По счастью, существует несколько таких наборов данных, например от канадского парламента (англо-французские пары) и Европейского союза (Europarl). Подмножество последнего, **набор данных WMT** (Workshop on Machine Translation), состоит из англо-немецких пар и широко используется в качестве эталона для проверки алгоритмов.

1.5.2.3. Другие задачи типа seq2seq

Обобщение машинного перевода – задача об обучении отображения одной последовательности x в другую последовательность y . Она называется **моделью seq2seq** и может рассматриваться как форма многомерной классификации (подробности см. в разделе 15.2.3). Эта постановка задачи очень общая, поэтому можно выделить много частных случаев: **реферирование документов, вопросно-ответные системы** и т. д. Например, в табл. 1.4 показано, как поставить задачу о вопросно-ответной системе как проблему seq2seq: входом является текст T и вопрос Q , а выходом – ответ A , представляющий собой множество слов, возможно, извлеченных из входного текста.

Таблица 1.4. Пары вопрос-ответ для примера текста из набора данных SQuAD. Каждый ответ представляет собой отрывок текста. Эту задачу можно решить с помощью разметки пар предложений. Входом является текстовый абзац T и вопрос Q . На выходе помечаются те слова T , которые отвечают на вопрос Q . Взято из работы [Raj+16]. Печатается с разрешения Перси Лианга

T: In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity . The main forms of precipitation include drizzle, rain, sleet, snow, graupe l and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud . Short, intense periods of rain in scattered locations are called “showers”.	
Q1: What causes precipitation to fall? A1: gravity	
Q2: What is another main form of precipitation besides drizzle, rain, snow, sleet and hail? A2: graupe	
Q3: Where do water droplets collide with ice crystals to form precipitation? A3: within a cloud	

1.5.2.4. Языковое моделирование

Под очень многоплановым термином «**языковое моделирование**» понимается задача создания безусловных порождающих моделей текстовых предложений, $p(x_1, \dots, x_T)$. Для нее требуются только входные предложения x , без ответственных «меток» y . Поэтому проблему можно рассматривать как одну из форм обучения без учителя, обсуждавшегося в разделе 1.3. Если языковая модель порождает в ответ на вход какой-то выход, как в случае seq2seq, то ее можно рассматривать как условную порождающую модель.

1.5.3. Предобработка дискретных входных данных

Во многих моделях МО предполагается, что данные представляют собой вещественные векторы признаков, $\mathbf{x} \in \mathbb{R}^D$. Но иногда входные данные могут содержать дискретные признаки, например категориальные величины типа расы и пола или слова из некоторого словаря. Ниже мы обсудим некоторые способы предобработки таких данных с целью преобразования их в векторную форму. Эта операция часто применяется в различных моделях.

1.5.3.1. Унитарное кодирование

Категориальные признаки необходимо преобразовать в какую-то числовую шкалу, чтобы вычисление их взвешенных комбинаций имело смысл. Стандартный способ такого преобразования называется **унитарным кодированием**, или **индикаторным кодированием** (dummy encoding). Если переменная x может принимать K значений, то результатом ее унитарного кодирования будет $\text{one-hot}(x) = [\mathbb{I}(x = 1), \dots, \mathbb{I}(x = K)]$. Например, если имеется 3 цвета (скажем, красный, зеленый и синий), то соответствующие им унитарные векторы имеют вид $\text{one-hot}(\text{красный}) = [1, 0, 0]$, $\text{one-hot}(\text{зеленый}) = [0, 1, 0]$, $\text{one-hot}(\text{синий}) = [0, 0, 1]$.

1.5.3.2. Перекрестные произведения признаков

Линейная модель, в которой используется унитарное кодирование всех категориальных переменных, может уловить **основной эффект** каждой переменной, но не **эффекты взаимодействия** переменных. Например, предположим, что требуется предсказать топливную эффективность транспортного средства по двум категориальным переменным: типу (скажем, городской внедорожник, грузовик или семейный автомобиль) и стране происхождения (скажем, США или Япония). Если конкатенировать унитарные коды тернарных и бинарных признаков, то входные данные будут представлены в следующем виде:

$$\phi(\mathbf{x}) = [1; \mathbb{I}(x_1 = S); \mathbb{I}(x_1 = T); \mathbb{I}(x_1 = F); \mathbb{I}(x_2 = U); \mathbb{I}(x_2 = J)], \quad (1.34)$$

где x_1 – тип, а x_2 – страна происхождения.

Эта модель не может уловить зависимости между признаками. Например, мы ожидаем от грузовиков меньшей топливной эффективности, но, быть может, американские грузовики даже менее эффективны, чем японские. Линейная модель (1.34) не может уловить этот факт, потому что вклад страны происхождения не зависит от типа автомобиля.

Это можно исправить, вычислив явные **перекрестные произведения признаков**. Например, мы можем определить новый составной признак с 3×2 возможными значениями, который будет улавливать взаимодействие между типом и страной происхождения. Тогда новая модель принимает вид:

$$f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) \quad (1.35)$$

$$\begin{aligned} &= w_0 + w_1 \mathbb{I}(x_1 = S) + w_2 \mathbb{I}(x_1 = T) + w_3 \mathbb{I}(x_1 = F) \\ &+ w_4 \mathbb{I}(x_2 = U) + w_5 \mathbb{I}(x_2 = J) \\ &+ w_6 \mathbb{I}(x_1 = S; x_2 = U) + w_7 \mathbb{I}(x_1 = T; x_2 = U) + w_8 \mathbb{I}(x_1 = F; x_2 = U) \\ &+ w_9 \mathbb{I}(x_1 = S; x_2 = J) + w_{10} \mathbb{I}(x_1 = T; x_2 = J) + w_{11} \mathbb{I}(x_1 = F; x_2 = J). \end{aligned} \quad (1.36)$$

Видно, что перекрестные произведения позволяют преобразовать исходный набор данных в **широкий формат** с гораздо большим числом столбцов.

1.5.4. Предобработка текстовых данных

В разделе 1.5.2 мы кратко обсудили классификацию текста и другие задачи NLP. Для подачи текста на вход классификатору нужно решить ряд вопросов. Во-первых, документы имеют переменную длину, а не являются векторами признаков фиксированной длины, как предполагается во многих моделях. Во-вторых, слова – это категориальные переменные, имеющие много значений (столько, сколько слов в словаре), поэтому размерность соответствующей унитарной кодировки будет очень велика, а в многомерном пространстве нет естественного понятия сходства. В-третьих, во время тестирования могут встретиться слова, которые не предъявлялись на этапе обучения (так называемые **внесловарные слова**, или OOV-слова). Некоторые решения этих проблем мы обсудим ниже, а дополнительные сведения можно найти, например, в работах [BKL10; MRS08; JM20].

1.5.4.1. Модель мешка слов

Простой подход к обработке текстовых документов переменной длины состоит в том, чтобы интерпретировать их как **мешок слов**, в котором порядок слов игнорируется. Чтобы преобразовать такой документ в вектор из фиксированного пространства входов, мы сначала преобразуем каждое слово в **лексему** из некоторого словаря.

Для уменьшения количества лексем применяются различные методы предобработки, например: отбрасывание знаков препинания, преобразование всех слов в нижний регистр, отбрасывание часто встречающихся, но не информативных слов типа «and» и «the» (это называется **исключением стоп-слов**), замена слов основной формой, например замена «running» и «runs» формой «run» (это называется **стеммингом**) и т. д. Детали см., например, в [BL12], а демонстрационный код на странице code.problm.ai/book1/text_preproc_torch.

Обозначим x_{nt} лексему в позиции t n -го документа. Если в словаре имеется D различных лексем, то n -й документа можно представить D -мерным вектором $\tilde{\mathbf{x}}_n$, где \tilde{x}_{nv} – количество вхождений слова v в документ n :

$$\tilde{x}_{nv} = \sum_{t=1}^T \mathbb{I}(x_{nt} = v), \quad (1.37)$$

где T – длина документа n . Теперь документы можно интерпретировать как векторы в пространстве \mathbb{R}^D . Это так называемая **модель векторного пространства** [SWY75; TP10].

Традиционно входные данные хранятся в матрице плана X размера $N \times D$, где D – число признаков. Но в контексте моделей векторного пространства принято представлять входные данные **терм-документной матрицей** размера $D \times N$, где TF_{ij} – частота термина i в документе j (см. рис. 1.15).

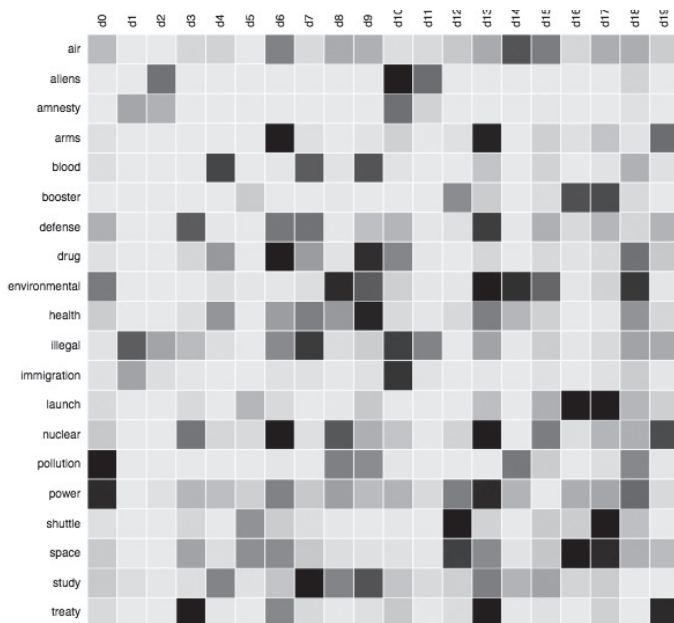


Рис. 1.15 ❖ Пример терм-документной матрицы, в которой исходные счетчики заменены значениями TF-IDF (см. раздел 1.5.4.2). Чем темнее ячейка, тем больше значение. Взято из статьи <https://bit.ly/2kByLQI>. Печатается с разрешения Кристофа Карла Клинга

1.5.4.2 TF-IDF

Одной из проблем, сопутствующих представлению документов векторами счетчиков слов, является тот факт, что часто встречающиеся слова могут оказывать несоразмерное их важности влияние, просто потому что счетчик вхождений для них больше, хотя никакого особого семантического смысла они не несут. Стандартное решение этой проблемы – заменить счетчики логарифмами, что уменьшает влияние слов, многократно встречающихся в одном документе.

А чтобы уменьшить влияние слов, которые вообще часто встречаются (во всех документах), мы вычисляем так называемую **обратную частоту документа**, которая определяется следующим образом: $IDF_i \triangleq \log N / (1 + DF_i)$, где DF_i – число документов, содержащих терм i . Оба преобразования можно объединить и вычислить матрицу **TF-IDF**:

$$\text{TFIDF}_{ij} = \log(\text{TF}_{ij} + 1) \times \text{IDF}_i \quad (1.38)$$

(дополнительно строки часто нормируются). Такая матрица дает семантически более осмысленное представление документов и используется в качестве входа во многих алгоритмах МО. Пример см. по адресу code.problml.ai/book1/tfidf_demo.

1.5.4.3. Погружения слов

Хотя преобразование в матрицу TF-IDF улучшает векторное представление слов, придавая больший вес «информативным» словам и меньший «неинформативным», оно не решает фундаментальную проблему, заключающуюся в том, что семантически родственные слова, например «man» и «woman», могут располагаться в векторном пространстве дальше друг от друга, чем семантически не связанные слова, например «man» и «banana». Поэтому предположение о том, что для точек, близких в пространстве входов, и выходы будут близки, неявно подразумеваемое в моделях логистической регрессии, неверно.

Стандартный способ решения этой проблемы – использование **погружений слов** (word embeddings), когда каждый разреженный унитарный вектор $\mathbf{x}_{nt} \in \{0, 1\}^V$ отображается в плотный вектор меньшей размерности, $\mathbf{e}_{nt} \in \mathbb{R}^K$ посредством преобразования $\mathbf{e}_{nt} = \mathbf{E}\mathbf{x}_{nt}$, где матрица \mathbf{E} обучена таким образом, что семантически родственные слова оказываются рядом. В разделе 20.5 мы увидим, что существует много способов обучиться такому погружению.

Имея матрицу погружений, мы можем представить текстовый документ переменной длины в виде **мешка погружений слов**. Затем этот мешок преобразуется в вектор фиксированной длины путем суммирования (или усреднения) погружений:

$$\bar{\mathbf{e}}_n = \sum_{t=1}^T \mathbf{e}_{nt} = \mathbf{E}\tilde{\mathbf{x}}_n, \quad (1.39)$$

где $\tilde{\mathbf{x}}_n$ – представление в виде мешка слов из формулы (1.37). Далее этот вектор можно использовать в классификаторе методом логистической регрессии, который мы кратко описали в разделе 1.2.1.5. Итоговая модель принимает вид:

$$p(y = c | \mathbf{x}_n; \boldsymbol{\theta}) = \mathcal{S}_c(\mathbf{W}\mathbf{E}\tilde{\mathbf{x}}_n). \quad (1.40)$$

Мы часто используем **предварительно обученную матрицу погружений** \mathbf{E} , и в этом случае модель линейна относительно \mathbf{W} , что упрощает оценивание параметров (см. главу 10). Смотрите также обсуждение контекстуального погружения слов в разделе 15.7.

1.5.4.4. Обработка новых слов

Во время тестирования модели могут встретиться совершенно новые слова, которые раньше не предъявлялись. Такие слова называются **внесловарными**.

ми (out of vocabulary – **OOV**). Их появление неизбежно, потому что множество слов – **открытый класс**. Так, множество имен собственных (имен людей и названий мест) не ограничено.

Стандартная эвристика для решения этой проблемы – заменить все новые слова специальным символом **UNK** (unknown – неизвестно). Но при этом теряется информация. Например, встретив слово «athazagoraphobia», мы можем догадаться, что это «боязнь чего-то», поскольку производный от греческого слова суффикс «phobia» часто встречается в английском языке и означает «боязнь» (кстати говоря, «атазагорафобия» означает боязнь быть забытым или проигнорированным).

Мы могли бы работать на уровне символов, но тогда пришлось бы обучать модель, как группировать частые комбинации букв в слова. Лучше воспользоваться тем фактом, что у слов есть внутренняя структура, и брать в качестве входных данных **подслова**, или **части слов** [SHB16; Wu+16]; они часто создаются методом **кодирования пар байтов** [Gag94], который представляет собой вариант сжатия данных, когда для представления общих подстрок создаются новые символы.

1.5.5. Обработка отсутствующих данных

Иногда некоторые данные отсутствуют, т. е. части входа **x** или выхода **y** неизвестны. Если во время обучения выход неизвестен, то пример останется не помеченным; такие сценарии обучения с частичным привлечением учителя рассматриваются в разделе 19.3. Поэтому мы сосредоточимся на случае, когда отсутствуют некоторые входные признаки, – на этапе обучения, этапе тестирования или на обоих.

Для моделирования этой ситуации обозначим **M** матрицу бинарных переменных размера $N \times D$, в которой $M_{nd} = 1$, если признак d в примере n отсутствует, и $M_{nd} = 0$ в противном случае. Пусть \mathbf{X}_v – присутствующие части матрицы входных признаков, соответствующие элементам $M_{nd} = 0$, а \mathbf{X}_h – отсутствующие части, соответствующие элементам $M_{nd} = 1$. Обозначим **Y** матрицу выходных меток и предположим, что она наблюдаема целиком. Если $p(\mathbf{M}|\mathbf{X}_v, \mathbf{X}_h, \mathbf{Y}) = p(\mathbf{M})$, то говорят, что данные **отсутствуют вполне случайно** (missing completely at random – **MCAR**), поскольку присутствие или отсутствие данных вообще не зависит от скрытых или наблюдаемых признаков. Если $p(\mathbf{M}|\mathbf{X}_v, \mathbf{X}_h, \mathbf{Y}) = p(\mathbf{M}|\mathbf{X}_v, \mathbf{Y})$, то говорят, что данные **отсутствуют случайно** (missing at random – **MAR**), поскольку отсутствие не зависит от скрытых признаков, но может зависеть от наблюдаемых. Если ни то, ни другое условие не выполняется, то говорят, что данные **отсутствуют не случайно** (not missing at random – **NMAR**).

В случаях MCAR и MAR мы можем игнорировать механизм отсутствия, поскольку он ничего не говорит о скрытых признаках. Но в случае NMAR необходимо смоделировать **механизм отсутствия данных**, потому что само отсутствие информации может нести полезную информацию. Например, тот факт, что человек не дал ответа на деликатный вопрос в анкете (например, «Болеете ли вы ковидом?»), может дать информацию об истинном значе-

нии. Дополнительные сведения о моделях отсутствия данных см. в работах [LR87; Mar08].

В этой книге мы всегда будем придерживаться предположения MAR. Но даже в этом предположении мы не можем напрямую воспользоваться дискриминантной моделью, например ГНС, при наличии отсутствующих входных признаков, потому что вход x будет иметь неизвестные значения. Стандартная эвристика, применяемая в этом случае, называется **подстановкой среднего значения** – вместо отсутствующих значений включаются их эмпирические средние. Вообще, мы можем аппроксимировать входные данные порождающей моделью и использовать эту модель для **восполнения** отсутствующих значений. Подходящие для этой цели порождающие модели мы кратко обсудим в главе 20, а более полно во втором томе этой книги, [Mur22].

1.6. ОБСУЖДЕНИЕ

В этом разделе мы определим место МО и этой книги в более общем контексте.

1.6.1. Связь МО с другими дисциплинами

Есть несколько сообществ, работающих в связанных с МО областях, называемых по-разному. **Прогнозная аналитика** напоминает обучение с учителем (в особенности классификацию и регрессию), но в большей степени относится к бизнес-приложениям. **Добыча данных** охватывает машинное обучение с учителем и без учителя, но применяется в основном к структурированным данным, которые обычно хранятся в крупных коммерческих базах данных. В **науке о данных** (data science) применяются методы машинного обучения и статистики, но много внимания уделяется также другим вопросам, в частности интеграции данных, визуализации данных и работе со специалистами в предметной области, зачастую в итеративном цикле с обратной связью (см., например, [BS17]). Различие между этими дисциплинами преимущественно терминологическое¹.

МО также тесно связано с математической **статистикой**. Действительно, Джерри Фридман, известный профессор статистики в Стенфордском университете, писал²:

[Если бы статистика] с самого начала включала методологию вычислений в качестве основополагающего инструмента, а не просто была удобным способом приложения имеющихся у нас инструментов, то нужды во многих других дисциплинах, связанных с данными [например, МО], и не возникло бы – они просто были бы частью статистики.

— Джерри Фридман [Fri97b]

¹ См. полезный «Глоссарий МО» на страницу <https://developers.google.com/machine-learning/glossary/>.

² Цитируется по <https://brenocon.com/blog/2008/12/statistics-vs-machine-learning-fight/>.

Машинное обучение также связано с **искусственным интеллектом (ИИ)**. Исторически предполагалось, что мы сможем программировать «интеллект» вручную (см., например, [RN10; PM17]), но этот подход не выдержал столкновения с реальностью, в основном потому что явно закодировать все знания, необходимые таким системам, оказалось слишком трудно. Поэтому возродился интерес к МО как к средству помочь ИИ в приобретении собственных знаний. (На самом деле связи между МО и ИИ настолько тесные, что зачастую эти термины употребляют как синонимы, хотя это, пожалуй, только вводит в заблуждение [Pre21].)

1.6.2. Структура книги

Мы видели, что МО тесно связано со многими другими разделами математики, статистики, информатики и т. д. Поэтому трудно решить, с чего начать.

В этой книге мы выбрали один конкретный путь по этому ландшафту взаимосвязанных дисциплин, используя теорию вероятностей в качестве объединяющей призмы. В части I мы излагаем основы статистики, в частях II–IV рассматриваем обучение с учителем, а в части V – обучение без учителя. Дополнительные сведения об этих (и других) вопросах читатель найдет во втором томе этой книги, [Mur22],

В дополнение к книге на сайте <http://probml.ai> имеются Python-блокноты.

1.6.3. Подводные камни

В этой книге мы увидим, как можно использовать машинное обучение для создания систем, которые могут (или хотя бы пытаются) предсказывать выходы по заданным входам, а затем использовать предсказания для выбора действий с целью минимизировать ожидаемую потерю. При разработке таких систем бывает трудно спроектировать функцию потерь, которые точно отражает все наши предпочтения; это может приводить к **«манипулированию вознаграждением»** (reward hacking) – машина усердно оптимизирует заданную нами функцию вознаграждения, а потом обнаруживается, что функция не улавливает различных ограничений или предпочтений, которые мы забыли описать [Wei76; Amo+16; D'A+20]. (Это особенно важно, когда необходимо искать компромиссы между несколькими целями.)

Манипулирование вознаграждением поднимает различные вопросы в контексте **этики** и **безопасности ИИ** (см., например, [KR19; Lia20]). В работе [Rus19] предлагается решать эту проблему, отказавшись от задания функции вознаграждения, а заставив вместо этого машину выводить вознаграждение путем наблюдения за поведением человека; этот подход называется **обратным обучением с подкреплением**. Однако слишком точное подражание текущему или прошлому поведению человека может быть нежелательно, на него могут повлиять доступные для обучения данные (см., например, [Pau+20]).

Другой подход – рассматривать МО как инструмент для построения адаптивных компонентов, являющихся частями более крупной системы. Такую систему следует проектировать и регулировать так же, как мы проектируем и регулируем другие сложные полуавтономные продукты человеческой деятельности: самолеты, платформы для интернет-трейдинга или системы медицинской диагностики (см. [Jor19]). МО играет важную роль в таких системах, но необходимы дополнительные сдержки и противовесы, чтобы закодировать наши предшествующие знания и предпочтения и гарантировать, что система будет действовать так, как запланировано.

Часть I



ОСНОВАНИЯ

Глава 2

Вероятность: одномерные модели

2.1. ВВЕДЕНИЕ

Эта глава содержит краткое введение в основы теории вероятностей. Существует много книг, где эта тема рассматривается подробнее, например [GS97; BT08].

2.1.1. Что такое вероятность?

Теория вероятностей есть в сущности не что иное, как здравый смысл, сведенный к исчислению.

— *Пьер Лаплас*, 1812

Все мы без малейших сомнений говорим, что вероятность выпадения орла при подбрасывании (симметричной) монеты равна 50 %. Но что это значит? На самом деле есть две интерпретации вероятности. Одна называется **частотной**. При такой точке зрения вероятности представляют частоты событий при многократном повторении. Так, приведенное выше утверждение означает, что если много раз подбрасывать монету, то орел будет выпадать примерно в половине случаев¹. Другой подход называется **байесовской** интерпретацией вероятности. В этом случае вероятность является количественной мерой **неопределенности** или отсутствия знаний о чем-то, так что налицо фундаментальная связь с информацией, а не с повторяющимися испытаниями [Jay03; Lin06]. С точки зрения байесовского подхода, предыдущее утверждение означает нашу веру в том, что при следующем подбрасывании шансы выпадения орла или решки равны.

¹ На самом деле статистик из Стэнфорда (а в прошлом профессиональный фокусник) Перси Дяконис показал, что с вероятностью примерно 51 % будет выпадать та же сторона монеты, что и при первом подбрасывании, такова уж физика процесса [DHM07].

Важное преимущество байесовской интерпретации состоит в том, что ее можно использовать для моделирования неуверенности в исходе однократных событий, для которых понятие частоты при большом числе повторений лишено смысла. Например, мы можем поставить задачу о вычислении вероятности того, что полярные шапки растают к 2030 году. Это событие произойдет ноль или один раз, но уж никак не многократно. Тем не менее ничто не мешает нам количественно выразить степень неопределенности этого события и в зависимости от того, какова, на наш взгляд, его вероятность, принять решение об оптимальных действиях (см. главу 5). Поэтому мы в этой книге будем придерживаться байесовской интерпретации. К счастью, основные правила теории вероятностей не зависят от интерпретации.

2.1.2. Типы неопределенности

Для неопределенности предсказания есть две принципиально различные причины. Первая связана с тем, что мы не знаем истинный скрытый механизм порождения данных. Это называется **эпистемической неопределенностью**, потому что эпистемология – философская дисциплина, исследующая знание как таковое. Но есть и более простой термин – **неопределенность модели**. Второй вид неопределенности связан с внутренне присущей вариативностью, которую нельзя уменьшить, даже если собрать больше данных. Иногда это называют **алеаторной неопределенностью** [Нас75; KD09]. Это слово – производное от латинского названия игральные костей, а более простой термин – **неопределенность данных**. В качестве конкретного примера рассмотрим подбрасывание симметричной монеты. Мы можем быть уверены, что вероятность выпадения орла $p = 0.5$, так что ни для какой эпистемической неопределенности нет места, но все равно точно предсказать исход мы не можем.

Это различие может играть важную роль в таких приложениях, как активное обучение. Типичная стратегия – запрашивать примеры, для которых $\mathbb{H}(p(y|x, D))$ велико (где $\mathbb{H}(p)$ – энтропия, обсуждаемая в разделе 6.1). Однако это может быть вызвано как неопределенностью параметров, т. е. большой величиной $\mathbb{H}(p(\theta|D))$, так и просто зашумленностью или изменчивостью меток и соответственно большой энтропией $p(y|x, \theta)$. Продолжение обсуждения см. в работе [Osb16].

2.1.3. Вероятность как обобщение логики

В этом разделе мы обсудим основные правила теории вероятностей, опираясь на изложение в работе [Jay03], где вероятность рассматривается как обобщение **булевой логики**.

2.1.3.1. Вероятность события

Мы определяем **событие**, обозначаемое бинарной переменной A , как некоторое состояние мира, которое имеет или не имеет места. Например, A

может быть событием «завтра будет дождь» или «вчера шел дождь», или «метка $y = 1$ », или «параметр θ находится между 1.5 и 2.0» и т. д. Выражение $\Pr(A)$ обозначает вероятность того, что, по нашему мнению, событие A истинно (или долю многократных испытаний, в которых A произойдет). Требуется, чтобы $0 \leq \Pr(A) \leq 1$, где $\Pr(A) = 0$ означает, что событие заведомо не произойдет, а $\Pr(A) = 1$ – что оно произойдет обязательно. $\Pr(\bar{A})$ будет обозначать вероятность того, что событие A не произойдет; по определению $\Pr(\bar{A}) = 1 - \Pr(A)$.

2.1.3.2. Вероятность конъюнкции двух событий

Будем обозначать **совместную вероятность** событий A и B , т. е. вероятность того, что оба они произойдут:

$$\Pr(A \wedge B) = \Pr(A, B). \quad (2.1)$$

Если события A и B независимы, то

$$\Pr(A; B) = \Pr(A)\Pr(B). \quad (2.2)$$

Например, предположим, что X и Y случайно выбраны из множества $\mathcal{X} = \{1, 2, 3, 4\}$. Обозначим A событие $X \in \{1, 2\}$, а B событие $Y \in \{3\}$. Тогда $\Pr(A, B) = \Pr(A)\Pr(B) = \frac{1}{2} \cdot \frac{1}{4}$.

2.1.3.3. Вероятность объединения двух событий

Вероятность того, что произойдет событие A или событие B , равна

$$\Pr(A \vee B) = \Pr(A) + \Pr(B) - \Pr(A \wedge B). \quad (2.3)$$

Если события взаимно исключающие (т. е. не могут произойти одновременно), то имеем

$$\Pr(A \vee B) = \Pr(A) + \Pr(B). \quad (2.4)$$

Например, предположим, что X случайно выбрано из множества $\mathcal{X} = \{1, 2, 3, 4\}$. Обозначим A событие $X \in \{1, 2\}$, а B событие $Y \in \{3\}$. Тогда $\Pr(A \vee B) = \frac{2}{4} + \frac{1}{4}$.

2.1.3.4. Условная вероятность одного события при условии другого

Условная вероятность события B при условии, что событие A произошло, определяется следующим образом:

$$\Pr(B|A) \triangleq \frac{\Pr(A, B)}{\Pr(A)}. \quad (2.5)$$

Эта величина не определена, если $\Pr(A) = 0$, поскольку нельзя выставлять в качестве условия невозможное событие.

2.1.3.5. Независимость событий

Говорят, что событие A независимо от события B , если

$$\Pr(A, B) = \Pr(A)\Pr(B). \quad (2.6)$$

2.1.3.6. Условная независимость событий

Говорят, что события A и B **условно независимы** при условии события C , если

$$\Pr(A, B|C) = \Pr(A|C)\Pr(B|C). \quad (2.7)$$

Это записывается в виде $A \perp B|C$. События часто зависят друг от друга, но могут считаться независимыми, если обусловить их подходящими промежуточными переменными; мы обсудим это подробнее ниже в этой главе.

2.2. СЛУЧАЙНЫЕ ВЕЛИЧИНЫ

Предположим, что X представляет некоторую интересующую нас величину, например выпавшую грань игральной кости или температуру на улице в данный момент времени. Если значение X неизвестно и (или) может изменяться, то мы называем его **случайной величиной**. Множество возможных значений, обозначаемое \mathcal{X} , называется **пространством элементарных событий**, или **пространством состояний**. **Событием** называется подмножество заданного пространства элементарных событий. Например, если X представляет выпавшее на кости число, т. е. $\mathcal{X} = \{1, 2, \dots, 6\}$, то событие «выпала 1» обозначается $X = 1$, событие «выпало нечетное число» — $X \in \{1, 3, 5\}$, событие «выпало число от 1 до 3» — $1 \leq X \leq 3$ и т. д.

2.2.1. Дискретные случайные величины

Если пространство элементарных событий \mathcal{X} конечно или счетно, то X называется **дискретной случайной величиной**. В этом случае вероятность события « X принимает значение x » обозначается $\Pr(X = x)$. По определению **функция массы вероятности**, или просто **функция вероятности** (англ. **pmf**), сопоставляет возможному значению случайной величины его вероятность:

$$p(x) \triangleq \Pr(X = x). \quad (2.8)$$

Функция вероятности удовлетворяет условиям $0 \leq p(x) \leq 1$ и $\sum_{x \in \mathcal{X}} p(x) = 1$.

Если \mathcal{X} содержит конечное число значений, скажем K , то функцию вероятности можно представить в виде списка K чисел и изобразить в виде

гистограммы. Например, на рис. 2.1 показаны две функции вероятности, определенные на множестве $\mathcal{X} = \{1, 2, 3, 4\}$. Слева показано равномерное распределение, $p(x) = 1/4$, а справа вырожденное распределение, $p(x) = \mathbb{I}(x = 1)$, где $\mathbb{I}()$ – бинарная индикаторная функция. Таким образом, распределение на рис. 2.1b представляет тот факт, что X всегда равна 1. (Так что случайные величины могут быть и постоянными.)

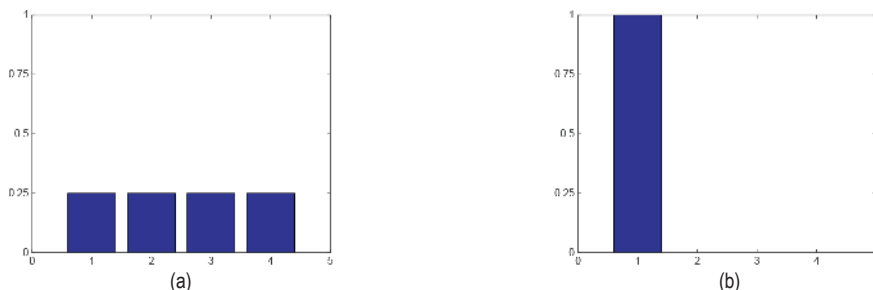


Рис. 2.1 ❖ Примеры дискретных распределений в пространстве элементарных событий $\mathcal{X} = \{1, 2, 3, 4\}$. (a) Равномерное распределение с $p(x = k) = 1/4$. (b) Вырожденное распределение (дельта-функция), вся масса которого сосредоточена в точке $x = 1$. Построено программой по адресу figures.problai/book1/2.1

2.2.2. Непрерывные случайные величины

Если $X \in \mathbb{R}$ вещественная, то она называется **непрерывной случайной величиной**. В этом случае уже не существует конечного (или счетного) множества ее возможных значений. Однако можно разбить вещественную прямую на счетное число *интервалов*. Если ассоциировать события с попаданием X в каждый из таких интервалов, то можно будет использовать методы, рассмотренные выше для дискретных случайных величин. Устремив размер интервалов к нулю, мы сможем представить вероятность равенства X конкретному значению, как будет показано ниже.

2.2.2.1. Функция распределения

Определим события $A = (X \leq a)$, $B = (X \leq b)$ и $C = (a < X \leq b)$, где $a < b$. Имеем $B = A \vee C$, а, поскольку A и C взаимно исключающие, то правило сложения вероятностей дает

$$\Pr(B) = \Pr(A) + \Pr(C), \quad (2.9)$$

и, следовательно, вероятность попадания в интервал C равна

$$\Pr(C) = \Pr(B) - \Pr(A). \quad (2.10)$$

В общем случае **функция распределения** случайной величины X определяется следующим образом:

$$P(x) \triangleq \Pr(X \leq x). \quad (2.11)$$

(Обратите внимание на заглавную букву P в определении функции распределения.) С ее помощью вероятность попадания в любой интервал вычисляется по формуле:

$$\Pr(a < X \leq b) = P(b) - P(a). \quad (2.12)$$

Любая функция распределения является неубывающей. На рис. 2.2а приведен пример стандартной функции гауссова распределения, $\mathcal{N}(x|0, 1)$; детали см. в разделе 2.6.

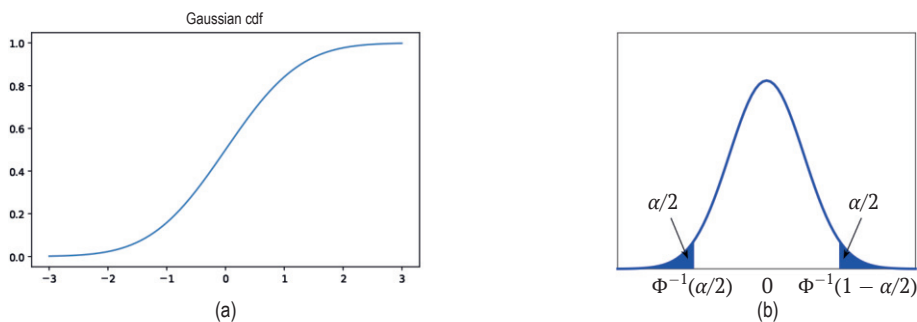


Рис. 2.2 ❖ (а) График стандартной функции нормального распределения, $\mathcal{N}(0, 1)$. Построено программой по адресу figures.probl.ai/book1/2.2. (б) Соответствующая функция плотности распределения. В каждой из закрашенных областей сосредоточена масса вероятности $\alpha/2$. Поэтому в незакрашенной области сосредоточена масса вероятности $1 - \alpha/2$. Координата левой точки отсечения равна $\Phi^{-1}(\alpha/2)$, где Φ – функция гауссова распределения. Из соображений симметрии координата правой точки отсечения равна $\Phi^{-1}(1 - \alpha/2) = -\Phi^{-1}(\alpha/2)$. Построено программой по адресу figures.probl.ai/book1/2.2

2.2.2.2. Функция плотности распределения

Определим **функцию плотности распределения** (ФПР) как производную функции распределения:

$$p(x) \triangleq \frac{d}{dx} \Pr(x). \quad (2.13)$$

(Отметим, что производная существует не всегда, а значит, и ФПР не всегда определена.) На рис. 2.2б приведен пример ФПР одномерного гауссова (нормального) распределения (детали см. в разделе 2.6). Зная ФПР, мы можем вычислить вероятность попадания непрерывной случайной величины в конечный интервал:

$$\Pr(a < X \leq b) = \int_a^b p(x) dx = P(b) - P(a). \quad (2.14)$$

При уменьшении ширины интервала получаем:

$$\Pr(x \leq X \leq x + dx) \approx p(x)dx. \quad (2.15)$$

Интуитивно это означает, что вероятность попадания X в малую окрестность точки x равна плотности вероятности в точке x , умноженной на ширину интервала.

2.2.2.3. Квантили

Если функция распределения P строго монотонно возрастает, то у нее имеется обратная функция, называемая **обратной функцией распределения**, **процентильной** или **квантильной функцией**.

Если P – функция распределения X , то $P^{-1}(q)$ равно такому значению x_q , что $\Pr(X \leq x_q) = q$; это называется q -м **квантилем** P . Значение $P^{-1}(0.5)$ называется **медианой** распределения; половина массы вероятности сосредоточена слева от него, а половина справа. Значения $P^{-1}(0.25)$ и $P^{-1}(0.75)$ называются **нижним** и **верхним квантилями**.

Пусть, например, Φ – функция гауссова распределения $\mathcal{N}(0, 1)$ и Φ^{-1} – обратная функция распределения. Тогда в области слева от $\Phi^{-1}(\alpha/2)$ сосредоточена часть $\alpha/2$ массы вероятности, как показано на рис. 2.2b. В силу симметрии в области справа от $\Phi^{-1}(1 - \alpha/2)$ также сосредоточена часть $\alpha/2$ массы. Поэтому в центральном интервале $(\Phi^{-1}(\alpha/2), \Phi^{-1}(1 - \alpha/2))$ сосредоточена масса $1 - \alpha$. Если положить $\alpha = 0.05$, то 95 % массы сосредоточено в центральном интервале:

$$(\Phi^{-1}(0.025), \Phi^{-1}(0.975)) = (-1.96, 1.96). \quad (2.16)$$

Для распределения $\mathcal{N}(\mu, \sigma^2)$ получается, что 95 % массы сосредоточено в интервале $(\mu - 1.96\sigma, \mu + 1.96\sigma)$. Часто вместо этого пишут приближенно: $\mu \pm 2\sigma$.

2.2.3. Множества связанных случайных величин

В этом разделе мы обсудим распределения множеств связанных случайных величин.

Пусть имеется две случайные величины, X и Y . Мы можем определить **совместное распределение** двух случайных величины, положив $p(x, y) = \Pr(X = x, Y = y)$ для всех возможных значений X и Y . Если обе величины принимают конечное множество значений, то совместное распределение можно представить в виде двумерной таблицы, сумма элементов которой равна 1. Например, рассмотрим такие две бинарных величины:

$p(X, Y)$	$Y = 0$	$Y = 1$
$X = 0$	0.2	0.3
$X = 1$	0.3	0.2

Если случайные величины независимы, то совместное распределение можно представить в виде произведения двух маргинальных распределений. Если величины принимают конечное множество значений, то таблицу совместного распределения можно разложить в произведение двух векторов, как показано на рис. 2.3.

Для данного совместного распределения определим **маргинальное распределение** случайной величины следующим образом:

$$p(X = x) = \sum_y p(X = x, Y = y), \quad (2.17)$$

где суммирование производится по всем возможным значениям Y . Иногда эту формулу называют **правилом сложения вероятностей** или **формулой полной вероятности**. Вероятность $p(Y = y)$ определяется аналогично. Например, из приведенной выше таблицы имеем $p(X = 0) = 0.2 + 0.3 = 0.5$ и $p(Y = 0) = 0.2 + 0.3 = 0.5$. (Термин «маргинальный» происходит от английского слова «margin», поле, и восходит к распространенной в бухгалтерском учете практике записывания сумм по строкам и по столбцам на полях таблицы.)

Определим **условное распределение** случайной величины:

$$p(Y = y | X = x) = \frac{p(X = x, Y = y)}{p(X = x)}. \quad (2.18)$$

Эту формулу можно переписать в виде:

$$p(x, y) = p(x)p(y|x). \quad (2.19)$$

Эта формула называется **правилом умножения вероятностей**.

Обобщая правило умножения вероятностей на D случайных величин, получаем **цепное правило**:

$$p(\mathbf{x}_{1:D}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)p(x_4|x_1, x_2, x_3) \dots p(x_D|x_{1:D-1}). \quad (2.20)$$

Оно дает способ строить многомерные совместные распределения из набора условных распределений. Более подробно мы обсудим этот вопрос в разделе 3.6.

2.2.4. Независимость и условная независимость

Говорят, что случайные величины X и Y **безусловно независимы** (обозначается $X \perp Y$), если их совместное распределение можно представить в виде произведения двух маргинальных распределений (см. рис. 2.3), т. е.

$$X \perp Y \Leftrightarrow p(X, Y) = p(X)p(Y). \quad (2.21)$$

В общем случае множество случайных величин X_1, \dots, X_n называется **независимым**, если их совместное распределение можно записать в виде произведения маргинальных распределений:

$$p(X_1, \dots, X_n) = \prod_{i=1}^n p(X_i). \quad (2.22)$$

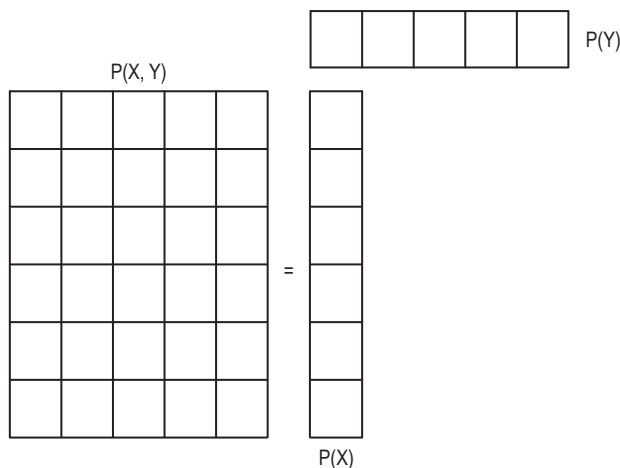


Рис. 2.3 ❖ Вычисление $p(x, y) = p(x)p(y)$, где $X \perp Y$. Здесь X и Y – дискретные случайные величины; X имеет 6 возможных состояний (значений), а Y – 5 возможных состояний. В общем случае для определения совместного распределения таких двух величин потребовалось бы $(6 \times 5) - 1 = 29$ параметров (1 вычитается, поскольку сумма всех вероятностей должна быть равна единице). Но в предположении безусловной независимости нужно только $(6 - 1) + (5 - 1) = 9$ параметров, чтобы определить $p(x, y)$

К сожалению, безусловная независимость встречается редко, потому что одни случайные величины, как правило, влияют на другие. Однако обычно это влияние не прямое, а опосредуется промежуточными величинами. Поэтому мы говорим, что X и Y **условно независимы** при условии Z , если условное совместное распределение можно записать в виде произведения условных маргинальных распределений:

$$X \perp Y | Z \Leftrightarrow p(X, Y | Z) = p(X | Z)p(Y | Z). \quad (2.23)$$

Это предположение можно выразить в виде графа $X - Z - Y$, который отражает интуитивную идею о том, что все зависимости между X и Y опосредованы Z . Воспользовавшись более крупными графами, мы сможем определить сложные совместные распределения; эти так называемые **графовые модели** обсуждаются в разделе 3.6.

2.2.5. Моменты распределения

В этом разделе мы опишем различные сводные статистики, которые можно вывести из распределения вероятностей.

2.2.5.1. Среднее распределения

Самое знакомое свойство распределения – его **среднее**, или **математическое ожидание**, которое часто обозначают μ . Для непрерывной случайной величины среднее определяется следующим образом:

$$\mathbb{E}[X] \triangleq \int_{\mathcal{X}} xp(x)dx. \quad (2.24)$$

Если этот интеграл расходится, то среднее не определено; такие примеры мы встретим ниже.

Для дискретной случайной величины среднее определяется формулой:

$$\mathbb{E}[X] \triangleq \sum_{x \in \mathcal{X}} xp(x). \quad (2.25)$$

Однако это определение имеет смысл, только если значения x каким-то образом упорядочены (например, представлены целыми числами).

Поскольку среднее – линейный оператор, имеем:

$$\mathbb{E}[aX + b] = a\mathbb{E}[X] + b. \quad (2.26)$$

Это тождество называется **линейностью математического ожидания**.

Для множества n случайных величин можно показать, что математическое ожидание их суммы равно сумме математических ожиданий:

$$\mathbb{E}\left[\sum_{i=1}^n X_i\right] = \sum_{i=1}^n \mathbb{E}[X_i]. \quad (2.27)$$

Если они к тому же независимы, то математическое ожидание произведения равно произведению математических ожиданий:

$$\mathbb{E}\left[\prod_{i=1}^n X_i\right] = \prod_{i=1}^n \mathbb{E}[X_i]. \quad (2.28)$$

2.2.5.2. Дисперсия распределения

Дисперсия – это мера «разброса» распределения, она часто обозначается σ^2 и определяется следующим образом:

$$\mathbb{V}[X] \triangleq \mathbb{E}[(X - \mu)^2] = \int (x - \mu)^2 p(x)dx \quad (2.29)$$

$$= \int x^2 p(x)dx + \mu^2 \int p(x)dx - 2\mu \int xp(x)dx = \mathbb{E}[X^2] - \mu^2. \quad (2.30)$$

Из этого определения можно вывести полезный результат:

$$\mathbb{E}[X^2] = \sigma^2 + \mu^2. \quad (2.31)$$

Стандартное отклонение по определению равно:

$$\text{std}[X] \triangleq \sqrt{\mathbb{V}[X]} = \sigma. \quad (2.32)$$

Оно полезно, потому что измеряется в тех же единицах, что сама X . Дисперсия сдвинутой и масштабированной случайной величины равна:

$$\mathbb{V}[aX + b] = a^2 \mathbb{V}[X]. \quad (2.33)$$

Если n случайных величин независимы, то дисперсия их суммы равна сумме дисперсий:

$$\mathbb{V}\left[\sum_{i=1}^n X_i\right] = \sum_{i=1}^n \mathbb{V}[X_i]. \quad (2.34)$$

Формулу дисперсии произведения можно вывести аналогично:

$$\mathbb{V}\left[\prod_{i=1}^n X_i\right] = \mathbb{E}\left[\left(\prod_i X_i\right)^2\right] - \left(\mathbb{E}\left[\prod_i X_i\right]\right)^2 \quad (2.35)$$

$$= \mathbb{E}\left[\prod_i X_i^2\right] - \left(\prod_i \mathbb{E}[X_i]\right)^2 \quad (2.36)$$

$$= \prod_i \mathbb{E}[X_i^2] - \prod_i (\mathbb{E}[X_i])^2 \quad (2.37)$$

$$= \prod_i (\mathbb{V}[X_i] + (\mathbb{E}[X_i])^2) - \prod_i (\mathbb{E}[X_i])^2 \quad (2.38)$$

$$= \prod_i (\sigma_i^2 + \mu_i^2) - \prod_i \mu_i^2. \quad (2.39)$$

2.2.5.3. Мода распределения

Модой распределения называется значение с наибольшей массой или плотностью вероятности:

$$\mathbf{x}^* - \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}). \quad (2.40)$$

Если распределение **многомодальное**, то мода может быть не единственной, как показано на рис. 2.4. Но, даже если мода единственна, она не всегда дает хорошее общее представление о распределении.

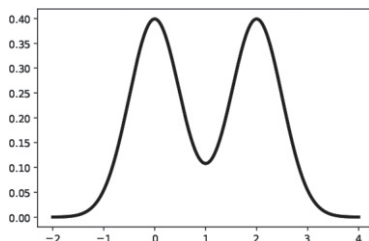


Рис. 2.4 ❖ Смесь двух одномерных гауссовых распределений,
 $p(x) = 0.5\mathcal{N}(x|0, 0.5) + 0.5\mathcal{N}(x|2, 0.5)$.

Построено программой по адресу figures.problm.ai/book1/2.4

2.2.5.4. Условные моменты

Если имеется две случайные величины или более, то можно вывести моменты одной, зная что-то о других. Например, согласно **формуле полного математического ожидания**:

$$\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|Y]]. \quad (2.41)$$

Чтобы доказать ее, предположим для простоты, что X и Y – дискретные случайные величины. Тогда

$$\mathbb{E}[\mathbb{E}[X|Y]] = \mathbb{E}\left[\sum_x xp(X=x|Y)\right] \quad (2.42)$$

$$= \sum_y \left[\sum_x xp(X=x|Y) \right] p(Y=y) = \sum_{x,y} xp(X=x, Y=y) = \mathbb{E}[X]. \quad (2.43)$$

Для иллюстрации рассмотрим простой пример¹. Пусть X – срок службы электрической лампочки, а Y – завод, на котором она была изготовлена. Предположим, что $\mathbb{E}[X|Y=1] = 5000$ и $\mathbb{E}[X|Y=2] = 4000$, т. е. на заводе 1 производятся более долговечные лампочки. Предположим еще, что на заводе 1 изготавливается 60 % лампочек, так что $p(Y=1) = 0.6$ и $p(Y=2) = 0.4$. Тогда ожидаемый срок службы случайной лампочки равен

$$\begin{aligned} \mathbb{E}[X] &= \mathbb{E}[X|Y=1]p(Y=1) + \mathbb{E}[X|Y=2]p(Y=2) \\ &= 5000 \times 0.6 + 4000 \times 0.4 = 4600. \end{aligned} \quad (2.44)$$

Аналогичная формула имеется для дисперсии. Именно, **формула полной дисперсии**, называемая также **формулой условной дисперсии**, имеет вид:

$$\mathbb{V}[X] = \mathbb{E}[\mathbb{V}[X|Y]] + \mathbb{V}[\mathbb{E}[X|Y]]. \quad (2.45)$$

Чтобы убедиться в этом, определим условные моменты, $\mu_{X|Y} = \mathbb{E}[X|Y]$, $s_{X|Y} = \mathbb{E}[X^2|Y]$ и $\sigma^2_{X|Y} = \mathbb{V}[X|Y] = s_{X|Y} - \mu^2_{X|Y}$, являющиеся функциями от Y (а потому случайными величинами). Тогда

$$\mathbb{V}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 = \mathbb{E}[s_{X|Y}] - (\mathbb{E}[\mu_{X|Y}])^2 \quad (2.46)$$

$$= \mathbb{E}[\sigma^2_{X|Y}] + \mathbb{E}[\mu^2_{X|Y}] - (\mathbb{E}[\mu_{X|Y}])^2 \quad (2.47)$$

$$= \mathbb{E}_Y[\mathbb{V}[X|Y]] + \mathbb{V}_Y[\mu_{X|Y}]. \quad (2.48)$$

Чтобы лучше уяснить эти формулы, рассмотрим смесь K одномерных гауссовых распределений. Обозначим Y скрытую индикаторную переменную, которая говорит, какая компонента смеси используется, и положим $X = \sum_{y=1}^K \pi_y \mathcal{N}(X|\mu_y, \sigma_y)$. На рис. 2.4 имеем $\pi_1 = \pi_2 = 0.5$, $\mu_1 = 0$, $\mu_2 = 2$, $\sigma_1 = \sigma_2 = 0.5$. Таким образом,

$$\mathbb{E}[\mathbb{V}[X|Y]] = \pi_1 \sigma_1^2 + \pi_2 \sigma_2^2 = 0.5; \quad (2.49)$$

¹ Пример взят из статьи https://en.wikipedia.org/wiki/Law_of_total_expectation, но приведен в других обозначениях.

$$\begin{aligned}\mathbb{V}[\mathbb{E}[X|Y]] &= \pi_1(\mu_1 - \bar{\mu})^2 + \pi_2(\mu_2 - \bar{\mu})^2 = 0.5(0 - 1)^2 + 0.5(2 - 1)^2 \\ &= 0.5 + 0.5 = 1.\end{aligned}\quad (2.50)$$

Мы получили интуитивно понятный результат: на дисперсию X большее влияние оказывает то, из какого центроида она выведена (т. е. разность средних), а не локальная дисперсия в окрестности каждого центроида.

2.2.6. Ограничения сводных статистик*

Хотя распределение вероятностей (или выбранные из него точки) принято характеризовать с помощью таких простых статистик, как среднее и дисперсия, при этом может теряться важная информация. Поразительный пример этого явления дает **квартет Энскомба** [Ans73], показанный на рис. 2.5. Мы видим 4 разных набора данных, состоящих из пар (x, y) , у всех них статистики низшего порядка одинаковы: $\mathbb{E}[x] = 9$, $\mathbb{V}[x] = 11$, $\mathbb{E}[y] = 7.50$, $\mathbb{V}[y] = 4.125$ и $\rho = 0.816$ (величина ρ – коэффициент корреляции, определенный в разделе 3.1.2). Однако совместные распределения $p(x, y)$, из которых выбраны эти точки, очевидно, сильно различаются. Энскомб придумал эти наборы данных, по 10 точек в каждом, чтобы развеять укоренившееся среди статистиков представление, будто числовые сводные характеристики лучше визуализации данных [Ans73].

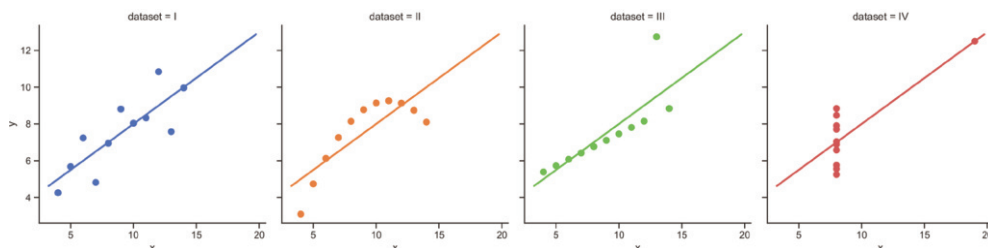


Рис. 2.5 ❖ Иллюстрация квартета Энскомба. У всех этих наборов данных одинаковые сводные статистики низшего порядка. Построено программой по адресу figures.problmlai/book1/2.5

Еще более убедительный пример этого феномена показан на рис. 2.6. Он включает набор данных, напоминающий динозавра¹ и еще 11 наборов; для всех них статистики низшего порядка одинаковы. Это собрание наборов данных называется **датазавровой дюжиной** (Datasaurus Dozen) [MF17]. Точные значения точек (x, y) можно найти в сети². Они были вычислены методом имитации отжига – методом оптимизации, не требующим вычис-

¹ Этот набор данных создал Альберто Каиро, он доступен по адресу <http://www.thefunctionalart.com/2016/08/download-datasaurus-never-trust-summary.html>.

² <https://www.autodesk.com/research/publications/same-stats-different-graphs>. На самом деле наборов данных 13, включая динозавра. Мы опустили набор данных «away», чтобы картинка на странице выглядела аккуратнее.

ления производных, который мы обсудим во втором томе книги, [Mur22]. (Оптимизируемая целевая функция равна сумме отклонения от целевой сводной статистики исходного динозавра и расстояния до конкретной целевой формы.)

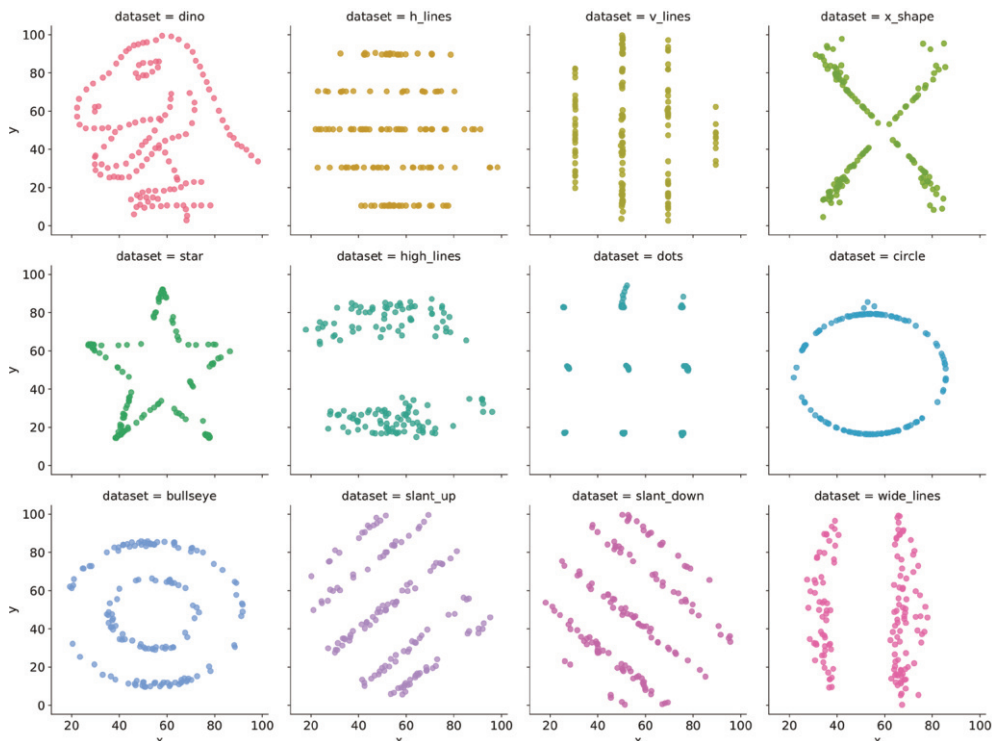


Рис. 2.6 ❖ Датазаврова дюжина.

У всех наборов данных одинаковые сводные статистики низшего порядка.

Основано на рис. 1 из работы [MF17].

Построено программой по адресу figures.probml.ai/book1/2.6

Имитацию отжига можно применить также к одномерным наборам данных, как показано на рис. 2.7. Мы видим, что наборы данных абсолютно различны, но у всех них одинаковые медиана и **межквартильный диапазон**, представленный центральными закрашенными «ящиками» на среднем рисунке. Еще более наглядную визуализацию дают **скрипичные диаграммы** на рисунке справа. На них показана оценка плотности одномерного ядра (два экземпляра) (раздел 16.3) распределения по вертикальной оси в дополнение к маркерам медианы и межквартильного диапазона. Эта визуализация способна лучше подчеркнуть различия распределений, однако она применима только к одномерным данным.



Рис. 2.7 ❖ Семь различных наборов данных (слева), соответствующие им диаграммы «ящик с усами» (в середине) и скрипичные диаграммы (справа). Взято из работы <https://www.autodesk.com/research/publications/same-stats-different-graphs> (рис. 8). Печатается с разрешения Юстина Матейка

2.3. Формула БАЙЕСА

Теорема Байеса играет в теории вероятностей такую же роль, как теорема Пифагора в геометрии.

— сэр Харольд Джеффрис, 1973 [Jef73]

В этом разделе мы обсудим основы **байесовского вывода**. Согласно словари Мерриам-Вебстера, «выводом» (inference) называется «акт перехода от выборочных данных к обобщениям, обычно сопровождаемый вычислением степени достоверности». Термин «байесовский» выделяет способы вывода, в которых «степень достоверности» вычисляется методами теории вероятностей, а для уточнения степени достоверности при появлении дополнительных данных применяется **формула Байеса**¹.

Сама формула Байеса очень проста: она служит для вычисления распределения вероятностей возможных значений неизвестной (или **скрытой**) величины H при наличии наблюдаемых данных $Y = y$:

$$p(H = h|Y = y) = \frac{p(H = h)p(Y = y|H = h)}{p(Y = y)}. \quad (2.51)$$

Это автоматически следует из тождества

$$p(h|y)p(y) = p(h)p(y|h) = p(h, y), \quad (2.52)$$

которое в свою очередь есть следствие **правила умножения вероятностей**.

В формуле (2.51) член $p(H)$ представляет то, что нам известно о возможных значениях H до наблюдения данных; это называется **априорным распределением**. (Если H может принимать K значений, то $p(H)$ – вектор, содержа-

¹ Томас Байес (1702–1761) – английский математик и пресвитерианский священник.

щий K вероятностей, сумма которых равна 1.) Член $p(Y|H = h)$ представляет распределение возможных исходов Y , которое мы ожидаем увидеть, если $H = h$; оно называется **распределением наблюдаемых значений**. Вычисляя его в точке, соответствующей фактическим наблюдениям, y , мы получаем функцию $p(Y = y|H = h)$, называемую **правдоподобием**. (Отметим, что это функция h , потому что y фиксировано, но она не является распределением вероятностей, так как сумма не равна 1.) Умножая априорное распределение $p(H = h)$ на функцию правдоподобия $p(Y = y|H = h)$ для каждого h , получаем ненормированное совместное распределение $p(H = h, Y = y)$. Его можно преобразовать в нормированное распределение, поделив на $p(Y = y)$; результат называется **маргинальным правдоподобием**, потому что вычислен с помощью маргинальных распределений неизвестной величины H :

$$p(Y = y) = \sum_{h' \in \mathcal{H}} p(H = h')p(Y = y|H = h') = \sum_{h' \in \mathcal{H}} p(H = h', Y = y'). \quad (2.53)$$

Нормировка совместного распределения путем вычисления $p(H = h, Y = y)/p(Y = y)$ для каждого h дает **апостериорное распределение** $p(H = h|Y = y)$, которое представляет наш новый взгляд на возможные значения H .

Словами формулу Байеса можно выразить так:

$$\text{апостериорное} \propto \text{априорное} \times \text{правдоподобие}. \quad (2.54)$$

Здесь символ \propto означает «пропорционально», поскольку мы игнорируем знаменатель, так как это константа, не зависящая от H . Применение формулы Байеса для уточнения распределения неизвестных значений некоторой представляющей интерес величины при наличии релевантных наблюдаемых данных называется **байесовским выводом**, или **апостериорным выводом**. А иногда говорят просто о **вероятностном выводе**.

Ниже мы приведем простые примеры байесовского вывода в действии. А впоследствии увидим и более интересные примеры.

2.3.1. Пример: тестирование на COVID-19

Допустим, вы подозреваете, что подхватили **COVID-19**, заразную болезнь, вызываемую вирусом **SARS-CoV-2**. Вы решили сдать диагностический тест и на основе его результата определить, больны вы или нет.

Пусть событие $H = 1$ означает, что вы заразились, а $H = 0$ – что не заразились. Пусть $Y = 1$, если тест положительный, а $Y = 0$, если отрицательный. Мы хотим вычислить $p(H = h|Y = y)$ для $h \in \{0, 1\}$, где y – наблюдаемый результат теста. (Для краткости будем записывать распределение значений $[p(H = 0|Y = y), p(H = 1|Y = y)]$ просто $p(H|y)$.) Эту задачу можно рассматривать как вариант бинарной классификации, когда H – неизвестная метка класса, а y – вектор признаков.

Сначала нужно определить правдоподобие. Эта величина, очевидно, зависит от надежности теста. Существует два основных параметра. **Чувствительность** (или **частота истинно положительных результатов**) определяется

как $p(Y = 1|H = 1)$, т. е. равна вероятности получить положительный результат теста, когда вы действительно больны. **Частота ложноотрицательных результатов** определяется как единица минус чувствительность. **Специфичность** (или **частота истинно отрицательных результатов**) определяется как $p(Y = 0|H = 0)$, т. е. равна вероятности получить отрицательный результат теста, когда вы действительно не больны. **Частота ложноположительных результатов** определяется как единица минус специфичность. Все эти величины сведены в табл. 2.1 (дополнительные сведения см. в разделе 5.1.3.1). Следуя статье <https://nyti.ms/31MTZgV>, будем считать, что чувствительность равна 87.5 %, а специфичность – 97.5 %.

Таблица 2.1. Функция правдоподобия $p(Y|H)$ для бинарного наблюдения Y при двух возможных скрытых состояниях H . Сумма по каждой строке равна 1. Аббревиатуры: TNR (true negative rate) – частота истинно отрицательных результатов, TPR (true positive rate) – частота истинно положительных результатов, FNR (false negative rate) – частота ложноотрицательных результатов, FPR (false positive rate) – частота ложноположительных результатов

		Наблюдение	
		0	1
Истина	0	TNR=Специфичность=0.975	FPR=1-TNR=0.025
	1	FNR=1-TPR=0.125	TPR=Чувствительность=0.875

Далее следует определить априорное распределение. Величина $p(H = 1)$ представляет **распространенность** заболевания в вашем регионе. Положим, $p(H = 1) = 0.1$ (т. е. 10 %), такова была распространенность в Нью-Йорке весной 2020 года. (Этот пример составлен на основе данных, приведенных в работе <https://nyti.ms/31MTZgV>.)

Предположим, что тест положительный. Имеем:

$$p(H = 1|Y = 1) = \frac{p(Y = 1|H = 1)p(H = 1)}{p(Y = 1|H = 1)p(H = 1) + p(Y = 1|H = 0)p(H = 0)} \quad (2.55)$$

$$= \frac{\text{TPR} \times \text{prior}}{\text{TPR} \times \text{prior} + \text{FPR} \times (1 - \text{prior})} \quad (2.56)$$

$$= \frac{0.875 \times 0.1}{0.875 \times 0.1 + 0.025 \times 0.9} = 0.795. \quad (2.57)$$

Таким образом, с вероятностью 79.5 % вы заразились.

Теперь предположим, что тест отрицательный. Тогда вероятность заражения равна:

$$p(H = 1|Y = 0) = \frac{p(Y = 0|H = 1)p(H = 1)}{p(Y = 0|H = 1)p(H = 1) + p(Y = 0|H = 0)p(H = 0)} \quad (2.58)$$

$$= \frac{\text{FNR} \times \text{prior}}{\text{FNR} \times \text{prior} + \text{TNR} \times (1 - \text{prior})} \quad (2.59)$$

$$= \frac{0.125 \times 0.1}{0.125 \times 0.1 + 0.975 \times 0.9} = 0.014. \quad (2.60)$$

То есть вы больны с вероятностью всего 1.4 %.

В настоящее время распространенность COVID-19 гораздо ниже. Повторим эти вычисления для коэффициента 1 %; теперь апостериорные вероятности уменьшаются до 26 и 0.13 % соответственно.

Тот факт, что вы больны COVID-19 всего лишь с вероятностью 26 % даже после положительного теста, противоречит интуиции. А причина в том, что единственный положительный тест, скорее всего, является ложноположительным, потому что болезнь встречается редко. Чтобы убедиться в этом, предположим, что имеется 100 000 человек, из которых 1000 больны. Для $875 = 0.875 \times 1000$ из этой тысячи тест положителен, а для здоровых тест будет положительным в $2475 = 0.025 \times 99\,000$ случаях. Таким образом, общее число положительных тестов равно $3350 = 875 + 2475$, поэтому апостериорная вероятность быть больным при положительном тесте равна $875/3350 = 0.26$.

Разумеется, в приведенных выше вычислениях предполагается, что мы знаем чувствительность и специфичность теста. В работе [GC20] описывается, как применять формулу Байеса к диагностическому тестированию, если эти параметры достоверно неизвестны.

2.3.2. Пример: парадокс Монти Холла

В этом разделе мы рассмотрим более «фривольное» применение формулы Байеса – **парадокс Монти Холла**.

Представьте себе игровое шоу со следующими правилами. Имеется три двери с табличками 1, 2, 3. За одной из дверей находится приз (например, автомобиль). Вы должны выбрать одну дверь. Затем ведущий, который знает, где автомобиль, открывает одну из двух других дверей, но не ту, за которой спрятан приз. После этого вам предоставляется возможность выбрать дверь еще раз: вы можете оставить прежнее решение или выбрать вторую из закрытых дверей. Затем все двери открываются, и вы получаете то, что находится за выбранной дверью.

Например, предположим, что вы выбрали дверь 1, а ведущий открыл дверь 3, за которой, конечно, ничего нет. Что для вас выгоднее: (а) остаться верным двери 1 или (б) изменить решение и выбрать дверь 2? Или же это не имеет значения?

Интуитивно кажется, что это должно быть безразлично, потому что первоначальный выбор двери не может повлиять на местоположение приза. Однако тот факт, что ведущий открыл дверь 3, сообщает нам кое-какую информацию о том, где приз, так как его выбор был обусловлен знанием истинного местоположения и вашим выбором. Как будет показано ниже, вы удвоите шансы на выигрыш, если измените решение и выберете дверь 3.

Чтобы убедиться в этом, воспользуемся формулой Байеса. Обозначим H_i гипотезу, согласно которой приз находится за дверью i . Сделаем следующие допущения: все три гипотезы H_1 , H_2 и H_3 априори равновероятны, т. е.:

$$P(H_1) = P(H_2) = P(H_3) = \frac{1}{3}. \quad (2.61)$$

После выбора двери 1 ведущий может выбрать $Y = 3$ или $Y = 2$ (т. е. открыть дверь 3 или 2 соответственно). Мы предполагаем, что эти два возможных исхода имеют следующие вероятности. Если приз находится за дверью 1, то ведущий случайно выбирает $Y = 2$ или $Y = 3$. В противном случае у ведущего нет выбора и вероятности равны 0 и 1.

$$\left| \begin{array}{l} P(Y = 2|H_1) = \frac{1}{2} \\ P(Y = 3|H_1) = \frac{1}{2} \end{array} \right| \left| \begin{array}{l} P(Y = 2|H_2) = 0 \\ P(Y = 3|H_2) = 1 \end{array} \right| \left| \begin{array}{l} P(Y = 2|H_3) = 1 \\ P(Y = 3|H_3) = 0 \end{array} \right|. \quad (2.62)$$

Теперь по формуле Байеса вычисляем апостериорные вероятности гипотез:

$$P(H_i|Y = 3) = \frac{P(Y = 3|H_i)P(H_i)}{P(Y = 3)}; \quad (2.63)$$

$$\left| P(H_1|Y = 3) = \frac{(1/2)(1/3)}{P(Y = 3)} \right| \left| P(H_2|Y = 3) = \frac{(1)(1/3)}{P(Y = 3)} \right| \left| P(H_3|Y = 3) = \frac{(0)(1/3)}{P(Y = 3)} \right|. \quad (2.64)$$

Знаменатель $P(Y = 3) = 1/6 + 1/3 = 1/2$. Следовательно,

$$\left| P(H_1|Y = 3) = \frac{1}{3} \right| \left| P(H_2|Y = 3) = \frac{2}{3} \right| \left| P(H_3|Y = 3) = 0 \right|. \quad (2.65)$$

Таким образом, участник должен изменить решение и выбрать дверь 2, если хочет максимизировать свои шансы получить приз. Конкретный пример приведен в табл. 2.2.

Таблица 2.2. Три возможных состояния игры Монти Холла, показывающие, что изменить решение в два раза выгоднее (в среднем), чем оставить первоначальное. Взято из работы [PM18] (табл. 6.1)

Дверь 1	Дверь 2	Дверь 3	Изменить	Оставить
Автомобиль	–	–	Проиграл	Выиграл
–	Автомобиль	–	Выиграл	Проиграл
–	–	Автомобиль	Выиграл	Проиграл

Многие не могут поверить в этот результат. Можно сделать его интуитивно более понятным, если поставить мысленный эксперимент, предположив, что дверей миллион. Теперь правила таковы: участник выбирает одну дверь, а затем ведущий открывает 999 998 дверей, за которыми приза заведомо нет. Остается одна дверь, выбранная участником, и одна закрытая дверь. Участник может оставить прежнее решение или переменить его. Поставьте себя на место участника, который видит перед собой миллион дверей, при

этом закрыты только двери 1 (первоначально выбранная) и 234 598. Как вы думаете, где находится приз?

2.3.3. Обратные задачи*

Теория вероятностей занимается предсказанием распределения исходов y при наличии знаний (или предположений) о состоянии окружающего мира, h . А теория **обратной вероятности** занимается выводом состояния мира из наблюдаемых исходов. Можно рассматривать это как обращение отображения $h \rightarrow y$.

Например, попробуем вывести форму трехмерного тела h из двумерного изображения y , это классическая задача **понимания визуальной сцены**. К сожалению, ее постановка принципиально **некорректна**, как следует из рис. 2.8, потому что существует много возможных h , совместимых с наблюдаемым y (см., например, [Piz01]). Аналогично можно считать некорректной задачу **понимания естественного языка**, когда слушатель должен вывести, что имел в виду говорящий h , из (зачастую неоднозначных) произнесенных им слов (см., например, [Sab21]).

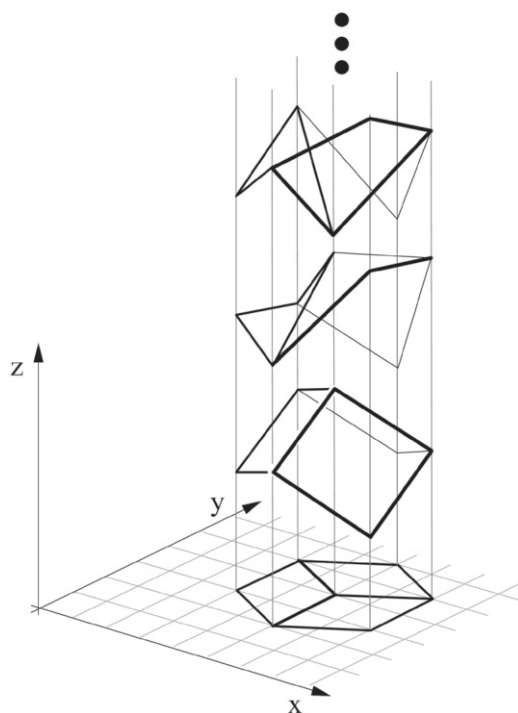


Рис. 2.8 ❖ Любая нарисованная на плоскости линия геометрически совместима с бесконечным множеством трехмерных тел. Взято из работы [SA93] (рис. 11). Печатается с разрешения Павана Синха

Для решения таких **обратных задач** можно воспользоваться формулой Байеса для вычисления апостериорного распределения $p(h|y)$, т. е. распределения возможных состояний окружающего мира. Для этого необходимо задать **прямую модель** $p(y|h)$, а также априорное распределение $p(h)$, которое можно использовать для исключения (или понижения веса) неправдоподобных состояний мира. Мы обсудим эту тему подробнее во втором томе книги, [Mur22].

2.4. РАСПРЕДЕЛЕНИЕ БЕРНУЛЛИ И БИНОМИАЛЬНОЕ РАСПРЕДЕЛЕНИЕ

Пожалуй, самым простым распределением является **распределение Бернулли**, которое служит для моделирования бинарных событий.

2.4.1. Определение

Рассмотрим подбрасывание монеты и предположим, что вероятность выпадения орла равна $0 \leq \theta \leq 1$. Обозначим это событие $Y = 1$, и пусть $Y = 0$ означает событие выпадения решки. Таким образом, мы предполагаем, что $p(Y = 1) = \theta$ и $p(Y = 0) = 1 - \theta$. Это и есть **распределение Бернулли**, которое можно записать в виде:

$$Y \sim \text{Ber}(\theta), \quad (2.66)$$

где символ \sim означает «выбрано из» или «распределено как», а Ber – сокращение от *Bernoulli* (Бернулли). Функция вероятности этого распределения определена следующим образом:

$$\text{Ber}(y|\theta) = \begin{cases} 1 - \theta, & \text{если } y = 0 \\ \theta, & \text{если } y = 1 \end{cases}. \quad (2.67)$$

(О функциях вероятности см. раздел 2.2.1.) Это можно записать короче:

$$\text{Ber}(y|\theta) \triangleq \theta^y (1 - \theta)^{1-y}. \quad (2.68)$$

Распределение Бернулли – частный случай **биномиального распределения**. Чтобы объяснить это, предположим, что наблюдаются исходы N испытаний Бернулли, которые мы обозначим $y_n \sim \text{Ber}(\cdot|\theta)$, $n = 1, \dots, N$. Конкретно будем говорить о подбрасывании монеты N раз. Пусть s – количество выпадений орла, $s \triangleq \sum_{n=1}^N \mathbb{I}(y_n = 1)$. Случайная величина s имеет биномиальное распределение:

$$\text{Bin}(s|N, \theta) \triangleq \binom{N}{s} \theta^s (1 - \theta)^{N-s}, \quad (2.69)$$

где

$$\binom{N}{k} \triangleq \frac{N!}{(N-k)!k!} \quad (2.70)$$

– число способов выбрать k элементов из N (эта величина называется **биномиальным коэффициентом** и произносится «выбор k из N »¹). Два примера биномиального распределения приведено на рис. 2.9. При $N = 1$ биномиальное распределение сводится к распределению Бернулли.

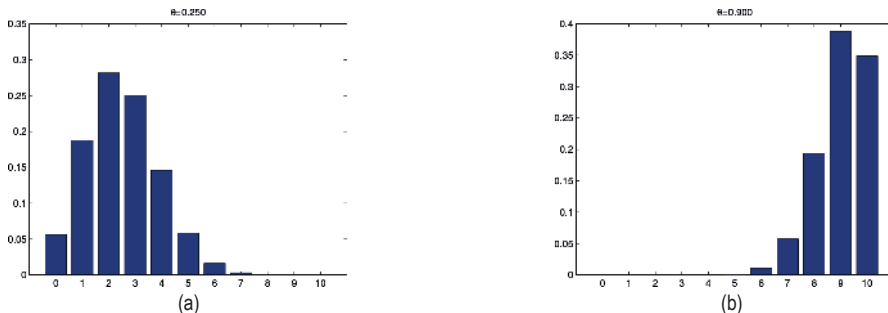


Рис. 2.9 ❖ Биномиальное распределение с $N = 10$ и (a) $\theta = 0.25$ и (b) $\theta = 0.9$. Построено программой по адресу figures.problm.ai/book1/2.9

2.4.2. Сигмоидная (логистическая) функция

Желая предсказать бинарную величину $y \in \{0, 1\}$ при известных входах $x \in \mathcal{X}$, мы должны использовать **условное распределение вероятности** вида:

Таблица 2.3. Некоторые полезные свойства сигмоидной (логистической) функции и связанных с ней функций. Заметим, что функция logit является обратной к сигмоидной и определена на отрезке $[0, 1]$

$$p(y|x, \theta) = \text{Ber}(y|f(x, \theta)), \quad (2.71)$$

$$\sigma(x) \triangleq \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}; \quad (2.72)$$

$$\frac{d}{dx} \sigma(x) = \sigma(x)(1 - \sigma(x)); \quad (2.73)$$

$$1 - \sigma(x) = \sigma(-x); \quad (2.74)$$

$$\sigma^{-1}(p) = \log\left(\frac{p}{1-p}\right) \triangleq \text{logit}(p); \quad (2.75)$$

$$\sigma_+(x) \triangleq \log(1 + e^x) \triangleq \text{spftplus}(x); \quad (2.76)$$

$$\frac{d}{dx} \sigma_+(x) = \sigma(x), \quad (2.77)$$

¹ В русскоязычной литературе принято обозначение C_n^k , которое произносится «С из n по k». – Прим. перев.

где $f(x; \theta)$ – некоторая функция, предсказывающая среднее выходного распределения. В частях II–IV мы будем рассматривать много вариантов функции f .

Чтобы избежать требования $0 \leq f(x; \theta) \leq 1$, мы можем считать функцию неограниченной и использовать следующую модель:

$$p(y|x, \theta) = \text{Ber}(y|\sigma(f(x, \theta))). \quad (2.78)$$

Здесь $\sigma()$ – **сигмоидная**, или **логистическая**, функция, определенная следующим образом:

$$\sigma(a) \triangleq \frac{1}{1 + e^{-a}}, \quad (2.79)$$

где $a = f(x; \theta)$. Слово «сигмоидная» означает S-образная: ее график изображен на рис. 2.10а. Мы видим, что эта функция отображает вещественную прямую в отрезок $[0, 1]$, что необходимо для ее интерпретации как вероятности (а значит, и для того, чтобы можно было использовать в качестве параметра распределения Бернулли θ). Сигмоидную функцию можно рассматривать как «сглаженный» вариант **ступенчатой функции Хевисайда**, определенной следующим образом:

$$H(a) \triangleq \mathbb{I}(a > 0), \quad (2.80)$$

как показано на рис. 2.10b.

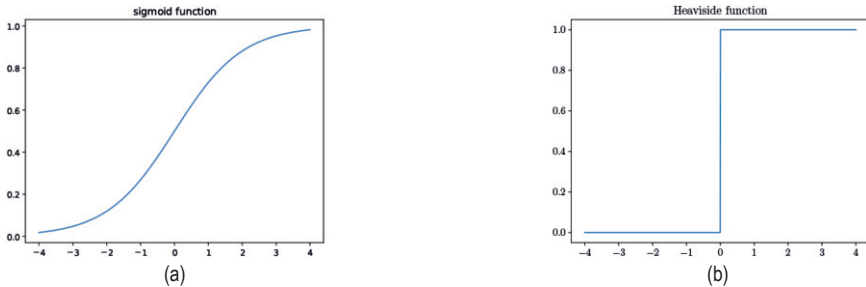


Рис. 2.10 ❖ (а) Сигмоидная (логистическая) функция $\sigma(a) = (1 + e^{-a})^{-1}$.
(б) Функция Хевисайда $\mathbb{I}(a > 0)$.

Построено программой по адресу figures.problml.ai/book1/2.10

Подставляя определение сигмоидной функции в формулу (2.78), получаем:

$$p(y = 1|x, \theta) = \frac{1}{1 + e^{-a}} = \frac{e^a}{1 + e^a} = \sigma(a); \quad (2.81)$$

$$p(y = 0|x, \theta) = 1 - \frac{1}{1 + e^{-a}} = \frac{e^{-a}}{1 + e^{-a}} = \frac{1}{1 + e^a} = \sigma(-a). \quad (2.82)$$

Величина a равна **логарифму отношения шансов**, $\log\left(\frac{p}{1-p}\right)$, где $p = p(y = 1|x; \theta)$. Чтобы убедиться в этом, заметим, что

$$\log\left(\frac{p}{1-p}\right) = \log\left(\frac{e^a}{1+e^a} \frac{1+e^a}{1}\right) = \log(e^a) = a. \quad (2.83)$$

Логистическая, или **сигмоидная**, **функция** переводит логарифм отношения шансов a в p :

$$p = \text{logistic}(a) = \theta(a) \triangleq \frac{1}{1+e^{-a}} = \frac{e^a}{1+e^a}. \quad (2.84)$$

Обратная к ней функция называется **logit** и переводит p в логарифм отношения шансов a :

$$a = \text{logit}(p) = \theta^{-1}(p) \triangleq \log\left(\frac{p}{1-p}\right). \quad (2.85)$$

Полезные свойства этих функций приведены в табл. 2.3.

2.4.3. Бинарная логистическая регрессия

В этом разделе мы используем условную модель Бернулли с линейным предиктором вида $f(\mathbf{x}; \theta) = \mathbf{w}^T \mathbf{x}$, т. е. модель описывается формулой:

$$p(y|x, \theta) = \text{Ber}(y|\sigma(\mathbf{w}^T \mathbf{x} + b)). \quad (2.86)$$

Иными словами,

$$p(y = 1|x; \theta) = \sigma(\mathbf{w}^T \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + b)}}. \quad (2.87)$$

Это называется **логистической регрессией**.

Например, рассмотрим одномерную версию набора данных Iris с двумя классами, в которой положительным является класс «виргинский», а отрицательным – класс «не виргинский». В качестве признака x будем использовать ширину лепестка. Мы аппроксимировали этот набор данных моделью логистической регрессии и показали результаты на рис. 2.11. **Решающая граница** соответствует значению x^* такому, что $p(y = 1|x = x^*; \theta) = 0.5$. Мы видим, что в этом примере $x^* \approx 1.7$. Когда x удаляется от этой границы, классификатор начинает более уверенно предсказывать метку класса.

Из этого примера должно быть ясно, почему линейная регрессия не годится для задач бинарной классификации. В такой модели вероятности становились бы больше 1, когда мы уходим достаточно далеко вправо, и меньше 0, когда мы уходим достаточно далеко влево.

Более подробное обсуждение логистической регрессии см. в главе 10.

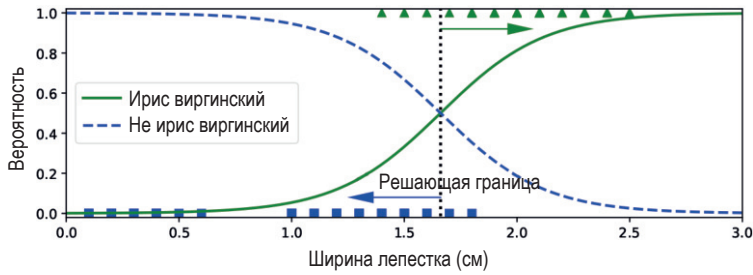


Рис. 2.11 ❖ Логистическая регрессия применительно к одномерной версии набора данных Iris с двумя классами. Построено программой по адресу figures.probml.ai/book1/2.11. За основу взят рис. 4.23 из работы [Gér19]

2.5. КАТЕГОРИАЛЬНОЕ И МУЛЬТИНОМИАЛЬНОЕ РАСПРЕДЕЛЕНИЕ

Чтобы представить распределение конечного множества меток, $y \in \{1, \dots, C\}$, мы можем воспользоваться **категориальным распределением**, обобщающим распределение Бернулли на $C > 2$ значений.

2.5.1. Определение

Категориальным распределением называется дискретное распределение вероятностей, в котором каждому классу соответствует один параметр:

$$\text{Cat}(y|\theta) \triangleq \prod_{c=1}^C \theta_c^{\mathbb{I}(y=c)}. \quad (2.88)$$

Иными словами, $p(y = c|\theta) = \theta_c$. Отметим, что параметры ограничены: $0 \leq \theta_c \leq 1$ и $\sum_{c=1}^C \theta_c = 1$, т. е. всего имеется $C - 1$ независимых параметров.

Мы можем записать категориальное распределение по-другому, преобразовав дискретную величину y в **унитарный вектор** с C элементами, которые все равны 0, кроме элемента, соответствующего метке класса. (Английский термин «one-hot», который переводится на русский как «унитарный», берет начало в электротехнике, где бинарные векторы кодируются электрическими токами на множестве проводов, которые могут быть активными («горячими», англ. *hot*) или неактивными («холодными», англ. *cold*.) Например, если $C = 3$, то классы 1, 2, 3 кодируются как $(1, 0, 0)$, $(0, 1, 0)$ и $(0, 0, 1)$. Вообще, можно закодировать классы с помощью **единичных векторов** e_c , все элементы которых равны 0, за исключением того, что соответствует измерению c . (Такая кодировка называется еще **индикаторной**.) Применив унитарную кодировку, мы можем записать категориальное распределение следующим образом:

$$\text{Cat}(y|\theta) \triangleq \prod_{c=1}^C \theta_c^{y_c}. \quad (2.89)$$

Категориальное распределение является частным случаем **мультиномиального распределения**. Предположим, что наблюдаем исходы N категориальных испытаний, $y_n \sim \text{Cat}(\cdot | \theta)$, $n = 1, \dots, N$. Для определенности пусть это будет бросание C -гранной кости N раз. Обозначим \mathbf{s} вектор, содержащий счетчики выпадения каждой грани, т. е. $s_c \triangleq \sum_{n=1}^N \mathbb{I}(y_n = c)$. (Эквивалентно, если воспользоваться унитарным кодированием, то $\mathbf{s} = \sum_n \mathbf{y}_n$.) Распределение \mathbf{s} называется **мультиномиальным** и описывается следующей формулой:

$$\text{Mu}(\mathbf{s} | N, \theta) \triangleq \binom{N}{s_1 \dots s_C} \prod_{c=1}^C \theta_c^{s_c}, \quad (2.90)$$

где θ_c – вероятность выпадения грани c , а

$$\binom{N}{s_1 \dots s_C} \triangleq \frac{N!}{s_1! s_2! \dots s_C!} \quad (2.91)$$

– **мультиномиальный коэффициент**, равный числу способов разбить множество размера $N = \sum_{c=1}^C s_c$ на подмножества размеров от s_1 до s_C . При $N = 1$ мультиномиальное распределение совпадает с категориальным.

2.5.2. Функция softmax

В условном случае можно определить

$$p(y | \mathbf{x}, \theta) = \text{Cat}(y | f(\mathbf{x}, \theta)), \quad (2.92)$$

что можно записать также в виде:

$$p(y | \mathbf{x}, \theta) = \text{Mu}(y | 1, f(\mathbf{x}, \theta)). \quad (2.93)$$

Мы требуем, чтобы $0 \leq f_c(\mathbf{x}; \theta) \leq 1$ и $\sum_{c=1}^C f_c(\mathbf{x}; \theta) = 1$.

Обычно мы не хотим требовать, чтобы f прямо представляла вектор вероятности, поэтому передаем результаты f функции **softmax** [Bri90], называемой также **мультиномиальной logit**. Определяется она следующим образом:

$$\mathcal{S}(\mathbf{a}) \triangleq \left[\frac{e^{a_1}}{\sum_{c'=1}^C e^{a_{c'}}}, \dots, \frac{e^{a_C}}{\sum_{c'=1}^C e^{a_{c'}}} \right]. \quad (2.94)$$

Эта функция отображает \mathbb{R}^C в $[0, 1]^C$ и удовлетворяет ограничениям $0 \leq \mathcal{S}(\mathbf{a})_c \leq 1$ и $\sum_{c=1}^C \mathcal{S}(\mathbf{a})_c = 1$. Входы функции softmax, $\mathbf{a} = f(\mathbf{x}; \theta)$, называются **логитами** и являются обобщениями логарифмического отношения шансов.

Название softmax связано с тем, что эта функция немного напоминает функцию argmax. Чтобы убедиться в этом, разделим каждый элемент a_c на постоянную T , называемую **температурой**¹. Тогда при $T \rightarrow 0$ имеем

¹ Эта терминология пришла из статистической физики. **Распределение Больцмана**,

$$\mathcal{S}(a/T)_c = \begin{cases} 1.0, & \text{если } c = \operatorname{argmax}_{c'} a_{c'} \\ 0.0, & \text{в противном случае} \end{cases}. \quad (2.95)$$

Иначе говоря, при низких температурах масса вероятности сосредоточена главным образом в самом вероятном состоянии (т. е. **победитель забирает все**), тогда как при высоких температурах масса распределена равномерно (см. рис. 2.12).

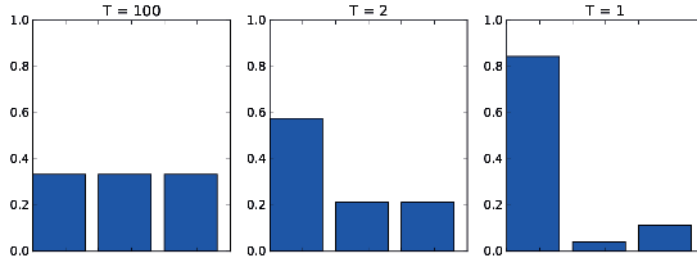


Рис. 2.12 ❖ Распределение softmax $\mathcal{S}(a/T)$, где $a = (3, 0, 1)$ при температурах $T = 100$, $T = 2$ и $T = 1$. Когда температура высокая (слева), распределение равномерное, а когда низкая (справа), в нем имеются пики, и большая часть массы сосредоточена в наибольшем элементе. Построено программой по адресу figures.problml.ai/book1/2.12

2.5.3. Многоклассовая логистическая регрессия

Если использовать линейный предиктор вида $f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}\mathbf{x} + \mathbf{b}$, где \mathbf{W} – матрица размера $C \times D$, в \mathbf{b} – C -мерный вектор смещения, то окончательная модель принимает вид:

$$p(y|\mathbf{x}; \boldsymbol{\theta}) = \operatorname{Cat}(y|\mathcal{S}(\mathbf{W}\mathbf{x} + \mathbf{b})). \quad (2.96)$$

Пусть $\mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{b}$ – C -мерный вектор логитов. Тогда формулу выше можно переписать следующим образом:

$$p(y = c|\mathbf{x}; \boldsymbol{\theta}) = \frac{e^{a_c}}{\sum_{c'=1}^C e^{a_{c'}}}. \quad (2.97)$$

Эта формула называется **мультиномиальной логистической регрессией**.

Если классов всего два, то она сводится к бинарной логистической регрессии. Чтобы убедиться в этом, заметим, что

$$\mathcal{S}(a)_0 = \frac{e^{a_0}}{e^{a_0} + e^{a_1}} = \frac{1}{1 + e^{a_1 - a_0}} = \sigma(a_0 - a_1), \quad (2.98)$$

определенное для состояний, имеет такую же форму, как функция softmax.

так что мы можем просто обучить модель предсказывать, что $a = a_1 - a_0$. Это можно сделать с помощью всего одного вектора весов \mathbf{w} ; а в многоклассовой постановке будет два вектора весов, \mathbf{w}_0 и \mathbf{w}_1 . Такая модель **перепараметризована**, что может негативно сказаться на интерпретируемости, но предсказания будут одинаковы.

Мы обсудим этот вопрос подробнее в разделе 10.3, а пока просто приведем пример. На рис. 2.13 показано, что происходит при аппроксимации этой моделью набора данных об ирисах с тремя классами при использовании всего двух признаков. Мы видим, что решающие границы между классами линейны. Можно создать и нелинейные границы, преобразовав признаки (например, с помощью полиномов), мы поговорим об этом в разделе 10.3.1.

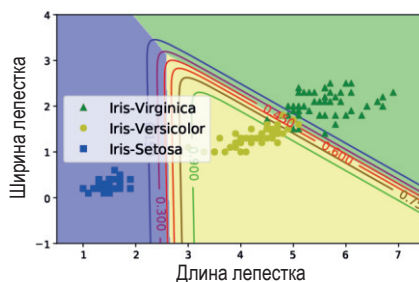


Рис. 2.13. Логистическая регрессия для версии набора данных Iris с двумя признаками и тремя классами. На основе рис. 4.25 из работы [Gér19]. Построено программой по адресу figures.problmlai/book1/2.13

2.5.4. Логарифмирование, суммирование, потенцирование

В этом разделе мы обсудим одну важную практическую деталь, на которую нужно обращать внимание при работе с распределением softmax. Пусть требуется вычислить нормированную вероятность $p_c = p(y = c|x)$ по формуле

$$p_c = \frac{e^{a_c}}{Z(\mathbf{a})} = \frac{e^{a_c}}{\sum_{c'=1}^C e^{a_{c'}}}, \quad (2.99)$$

где $\mathbf{a} = f(\mathbf{x}; \boldsymbol{\theta})$ – логиты. При вычислении **функции разбиения** (partition function)¹ Z мы можем столкнуться с численными проблемами. Например, пусть имеется 3 класса с логитами $\mathbf{a} = (0, 1, 0)$. Тогда мы находим $Z = e^0 + e^1 + e^0 = 4.71$. А теперь предположим, что $\mathbf{a} = (1000, 1001, 1000)$; тогда $Z = \infty$, потому что даже на компьютере с точностью 64 бита $\text{pr.exp}(1000)=\text{inf}$. Точно так же предположим, что $\mathbf{a} = (-1000, -999, -1000)$; тогда $Z = 0$, потому что

¹ В статистической механике аналогичная величина называется **статистической суммой**. – Прим. перев.

пр. $\exp(-1000)=0$. Чтобы избежать численных трудностей, мы воспользуемся следующим тождеством:

$$\log \sum_{c=1}^C \exp(a_c) = m + \log \sum_{c=1}^C \exp(a_c - m), \quad (2.100)$$

справедливым для любого m . Обычно полагают $m = \max_c a_c$, чтобы наибольший показатель степени был равен нулю. Так мы точно избежим переполнения и даже в случае потери значимости ответ будет разумным. Этот прием называется трюком **log-sum-exp** (логарифмирование, суммирование, потенцирование). Мы воспользуемся им при реализации функции **lse**:

$$\text{lse}(\mathbf{a}) \triangleq \log \sum_{c=1}^C \exp(a_c). \quad (2.101)$$

Описанный прием можно применить для вычисления вероятностей по логитам:

$$p(y = c|\mathbf{x}) = \exp(a_c - \text{lse}(\mathbf{a})). \quad (2.102)$$

Затем результат можно передать функции потерь перекрестной энтропии, определенной формулой (5.41).

Но чтобы уменьшить объем вычислений и гарантировать численную устойчивость, потерю перекрестной энтропии часто модифицируют, так что она принимает на входе логиты, а не вектор вероятностей \mathbf{p} . Рассмотрим, к примеру, бинарный случай. Потеря перекрестной энтропии для одного примера равна

$$\mathcal{L} = -[\mathbb{I}(y = 0)\log p_0 + \mathbb{I}(y = 1)\log p_1], \quad (2.103)$$

где

$$\log p_1 = \log \left(\frac{1}{1 + \exp(-a)} \right) - \log(1) - \log(1 + \exp(-a)) = 0 - \text{lse}([0, -a]); \quad (2.104)$$

$$\log p_0 = 0 - \text{lse}([0, +a]). \quad (2.105)$$

2.6. ОДНОМЕРНОЕ ГАУССОВО (НОРМАЛЬНОЕ) РАСПРЕДЕЛЕНИЕ

Из всех распределений вещественной случайной величины $y \in \mathbb{R}$ самым известным и широко используемым является **нормальное распределение**, которое часто называют также **гауссовым** (о происхождении этих названий см. раздел 2.6.4).

2.6.1. Функция распределения

Определим **функцию распределения** (англ. cdf) непрерывной случайной величины Y следующим образом:

$$P(y) \triangleq \Pr(Y \leq y). \quad (2.106)$$

(Обратите внимание на прописную букву P , так обозначается функция распределения). С ее помощью мы можем вычислить вероятность попадания в любой интервал:

$$\Pr(a < Y \leq b) = P(b) - P(a). \quad (2.107)$$

Любая функция распределения является монотонно неубывающей. Функция гауссова распределения имеет вид

$$\Phi(y; \mu, \sigma^2) \triangleq \int_{-\infty}^y \mathcal{N}(z | \mu, \sigma^2) dz = \frac{1}{2} [1 + \operatorname{erf}(z/\sqrt{2})], \quad (2.108)$$

где $z = (y - \mu)/\sigma$, а **функция ошибок** $\operatorname{erf}(u)$ определена следующим образом:

$$\operatorname{erf}(u) \triangleq \frac{2}{\sqrt{\pi}} \int_0^u e^{-t^2} dt. \quad (2.109)$$

Эта функция включена в большинство программных пакетов. Ее график приведен на рис. 2.2а.

Параметр μ равен среднему значению распределения и совпадает с модой, потому что распределение унимодальное. Параметр σ^2 равен дисперсии. (Иногда встречается термин **точность** гауссова распределения, это величина, обратная дисперсии: $\lambda = 1/\sigma^2$.) Если $\mu = 0$ и $\sigma = 1$, то говорят о **стандартном нормальном** распределении.

Если P – функция распределения Y , то $P^{-1}(q)$ – это такое значение y_q , что $p(Y \leq y_q) = q$; оно называется q -м **квантилем** P . Значение $P^{-1}(0.5)$ называется **медианой** распределения, половина массы вероятности находится слева от нее, а половина – справа. Значения $P^{-1}(0.25)$ и $P^{-1}(0.75)$ называются нижним и верхним **квартлями**.

Например, пусть Φ – функция гауссова распределения $\mathcal{N}(0, 1)$ и Φ^{-1} – обратная функция распределения, называемая также **пробит-функцией**. Тогда точки слева от $\Phi^{-1}(\alpha/2)$ содержат массу вероятности $\alpha/2$, как показано на рис. 2.2b. В силу симметрии точки справа от $\Phi^{-1}(1 - \alpha/2)$ также содержат массу вероятности $\alpha/2$. Поэтому центральный интервал $(\Phi^{-1}(\alpha/2), \Phi^{-1}(1 - \alpha/2))$ содержит массу $1 - \alpha$. Если положить $\alpha = 0.05$, то 95 % массы будет сосредоточено в центральном интервале:

$$(\Phi^{-1}(0.025), \Phi^{-1}(0.975)) = (-1.96, 1.96). \quad (2.110)$$

Для распределения $\mathcal{N}(\mu, \sigma^2)$ 95%-ный интервал принимает вид $(\mu - 1.96\sigma, \mu + 1.96\sigma)$. Часто его аппроксимируют и записывают в виде $\mu \pm 2\sigma$.

2.6.2. Функция плотности вероятности

Функция плотности вероятности (ФПВ, англ. pdf) определяется как производная функции распределения:

$$p(y) \triangleq \frac{d}{dy} P(y). \quad (2.111)$$

ФПВ гауссова распределения имеет вид

$$\mathcal{N}(y|\mu, \sigma^2) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-\mu)^2}, \quad (2.112)$$

где $\sqrt{2\pi\sigma^2}$ – нормировочная постоянная, необходимая, для того чтобы полная плотность была равна 1 (см. упражнение 2.12). График этой функции показан на рис. 2.2b.

Зная ФПВ, мы можем вычислить вероятность нахождения непрерывной случайной величины в конечном интервале:

$$\Pr(a < Y \leq b) = \int_a^b p(y) dy = P(b) - P(a). \quad (2.113)$$

Устремляя длину интервала к нулю, можно написать:

$$\Pr(y \leq Y \leq y + dy) \approx p(y) dy. \quad (2.114)$$

Интуитивно это означает, что вероятность попадания Y в малую окрестность y равна произведению плотности в точке y на длину интервала. Из этого результата вытекает важное следствие: ФПВ в точке может быть больше 1. Например, $\mathcal{N}(0|0, 0.1) = 3.99$.

ФПВ можно использовать для вычисления **среднего**, или **математического, ожидания** распределения:

$$\mathbb{E}(Y) \triangleq \int_{\mathcal{Y}} yp(y) dy. \quad (2.115)$$

Для гауссова распределения имеем хорошо знакомый результат:

$$\mathbb{E}[\mathcal{N}(\cdot|\mu, \sigma^2)] = \mu.$$

(Отметим, однако, что для некоторых распределений этот интеграл расходится, а значит, среднее не определено.)

ФПВ может также использовать для вычисления **дисперсии** распределения. Это мера разброса, которая часто обозначается σ^2 и определяется следующим образом:

$$\mathbb{V}(Y) \triangleq \mathbb{E}[(Y - \mu)^2] = \int (y - \mu)^2 p(y) dy \quad (2.116)$$

$$= \int y^2 p(y) dy + \mu^2 \int p(y) dy - 2\mu \int yp(y) dy = \mathbb{E}[Y^2] - \mu^2, \quad (2.117)$$

откуда вытекает полезный результат:

$$\mathbb{E}[Y^2] = \sigma^2 + \mu^2. \quad (2.118)$$

Стандартное отклонение по определению равно:

$$\text{std}[Y] \triangleq \sqrt{\mathbb{V}[Y]} = \sigma. \quad (2.119)$$

(Стандартное отклонение проще интерпретировать, чем дисперсию, потому что оно измеряется в тех же единицах, что сама Y .) Для гауссова распределения имеет место знакомый результат: $\text{std}[\mathcal{N}(\cdot|\mu, \sigma^2)] = \sigma$.

2.6.3. Регрессия

До сих пор мы рассматривали безусловное гауссово распределение. Но в некоторых случаях полезно сделать параметры гауссианы функциями входных величин, т. е. создать модель условной плотности вида

$$p(y|\mathbf{x}; \boldsymbol{\theta}) = \mathcal{N}(y|f_{\mu}(\mathbf{x}; \boldsymbol{\theta}); f_{\sigma}(\mathbf{x}; \boldsymbol{\theta})^2), \quad (2.120)$$

где $f_{\mu}(\mathbf{x}; \boldsymbol{\theta}) \in \mathbb{R}$ предсказывает среднее, а $f_{\sigma}(\mathbf{x}; \boldsymbol{\theta}) \in \mathbb{R}^+$ предсказывает дисперсию.

Обычно предполагают, что дисперсия фиксирована и не зависит от входных данных. Такая модель называется **гомоскедастической регрессией**. Также часто предполагают, что среднее линейно зависит от входных данных – такая модель называется **линейной регрессией**.

$$p(y|\mathbf{x}; \boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{w}^T \mathbf{x} + b; \sigma^2), \quad (2.121)$$

где $\boldsymbol{\theta} = (\mathbf{w}, b, \sigma^2)$. На рис. 2.14а эта модель показана в одномерном случае, а в разделе 11.2 описана более подробно.

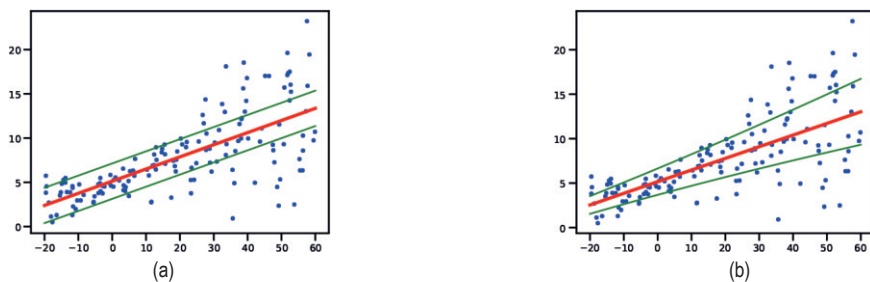


Рис. 2.14 ❖ Линейная регрессия с использованием гауссова распределения со средним $\mu(x) = b + wx$ и (а) фиксированной дисперсией σ^2 (гомоскедастическое) и (б) дисперсией, зависящей от входных данных $\sigma(x)^2$ (гетероскедастическое).

Построено программой по адресу figures.problml.ai/book1/2.14

Но дисперсия может и зависеть от входных данных, такая модель называется **гетероскедастической регрессией**. В линейной постановке имеем

$$p(y|\mathbf{x}; \boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{w}_\mu^\top \mathbf{x} + b; \sigma_+(\mathbf{w}_\sigma^\top \mathbf{x})), \quad (2.122)$$

где $\boldsymbol{\theta} = (\mathbf{w}_\mu, \mathbf{w}_\sigma)$ – два вида весов регрессии, а

$$\sigma_+(a) = \log(1 + e^a) \quad (2.123)$$

– функция **softplus**, которая отображает \mathbb{R} в \mathbb{R}^+ , чтобы стандартное отклонение было гарантированно неотрицательным. В одномерном случае эта модель показана на рис. 2.14b.

Отметим, что на рис. 2.14 показан 95%-ный прогностический интервал, $[\mu(x) - 2\sigma(x); \mu(x) + 2\sigma(x)]$. Это неопределенность предсказанного *наблюдения* у при данном x , она улавливает изменчивость синих точек. С другой стороны, неопределенность истинной (незашумленной) функции представлена величиной $\sqrt{\mathbb{V}[f_\mu(\mathbf{x}; \boldsymbol{\theta})]}$, не содержащей члена σ ; теперь неопределенность сосредоточена в параметрах $\boldsymbol{\theta}$, а не в выходе y . О моделировании неопределенности параметров см. раздел 11.7.

2.6.4. Почему гауссово распределение так широко используется?

Гауссово распределение используется чаще любого другого в статистике и машинном обучении. Тому есть несколько причин. Во-первых, у него два параметра, которые легко интерпретировать и которые отражают важнейшие свойства распределения: среднее и дисперсию. Во-вторых, согласно центральной предельной теореме (раздел 2.8.6), сумма независимых случайных величин имеет приближенно гауссово распределение, что делает его удобным средством для моделирования невязок или «шума». В-третьих, гауссово распределение делает наименьшее число допущений (имеет максимальную энтропию) среди всех распределений с заданным средним и дисперсией, как будет показано в разделе 3.4.4, поэтому во многих случаях оно является хорошим выбором по умолчанию. Наконец, у него простая математическая форма, что ведет к простым в реализации, но при этом весьма эффективным методам, в чем мы убедимся в разделе 3.2.

Стоит заметить, что исторически термин «гауссово распределение» неточен, поскольку, как пишет Джейнс [Jay03, стр. 241], «фундаментальная природа этого распределения и его основные свойства были отмечены Лапласом, когда Гауссу было всего шесть лет, а само распределение было известно Муавру еще до рождения Лапласа». Однако именно Гаусс популяризировал использование этого распределения в 1800-е годы, поэтому термин «гауссово» широко применяется в науке и технике.

Название «нормальное распределение», похоже, возникло в связи с нормальными уравнениями линейной регрессии (см. раздел 11.2.2.2). Однако мы предпочитаем не использовать слово «нормальное», потому что оно предполагает, что другие распределения «аномальны», тогда как, по замечанию Джейнса [Jay03], именно гауссово распределение аномально в том смысле, что обладает многими специальными свойствами, нетипичными для распределений общего вида.

2.6.5. Дельта-функция Дирака как предельный случай

Когда дисперсия гауссова распределения стремится к 0, распределение становится бесконечно узким и бесконечно высоким с «пиком» в среднем значении. Это можно записать следующим образом:

$$\lim_{\sigma \rightarrow 0} \mathcal{N}(y|\mu, \sigma^2) \rightarrow \delta(y - \mu), \quad (2.124)$$

где δ – дельта-функция Дирака:

$$\delta(x) = \begin{cases} +\infty, & \text{если } x = 0 \\ 0, & \text{если } x \neq 0 \end{cases}, \quad (2.125)$$

где

$$\int_{-\infty}^{\infty} \delta(x) dx = 1. \quad (2.126)$$

Это определение можно немного модифицировать:

$$\delta_y(x) = \begin{cases} +\infty, & \text{если } x = y \\ 0, & \text{если } x \neq y \end{cases}. \quad (2.127)$$

Отметим, что

$$\delta_y(x) = \delta(x - y). \quad (2.128)$$

Дельта-функция обладает следующим **фильтрующим свойством**, которое пригодится нам ниже:

$$\int_{-\infty}^{\infty} f(y) \delta(x - y) dy = f(x). \quad (2.129)$$

2.7. ДРУГИЕ ЧАСТО ВСТРЕЧАЮЩИЕСЯ ОДНОМЕРНЫЕ РАСПРЕДЕЛЕНИЯ*

В этом разделе мы кратко упомянем некоторые другие одномерные распределения, встречающиеся в этой книге.

2.7.1. Распределение Стьюдента

Гауссово распределение очень чувствительно к **выбросам**. Робастной альтернативой гауссову распределению является **t-распределение Стьюдента**,

которое мы для краткости будем называть просто **распределением Стьюдента**¹. Его ФПВ имеет вид

$$\mathcal{T}(y|\mu, \sigma^2, \nu) \propto \left[1 + \frac{1}{\nu} \left(\frac{y - \mu}{\sigma} \right)^2 \right]^{-\left(\frac{\nu+1}{2}\right)}, \quad (2.130)$$

где μ – среднее, $\sigma > 0$ – параметр масштаба (не стандартное отклонение), а $\nu > 0$ называется **числом степеней свободы** (хотя правильнее было бы говорить о **степени нормальности** [Kru13], поскольку при больших значениях ν распределение становится похожим на гауссово).

Мы видим, что плотность вероятности полиномиально, а не экспоненциально зависит от квадрата расстояния от центра, поэтому в хвостах сосредоточено больше массы вероятности, чем в случае гауссова распределения (см. рис. 2.15). Говорят, что у распределения Стьюдента **тяжелые хвосты**, из-за чего оно более робастно к выбросам.

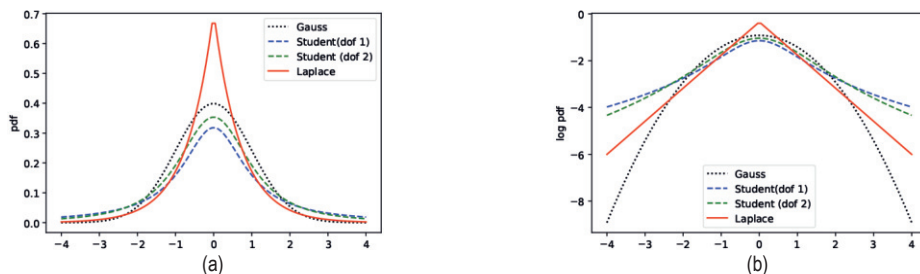


Рис. 2.15 ❖ (а) ФПВ для $\mathcal{N}(0, 1)$, $\mathcal{T}(\mu = 0, \sigma = 1, \nu = 1)$, $\mathcal{T}(\mu = 0, \sigma = 1, \nu = 2)$ и $\text{Lap}(0, 1/\sqrt{2})$. Для распределений Гаусса и Лапласа среднее равно 0, а дисперсия равна 1. При $\nu = 1$ распределение Стьюдента совпадает с распределением Коши, для которого среднее и дисперсия не определены. (б) Логарифмы этих ФПВ. Отметим, что распределение Стьюдента не является логарифмически выпуклой функцией при любом значении параметра, в отличие от распределения Лапласа. Тем не менее оба они унимодальны. Построено программой по адресу figures.problml.ai/book1/2.15

Для иллюстрации робастности распределения Стьюдента рассмотрим рис. 2.16. Слева показано, как распределения Гаусса и Стьюдента аппроксимируют данные без выбросов. Справа же мы добавили несколько выбросов. Мы видим, что на гауссово распределение это оказало сильное влияние, а распределение Стьюдента почти не изменилось. Как воспользоваться рас-

¹ У этого названия колоритная этимология. Впервые его опубликовал в 1908 году Уильям Сили Госсет, который работал на пивоварне Гиннеса в Дублине, Ирландия. Поскольку работодатель не разрешил ему публиковаться под собственным именем, Госсет назвал его распределением «Стьюдента». Буква t , вероятно, появилась в контексте таблиц распределения Стьюдента, которые Фишер использовал при разработке основ классического статистического вывода. Дополнительные исторические детали см. в статье <http://jeff560.tripod.com/s.html>.

пределением Стьюдента для робастной линейной регрессии, мы обсудим в разделе 11.6.2.

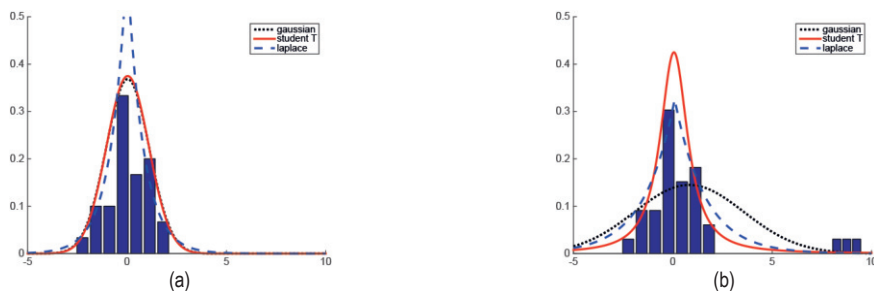


Рис. 2.16 ❖ Влияние выбросов на аппроксимацию распределениями Гаусса, Стьюдента и Лапласа. (а) Без выбросов (кривые распределений Гаусса и Стьюдента совпадают). (б) С выбросами. Как видим, распределение Гаусса в большей степени подвержено влиянию выбросов, чем распределения Стьюдента и Лапласа. На основе рис. 2.16 из работы [Bis06]. Построено программой по адресу figures.probml.ai/book1/2.16

На будущее отметим, что распределение Стьюдента обладает следующими свойствами:

$$\text{среднее} = \mu, \text{ мода} = \mu, \text{ дисперсия} = \frac{v\sigma^2}{(v-2)}. \quad (2.131)$$

Среднее определено, только если $v > 1$. Дисперсия определена, если $v > 2$. При $v \gg 5$ распределение Стьюдента быстро приближается к гауссову и теряет свойства робастности. Чаще всего берут $v = 4$, поскольку это значение дает хорошее качество для широкого круга задач [LLT89].

2.7.2. Распределение Коши

При $v = 1$ распределение Стьюдента называется распределением **Коши** или **Лоренца**. Его ФПВ имеет вид:

$$\mathcal{C}(x|\mu, \gamma) = \frac{1}{\gamma\pi} \left[1 + \left(\frac{x - \mu}{\gamma} \right)^2 \right]^{-1}. \quad (2.132)$$

Это распределение обладает очень тяжелыми хвостами по сравнению с гауссовым. Например, для стандартного нормального распределения 95 % массы вероятности сосредоточено в интервале между -1.96 и 1.96 , а для стандартного распределения Коши – в интервале между -12.7 и 12.7 . Причина такой тяжести хвостов заключается в том, что интеграл, определяющий среднее, расходится. **Половинное распределение Коши** – это вариант распределения Коши ($\mu = 0$), «сложенный вдвое», так что вся плотность вероят-

ности приходится на положительные вещественные числа. Поэтому функция плотности вероятности имеет вид:

$$C_+(x|\gamma) \triangleq \frac{2}{\pi\gamma} \left[1 + \left(\frac{x}{\gamma} \right)^2 \right]^{-1}. \quad (2.133)$$

Это полезно в байесовском моделировании, где мы хотим иметь распределение положительных вещественных чисел с тяжелыми хвостами, но с конечной плотностью в начале координат.

2.7.3. Распределение Лапласа

Еще одним распределением с тяжелыми хвостами является **распределение Лапласа**¹, известное также под названием **двойного экспоненциального** распределения. Его ФПВ имеет вид:

$$\text{Lap}(y|\mu, b) \triangleq \frac{1}{2b} \exp\left(-\frac{|y - \mu|}{b}\right). \quad (2.134)$$

График этой функции показан на рис. 2.15. Здесь μ – параметр сдвига, а $b > 0$ – параметр масштаба. Это распределение обладает следующими свойствами:

$$\text{среднее} = \mu, \text{ мода} = \mu, \text{ дисперсия} = 2b^2. \quad (2.135)$$

В разделе 11.6.1 мы обсудим, как использовать распределение Лапласа для робастной линейной регрессии, а в разделе 11.4 – для разреженной линейной регрессии.

2.7.4. Бета-распределение

Бета-распределение определено на интервале $[0, 1]$ следующим образом:

$$\text{Beta}(x|a, b) = \frac{1}{B(a, b)} x^{a-1} (1-x)^{b-1}. \quad (2.136)$$

Здесь $B(a, b)$ – **бета-функция**:

$$B(a, b) \triangleq \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}, \quad (2.137)$$

где $\Gamma(a)$ – гамма-функция:

¹ Пьер-Симон Лаплас (1749–1827) – французский математик, сыгравший важную роль в создании байесовской статистики как научной дисциплины.

$$\Gamma(a) \triangleq \int_0^{\infty} x^{a-1} e^{-x} dx. \quad (2.138)$$

Графики нескольких бета-распределений показаны на рис. 2.17а.

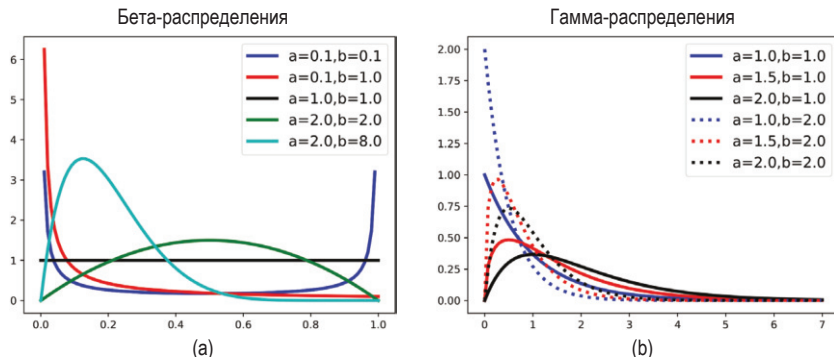


Рис. 2.17 ❖ (а) Бета-распределения. Если $a < 1$, то имеет место пик слева, а если $b < 1$, то пик справа; при $a = b = 1$ распределение равномерное. Если $a > 1$ и $b > 1$, то распределение унимодальное. Построено программой по адресу figures.probl.ai/book1/2.17. (б) Гамма-распределения. Если $a \leq 1$, то мода находится в точке 0, в противном случае вдали от 0. Увеличение коэффициента b приводит к уменьшению горизонтального масштаба, т. е. график сжимается влево и вверх. Построено программой по адресу figures.probl.ai/book1/2.17

Мы требуем выполнения условий $a, b > 0$, чтобы распределение было интегрируемым (это необходимое условие существования $B(a, b)$). Если $a = b = 1$, то получается равномерное распределение. Если a и b меньше 1, то получается бимодальное распределение с пиками в точках 0 и 1; если a и b больше 1, то распределение унимодальное.

Для справки отметим, что распределение обладает следующими свойствами (упражнение 2.8):

$$\text{среднее} = \frac{a}{a+b}, \quad \text{мода} = \frac{a-1}{a+b-2}, \quad \text{дисперсия} = \frac{ab}{(a+b)^2(a+b+1)}. \quad (2.139)$$

2.7.5. Гамма-распределение

Гамма-распределение определено для положительных вещественных случайных величин, $x > 0$. У него имеется два параметра: форма (shape) $a > 0$ и коэффициент (rate) $b > 0$:

$$\text{Ga}(x|\text{shape} = a, \text{rate} = b) = \frac{b^a}{\Gamma(a)} x^{a-1} e^{-xb}. \quad (2.140)$$

Иногда это распределение параметризуется формой a и масштабом (scale) $s = 1/b$:

$$\text{Ga}(x|\text{shape} = a, \text{scale} = s) \triangleq \frac{1}{s^a \Gamma(a)} x^{a-1} e^{-x/s}. \quad (2.141)$$

На рис. 2.17b приведены графики нескольких ФПВ гамма-распределения. Для справки отметим, что распределение обладает следующими свойствами:

$$\text{среднее} = \frac{a}{b}, \quad \text{мода} = \frac{a-1}{b}, \quad \text{дисперсия} = \frac{a}{b^2}. \quad (2.142)$$

Ниже обсуждается несколько частных случаев гамма-распределения.

○ **Экспоненциальное распределение.** Определено следующим образом:

$$\text{Expon}(x|\lambda) \triangleq \text{Ga}(x|\text{shape} = 1, \text{rate} = \lambda). \quad (2.143)$$

Это распределение описывает промежутки времени между событиями в пуассоновском процессе, т. е. процессе, в котором события возникают непрерывно и независимо с постоянной средней скоростью λ .

○ **Распределение хи-квадрат.** Определено следующим образом:

$$\chi_v^2(x) \triangleq \text{Ga}\left(x|\text{shape} = \frac{v}{2}, \text{rate} = \frac{1}{2}\right), \quad (2.144)$$

где v называется числом степеней свободы. Это распределение суммы квадратов гауссовых случайных величин. Точнее, если $Z_i \sim \mathcal{N}(0, 1)$ и $S = \sum_{i=1}^v Z_i^2$, то $S \sim \chi_v^2$.

○ **Обратное гамма-распределение** определено следующим образом:

$$\text{IG}(x|\text{shape} = a, \text{scale} = b) \triangleq \frac{b^a}{\Gamma(a)} x^{-(a+1)} e^{-b/x}. \quad (2.145)$$

Оно обладает следующими свойствами:

$$\text{среднее} = \frac{b}{a-1}, \quad \text{мода} = \frac{b}{a+1}, \quad \text{дисперсия} = \frac{b^2}{(a+1)^2(a-2)}. \quad (2.146)$$

Среднее существует, только если $a > 1$. Дисперсия существует, только если $a > 2$. Примечание: если $X \sim \text{Ga}(\text{shape} = a, \text{rate} = b)$, то $1/X \sim \text{IG}(\text{shape} = a, \text{scale} = b)$. (Отметим, что в этом случае b играет две разные роли.)

2.7.6. Эмпирическое распределение

Пусть имеется выборка $\mathcal{D} = \{x^{(1)}, \dots, x^{(N)}\}$ объема N из распределения $p(X)$, где $X \in \mathbb{R}$. Мы можем аппроксимировать ФПВ множеством дельта-функций (раздел 2.6.5) с «пиками» в точках из этой выборки:

$$\hat{p}_N(x) = \frac{1}{N} \sum_{n=1}^N \delta_{x^{(i)}}(x). \quad (2.147)$$

Это называется **эмпирическим распределением** набора данных \mathcal{D} . Пример для $N = 5$ показан на рис. 2.18а.

Соответствующая функция распределения имеет вид

$$\hat{P}_N(x) = \frac{1}{N} \sum_{n=1}^N \mathbb{I}(x^{(i)} \leq x) = \frac{1}{N} \sum_{n=1}^N u_{x^{(i)}}(x), \quad (2.148)$$

где $u_y(x)$ – **ступенчатая функция**:

$$u_y(x) = \begin{cases} 1, & \text{если } x \geq y \\ 0, & \text{если } x < y \end{cases}. \quad (2.149)$$

Это можно визуализировать в виде «лестницы», как показано на рис. 2.18б, где в каждом примере имеет место ступенька высоты $1/N$.

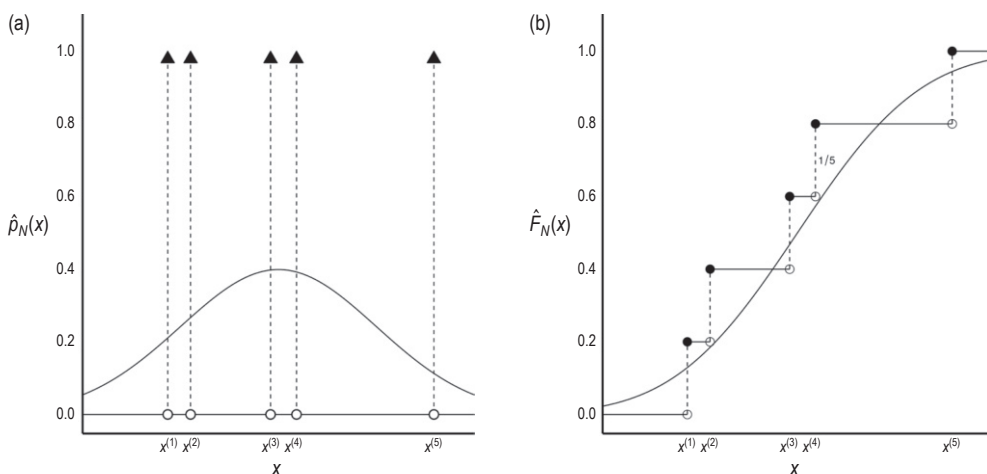


Рис. 2.18 ❖ (а) ФПВ эмпирического распределения и (б) функция эмпирического распределения для выборки объемом $N = 5$. Взято из <https://bit.ly/3hFgi0e>.

Печатается с разрешения Мауро Эскудеро

2.8. ПРЕОБРАЗОВАНИЯ СЛУЧАЙНЫХ ВЕЛИЧИН*

Пусть $x \sim p()$ – случайная величина, а $y = f(x)$ – ее детерминированное преобразование. В этом разделе мы обсудим, как вычислить $p(y)$.

2.8.1. Дискретный случай

Если X – дискретная случайная величина, то мы можем вывести ФПВ Y , просто просуммировав массу вероятности всех x , для которых $f(x) = y$:

$$p_y(y) = \sum_{x:f(x)=y} p_x(x). \quad (2.150)$$

Например, если $f(X) = 1$, когда X четно, и $f(X) = 0$ в противном случае, а $p_x(X)$ равномерное распределение на множестве $\{1, \dots, 10\}$, то $p_y(1) = \sum_{x \in \{2,4,6,8,10\}} p_x(x) = 0.5$, а потому также $p(0) = 0.5$. Заметим, что в этом примере f функция типа «многие к одному».

2.8.2. Непрерывный случай

Если X непрерывна, то мы можем воспользоваться формулой (2.150), потому что $p_x(x)$ – плотность, а не ФПВ, а суммировать плотности нельзя. Вместо этого мы будем работать с функциями распределения следующим образом:

$$P_y(y) \triangleq \Pr(Y \leq y) = \Pr(f(X) \leq y) = \Pr(X \in \{x|f(x) \leq y\}). \quad (2.151)$$

Если f обратима, то вывести ФПВ y можно путем дифференцирования функции распределения, как будет показано ниже. Если же f необратима, то можно воспользоваться численным интегрированием или аппроксимацией методом Монте-Карло.

2.8.3. Обратимые преобразования (биекции)

В этом разделе мы рассмотрим случай монотонной, а потому обратимой функции. (Отметим, что функция обратима тогда и только тогда, когда она является **биекцией**.) В этом предположении существует простая формула для ФПВ y , как мы увидим ниже. (Ее можно обобщить на обратимые, но не монотонные функции, однако этот случай мы рассматривать не будем.)

2.8.3.1. Замена переменных: скалярный случай

Начнем с примера. Пусть $x \sim \text{Unif}(0, 1)$ и $y = f(x) = 2x + 1$. Эта функция растягивает и сдвигает распределение вероятностей, как показано на рис. 2.19а. Рассмотрим ее в точке x и в другой бесконечно близкой точке $x + dx$. Мы видим, что этот интервал отображается в $(y, y + dy)$. Масса вероятности в этих интервалах должна быть одинаковой, поэтому $p(x)dx = p(y)dy$ и, следовательно, $p(y) = p(x)dx/dy$. Однако, поскольку не имеет значения (в терминах сохранения вероятности), будет ли $dx/dy > 0$, или $dx/dy < 0$, имеем:

$$p_y(y) = p_x(x) \left| \frac{dx}{dy} \right|. \quad (2.152)$$

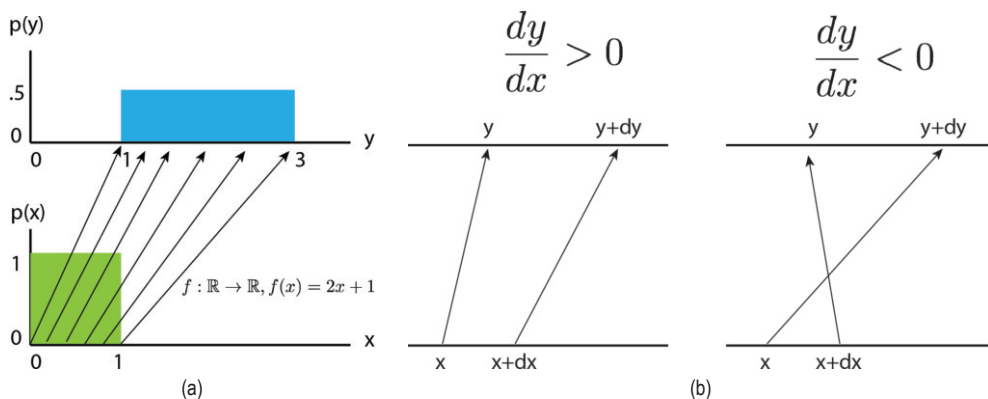


Рис. 2.19 ❖ (а) Отображение равномерной ФПВ посредством функции $f(x) = 2x + 1$. (б) Как две близкие точки, x и $x + dx$, преобразуются отображением f . Если $dy/dx > 0$, то функция локально возрастает, а если $dy/dx < 0$, то локально убывает.

Взято из работы [Jan18]. Печатается с разрешения Эрика Янга

Теперь рассмотрим общий случай для любой $p_x(x)$ и любой монотонной функции $f: \mathbb{R} \rightarrow \mathbb{R}$. Пусть $g = f^{-1}$, т. е. $y = f(x)$ и $x = g(y)$. Предположив, что $f: \mathbb{R} \rightarrow \mathbb{R}$ монотонно возрастает, имеем:

$$P_y(y) = \Pr(f(X) \leq y) = \Pr(X \leq f^{-1}(y)) = P_x(f^{-1}(y)) = P_x(g(y)). \quad (2.153)$$

Дифференцируя, получаем:

$$p_y(y) \triangleq \frac{d}{dy} P_y(y) = \frac{d}{dy} P_x(x) = \frac{dx}{dy} \frac{d}{dx} P_x(x) = \frac{dx}{dy} p_x(x). \quad (2.154)$$

Мы можем вывести аналогичное выражение (но с противоположным знаком) для случая, когда f монотонно убывает. В общем случае возьмем абсолютную величину:

$$p_y(y) = p_x(g(y)) \left| \frac{d}{dy} g(y) \right|. \quad (2.155)$$

Это называется формулой **замены переменных**.

2.8.3.2. Замена переменных: многомерный случай

Полученные выше результаты можно обобщить на многомерные распределения. Пусть f – обратимая функция, отображающая \mathbb{R}^n в \mathbb{R}^n , а g – обратная к ней функция. Пусть требуется вычислить ФПВ распределения $y = f(x)$. По аналогии со скалярным случаем имеем

$$p_y(y) = p_x(g(y)) |\det[J_g(y)]|, \quad (2.156)$$

где $\mathbf{J}_g = d\mathbf{g}(\mathbf{y})/d\mathbf{y}^T$ – якобиан \mathbf{g} , а $|\det \mathbf{J}(\mathbf{y})|$ – абсолютная величина определителя \mathbf{J} , вычисленная в точке \mathbf{y} . (Обсуждение якобианов см. в разделе 7.8.5.) В упражнении 3.6 мы воспользуемся этой формулой для вывода нормировочной постоянной многомерного гауссова распределения.

На рис. 2.20 этот результат показан для двумерного случая, когда $\mathbf{f}(\mathbf{x}) = \mathbf{Ax} + \mathbf{b}$, где $\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. Мы видим, что отношение площади параллелограмма к площади квадрата равно $\det(\mathbf{A}) = ad - bc$.

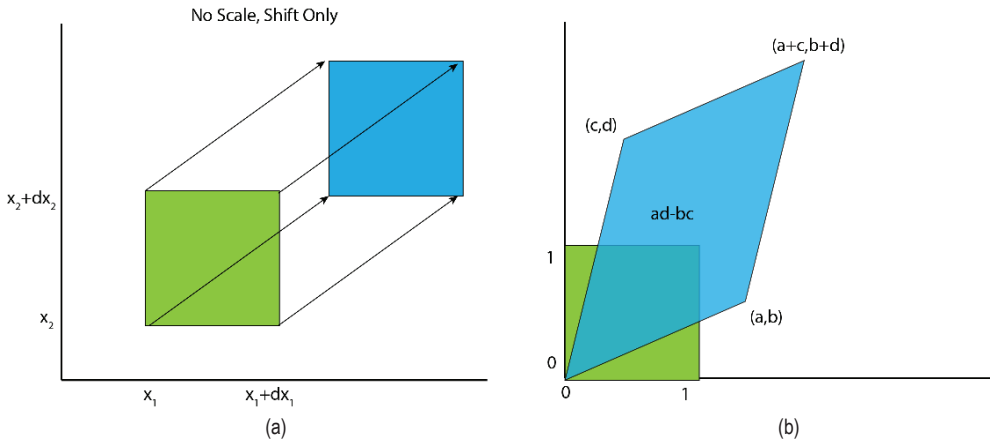


Рис. 2.20 ❖ Применение аффинного преобразования $\mathbf{f}(\mathbf{x}) = \mathbf{Ax} + \mathbf{b}$ к единичному квадрату. (а) Здесь $\mathbf{A} = \mathbf{I}$. (б) Здесь $\mathbf{b} = \mathbf{0}$. Взято из работы [Jan18]. Печатается с разрешения Эрика Янга

Рассмотрим другой пример: преобразование плотности из декартовых координат $\mathbf{x} = (x_1, x_2)$ в полярные $\mathbf{y} = \mathbf{f}(x_1, x_2)$, так что $\mathbf{g}(r, \theta) = (r \cos \theta, r \sin \theta)$. Тогда

$$\mathbf{J}_g = \begin{pmatrix} \frac{\partial x_1}{\partial r} & \frac{\partial x_1}{\partial \theta} \\ \frac{\partial x_2}{\partial r} & \frac{\partial x_2}{\partial \theta} \end{pmatrix} = \begin{pmatrix} \cos \theta & -r \sin \theta \\ \sin \theta & r \cos \theta \end{pmatrix}; \quad (2.157)$$

$$|\det(\mathbf{J}_g)| = |r \cos^2 \theta + r \sin^2 \theta| = |r|. \quad (2.158)$$

Отсюда

$$p_{r,\theta}(r, \theta) = p_{x_1, x_2}(r \cos \theta, r \sin \theta) r. \quad (2.159)$$

Чтобы геометрически интерпретировать эту формулу, заметим, что площадь закрашенного участка на рис. 2.21 равна:

$$\Pr(r \leq R \leq r + dr, \theta \leq \Theta \leq \theta + d\theta) = p_{r,\theta}(r, \theta) dr d\theta. \quad (2.160)$$

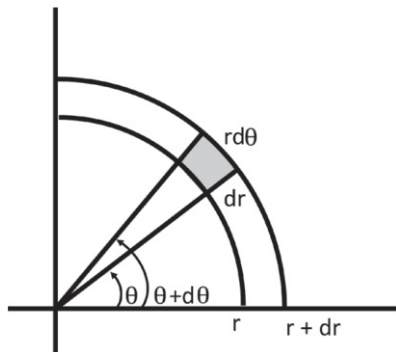


Рис. 2.21 ❖ Замена декартовых координат полярными.
Площадь закрашенного участка равна $r dr d\theta$.
На основе рис. 3.16 из работы [Ric95]

В пределе это выражение равно плотности в центре участка, умноженной на его площадь, равную $r dr d\theta$. Поэтому

$$p_{r,\theta}(r, \theta) dr d\theta = p_{x_1, x_2}(r \cos \theta, r \sin \theta) r dr d\theta. \quad (2.161)$$

2.8.4. Моменты линейного преобразования

Пусть f – аффинная функция, т. е. $y = Ax + b$. В этом случае легко вычислить среднее и ковариацию y . Среднее вычисляется так:

$$\mathbb{E}[y] = \mathbb{E}[Ax + b] = A\mu + b, \quad (2.162)$$

где $\mu = \mathbb{E}[x]$. Если f – скалярная функция, $f(x) = a^T x + b$, то соответствующий результат имеет вид:

$$\mathbb{E}[a^T x + b] = a^T \mu + b. \quad (2.163)$$

Что касается ковариации, то имеем

$$\text{Cov}[y] = \text{Cov}[Ax + b] = A\Sigma A^T, \quad (2.164)$$

где $\Sigma = \text{Cov}[x]$. Оставляем доказательство этого факта в качестве упражнения.

В частном случае, когда $y = a^T x + b$, получаем:

$$\mathbb{V}[y] = \mathbb{V}[a^T x + b] = a^T \Sigma a. \quad (2.165)$$

Например, чтобы вычислить дисперсию суммы двух скалярных случайных величин, мы можем положить $a = [1, 1]$, тогда

$$\mathbb{V}[x_1 + x_2] = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad (2.166)$$

$$= \Sigma_{11} + \Sigma_{22} + 2\Sigma_{12} = \mathbb{V}[x_1] + \mathbb{V}[x_2] + 2\text{Cov}[x_1, x_2]. \quad (2.167)$$

Заметим, однако, что, хотя некоторые распределения (например, гауссово) полностью описываются средним и ковариацией, в общем случае для вывода полного распределения у необходимо применять описанный выше метод.

2.8.5. Теорема о свертке

Пусть $y = x_1 + x_2$, где x_1 и x_2 – независимые случайные величины. Если эти величины дискретны, то функцию вероятности их суммы можно вычислить следующим образом:

$$p(y = j) = \sum_k p(x_1 = k)p(x_2 = j - k), \quad (2.168)$$

где $j = \dots, -2, -1, 0, 1, 2, \dots$.

Если ФПВ x_1 и x_2 равны соответственно $p_1(x_1)$ и $p_2(x_2)$, то каково распределение у? Функция распределения для у описывается формулой

$$p_y(y^*) = \Pr(y \leq y^*) = \int_{-\infty}^{\infty} p_1(x_1) dx_1 \int_{-\infty}^{y^* - x_1} p_2(x_2) dx_2, \quad (2.169)$$

где интегрирование производится по области R , определенной неравенством $x_1 + x_2 < y^*$. Таким образом, ФПВ для у имеет вид:

$$p(y) = \left[\frac{d}{dy^*} P_y(y^*) \right]_{y^*=y} = \int p_1(x_1) p_2(y - x_1) dx_1, \quad (2.170)$$

где мы воспользовались правилом **дифференцирования под знаком интеграла**:

$$\frac{d}{dx} \int_{a(x)}^{b(x)} f(t) dt = f(b(x)) \frac{db(x)}{dx} - f(a(x)) \frac{da(x)}{dx}. \quad (2.171)$$

Формулу (2.170) можно переписать в виде:

$$p = p_1 \circledast p_2, \quad (2.172)$$

где \circledast обозначает оператор **свертки**. Для векторов конечной длины интегралы заменяются суммами, а операция свертки иллюстрируется в табл. 2.4. Формула (2.170) называется **теоремой о свертке**.

Например, предположим, что бросаются две кости, так что дискретные случайные величины p_1 и p_2 имеют равномерное распределение над множеством $\{1, 2, \dots, 6\}$. Обозначим $y = x_1 + x_2$ сумму, выпавшую на костях. Имеем

$$p(y = 2) = p(x_1 = 1)p(x_2 = 1) = \frac{1}{6} \frac{1}{6} = \frac{1}{36}; \quad (2.173)$$

$$p(y = 3) = p(x_1 = 1)p(x_2 = 2) + p(x_1 = 2)p(x_2 = 1) = \frac{1}{6} \frac{1}{6} + \frac{1}{6} \frac{1}{6} = \frac{2}{36}; \quad (2.174)$$

$$\dots \quad (2.175)$$

Таблица 2.4. Дискретная свертка $x = [1, 2, 3, 4]$ с $y = [5, 6, 7]$; в результате получается $z = [5, 16, 34, 52, 45, 28]$. В общем случае $z_n = \sum_{k=-\infty}^{\infty} x_k y_{n-k}$. Мы видим, что эта операция состоит из почленного умножения элемента x на «дополнительный» элемент y и сложения результатов

–	–	1	2	3	4	–	–	
7	6	5	–	–	–	–	–	$z_0 = x_0 y_0 = 5$
–	7	6	5	–	–	–	–	$z_1 = x_0 y_1 + x_1 y_0 = 16$
–	–	7	6	5	–	–	–	$z_2 = x_0 y_2 + x_1 y_1 + x_2 y_0 = 34$
–	–	–	7	6	5	–	–	$z_3 = x_1 y_2 + x_2 y_1 + x_3 y_0 = 52$
–	–	–	–	7	6	5	–	$z_4 = x_2 y_2 + x_3 y_1 = 45$
–	–	–	–	–	7	6	5	$z_5 = x_3 y_2 = 28$

Продолжая таким образом, находим $p(y = 4) = 3/36$, $p(y = 5) = 4/36$, $p(y = 6) = 5/36$, $p(y = 7) = 6/36$, $p(y = 8) = 5/36$, $p(y = 9) = 4/36$, $p(y = 10) = 3/36$, $p(y = 11) = 2/36$ и $p(y = 12) = 1/36$. Графически это показано на рис. 2.22. Мы видим, что распределение похоже на гауссово, причины этого будут объяснены в разделе 2.8.6.

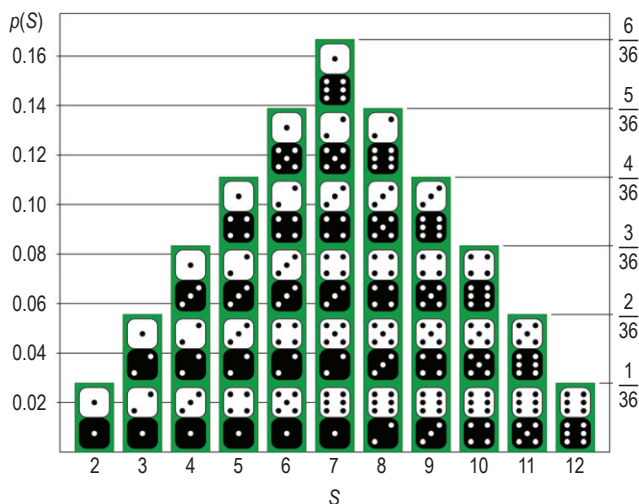


Рис. 2.22 ❖ Распределение суммы очков на двух костях, т. е. функция $p(y)$, где $y = x_1 + x_2$ и $x_i \sim \text{Unif}(\{1, 2, \dots, 6\})$. Взято из статьи https://en.wikipedia.org/wiki/Probability_distribution. Печатается с разрешения автора «Википедии» Тима Стеллмача

Мы также можем вычислить ФПВ суммы двух непрерывных случайных величин. Например, в случае двух гауссовых распределений, когда $x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ и $x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$, можно показать (см. упражнение 2.4), что для $y = x_1 + x_2$

$$p(y) = \mathcal{N}(x_1 | \mu_1, \sigma_1^2) \otimes \mathcal{N}(x_2 | \mu_1, \sigma_2^2) = \mathcal{N}(y | \mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2). \quad (2.176)$$

Поэтому свертка двух гауссовых распределений является гауссовым распределением.

2.8.6. Центральная предельная теорема

Теперь рассмотрим N случайных величин с ФПВ (необязательно гауссовыми) $p_n(x)$ с одинаковыми средними μ и дисперсиями σ^2 . Предполагается, что все величины **независимы и одинаково распределены**, т. е. $X_n \sim p(X)$ – независимые примеры, выбранные из одного и того же распределения. Обозначим $S_N = \sum_{n=1}^N X_n$ случайных величин. Можно показать, что при увеличении N распределение этой суммы стремится к

$$p(S_N = u) = \frac{1}{\sqrt{2\pi N\sigma^2}} \exp\left(-\frac{(u - N\mu)^2}{2N\sigma^2}\right). \quad (2.177)$$

Поэтому распределение величины

$$Z_N \triangleq \frac{S_N - N\mu}{\sigma\sqrt{N}} = \frac{\bar{X} - \mu}{\sigma/\sqrt{N}} \quad (2.178)$$

сходится к стандартному нормальному, где $\bar{X} = S_N/N$ – выборочное среднее. Этот результат называется **центральной предельной теоремой**. Доказательство см., например, в книгах [Jay03, стр. 222] или [Ric95, стр. 169].

На рис. 2.23 приведен пример вычисления выборочного среднего случайных величин, выбранных из бета-распределения. Мы видим, что распределение этого среднего быстро сходится к гауссову.

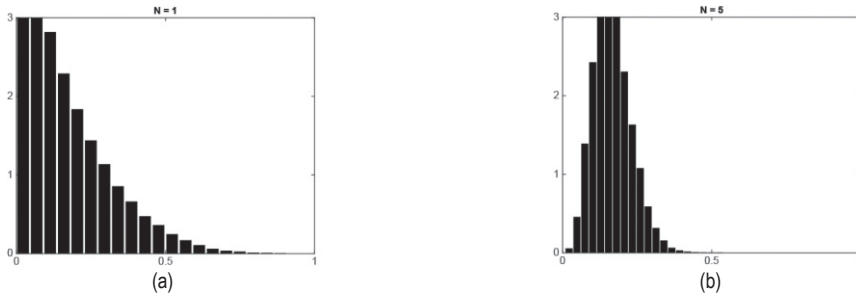


Рис. 2.23 ❖ Центральная предельная теорема в картинках. Мы изобраили гистограмму $\hat{\mu}_N^s = (1/N) \sum_{n=1}^N x_{ns}$, где $x_{ns} \sim \text{Beta}(1, 5)$, $s = 1, \dots, 10\,000$. При $N \rightarrow \infty$ распределение сходится к гауссову. (a) $N = 1$. (b) $N = 5$. На основе рис. 2.6 из работы [Bis06]. Построено программой по адресу figures.problml.ai/book1/2.23

2.8.7. Аппроксимация Монте-Карло

Пусть \mathbf{x} – случайная величина, а $y = f(\mathbf{x})$ – некоторая функция от \mathbf{x} . Часто трудно вычислить индуцированное распределение $p(y)$ аналитически. Простая, но действенная альтернатива – произвести большую выборку из распределения \mathbf{x} и использовать эти примеры (вместо самого распределения) для аппроксимации $p(y)$.

Например, предположим, что $x \sim \text{Unif}(-1, 1)$ и $y = f(x) = x^2$. Чтобы аппроксимировать $p(y)$, мы можем выбрать много примеров из $p(x)$ (с помощью **генератора случайных чисел**), возвести их в квадрат и вычислить эмпирическое распределение:

$$p_S(y) \triangleq \frac{1}{N_S} \sum_{s=1}^{N_S} \delta(y - y_s). \quad (2.179)$$

Это просто взвешенная с одинаковыми весами «сумма пиков» с центрами в каждом из примеров (см. раздел 2.7.6). Имея достаточно примеров, мы сможем аппроксимировать $p(y)$ с приемлемой точностью (см. иллюстрацию на рис. 2.24).

Этот подход называется **аппроксимацией Монте-Карло** распределения. (Термин «Монте-Карло» – отсылка к знаменитому казино в Монако.) Метод Монте-Карло впервые был разработан для целей статистической физики – в частности, при создании атомной бомбы, – но теперь широко используется в статистике и машинном обучении. Дополнительные сведения можно найти во втором томе этой книги, [Mur22], а также в книгах, специально посвященных этой теме, например [Liu01; RC04; KTB11; BZ20].

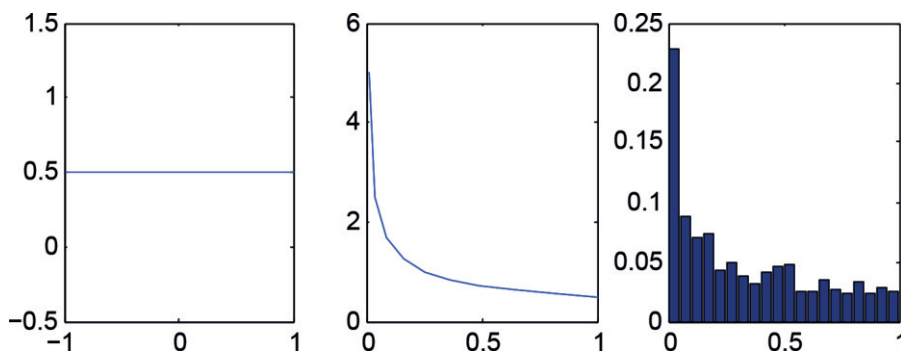


Рис. 2.24 ❖ Вычисление распределения $y = x^2$, где $p(x)$ – равномерное распределение (слева). Аналитический результат показан в центре, а аппроксимация Монте-Карло справа. Построено программой по адресу figures.problai/book1/2.24

2.9. УПРАЖНЕНИЯ

Упражнение 2.1 [условная независимость*].

(Источник: Коллер.)

- Пусть $H \in \{1, \dots, K\}$ – дискретная случайная величина, а e_1 и e_2 – наблюдаемые значения двух других случайных величин E_1 и E_2 . Требуется вычислить вектор:

$$\vec{P}(H|e_1, e_2) = (P(H = 1|e_1, e_2), \dots, P(H = K|e_1, e_2)).$$

Каких из следующих наборов чисел достаточно для вычисления?

- i. $P(e_1, e_2), P(H), P(e_1|H), P(e_2|H)$.
 - ii. $P(e_1, e_2), P(H), P(e_1, e_2|H)$.
 - iii. $P(e_1|H), P(e_2|H), P(H)$.
- b. Теперь предположим, что $E_1 \perp E_2|H$ (т. е. E_1 и E_2 условно независимы при условии H). Тогда каких из этих трех наборов достаточно?
Представьте вычисления и конечный результат. *Указание:* воспользуйтесь формулой Байеса.

Упражнение 2.2 [из попарной независимости не следует взаимная независимость].

Говорят, что две случайные величины попарно независимы, если

$$p(X_2|X_1) = p(X_2) \quad (2.180)$$

и, следовательно,

$$p(X_2, X_1) = p(X_1)p(X_2|X_1) = p(X_1)p(X_2). \quad (2.181)$$

Говорят, что n случайных величин взаимно независимы, если

$$p(X_i|X_S) = p(X_i) \quad \forall S \subseteq \{1, \dots, n\} \setminus \{i\} \quad (2.182)$$

и, следовательно,

$$p(X_{1:n}) = \prod_{i=1}^n p(X_i). \quad (2.183)$$

Покажите, что из попарной независимости всех пар случайных величин необязательно следует взаимная независимость. Достаточно привести контрпример.

Упражнение 2.3 [условная независимость равносильна факторизуемости совместного распределения*].

В основном тексте мы дали такое определение: $X \perp Z$, если

$$p(x, y|z) = p(x|z)p(y|z) \quad (2.184)$$

для всех x, y, z таких, что $p(z) > 0$. Докажите, что $X \perp Y|Z$ тогда и только тогда, когда существуют функции g и h такие, что

$$p(x, y|z) = g(x, z)h(y, z) \quad (2.185)$$

для всех x, y, z таких, что $p(z) > 0$, т. е. оба определения эквивалентны.

Упражнение 2.4 [свертка двух гауссовых распределений является гауссовым распределением].

Покажите, что свертка двух гауссовых распределений является гауссовым распределением, т. е.

$$p(y) = \mathcal{N}(x_1|\mu_1, \sigma_1^2) \otimes \mathcal{N}(x_2|\mu_2, \sigma_2^2) = \mathcal{N}(y|\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2), \quad (2.186)$$

где $y = x_1 + x_2$, $x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ и $x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$.

Упражнение 2.5 [математическое ожидание минимума двух случайных величин*].

Пусть X, Y – две точки, случайно и равномерно выбранные из интервала $[0, 1]$. Каково ожидаемое местоположение левой точки?

Упражнение 2.6 [дисперсия суммы].

$$\mathbb{V}[X + Y] = \mathbb{V}[X] + \mathbb{V}[Y] + 2\text{Cov}[X, Y], \quad (2.187)$$

где $\text{Cov}[X, Y]$ – ковариация X и Y .

Упражнение 2.7 [вывод плотности обратного гамма-распределения*].

Пусть $X \sim \text{Ga}(a, b)$ и $Y = 1/X$. Выведите распределение Y .

Упражнение 2.8 [среднее, мода и дисперсия бета-распределения].

Пусть $\theta \sim \text{Beta}(a, b)$. Покажите, что среднее, мода и дисперсия равны:

$$\mathbb{E}[\theta] = \frac{a}{a + b}; \quad (2.188)$$

$$\mathbb{V}[\theta] = \frac{ab}{(a + b)^2(a + b + 1)}; \quad (2.189)$$

$$\text{mode}[\theta] = \frac{a - 1}{a + b - 2}. \quad (2.190)$$

Упражнение 2.9 [формула Байеса в медицинской диагностике].

По результатам ежегодного обследования у врача есть для вас хорошая и плохая новость. Плохая – что ваш тест на серьезное заболевание оказался положительным, и верность этого теста составляет 99 % (т. е. вероятность положительного теста при условии, что вы больны, равна 0.99, как и вероятность отрицательного теста при условии, что вы не больны). Хорошая новость – что это редкое заболевание, которое обнаруживается у одного человека из 10 000. Каковы шансы, что вы действительно больны? (Представьте вычисления и конечный результат.)

Упражнение 2.10 [судебное разбирательство].

(Источник: Питер Ли.) Совершено преступление. На месте преступления найдены следы крови, которым нет безобидного объяснения. Такая группа крови встречается у 1 % людей.

- Прокурор говорит: «Вероятность, что у обвиняемого была бы найденная на месте преступления группа крови, если бы он был невиновен, равна 1 %. Следовательно, с вероятностью 99 % он виновен». Это рассуждение называется **ошибкой прокурора**. Что в нем не так?
- Защитник говорит: «Преступление совершено в городе с населением 800 000 человек. Такая группа крови встречается примерно у 8000 человек. Следовательно, вероятность виновности обвиняемого равна всего 1 к 8000, это пренебрежимо мало». Это рассуждение называется **ошибкой защитника**. Что в нем не так?

Упражнение 2.11 [вероятность зависит от формы вопроса, на который требуется дать ответ*].

(Источник: Минка.) У моего соседа двое детей. В предположении, что пол ребенка можно уподобить исходу подбрасывания монеты, априорная вероятность, что у соседа мальчик и девочка, равна $1/2$. Другие варианты – два мальчика или две девочки – имеют вероятность $1/4$ и $1/4$.

- а. Предположим, что в ответ на вопрос, есть ли у него мальчики, сосед отвечает да. Какова вероятность, что один ребенок – девочка?
- б. Предположим, что я видел одного из соседских детей, и это был мальчик. Какова вероятность, что другой ребенок – девочка?

Упражнение 2.12 [нормировочная постоянная для одномерного гауссова распределения].

Нормировочная постоянная для гауссова распределения с нулевым средним равна

$$Z = \int_a^b \exp\left(-\frac{x^2}{2\sigma^2}\right) dx, \quad (2.191)$$

где $a = -\infty$ и $b = \infty$. Для вычисления рассмотрим ее квадрат:

$$Z^2 = \int_a^b \int_a^b \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) dx dy. \quad (2.192)$$

Перейдем от декартовых координат (x, y) к полярным (r, θ) , совершив замену переменных $x = r \cos \theta$, $y = r \sin \theta$. Поскольку $dx dy = r dr d\theta$ и $\cos^2 \theta + \sin^2 \theta = 1$, имеем

$$Z^2 = \int_0^{2\pi} \int_0^\infty r \exp\left(-\frac{r^2}{2\sigma^2}\right) dr d\theta. \quad (2.193)$$

Вычислите этот интеграл и покажите, что $Z = \sqrt{\sigma^2 2\pi}$. *Указание 1:* представьте интеграл в виде произведения двух сомножителей, один из которых (включающий $d\theta$) постоянный, так что вычислить его легко. *Указание 2:* если $u = e^{-r^2/2\sigma^2}$, то $du/dr = -(1/\sigma^2) r e^{-r^2/2\sigma^2}$, так что второй интеграл тоже берется легко (поскольку $\int u'(r) dr = u(r)$).

Глава 3

Вероятность: многомерные модели

3.1. СОВМЕСТНЫЕ РАСПРЕДЕЛЕНИЯ НЕСКОЛЬКИХ СЛУЧАЙНЫХ ВЕЛИЧИН

В этом разделе мы обсудим различные способы измерения зависимости одной или нескольких случайных величин друг от друга.

3.1.1. Ковариация

Ковариация случайных величин X и Y измеряет степень их (линейной) зависимости и определяется следующим образом:

$$\text{Cov}[X, Y] \triangleq \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]. \quad (3.1)$$

Если \mathbf{x} – D -мерный случайный вектор, то его **ковариационной матрицей** называется следующая симметричная положительно полуопределенная матрица:

$$\text{Cov}[\mathbf{x}] \triangleq \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^T] \triangleq \mathbf{\Sigma}. \quad (3.2)$$

$$= \begin{pmatrix} \mathbb{V}[X_1] & \text{Cov}[X_1, X_2] & \cdots & \text{Cov}[X_1, X_D] \\ \text{Cov}[X_2, X_1] & \mathbb{V}[X_2] & \cdots & \text{Cov}[X_2, X_D] \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}[X_D, X_1] & \text{Cov}[X_D, X_2] & \cdots & \mathbb{V}[X_D] \end{pmatrix}. \quad (3.3)$$

Из этого определения следует важный результат:

$$\mathbb{E}[\mathbf{x}\mathbf{x}^T] = \mathbf{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^T. \quad (3.4)$$

Еще один полезный результат состоит в том, что ковариация линейного преобразования удовлетворяет соотношению:

$$\text{Cov}[\mathbf{A}\mathbf{x} + \mathbf{b}] = \mathbf{A}\text{Cov}[\mathbf{x}]\mathbf{A}^T, \quad (3.5)$$

как показано в упражнении 3.4.

Взаимная ковариация двух случайных векторов определяется как

$$\text{Cov}[\mathbf{x}, \mathbf{y}] = \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{y} - \mathbb{E}[\mathbf{y}])^T]. \quad (3.6)$$

3.1.2. Корреляция

Ковариация изменяется от минус до плюс бесконечности. Иногда удобнее использовать нормированную меру с конечной нижней и верхней границей.

Коэффициент корреляции Пирсона между X и Y определяется как

$$\rho \triangleq \text{corr}[X, Y] \triangleq \frac{\text{Cov}[X, Y]}{\sqrt{\mathbb{V}[X]\mathbb{V}[Y]}}. \quad (3.7)$$

Можно показать (упражнение 3.2), что $-1 \leq \rho \leq 1$.

Также можно показать, что $\text{corr}[X, Y] = 1$ тогда и только тогда, когда $Y = aX + b$ (и $a > 0$) для некоторых параметров a и b , т. е. когда между X и Y существует *линейная* зависимость (см. упражнение 3.3). Интуитивно кажется, что коэффициент корреляции как-то связан с наклоном регрессионной прямой, т. е. с коэффициентом a в выражении $Y = aX + b$. Однако, как будет показано в формуле (11.27), коэффициент регрессии на самом деле равен $a = \text{Cov}[X, Y]/\mathbb{V}[X]$. На рис. 3.1 видно, что коэффициент корреляции может быть равен 0 при наличии сильной, но нелинейной зависимости (сравните с рис. 6.6). Поэтому лучше представлять коэффициент корреляции как *степень линейности* (см. демонстрацию этой идеи программой по адресу code.problml.ai/book1/correlation2d).

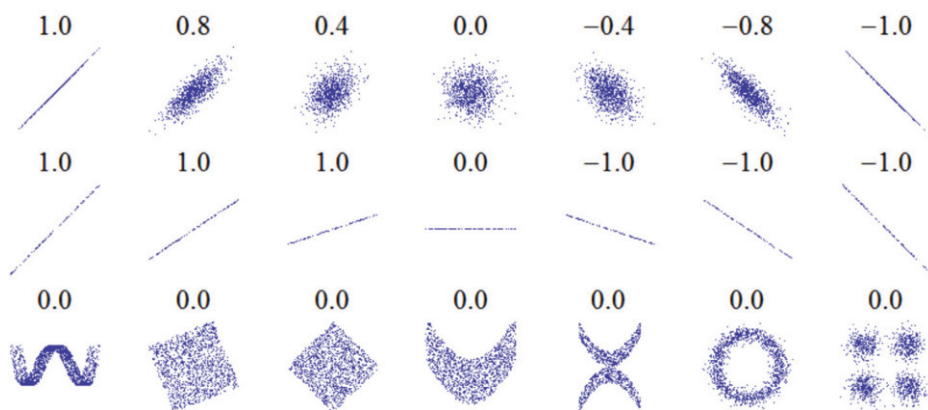


Рис. 3.1 ❖ Несколько наборов точек (x, y) и коэффициент корреляции x и y для каждого набора. Отметим, что корреляция отражает зашумленность и направление линейной зависимости (верхний ряд), но не отражает ее угловой коэффициент (средний ряд), а также многие аспекты нелинейных связей (нижний ряд). (Примечание: на рисунке в центре угловой коэффициент равен 0, но в этом случае коэффициент корреляции не определен, потому что дисперсия Y нулевая.) Взято из статьи https://en.wikipedia.org/wiki/Pearson_correlation_coefficient. Печатается с разрешения автора «Википедии» Imagecreator

В случае, когда \mathbf{x} — вектор взаимосвязанных случайных величин, **корреляционная матрица** определяется как

$$\text{corr}(\mathbf{x}) = \begin{pmatrix} 1 & \frac{\mathbb{E}[(X_1 - \mu_1)(X_2 - \mu_2)]}{\sigma_1 \sigma_2} & \dots & \frac{\mathbb{E}[(X_1 - \mu_1)(X_D - \mu_D)]}{\sigma_1 \sigma_D} \\ \frac{\mathbb{E}[(X_2 - \mu_2)(X_1 - \mu_1)]}{\sigma_2 \sigma_1} & 1 & \dots & \frac{\mathbb{E}[(X_2 - \mu_2)(X_D - \mu_D)]}{\sigma_2 \sigma_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\mathbb{E}[(X_D - \mu_D)(X_1 - \mu_1)]}{\sigma_D \sigma_1} & \frac{\mathbb{E}[(X_D - \mu_D)(X_2 - \mu_2)]}{\sigma_D \sigma_2} & \dots & 1 \end{pmatrix}. \quad (3.8)$$

Это можно более компактно записать в виде

$$\text{corr}(\mathbf{x}) = (\text{diag}(\mathbf{K}_{xx}))^{-\frac{1}{2}} \mathbf{K}_{xx} (\text{diag}(\mathbf{K}_{xx}))^{-\frac{1}{2}}, \quad (3.9)$$

где \mathbf{K}_{xx} — **автоковариационная матрица**:

$$\mathbf{K}_{xx} = \Sigma = \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^T] = \mathbf{R}_{xx} - \boldsymbol{\mu}\boldsymbol{\mu}^T, \quad (3.10)$$

а $\mathbf{R}_{xx} = \mathbb{E}[\mathbf{x}\mathbf{x}^T]$ — **автокорреляционная матрица**.

3.1.3. Некоррелированные не значит независимые

Если X и Y независимы, т. е. $p(X, Y) = p(X)p(Y)$, то $\text{Cov}[X, Y] = 0$ и, значит, $\text{corr}[X, Y] = 0$. Поэтому из независимости вытекает некоррелированность. Однако обратное неверно: из некоррелированности не следует независимость. Например, пусть $X \sim U(-1, 1)$ и $Y = X^2$. Очевидно, что Y зависит от X (более того, Y однозначно определено X), но можно показать (упражнение 3.1), что $\text{corr}[X, Y] = 0$. Некоторые поразительные примеры этого факта показаны на рис. 3.1. Мы видим несколько наборов данных, для которых наличие зависимости между X и Y не вызывает сомнения, а коэффициент корреляции тем не менее равен 0. Более общей мерой зависимости между случайными величинами является взаимная информация, рассматриваемая в разделе 6.3. Она равна 0 только тогда, когда величины действительно независимы.

3.1.4. Из коррелированности не следует наличие причинно-следственной связи

Хорошо известно, что «**коррелированность не означает причинно-следственную связь**». Например, рассмотрим рис. 3.2. Красным цветом изображен график $x_{1:T}$, где x_t — объем продаж мороженого в месяце t . Желтым цветом изображен график $y_{1:T}$, где y_t — частота насильственных преступлений в месяце t . (Величины приведены к общему масштабу, так чтобы графики накладывались друг на друга.) Мы наблюдаем сильную корреляцию между сигналами.

Иногда по этой причине заявляют, что «поедание мороженого провоцирует преступления» [Pet13]. Конечно, это всего лишь **ложная корреляция**, вызванная **общей скрытой причиной**, а именно погодой. Когда жарко, естественно, растет потребление мороженого. Но жара также приводит к росту насильственных преступлений; почему это так – предмет (кхе-кхе) жарких споров; одни считают, что жара провоцирует гнев [And01], другие – что дело в том, что люди чаще выходят на улицу [Ash18], где и происходят убийства.

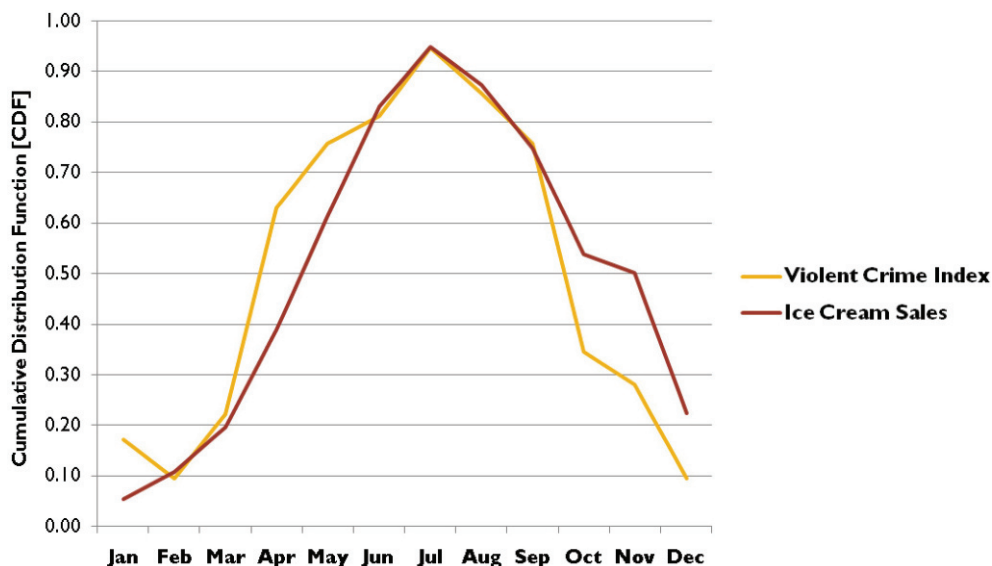


Рис. 3.2 ❖ Примеры ложных корреляций между временными рядами, не имеющими причинно-следственной связи. Потребление мороженого (красная кривая) и частота насильственных преступлений (желтая кривая) в зависимости от времени года. Взято из статьи <http://icbseverywhere.com/blog/2014/10/the-logic-of-causal-conclusions/>. Печатается с разрешения Барбары Дрешер

Еще один знаменитый пример касается положительной корреляции между рождаемостью и наличием аистов. Он породил городскую легенду о том, что аисты приносят детей [Mat00]. Конечно, истинная причина корреляции связана, скорее, со скрытыми факторами, например повышением жизненного уровня и, следовательно, обилием пищи. Много других забавных примеров ложной корреляции приведено в работе [Vig15].

Эти примеры служат предупреждением о том, что не следует рассматривать способность x предсказывать y как признак того, что x является причиной y .

3.1.5. Парадокс Симпсона

Парадокс Симпсона говорит, что статистический тренд или связь, наблюдаемая в нескольких группах данных, может исчезнуть или изменить знак,

когда эти группы объединяются. Это приводит к противоречащему интуиции поведению, если ошибочно воспринять статистическую зависимость как причинно-следственную связь. Визуализация этого парадокса показана на рис. 3.3. Мы видим, что в целом y уменьшается с ростом x , но внутри каждой подгруппы y увеличивается.

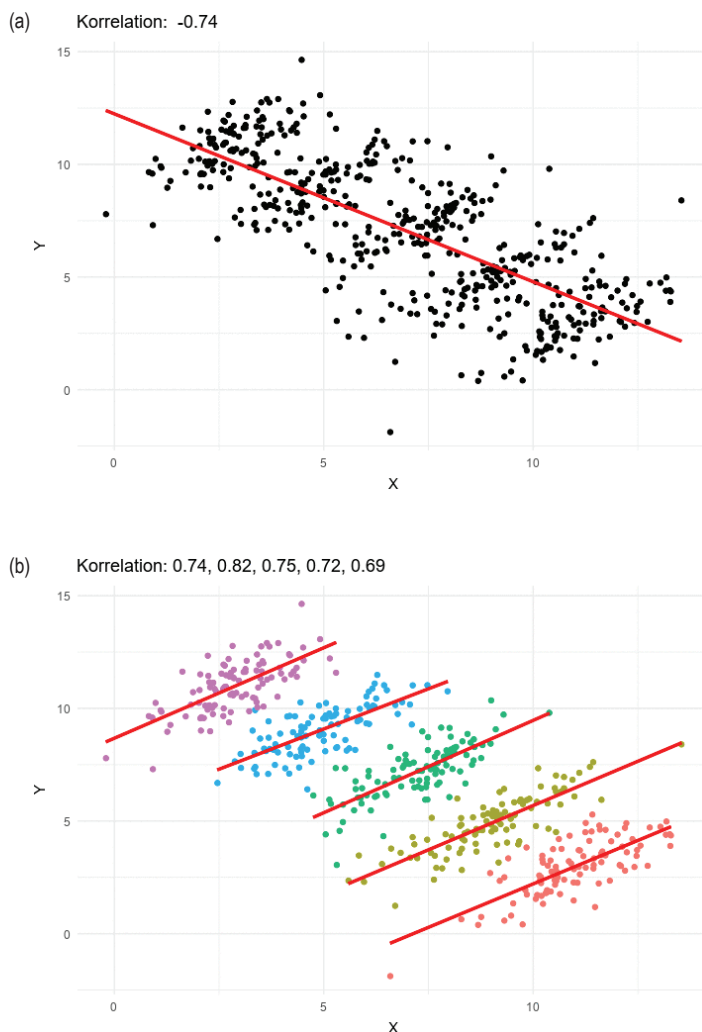


Рис. 3.3 ❖ Парадокс Симпсона. (а) В целом y уменьшается с ростом x . (б) В каждой группе y увеличивается с ростом x . Взято из статьи https://en.wikipedia.org/wiki/Simpson%27s_paradox. Печатается с разрешения автора «Википедии» Рэсе svwiki

Реальный пример парадокса Симпсона в контексте COVID-19 представлен на рис. 3.4а. Здесь показано, что в Италии коэффициент смертности (case fatality rate – CFR) от COVID-19 меньше, чем в Китае, в каждой возрастной группе, но в целом смертность выше. Причина в том, что в Италии больше пожилых людей, как показано на рис. 3.4б. Иными словами, рис. 3.4а показывает $p(F = 1|A, C)$, где A – возраст, C – страна, а $F = 1$ – событие, заключающееся в смерти от COVID-19. А рис. 3.4б показывает $p(A|C)$, т. е. вероятность оказаться в возрастной группе A в стране C . Объединяя то и другое, находим $p(F = 1|C = \text{Италия}) > p(F = 1|C = \text{Китай})$. Дополнительные сведения см. в работе [KGS20].

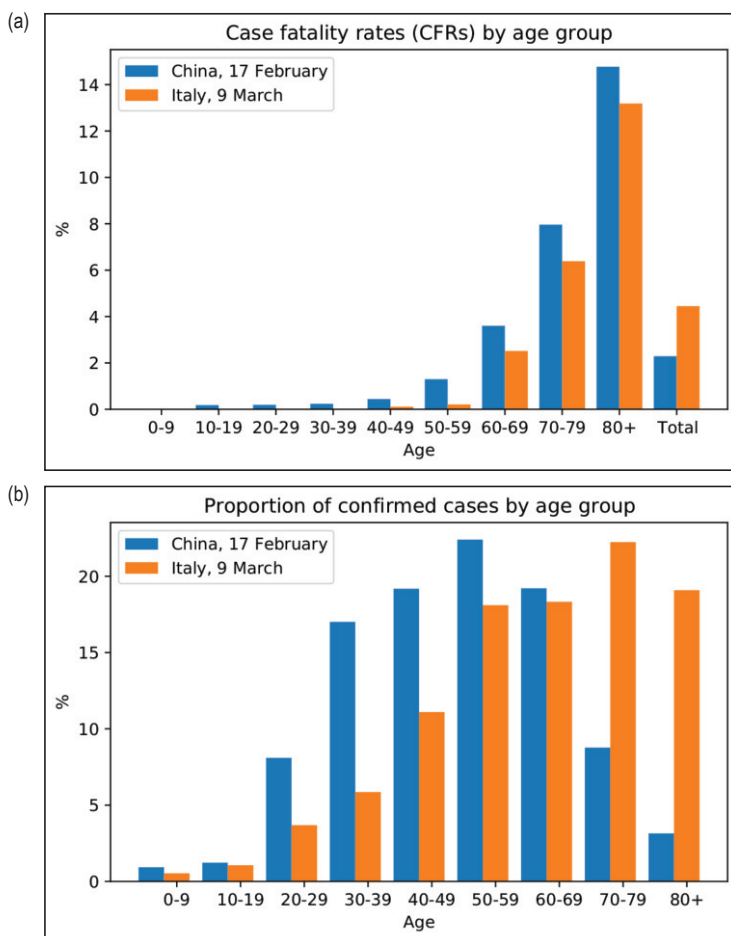


Рис. 3.4 ❖ Парадокс Симпсона на примере COVID-19. (а) Коэффициенты смертности (CFR) в Италии и в Китае в разрезе возрастных групп и в агрегированной форме (последние два столбика, «Total») за период до момента публикации отчета (см. надписи). (б) Доля всех подтвержденных случаев, включенных в (а) в каждой возрастной группе по странам. Рисунок 1 из работы [KGS20]. Печатается с разрешения Юлиуса фон Кюгельгена

3.2. МНОГОМЕРНОЕ ГАУССОВО (НОРМАЛЬНОЕ) РАСПРЕДЕЛЕНИЕ

Самым распространенным из совместных распределений непрерывных случайных величин является **многомерное гауссово**, или **нормальное, распределение** (multivariate normal – MVN). Объясняется это в первую очередь математическим удобством, но также и тем, что предположение о гауссовости во многих случаях вполне разумно (см. обсуждение в разделе 2.6.4).

3.2.1. Определение

Плотность MVN по определению равна

$$\mathcal{N}(\mathbf{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \triangleq \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{y} - \boldsymbol{\mu})\right], \quad (3.11)$$

где $\boldsymbol{\mu} = \mathbb{E}[\mathbf{y}] \in \mathbb{R}^D$ – вектор средних, а $\boldsymbol{\Sigma} = \text{Cov}[\mathbf{y}]$ – ковариационная матрица размера $D \times D$, определенная следующим образом:

$$\text{Cov}[\mathbf{y}] \triangleq \mathbb{E}[(\mathbf{y} - \mathbb{E}[\mathbf{y}])(\mathbf{y} - \mathbb{E}[\mathbf{y}])^\top] \quad (3.12)$$

$$= \begin{pmatrix} \mathbb{V}[Y_1] & \text{Cov}[Y_1, Y_2] & \cdots & \text{Cov}[Y_1, Y_D] \\ \text{Cov}[Y_2, Y_1] & \mathbb{V}[Y_2] & \cdots & \text{Cov}[Y_2, Y_D] \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}[Y_D, Y_1] & \text{Cov}[Y_D, Y_2] & \cdots & \mathbb{V}[Y_D] \end{pmatrix}. \quad (3.13)$$

где

$$\text{Cov}[Y_i, Y_j] \triangleq \mathbb{E}[(Y_i - \mathbb{E}[Y_i])(Y_j - \mathbb{E}[Y_j])] = \mathbb{E}[Y_i Y_j] - \mathbb{E}[Y_i]\mathbb{E}[Y_j]. \quad (3.14)$$

и $\mathbb{V}[Y_i] = \text{Cov}[Y_i, Y_i]$. Из (3.12) получаем важный результат:

$$\mathbb{E}[\mathbf{y}\mathbf{y}^\top] = \boldsymbol{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^\top. \quad (3.15)$$

Нормировочная постоянная в формуле (3.11) $Z = (2\pi)^{D/2} |\boldsymbol{\Sigma}|^{1/2}$ нужна для того, чтобы интеграл ФПВ был равен 1 (см. упражнение 3.6).

В двумерном случае многомерное нормальное распределение называется **двумерным гауссовым** распределением. Его ФПВ имеет вид $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, где $\mathbf{y} \in \mathbb{R}^2$, $\boldsymbol{\mu} \in \mathbb{R}^2$ и

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \sigma_{12}^2 \\ \sigma_{21}^2 & \sigma_2^2 \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}, \quad (3.16)$$

где ρ – **коэффициент корреляции**, равный

$$\text{corr}[Y_1, Y_2] \triangleq \frac{\text{Cov}[Y_1, Y_2]}{\sqrt{\mathbb{V}[Y_1]\mathbb{V}[Y_2]}} = \frac{\sigma_{12}^2}{\sigma_1 \sigma_2}. \quad (3.17)$$

Можно показать (упражнение 3.2), что $-1 \leq \text{corr}[Y_1, Y_2] \leq 1$. Обобщение ФПВ на двумерный случай дает следующий устрашающий результат:

$$p[y_1, y_2] = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)} \times \left[\frac{(y_1-\mu_1)^2}{\sigma_1^2} + \frac{(y_2-\mu_2)^2}{\sigma_2^2} - 2\rho\frac{(y_1-\mu_1)(y_2-\mu_2)}{\sigma_1\sigma_2}\right]\right). \quad (3.18)$$

На рис. 3.5 и 3.6 изображены графики плотности двумерного гауссова распределения для нескольких вариантов ковариационной матрицы. В **полной ковариационной матрице** имеется $D(D+1)/2$ параметров (на 2 делится, потому что матрица Σ симметрична). (Почему форма линий уровня эллиптическая, объясняется в разделе 7.4.4 при обсуждении геометрии квадратичных форм.) В **диагональной ковариационной матрице** D параметров, а все внедиагональные элементы равны 0. **Сферическая ковариационная матрица**, называемая также **изотропной ковариационной матрицей**, имеет вид $\Sigma = \sigma^2 \mathbf{I}_D$, так что свободный параметр только один, а именно σ^2 .

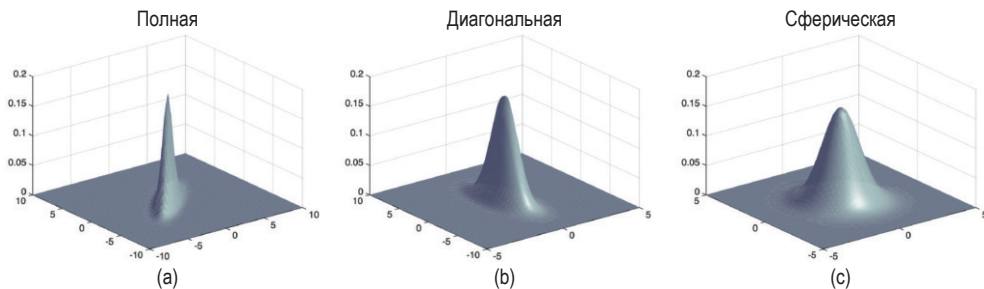


Рис. 3.5 ❖ Визуализация плотности двумерного гауссова распределения в виде поверхности. (a) Распределение с полной ковариационной матрицей можно ориентировать под любым углом. (b) Распределение с диагональной ковариационной матрицей должно быть параллельно оси. (c) Распределение со сферической ковариационной матрицей должно быть симметрично. Построено программой по адресу figures.problmlai/book1/3.5

3.2.2. Расстояние Махаланобиса

В этом разделе мы постараемся пролить свет на геометрическую форму функции плотности вероятности многомерного гауссова распределения. Для этого рассмотрим форму **линий уровня** постоянной (логарифмической) вероятности.

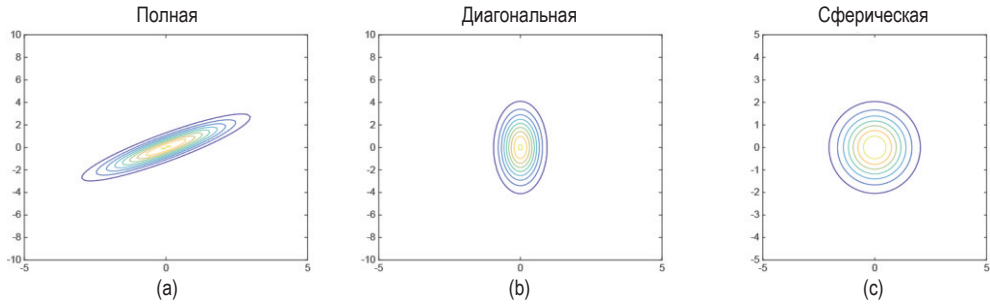


Рис. 3.6. Визуализация плотности двумерного гауссова распределения в терминах линий уровня постоянной плотности вероятности. (a) В случае полной ковариационной матрицы контуры эллиптические. (b) В случае диагональной ковариационной матрицы оси эллипса параллельны осям координат. (c) В случае сферической ковариационной матрицы линии уровня – окружности. Построено программой по адресу figures.problml.ai/book1/3.6

Логарифмическая вероятность в точке \mathbf{y} равна

$$\log p(\mathbf{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{y} - \boldsymbol{\mu}) + \text{const}. \quad (3.19)$$

Зависимость от \mathbf{y} можно выразить в терминах **расстояния Махаланобиса** Δ между \mathbf{y} и $\boldsymbol{\mu}$, квадрат которого определяется формулой:

$$\Delta^2 \triangleq (\mathbf{y} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{y} - \boldsymbol{\mu}). \quad (3.20)$$

Поэтому линии постоянной (логарифмической) вероятности эквивалентны линиям постоянного расстояния Махаланобиса.

Чтобы понять, как выглядят линии постоянного расстояния Махаланобиса, воспользуемся тем фактом, что $\boldsymbol{\Sigma}$, а значит, и $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ – положительно определенные матрицы (по предположению). Рассмотрим следующее спектральное разложение (см. раздел 7.4) матрицы $\boldsymbol{\Sigma}$:

$$\boldsymbol{\Sigma} = \sum_{d=1}^D \lambda_d \mathbf{u}_d \mathbf{u}_d^\top. \quad (3.21)$$

Точно так же можно написать:

$$\boldsymbol{\Sigma}^{-1} = \sum_{d=1}^D \frac{1}{\lambda_d} \mathbf{u}_d \mathbf{u}_d^\top. \quad (3.22)$$

Определим $z_d \triangleq \mathbf{u}_d^\top (\mathbf{y} - \boldsymbol{\mu})$, так что $\mathbf{z} = \mathbf{U}(\mathbf{y} - \boldsymbol{\mu})$. Тогда расстояние Махаланобиса можно переписать в виде:

$$(\mathbf{y} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{y} - \boldsymbol{\mu}) = (\mathbf{y} - \boldsymbol{\mu})^\top \left(\sum_{d=1}^D \frac{1}{\lambda_d} \mathbf{u}_d \mathbf{u}_d^\top \right) (\mathbf{y} - \boldsymbol{\mu}) \quad (3.23)$$

$$= \sum_{d=1}^D \frac{1}{\lambda_d} (\mathbf{y} - \boldsymbol{\mu})^\top \mathbf{u}_d \mathbf{u}_d^\top (\mathbf{y} - \boldsymbol{\mu}) = \sum_{d=1}^D \frac{z_d^2}{\lambda_d}. \quad (3.24)$$

Как мы увидим в разделе 7.4.4, это означает, что расстояние Махаланобиса можно интерпретировать как евклидово расстояние в новой системе координат \mathbf{z} , в которой \mathbf{y} повернуто на \mathbf{U} и масштабировано на $\boldsymbol{\Lambda}$.

Например, в двумерной системе координат рассмотрим множество точек (z_1, z_2) , удовлетворяющих уравнению:

$$\frac{z_1^2}{\lambda_1} + \frac{z_2^2}{\lambda_2} = r. \quad (3.25)$$

Поскольку для этих точек расстояние Махаланобиса одинаково, они соответствуют точкам равной вероятности. Таким образом, мы видим, что линии равной плотности вероятности двумерного гауссова распределения имеют форму эллипсов. Это показано на рис. 7.6. Собственные векторы определяют ориентацию эллипса, а собственные значения – его вытянутость.

3.2.3. Маргинальные и условные распределения для многомерного нормального распределения*

Пусть $\mathbf{y} = (y_1, y_2)$ имеют совместное гауссово распределение с параметрами

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}, \quad \boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{pmatrix}, \quad (3.26)$$

где $\boldsymbol{\Lambda}$ – **матрица точности**. Тогда маргинальные распределения определяются формулами:

$$\boxed{\begin{aligned} p(y_1) &= \mathcal{N}(y_1 | \mu_1, \Sigma_{11}); \\ p(y_2) &= \mathcal{N}(y_2 | \mu_2, \Sigma_{22}), \end{aligned}} \quad (3.27)$$

а апостериорное условное распределение – формулами:

$$\boxed{\begin{aligned} p(y_1 | y_2) &= \mathcal{N}(y_1 | \mu_{1|2}, \Sigma_{1|2}); \\ \mu_{1|2} &= \mu_1 + \Sigma_{12} \Sigma_{22}^{-1} (y_2 - \mu_2) \\ &= \mu_1 - \Lambda_{11}^{-1} \Lambda_{12} (y_2 - \mu_2) \\ &= \Sigma_{1|2} (\Lambda_{11} \mu_1 - \Lambda_{12} (y_2 - \mu_2)); \\ \Sigma_{1|2} &= \Sigma_{11} - \Sigma_{12} \Sigma_{22}^{-1} \Sigma_{21} = \Lambda_{11}^{-1}. \end{aligned}} \quad (3.28)$$

Важность этих формул настолько велика, что мы обвели их рамочкой, чтобы потом было проще найти. Вывод этих результатов (опирающийся на дополнение Шура $\boldsymbol{\Sigma}/\Sigma_{22} = \Sigma_{11} - \Sigma_{12} \Sigma_{22}^{-1} \Sigma_{21}$) см. в разделе 7.3.5.

Как видим, маргинальные и условные распределения сами являются гауссовыми. В случае маргинальных распределений мы просто выделяем строки и столбцы, соответствующие y_1 или y_2 . В случае условного распределения придется поработать побольше. Впрочем, и это не сложно: условное среднее является линейной функцией от y_2 , а условная ковариация – постоянная матрица, не зависящая от y_2 . Мы приводим три разных (но эквивалентных) выражения для апостериорного среднего и два разных (но эквивалентных) выражения для апостериорной ковариации; все они полезны в определенных обстоятельствах.

3.2.4. Пример: обусловливание двумерного гауссова распределения

Рассмотрим двумерный случай. Ковариационная матрица имеет вид:

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}. \quad (3.29)$$

Маргинальное распределение $p(y_1)$ является одномерным гауссовым и получается проецированием совместного распределения на ось y_1 :

$$p(y_1) = \mathcal{N}(y_1 | \mu_1, \sigma_1^2). \quad (3.30)$$

Предположим, что наблюдается $Y_2 = y_2$; условное распределение $p(y_1|y_2)$ получается «прорезанием» совместного распределения прямой $Y_2 = y_2$:

$$p(y_1|y_2) = \mathcal{N}\left(y_1 | \mu_1 + \frac{\rho\sigma_1\sigma_2}{\sigma_2^2}(y_2 - \mu_2), \sigma_1^2 - \frac{(\rho\sigma_1\sigma_2)^2}{\sigma_2^2}\right). \quad (3.31)$$

Если $\sigma_1 = \sigma_2 = \sigma$, то получаем:

$$p(y_1|y_2) = \mathcal{N}(y_1 | \mu_1 + \rho(y_2 - \mu_2), \sigma^2(1 - \rho^2)). \quad (3.32)$$

Например, предположим, что $\rho = 0.8$, $\sigma_1 = \sigma_2 = 1$, $\mu_1 = \mu_2 = 0$ и $y_2 = 1$. Мы видим, что $\mathbb{E}[y_1|y_2 = 1] = 0.8$; это имеет смысл, т. к. $\rho = 0.8$ означает нашу веру в то, что если y_2 увеличивается на 1 (от среднего), то y_1 увеличивается на 0.8. Мы также видим, что $\mathbb{V}[y_1|y_2 = 1] = 1 - 0.8^2 = 0.36$. Это тоже имеет смысл: наша неуверенность в значении y_1 снизилась, поскольку мы что-то узнали о y_1 (косвенно) из наблюдения y_2 . Если $\rho = 0$, то получаем $p(y_1|y_2) = \mathcal{N}(y_1 | \mu_1, \sigma_1^2)$, потому что y_2 не несет никакой информации о y_1 , если они некоррелированы (и, стало быть, независимы).

3.2.5. Пример: подстановка отсутствующих значений*

В качестве примера применения описанных выше результатов предположим, что наблюдаются некоторые части (измерения) y , а остальные части отсутствуют или не наблюдаемы. Мы можем воспользоваться корреляцией между измерениями (представленной ковариационной матрицей), чтобы вывести отсутствующие значения; это называется **подстановкой отсутствующих значений**.

На рис. 3.7 приведен простой пример. Мы выбрали N векторов из $D = 10$ -мерного гауссова распределения, а затем сознательно «скрыли» 50 % данных в каждом примере (строке). Затем мы вывели отсутствующие значения, зная наблюдаемые значения и истинные параметры модели¹. Точнее, для каждой строки n матрицы данных мы вычисляем $p(y_{n,h}|y_{n,v}, \theta)$, где v – индексы видимых элементов данной строки, h – индексы остальных, скрытых, элементов и $\theta = (\mu, \Sigma)$. Далее вычисляем маргинальное распределение каждой отсутствующей переменной $i \in h$, $p(y_{n,i}|y_{n,v}, \theta)$. А зная маргинальное распределение, вычисляем апостериорное среднее $\bar{y}_{n,i} = \mathbb{E}[y_{n,i}|y_{n,v}, \theta]$.

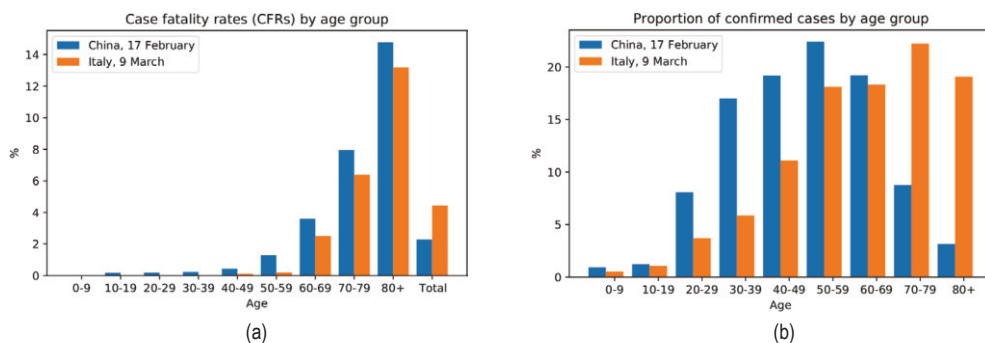


Рис. 3.7 ❖ Подстановка данных с использованием многомерного гауссова распределения. (а) Визуализация матрицы данных. Пустые ячейки означают, что данные отсутствуют. (Это так называемая диаграмма Хинтона, названная в честь Джеффри Хинтона, хорошо известного специалиста по МО.) (б) Истинная матрица данных (скрытая). (с) Среднее апостериорного прогнозного распределения, основанное на частично наблюдаемых данных в строке и истинных параметрах модели. Построено программой по адресу figures.problml.ai/book1/3.7

Апостериорное среднее представляет нашу «лучшую гипотезу» об истинном значении элемента в том смысле, что оно минимизирует ожидаемую квадратичную ошибку, как объясняется в главе 5. Мы можем использовать

¹ На практике следовало бы оценить параметры по частично наблюдаемым данным. К сожалению, оценка максимального правдоподобия (MLE) из раздела 4.2.6 здесь неприменима, но можно воспользоваться ЕМ-алгоритмом для аппроксимации MLE в условиях отсутствия части данных. Детали см. во втором томе.

$\mathbb{V}[y_{n,i}|\mathbf{y}_{n,v}, \boldsymbol{\theta}]$ как меру уверенности в этой гипотезе, хотя на рисунке это не показано. С другой стороны, мы могли бы выбрать несколько апостериорных примеров из распределения $p(\mathbf{y}_{n,h}|\mathbf{y}_{n,v}, \boldsymbol{\theta})$; это называется **множественной подстановкой** и дает более робастную оценку для последующих алгоритмов, потребляющих «подставленные» данные.

3.3. ЛИНЕЙНЫЕ ГАУССОВЫ СИСТЕМЫ*

В разделе 3.2.3 мы выводили апостериорное распределение скрытых частей гауссова случайного вектора, используя в качестве условия незашумленные наблюдения. Сейчас мы обобщим этот подход на зашумленные наблюдения. Пусть $\mathbf{z} \in \mathbb{R}^L$ – неизвестный вектор значений, а $\mathbf{y} \in \mathbb{R}^D$ – зашумленное измерение \mathbf{z} . Мы предполагаем, что эти величины связаны следующим совместным распределением:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z); \quad (3.33)$$

$$p(\mathbf{y}|\mathbf{z}) = \mathcal{N}(\mathbf{y}|\mathbf{W}\mathbf{z} + \mathbf{b}, \boldsymbol{\Sigma}_y), \quad (3.34)$$

где \mathbf{W} – матрица размера $D \times L$. Это пример **линейной гауссовой системы**.

Соответствующее совместное распределение, $p(\mathbf{z}, \mathbf{y}) = p(\mathbf{z})p(\mathbf{y}|\mathbf{z})$, является $(L + D)$ -мерным гауссовым, а его среднее и ковариация равны соответственно:

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_z \\ \mathbf{W}\boldsymbol{\mu}_z + \mathbf{b} \end{pmatrix}; \quad (3.35)$$

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_z & \boldsymbol{\Sigma}_z \mathbf{W}^\top \\ \mathbf{W}\boldsymbol{\Sigma}_z & \boldsymbol{\Sigma}_y + \mathbf{W}\boldsymbol{\Sigma}_z \mathbf{W}^\top \end{pmatrix}. \quad (3.36)$$

Применив формулу гауссова обусловливания (3.28) к совместному распределению $p(\mathbf{y}, \mathbf{z})$, мы сможем вычислить апостериорное распределение $p(\mathbf{z}|\mathbf{y})$, как будет объяснено ниже. Это можно интерпретировать как обращение стрелки $\mathbf{z} \rightarrow \mathbf{y}$ в порождающей модели, так что она ведет от латентных значений к наблюдаемым.

3.3.1. Формула Байеса для гауссовых распределений

Апостериорное распределение латентных значений имеет вид:

$$\begin{aligned} p(\mathbf{z}|\mathbf{y}) &= \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{z|\mathbf{y}}, \boldsymbol{\Sigma}_{z|\mathbf{y}}); \\ \boldsymbol{\Sigma}_{z|\mathbf{y}}^{-1} &= \boldsymbol{\Sigma}_z^{-1} + \mathbf{W}^\top \boldsymbol{\Sigma}_y^{-1} \mathbf{W}; \\ \boldsymbol{\mu}_{z|\mathbf{y}} &= \boldsymbol{\Sigma}_{z|\mathbf{y}} [\mathbf{W}^\top \boldsymbol{\Sigma}_y^{-1} (\mathbf{y} - \mathbf{b}) + \boldsymbol{\Sigma}_z^{-1} \boldsymbol{\mu}_z]. \end{aligned}$$

(3.37)

Эти выражения называются **формулой Байеса для гауссовых распределений**. Нормировочная постоянная апостериорного распределения равна:

$$p(y) = \int \mathcal{N}(z|\mu_z, \Sigma_z) \mathcal{N}(y|Wz + b, \Sigma_y) dz = \mathcal{N}(y|W\mu_z + b, \Sigma_y + W\Sigma_z W^T). \quad (3.38)$$

Мы видим, что гауссово априорное распределение $p(z)$ в сочетании с гауссовым правдоподобием $p(y|z)$ дает гауссово апостериорное распределение $p(z|y)$. Таким образом, множество гауссовых распределений замкнуто относительно байесовского обусловливания. Чтобы описать этот факт более общим образом, будем говорить, что гауссово априорное распределение является **сопряженным априорным** для гауссова правдоподобия, потому что апостериорное распределение имеет тот же тип, что априорное. Мы обсудим понятие сопряженных априорных распределений более подробно в разделе 4.6.1.

В разделах ниже приведены различные применения этого результата. Но сначала выведем его.

3.3.2. Вывод*

Выведем формулы (3.37). Основная идея заключается в том, чтобы вывести совместное распределение $p(z, y) = p(z)p(y|z)$, а затем воспользоваться результатами из раздела 3.2.3 для вычисления $p(y|z)$.

Будем рассуждать следующим образом. Логарифм совместного распределения имеет вид (опуская несущественные постоянные):

$$\log p(z, y) = -\frac{1}{2}(z - n_z)^T \Sigma_z^{-1}(z - n_z) - \frac{1}{2}(y - Wz - b)^T \Sigma_y^{-1}(y - Wz - b). \quad (3.39)$$

Очевидно, что это совместное гауссово распределение, потому что оно является экспонентой квадратичной формы.

Раскрывая квадратичные члены, содержащие z и y , и игнорируя линейные и постоянные члены, получаем:

$$Q = -\frac{1}{2}z^T \Sigma_z^{-1}z - \frac{1}{2}y^T \Sigma_y^{-1}y - \frac{1}{2}(Wz)^T \Sigma_y^{-1}(Wz) + y^T \Sigma_y^{-1}Wz \quad (3.40)$$

$$= -\frac{1}{2} \begin{pmatrix} z \\ y \end{pmatrix}^T \begin{pmatrix} \Sigma_z^{-1} + W^T \Sigma_y^{-1} W & -W^T \Sigma_y^{-1} \\ -\Sigma_y^{-1} W & \Sigma_y^{-1} \end{pmatrix} \begin{pmatrix} z \\ y \end{pmatrix} \quad (3.41)$$

$$= -\frac{1}{2} \begin{pmatrix} z \\ y \end{pmatrix}^T \Sigma^{-1} \begin{pmatrix} z \\ y \end{pmatrix}, \quad (3.42)$$

где матрица точности определена как

$$\Sigma^{-1} = \begin{pmatrix} \Sigma_z^{-1} + W^T \Sigma_y^{-1} W & -W^T \Sigma_y^{-1} \\ -\Sigma_y^{-1} W & \Sigma_y^{-1} \end{pmatrix} \triangleq \Lambda = \begin{pmatrix} \Lambda_{xx} & \Lambda_{xy} \\ \Lambda_{yx} & \Lambda_{yy} \end{pmatrix}. \quad (3.43)$$

Из (3.28) и того факта, что $\mu_y = \mathbf{W}\mu_z + \mathbf{b}$, имеем

$$p(\mathbf{z}|\mathbf{y}) = \mathcal{N}(\mu_{z|\mathbf{y}}, \Sigma_{z|\mathbf{y}}); \quad (3.44)$$

$$\Sigma_{z|\mathbf{y}} = \Lambda_{xx}^{-1} = (\Sigma_z^{-1} + \mathbf{W}^T \Sigma_y^{-1} \mathbf{W})^{-1}; \quad (3.45)$$

$$\mu_{z|\mathbf{y}} = \Sigma_{z|\mathbf{y}}(\Lambda_{xx}\mu_z - \Lambda_{xy}(\mathbf{y} - \mu_y)) \quad (3.46)$$

$$= \Sigma_{z|\mathbf{y}}(\Sigma_z^{-1}\mu_z + \mathbf{W}^T \Sigma_y^{-1} \mathbf{W}\mu_z + \mathbf{W}^T \Sigma_y^{-1}(\mathbf{y} - \mu_y)) \quad (3.47)$$

$$= \Sigma_{z|\mathbf{y}}(\Sigma_z^{-1}\mu_z + \mathbf{W}^T \Sigma_y^{-1}(\mathbf{W}\mu_z + \mathbf{y} - \mu_y)) \quad (3.48)$$

$$= \Sigma_{z|\mathbf{y}}(\Sigma_z^{-1}\mu_z + \mathbf{W}^T \Sigma_y^{-1}(\mathbf{y} - \mathbf{b})). \quad (3.49)$$

3.3.3. Пример: вывод неизвестного скаляра

Предположим, что мы произвели N зашумленных измерений y_i некоторой величины z ; предположим также, что шум измерений имеет фиксированную точность $\lambda_y = 1/\sigma^2$, так что правдоподобие равно:

$$p(y_i|z) = \mathcal{N}(y_i|z, \lambda_y^{-1}). \quad (3.50)$$

Теперь используем гауссово априорное распределение для значения неизвестного источника:

$$p(z) = \mathcal{N}(z|\mu_0, \lambda_0^{-1}). \quad (3.51)$$

Мы хотим вычислить $p(z|y_1, \dots, y_N, \sigma^2)$. Преобразуем это к виду, который позволит применить формулу Байеса для гауссовых распределений, для чего определим $\mathbf{y} = (y_1, \dots, y_N)$, $\mathbf{W} = \mathbf{1}_N^T$ (вектор-строка $1 \times N$, состоящий из единиц) и $\Sigma_y^{-1} = \text{diag}(\lambda_y \mathbf{I})$. Тогда получаем:

$$p(z|\mathbf{y}) = \mathcal{N}(z|\mu_N, \lambda_N^{-1}); \quad (3.52)$$

$$\lambda_N = \lambda_0 + N\lambda_y; \quad (3.53)$$

$$\mu_N = \frac{N\lambda_y \bar{y} + \lambda_0 \mu_0}{\lambda_N} = \frac{N\lambda_y + \lambda_0 \mu_0}{N\lambda_y + \lambda_0} \bar{y} + \frac{\lambda_0}{N\lambda_y + \lambda_0} \mu_0. \quad (3.54)$$

Эти формулы интуитивно понятны: апостериорная точность λ_N является априорной точностью λ_0 плюс N единиц точности измерений λ_y . Кроме того, апостериорное среднее μ_N является выпуклой комбинацией оценки максимального правдоподобия (MLE) \bar{y} и априорного среднего μ_0 . Отсюда понятно, что апостериорное среднее – это компромисс между MLE и априорным средним. Если априорное среднее мало, по сравнению с силой сигнала (μ_0 мало относительно λ_y), то мы назначаем больший вес MLE. Если же априорное среднее велико, по сравнению с силой сигнала (μ_0 велико относительно λ_y), то больший вес назначается априорному среднему. Это показано на рис. 3.8.

Заметим, что апостериорное среднее записывается в терминах $N\lambda_y \bar{y}$, так что наличие N измерений с точностью λ_y каждое сопоставимо с наличием одного измерения со значением \bar{y} и точностью $N\lambda_y$.

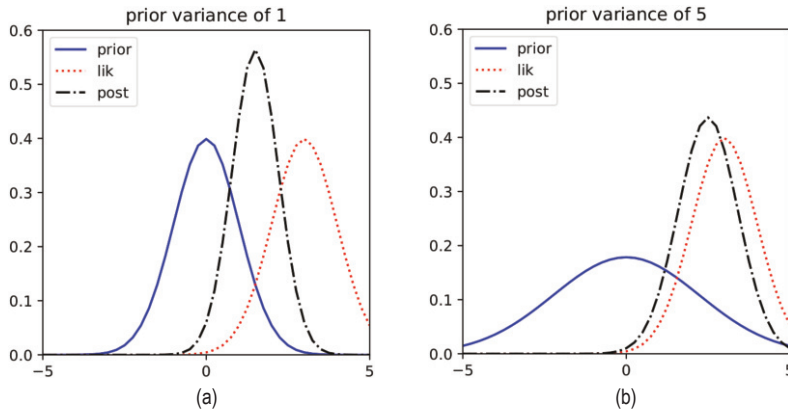


Рис. 3.8 ❖ Вывод z по зашумленному наблюдению $y = 3$. (a) Сильное априорное распределение $\mathcal{N}(0, 1)$. Апостериорное среднее, «усевшее» в сторону априорного среднего, равного 0. (b) Слабое априорное распределение $\mathcal{N}(0, 5)$. Апостериорное среднее близко к MLE. Построено программой по адресу figures.probl.ai/book1/3.8

Мы можем переписать результаты в терминах апостериорной дисперсии, а не апостериорной точности:

$$p(z|\mathcal{D}, \sigma^2) = \mathcal{N}(z|\mu_N, \tau_N^2); \quad (3.55)$$

$$\tau_N^2 = \frac{1}{\frac{N}{\sigma^2} + \frac{1}{\tau_0^2}} = \frac{\sigma^2 \tau_0^2}{N\tau_0^2 + \sigma^2}; \quad (3.56)$$

$$\mu_N = \tau_N^2 \left(\frac{\mu_0}{\tau_0^2} + \frac{N\bar{y}}{\sigma^2} \right) = \frac{\sigma^2}{N\tau_0^2 + \sigma^2} \mu_0 + \frac{N\tau_0^2}{N\tau_0^2 + \sigma^2} \bar{y}, \quad (3.57)$$

где $\tau_0^2 = 1/\lambda_0$ – априорная дисперсия, а $\tau_N^2 = 1/\lambda_N$ – апостериорная дисперсия.

Мы также можем вычислять апостериорное распределение последовательно, обновляя его после каждого наблюдения. Если $N = 1$, то после одного наблюдения можно переписать апостериорное распределение следующим образом (где $\Sigma_y = \sigma^2$, $\Sigma_0 = \tau_0^2$ и $\Sigma_1 = \tau_1^2$ – дисперсии правдоподобия, априорного и апостериорного распределения):

$$p(z|y) = \mathcal{N}(z|\mu_1, \Sigma_1); \quad (3.58)$$

$$\Sigma_1 = \left(\frac{1}{\Sigma_0} + \frac{1}{\Sigma_y} \right)^{-1} = \frac{\Sigma_y \Sigma_0}{\Sigma_0 + \Sigma_y}; \quad (3.59)$$

$$\mu_1 = \Sigma_1 \left(\frac{\mu_0}{\Sigma_0} + \frac{y}{\Sigma_y} \right). \quad (3.60)$$

Апостериорное среднее можно переписать тремя разными способами:

$$\mu_1 = \frac{\Sigma_y}{\Sigma_y + \Sigma_0} \mu_0 + \frac{\Sigma_0}{\Sigma_y + \Sigma_0} y \quad (3.61)$$

$$= \mu_0 + (y - \mu_0) \frac{\Sigma_0}{\Sigma_y + \Sigma_0} \quad (3.62)$$

$$= y - (y - \mu_0) \frac{\Sigma_y}{\Sigma_y + \Sigma_0}. \quad (3.63)$$

Первое равенство – выпуклая комбинация априорного распределения и данных, второе – априорное среднее, скорректированное в сторону данных, а третье – данные, скорректированные в сторону априорного среднего (это называется **усадкой**). Все три формы – эквивалентные способы выражения компромисса между правдоподобием и априорным распределением. Если Σ_0 мало, по сравнению с Σ_y , что соответствует сильному априорному распределению, то величина усадки велика (см. рис. 3.8a). Если же Σ_0 велико, по сравнению с Σ_y , что соответствует слабому априорному распределению, то величина усадки мала (см. рис. 3.8b).

Величину усадки можно также выразить в терминах отношения сигнал–шум, которое определено следующим образом:

$$\text{SNR} \triangleq \frac{\mathbb{E}[Z^2]}{\mathbb{E}[\mu^2]} = \frac{\Sigma_0 + \mu_0^2}{\Sigma_y}, \quad (3.64)$$

где $z \sim \mathcal{N}(\mu_0, \Sigma_0)$ – истинный сигнал, $y = z + \varepsilon$ – наблюдаемый сигнал, а $\varepsilon \sim \mathcal{N}(0, \Sigma_y)$ – шум.

3.3.4. Пример: вывод неизвестного вектора

Пусть имеется неизвестная величина $\mathbf{z} \in \mathbb{R}^D$, для которой мы предположим априорное гауссово распределение $p(\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$. Если мы ничего не знаем о \mathbf{z} заранее, то можем положить $\boldsymbol{\Sigma}_z = \infty \mathbf{I}$, это означает, что мы абсолютно не уверены в том, каково должно быть значение \mathbf{z} . (На практике можно использовать большую, но конечную ковариацию). В силу симметрии представляется разумным положить $\boldsymbol{\mu}_z = \mathbf{0}$.

Теперь предположим, что произведено N зашумленных, но независимых измерений \mathbf{z} , $\mathbf{y}_n \sim \mathcal{N}(\mathbf{z}, \boldsymbol{\Sigma}_y)$, размера D каждое. Тогда правдоподобие можно записать в виде:

$$p(\mathcal{D}|\mathbf{z}) = \prod_{n=1}^N N(\mathbf{y}_n|\mathbf{z}, \boldsymbol{\Sigma}_y) = N\left(\mathbf{y}|\mathbf{z}, \frac{1}{N} \boldsymbol{\Sigma}_y\right). \quad (3.65)$$

Заметим, что N наблюдений можно заменить их средним $\bar{\mathbf{y}}$, компенсировав это умножением ковариации на $1/N$. Положив $\mathbf{W} = \mathbf{I}$, $\mathbf{b} = \mathbf{0}$, мы можем воспользоваться формулой Байеса для гауссова распределения и вычислить апостериорное распределение \mathbf{z} :

$$p(\mathbf{z}|\mathbf{y}_1, \dots, \mathbf{y}_N) = N(\mathbf{z}|\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}); \quad (3.66)$$

$$\hat{\boldsymbol{\Sigma}}^{-1} = \boldsymbol{\Sigma}_z^{-1} + N\hat{\boldsymbol{\Sigma}}_y^{-1}; \quad (3.67)$$

$$\hat{\boldsymbol{\mu}} = \hat{\boldsymbol{\Sigma}}(\boldsymbol{\Sigma}_z^{-1}(N\bar{\mathbf{y}}) + \boldsymbol{\Sigma}_y^{-1}\boldsymbol{\mu}_z), \quad (3.68)$$

где $\hat{\boldsymbol{\mu}}$ и $\hat{\boldsymbol{\Sigma}}$ – параметры апостериорного распределения.

На рис. 3.9 приведен пример для двумерного случая. Можно считать, что \mathbf{z} представляет истинное, но неизвестное положение объекта на двумерной плоскости, например ракеты или самолета, а \mathbf{y}_n – зашумленные наблюдения, например радиолокационные отметки. Чем больше отметок, тем точнее мы можем локализовать источник. (Во втором томе, [Mur22], обсуждается **фильтр Калмана**, который обобщает эту идею на временную последовательность наблюдений.)

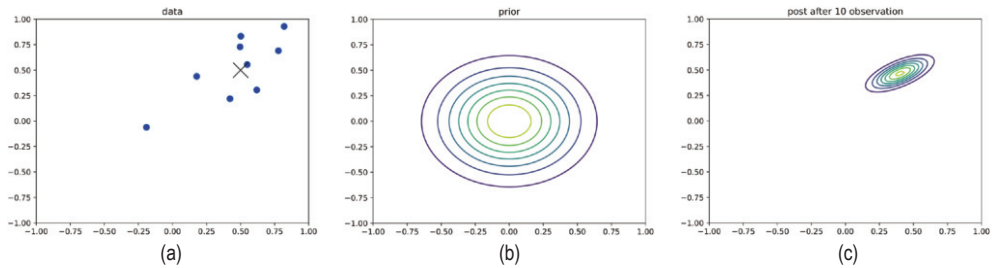


Рис. 3.9 ❖ Байесовский вывод для двумерного гауссова случайного вектора \mathbf{z} . (а) Данные сгенерированы в соответствии с распределением $\mathbf{y}_n \sim \mathcal{N}(\mathbf{z}, \boldsymbol{\Sigma}_y)$, где $\mathbf{z} = [0.5, 0.5]^T$ и $\boldsymbol{\Sigma}_y = 0.1[2, 1; 1, 1]$. Предполагается, что ковариация шума в датчиках $\boldsymbol{\Sigma}_y$ известна, но \mathbf{z} неизвестно. Черные крестики представляют \mathbf{z} . (б) Априорное распределение имеет вид $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, 0.1\mathbf{I}_2)$. (с) Показано апостериорное распределение после наблюдения 10 точек. Построено программой по адресу figures.problm.ai/book1/3.9

Апостериорная неопределенность каждой компоненты вектора \mathbf{z} зависит от надежности показаний датчика при каждом измерении. В нашем примере шум измерений в направлении 1 больше, чем в направлении 2, поэтому апостериорная неопределенность z_1 (по горизонтальной оси) больше, чем z_2 (по вертикальной оси).

3.3.5. Пример: слияние показаний датчиков

В этом разделе мы обобщим результаты из раздела 3.3.4 на случай нескольких измерений от разных датчиков с различной надежностью. То есть модель имеет вид:

$$p(\mathbf{z}, \mathbf{y}) = p(\mathbf{z}) \prod_{m=1}^M \prod_{n=1}^{N_m} \mathcal{N}(\mathbf{y}_{n,m}|\mathbf{z}, \boldsymbol{\Sigma}_m), \quad (3.69)$$

где M – количество датчиков (измерительных устройств), N_m – количество наблюдений, произведенных m -м датчиком, а $\mathbf{y} = \mathbf{y}_{1:N;1:M} \in \mathbb{R}^K$. Наша цель – объединить имеющиеся факты и вычислить $p(\mathbf{z}|\mathbf{y})$. Это называется **слиянием показаний датчиков**.

Теперь приведем простой пример, когда имеется всего два датчика $y_1 \sim \mathcal{N}(\mathbf{z}, \Sigma_1)$ и $y_2 \sim \mathcal{N}(\mathbf{z}, \Sigma_2)$. Схематически этот пример можно представить в виде $\mathbf{y}_1 \leftarrow \mathbf{z} \rightarrow \mathbf{y}_2$. Мы можем объединить \mathbf{y}_1 и \mathbf{y}_2 в один вектор \mathbf{y} и представить модель в виде $\mathbf{z} \rightarrow [\mathbf{y}_1, \mathbf{y}_2]$, где $p(\mathbf{y}|\mathbf{z}) = \mathcal{N}(\mathbf{y}|\mathbf{W}\mathbf{z}, \Sigma_y)$, где $\mathbf{W} = [\mathbf{I}; \mathbf{I}]$ и $\Sigma_y = [\Sigma_1, 0; 0, \Sigma_2]$ – блочные матрицы. Затем можно применить формулу Байеса для гауссовых распределений и вычислить $p(\mathbf{y}|\mathbf{z})$.

На рис. 3.10а приведен пример для двумерного случая, где мы положили $\Sigma_1 = \Sigma_2 = 0.01\mathbf{I}_2$, т. е. оба датчика одинаково надежны. В этом случае апостериорное среднее расположено посередине между двумя наблюдениями, \mathbf{y}_1 и \mathbf{y}_2 . На рис. 3.10b мы положили $\Sigma_1 = 0.05\mathbf{I}_2$ и $\Sigma_2 = 0.01\mathbf{I}_2$, т. е. датчик 2 надежнее, чем датчик 1. В этом случае апостериорное среднее расположено ближе к \mathbf{y}_2 . На рис. 3.10(c) мы положили

$$\Sigma_1 = 0.01 \begin{pmatrix} 10 & 1 \\ 1 & 1 \end{pmatrix}, \quad \Sigma_2 = 0.01 \begin{pmatrix} 1 & 1 \\ 1 & 10 \end{pmatrix}, \quad (3.70)$$

т. е. датчик 1 дает более надежные измерения второй компоненты (в вертикальном направлении), а датчик 2 – первой (в горизонтальном направлении). В этом случае апостериорное среднее заимствует вертикальную компоненту \mathbf{y}_1 и горизонтальную компоненту \mathbf{y}_2 .

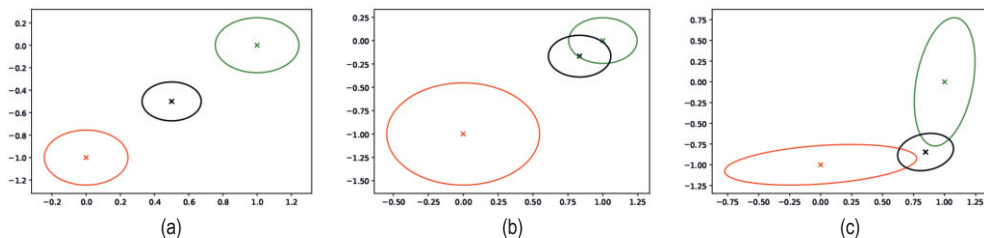


Рис. 3.10 ❖ Мы наблюдаем $\mathbf{y}_1 = (0, -1)$ (красный крестик) и $\mathbf{y}_2 = (1, 0)$ (зеленый крестик) и оцениваем $\mathbb{E}[\mathbf{z}|\mathbf{y}_1, \mathbf{y}_2]$ (черный крестик). (а) Датчики одинаково надежны, поэтому оценка апостериорного среднего расположена посередине между двумя окружностями. (б) Датчик 2 надежнее, поэтому оценка сдвинута ближе к зеленой окружности. (с) Датчик 1 надежнее в вертикальном направлении, а датчик 2 – в горизонтальном. Оценка является комбинацией обоих измерений. Построено программой по адресу figures.problml.ai/book1/3.10

3.4. ЭКСПОНЕНЦИАЛЬНОЕ СЕМЕЙСТВО РАСПРЕДЕЛЕНИЙ*

В этом разделе мы определим **экспоненциальное семейство распределений**, включающее много распространенных распределений вероятностей. Оно играет весьма важную роль в статистике и машинном обучении. В этой книге мы в основном используем его в контексте обобщенных линейных моделей, обсуждаемых в главе 12. Другие приложения экспоненциального семейства распределений мы встретим во втором томе.

3.4.1. Определение

Рассмотрим семейство распределений вероятностей с параметром $\boldsymbol{\eta} \in \mathbb{R}^K$ с фиксированным носителем $\mathcal{Y}^D \subseteq \mathbb{R}^D$. Говорят, что распределение $p(\mathbf{y}|\boldsymbol{\eta})$ является **экспоненциальным семейством распределений**, если его плотность можно записать в виде:

$$p(\mathbf{y}|\boldsymbol{\eta}) \triangleq \frac{1}{Z(\boldsymbol{\eta})} h(\mathbf{y}) \exp[\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{y})] = h(\boldsymbol{\eta}) \exp[\boldsymbol{\eta}^\top \mathcal{T}(\boldsymbol{\eta}) - A(\boldsymbol{\eta})], \quad (3.71)$$

где $h(\mathbf{y})$ – масштабная постоянная (ее также называют **базовой мерой**, и часто она равна 1), $\mathcal{T}(\mathbf{y}) \in \mathbb{R}^K$ – **достаточные статистики**, $\boldsymbol{\eta}$ – **естественные**, или **канонические, параметры**, $Z(\boldsymbol{\eta})$ – нормировочная постоянная, называемая также **функцией разбиения**, а $A(\boldsymbol{\eta}) = \log Z(\boldsymbol{\eta})$ – **логарифмическая функция разбиения**. Можно показать, что A – выпуклая функция, определенная на выпуклом множестве $\Omega \triangleq \{\boldsymbol{\eta} \in \mathbb{R}^K : A(\boldsymbol{\eta}) < 1\}$.

Удобно, когда естественные параметры независимы. Формально мы говорим, что экспоненциальное семейство **минимально**, если не существует такого $\boldsymbol{\eta} \in \mathbb{R}^K \setminus \{0\}$, что $\boldsymbol{\eta}^\top \mathcal{T}(\mathbf{y}) = 0$. Последнее условие может быть нарушено в случае мультиномиальных распределений из-за того, что сумма значений параметров должна быть равна 1; однако распределение легко перепараметризовать, взяв $K - 1$ независимых параметров, как показано ниже.

Формулу (3.71) можно обобщить, определив $\boldsymbol{\eta} = f(\boldsymbol{\phi})$, где $\boldsymbol{\phi}$ – некоторое, возможно меньшее, множество параметров. В этом случае распределение имеет вид:

$$p(\mathbf{y}|\boldsymbol{\phi}) = h(\mathbf{y}) \exp[f(\boldsymbol{\phi})^\top \mathcal{T}(\mathbf{y}) - A(f(\boldsymbol{\phi}))]. \quad (3.72)$$

Если отображение $\boldsymbol{\phi}$ в $\boldsymbol{\eta}$ нелинейно, то мы говорим об **искривленном экспоненциальном семействе**. Если $\boldsymbol{\eta} = f(\boldsymbol{\phi}) = \boldsymbol{\phi}$, то говорят, что модель имеет **каноническую форму**. Если, кроме того, $\mathcal{T}(\mathbf{y}) = \mathbf{y}$, то говорят, что это **естественное экспоненциальное семейство** (natural exponential family – NEF). В таком случае его можно записать в виде:

$$p(\mathbf{y}|\boldsymbol{\eta}) = h(\mathbf{y}) \exp[\boldsymbol{\eta}^\top \mathbf{y} - A(\boldsymbol{\eta})]. \quad (3.73)$$

3.4.2. Пример

В качестве простого примера рассмотрим распределение Бернулли. Его можно записать в виде экспоненциального семейства следующим образом:

$$\text{Ber}(y|\mu) = \mu^y(1 - \mu)^{1-y} \quad (3.74)$$

$$= \exp[y\log(\mu) + (1 - y)\log(1 - \mu)] \quad (3.75)$$

$$= \exp[\mathcal{T}(y)^\top \boldsymbol{\eta}], \quad (3.76)$$

где $\mathcal{T}(y) = [\mathbb{I}(y = 1), \mathbb{I}(y = 0)]$, $\boldsymbol{\eta} = [\log(\mu), \log(1 - \mu)]$, а μ – параметр, определяющий среднее. Однако это **сверхполное представление**, поскольку существует линейная зависимость между признаками. В этом можно убедиться следующим образом:

$$\mathbf{1}^\top \mathcal{T}(y) = \mathbb{I}(y = 0) + \mathbb{I}(y = 1) = 1. \quad (3.77)$$

Если представление сверхполное, то $\boldsymbol{\eta}$ определено неоднозначно. Обычно используют **минимальное представление**, когда с распределением ассоциировано единственное $\boldsymbol{\eta}$. В таком случае мы просто определяем:

$$\text{Ber}(y|\mu) = \exp\left[y\log\left(\frac{\mu}{1-\mu}\right) + \log(1 - \mu)\right]. \quad (3.78)$$

Это можно превратить в экспоненциальное семейство, определив

$$\boldsymbol{\eta} = \log\left(\frac{\mu}{1-\mu}\right); \quad (3.79)$$

$$\mathcal{T}(y) = y; \quad (3.80)$$

$$A(\boldsymbol{\eta}) = -\log(1 - \mu) = \log(1 + e^\eta); \quad (3.81)$$

$$h(y) = 1. \quad (3.82)$$

Мы можем восстановить параметр среднего μ по каноническому параметру η , положив

$$\mu = \sigma(\eta) = \frac{1}{1 + e^{-\eta}}. \quad (3.83)$$

В этой формуле мы узнаем логистическую (сигмоидную) функцию. Другие примеры см. во втором томе, [Mur22].

3.4.3. Логарифмическая функция разбиения является производящей функцией полуинвариантов

Первым и вторым **полуинвариантами** распределения являются его среднее $\mathbb{E}[Y]$ и дисперсия $\mathbb{V}[Y]$, а первым и вторым моментами – $\mathbb{E}[Y]$ и $\mathbb{E}[Y^2]$. Можно также вычислить полуинварианты (и моменты) высших порядков. Важное свойство экспоненциального семейства заключается в том, что производные логарифмической функции разбиения можно использовать для порождения всех полуинвариантов достаточных статистик. В частности, первый и второй полуинварианты равны соответственно

$$\nabla A(\boldsymbol{\eta}) = \mathbb{E}[\mathcal{T}(\mathbf{y})]; \quad (3.84)$$

$$\nabla^2 A(\boldsymbol{\eta}) = \text{Cov}[\mathcal{T}(\mathbf{y})]. \quad (3.85)$$

Из этого результата мы видим, что гессиан – положительно определенная матрица, а потому $A(\boldsymbol{\eta})$ – выпуклая относительно $\boldsymbol{\eta}$ функция. Поскольку логарифмическое правдоподобие имеет вид $\log p(\mathbf{y}|\boldsymbol{\eta}) = \boldsymbol{\eta}^T \mathcal{T}(\mathbf{y}) - A(\boldsymbol{\eta}) + \text{const}$, оно является выпуклой функцией и, следовательно, оценка максимального правдоподобия имеет глобальный максимум.

3.4.4. Вывод максимальной энтропии экспоненциального семейства

Пусть требуется найти распределение $p(\mathbf{x})$, описывающее данные, а знаем мы только математические ожидания (F_k) некоторых признаков, или функций $f_k(\mathbf{x})$:

$$\int d\mathbf{x} p(\mathbf{x}) f_k(\mathbf{x}) = F_k. \quad (3.86)$$

Например, f_1 могла бы вычислять x , а $f_2 = x^2$, так что F_1 – эмпирическое среднее, а F_2 – эмпирический второй момент. Априорно мы полагаем, что распределение описывается функцией $q(\mathbf{x})$.

Чтобы формализовать понятие «наименьшего числа допущений», будем искать распределение, которое максимально близко к априорному распределению $q(\mathbf{x})$ в смысле расхождения Кульбака–Лейблера (КЛ) (раздел 6.2) и удовлетворяет нашим ограничениям:

$$p = \text{argmin}_p \text{KL}(p||q) \text{ с ограничениями.} \quad (3.87)$$

Если используется равномерное априорное распределение $q(\mathbf{x}) \propto 1$, то минимизация расхождения КЛ эквивалентна максимизации энтропии (раздел 6.1):

$$p = \text{argmax}_p \mathbb{H}(p) \text{ с ограничениями.} \quad (3.88)$$

Результат называется **моделью максимальной энтропии**.

Для минимизации расхождения КЛ при ограничениях (3.86) и при условии, что $p(\mathbf{x}) \geq 0$ и $\sum_{\mathbf{x}} p(\mathbf{x}) = 1$, воспользуемся множителями Лагранжа (см. раздел 8.5.1). Лагранжиан равен:

$$J(p, \lambda) = -\sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} + \lambda_0 \left(1 - \sum_{\mathbf{x}} p(\mathbf{x})\right) + \sum_k \lambda_k \left(F_k - \sum_{\mathbf{x}} p(\mathbf{x}) f_k(\mathbf{x})\right). \quad (3.89)$$

Чтобы взять производные по функции p , можно было бы применить вариационное исчисление, но мы поступим проще и будем рассматривать p как вектор фиксированной длины (поскольку предполагается, что величина \mathbf{x} дискретная). Тогда имеем:

$$\frac{\partial J}{\partial p_c} = -1 - \log \frac{p(x=c)}{q(x=c)} - \lambda_0 - \sum_k \lambda_k f_k(x=c). \quad (3.90)$$

Приравнивая $\partial J / \partial p_c$ нулю для каждого c , получаем

$$p(\mathbf{x}) = \frac{q(\mathbf{x})}{Z} \exp\left(-\sum_k \lambda_k f_k(\mathbf{x})\right), \quad (3.91)$$

где по определению $Z \triangleq e^{1+\lambda_0}$. Вспоминая, что сумма вероятностей должна быть равна 1, получаем

$$1 = \sum_{\mathbf{x}} p(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{x}} q(\mathbf{x}) \exp\left(-\sum_k \lambda_k f_k(\mathbf{x})\right). \quad (3.92)$$

Поэтому нормировочная постоянная равна:

$$Z = \sum_{\mathbf{x}} q(\mathbf{x}) \exp\left(-\sum_k \lambda_k f_k(\mathbf{x})\right). \quad (3.93)$$

Это в точности форма экспоненциального семейства, где $\mathbf{f}(\mathbf{x})$ – вектор достаточных статистик, $-\lambda$ – естественные параметры, а $q(\mathbf{x})$ – базовая мера.

Например, если признаками являются $f_1(x) = x$ и $f_2(x) = x^2$ и мы ищем совпадения с первым и вторым моментами, то получается гауссово распределение.

3.5. СМЕСОВЫЕ МОДЕЛИ

Один из способов создания более сложных вероятностных моделей – взять выпуклую комбинацию простых распределений. Это называется **смесовой моделью**. Такая модель имеет вид

$$p(\mathbf{y}|\boldsymbol{\theta}) = \sum_{k=1}^K \pi_k p_k(\mathbf{y}), \quad (3.94)$$

где p_k – k -я компонента смеси, а π_k – веса компонент, удовлетворяющие условиям $0 \leq \pi_k \leq 1$ и $\sum_{k=1}^K \pi_k = 1$.

Эту модель можно по-другому выразить в виде иерархической модели, в которую мы введем дискретную **латентную переменную** $z \in \{1, \dots, K\}$, определяющую, какое распределение использовать для порождения выхода y . Априорное распределение этой латентной переменной имеет вид $p(z = k) = \pi_k$, а условное – $p(y|z = k) = p_k(y) = p(y|\theta_k)$. То есть мы определяем следующую совместную модель:

$$p(z|\theta) = \text{Cat}(z|\pi); \quad (3.95)$$

$$p(y|z = k, \theta) = p(y|\theta_k). \quad (3.96)$$

«История порождения» данных выглядит так: сначала делаем выборку из конкретной компоненты z , а затем порождаем наблюдения y , используя параметры, выбранные в зависимости от значения z . Исключая z , приходим к формуле (3.94):

$$p(y|\theta) = \sum_{k=1}^K p(z = k|\theta)p(y|z = k, \theta) = \sum_{k=1}^K \pi_k p(y|\theta_k). \quad (3.97)$$

Можно создавать различные виды смесовых моделей, варьируя базовое распределение p_k , как показано ниже.

3.5.1. Модель гауссовой смеси

Модель гауссовой смеси (Gaussian mixture model – **GMM**), называемая также **смесью гауссиан** (mixture of Gaussians – **MoG**), определена следующим образом:

$$p(y) = \sum_{k=1}^K \pi_k \mathcal{N}(y|\mu_k, \Sigma_k). \quad (3.98)$$

На рис. 3.11 показана плотность смеси трех двумерных гауссиан. Каждая компонента смеси представлена своим множеством эллиптических линий уровня. При достаточно большом числе компонент смеси GMM может аппроксимировать любое гладкое распределение на \mathbb{R}^D .

GMM часто используются для кластеризации без учителя вещественных выборочных данных $y_n \in \mathbb{R}^D$. Делается это в два этапа. Сначала мы обучаем модель, например вычисляя MLE $\hat{\theta} = \arg\max \log p(\mathcal{D}|\theta)$, где $\mathcal{D} = \{y_n : n = 1 \dots N\}$. (Мы обсудим, как вычисляется эта MLE в разделе 8.7.3.) Затем ассоциируем с каждой точкой y_n дискретную латентную (скрытую) переменную $z_n \in \{1, \dots, K\}$, которая задает номер компоненты смеси или кластер, который был использован для порождения y_n . Эти латентные переменные неизвестны, но мы можем вычислить их апостериорное распределение по формуле Байеса:

$$r_{nk} \triangleq p(z_n = k | \mathbf{x}_n, \boldsymbol{\theta}) = \frac{p(z_n = k | \boldsymbol{\theta}) p(\mathbf{x}_n | z_n = k, \boldsymbol{\theta})}{\sum_{k'=1}^K p(z_n = k' | \boldsymbol{\theta}) p(\mathbf{x}_n | z_n = k', \boldsymbol{\theta})}. \quad (3.99)$$

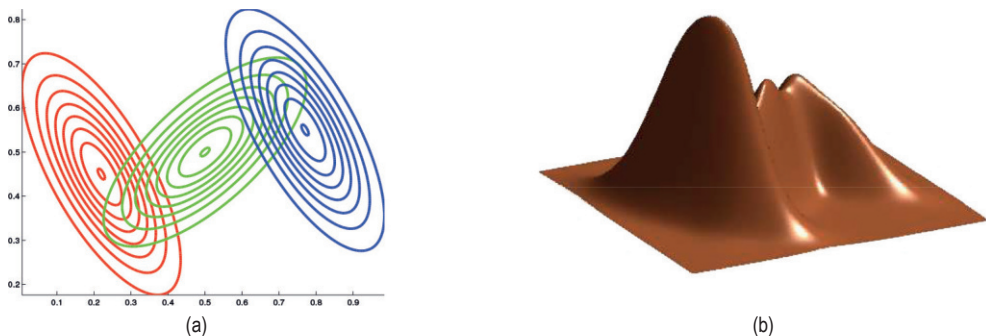


Рис. 3.11 ❖ Смесь трех двумерных гауссовых распределений. (а) Показаны линии постоянной вероятности для каждой компоненты смеси. (б) Поверхность суммарной плотности. На основе рис. 2.23 из работы [Bis06]. Построено программой по адресу figures.problml.ai/book1/3.11

Величина r_{nk} называется **ответственностью** кластера k за точку данных n . Зная ответственности, мы можем вычислить самый вероятный кластер:

$$\hat{z}_n = \underset{k}{\operatorname{argmax}} r_{nk} = \underset{k}{\operatorname{argmax}} [\log p(\mathbf{x}_n | z_n = k, \boldsymbol{\theta}) + \log p(z_n = k | \boldsymbol{\theta})]. \quad (3.100)$$

Это называется **жесткой кластеризацией**. (Если ответственности используются для частичного отнесения каждой точки к разным кластерам, то говорят о **мягкой кластеризации**.) Пример показан на рис. 3.12.

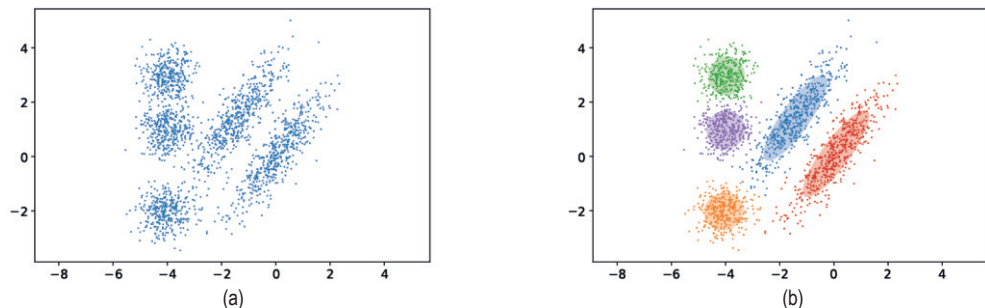


Рис. 3.12 ❖ (а) Двумерные данные. (б) Возможная кластеризация с $K = 5$ кластерами, вычисленная с помощью GMM. Построено программой по адресу figures.problml.ai/book1/3.12

Если имеется равномерное априорное распределение z_n и используются сферические гауссовы распределения с $\boldsymbol{\Sigma}_k = \mathbf{I}$, то задача жесткой кластеризации сводится к

$$z_n = \underset{k}{\operatorname{argmin}} \|y_n - \hat{\mu}_k\|_2^2. \quad (3.101)$$

Иными словами, мы сопоставляем каждой точке ближайший к ней (в смысле евклидова расстояния) центроид. Эта идея лежит в основе **кластеризации методом К средних**, который мы обсудим в разделе 21.3.

3.5.2. Модели бернуллиевой смеси

Если данные двоичные, то можно использовать **модель бернуллиевой смеси**, или **ВММ** (ее также называют **смесью распределений Бернулли**), каждая компонента которой имеет вид:

$$p(y|z = k, \theta) = \prod_{d=1}^D \operatorname{Ber}(y_d | \mu_{dk}) = \prod_{d=1}^D \mu_{dk}^{y_d} (1 - \mu_{dk})^{1-y_d}. \quad (3.102)$$

Здесь μ_{dk} – вероятность, что бит d поднят в кластере k .

В качестве примера аппроксимируем смесью ВММ с $K = 20$ компонентами набор данных MNIST (раздел 3.5.2). (Для обучения модели применялся ЕМ-алгоритм, похожий на ЕМ-алгоритм для GMM, рассматриваемый в разделе 8.7.3, однако можно также использовать стохастический градиентный спуск, более эффективный для больших наборов данных¹.) Получившиеся параметры каждой компоненты смеси (т. е. μ_k и π_k) показаны на рис. 3.13. Мы видим, что модель «обнаружила» представление каждой цифры. (Некоторые цифры представлены несколько раз, потому что модель не знала «истинное» количество классов. О том, как выбирать K для смесовых моделей, см. раздел 21.3.7.)

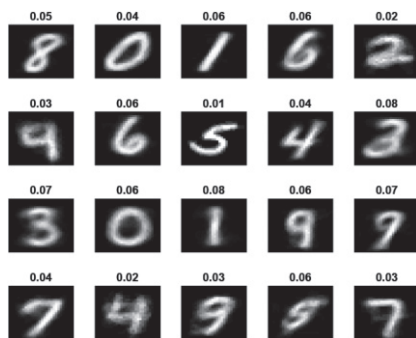


Рис. 3.13 ❖ Мы аппроксимируем набор бинаризованных данных о рукописных цифрах MNIST смесью 20 распределений Бернулли. Показаны оценки средних кластеров $\hat{\mu}_k$. Числа над каждым изображением представляют оценки весов в смеси $\hat{\pi}_k$. При обучении модели никакие метки не использовались. Построено программой по адресу figures.probl.ai/book1/3.13

¹ Код с использованием CGF см. по адресу code.probl.ai/book1/mix_bernoulli_sgd_mnist.

3.6. ГРАФОВЫЕ ВЕРОЯТНОСТНЫЕ МОДЕЛИ*

Мне известно два основных принципа упрощения сложных систем: принцип модульности и принцип абстрагирования. Я приверженец применения вычисления вероятностей в машинном обучении, потому что верю, что теория вероятностей реализует оба принципа глубокими и неожиданными способами, а именно с помощью факторизации и усреднения. Максимально полное использование этих двух механизмов представляется мне дорогой вперед в машинном обучении.

— Майкл Джордан, 1997 (цитируется по изданию [Fre98])

Мы познакомились с несколькими простыми строительными блоками, основанными на теории вероятностей. В разделе 3.3 мы продемонстрировали, как построить многомерное гауссово распределение $p(\mathbf{y})$ из более простых частей: маргинального распределения $p(\mathbf{y}_1)$ и условного распределения $p(\mathbf{y}_2|\mathbf{y}_1)$. Эту идею можно обобщить и определить совместные распределения многих случайных величин. Ключевое предположение будет заключаться в том, что некоторые величины **условно независимы** от других. Мы будем представлять предположения об условной независимости (CI) в виде графов, как описано ниже. (Дополнительные сведения см. во втором томе этой книги, [Mur22].)

3.6.1. Представление

Графовая вероятностная модель (probabilistic graphical model – **PGM**) – это совместное распределение вероятностей, представленное в виде графа, в котором закодированы предположения об условной независимости. Если граф **ориентированный** и **ациклический** (directed acyclic graph – **DAG**), то модель иногда называют **байесовской сетью**, хотя ничего специфически байесовского в таких моделях нет.

Основная идея PGM заключается в том, что каждая вершина графа представляет случайную величину, а каждое ребро – прямую зависимость. Точнее, отсутствие ребра означает, что величины условно независимы. В случае DAG мы можем занумеровать вершины в **топологическом порядке** (родители предшествуют потомкам), а затем соединить их так, что каждая вершина будет условно независима от всех своих предшественников при условии своих родителей:

$$Y_i \perp \mathbf{Y}_{\text{pred}(i) \setminus \text{pa}(i)} | \mathbf{Y}_{\text{pa}(i)}, \quad (3.103)$$

где $\text{pa}(i)$ – родители вершины i , а $\text{pred}(i)$ – предшественники вершины i в данном порядке (это называется **упорядоченным марковским свойством**). Следовательно, совместное распределение можно представить следующим образом:

$$p(\mathbf{Y}_{1:V}) = \prod_{i=1}^V p(Y_i | \mathbf{Y}_{\text{pa}(i)}), \quad (3.104)$$

где V – число вершин графа.

3.6.1.1. Пример: оросительная система

Пусть требуется смоделировать зависимости между 4 случайными величинами: C (сезон ненастный или нет), R (идет ли дождь), S (включен ли ороситель) и W (трава мокрая или нет). Мы знаем, что если сезон ненастный, то дождь более вероятен, поэтому добавляем ребро $C \rightarrow R$. Известно также, что в ненастный сезон включать оросители приходится реже, поэтому добавляем ребро $C \rightarrow S$. Наконец, мы знаем, что когда идет дождь или работают оросители, трава мокрая, поэтому добавляем ребра $S \rightarrow W$ и $R \rightarrow W$.

Формально эти предположения определяют следующее совместное распределение:

$$p(C, S, R, W) = p(C)p(S|C)p(R|C, S)p(W|S, R, \cancel{C}), \quad (3.105)$$

где вычеркнуты члены, ненужные в силу свойств условной независимости.

Каждый член $p(Y_i | \mathbf{Y}_{\text{pa}(i)})$ называется **условным распределением вероятностей** (conditional probability distribution – **CPD**) для вершины i . Распределение может быть любым. На рис. 3.14 предполагается, что все CPD – условные категориальные распределения, которые можно представить **таблицей условных вероятностей** (conditional probability table – **CPT**). Мы можем представить i -ю CPT следующим образом:

$$\theta_{ijk} \triangleq p(Y_i = k | \mathbf{Y}_{\text{pa}(i)} = j). \quad (3.106)$$

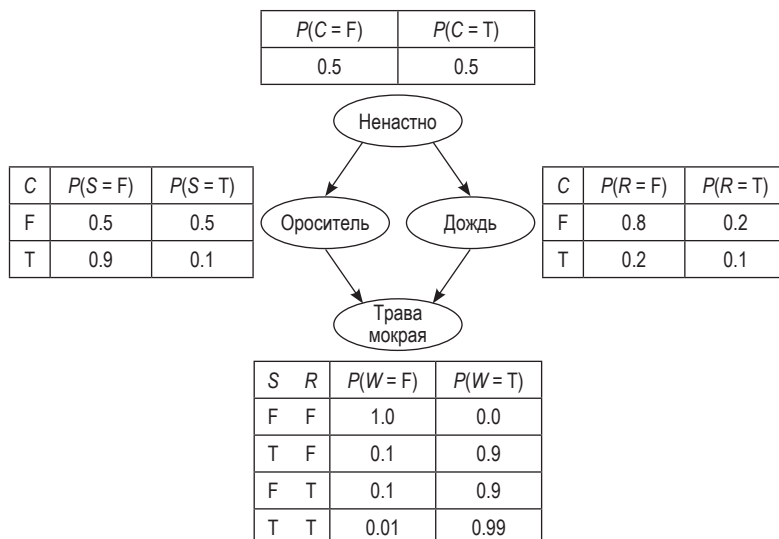


Рис. 3.14 ❖ Графовая вероятностная модель оросительной системы с бинарными таблицами условных вероятностей. Буквы Т и F означают true (истина) и false (ложь)

Такое представление обладает свойствами $0 \leq \theta_{ijk} \leq 1$ и $\sum_{k=1}^{K_i} \theta_{ijk} = 1$ для каждой строки j . Здесь i – индекс вершины, $i \in [V]$; k – индекс состояния вер-

шины, $k \in [K_i]$, где K_i – число состояний вершины i ; j – индекс совместного состояния родителей, $j \in [J_i]$, где $J_i = \prod_{p \in \text{pa}(i)} K_p$. Например, вершина «травя мокрая» имеет двух бинарных родителей, поэтому всего имеется четыре совместных состояния родителей.

3.6.1.2. Пример: марковская цепь

Пусть требуется создать совместное распределение вероятностей последовательностей переменной длины, $p(y_{1:T})$. Если каждая величина y_t представляет слово из словаря, содержащего K возможных значений, т. е. $y_t \in \{1, \dots, K\}$, то результирующая модель представляет распределение возможных последовательностей длины T ; часто ее называют **языковой моделью**.

Согласно цепному правилу, любое совместное распределение T случайных величин можно представить следующим образом:

$$p(y_{1:T}) = p(y_1)p(y_2|y_1)p(y_3|y_2, y_1)p(y_4|y_3, y_2, y_1)\dots = \prod_{t=1}^T p(y_t|y_{1:t-1}). \quad (3.107)$$

К сожалению, число параметров, необходимых для представления каждого условного распределения $p(y_t|y_{1:t-1})$ растет экспоненциально с ростом t . Но допустим, что имеет место предположение условной независимости, т. е. будущее $y_{t+1:T}$ не зависит от прошлого, $y_{1:t-1}$, при условии настоящего y_t . Оно называется **марковским условием первого порядка** и представляется PGM на рис. 3.15а. В этом предположении совместное распределение можно записать так:

$$p(y_{1:T}) = p(y_1)p(y_2|y_1)p(y_3|y_2)p(y_4|y_3)\dots = p(y_1)\prod_{t=2}^T p(y_t|y_{t-1}). \quad (3.108)$$



Рис. 3.15 ❖ Авторегрессионные модели (марковские) первого и второго порядка

Это называется **марковской цепью**, **марковской моделью** или **авторегрессионной моделью** первого порядка. Функция $p(y_t|y_{t-1})$ называется **переходной функцией**, **переходным ядром** или **марковским ядром**. Это просто условное распределение состояний в момент t при условии состояния в момент $t-1$, а потому для нее справедливы соотношения $p(y_t|y_{t-1}) \geq 0$ и $\sum_{k=1}^K p(y_t = k|y_{t-1} = j) = 1$. Мы можем представить эту СРТ в виде **стохастической матрицы**, $A_{jk} = p(y_t = k|y_{t-1} = j)$, в которой сумма элементов в каждой строке равна 1. Эта матрица называется **матрицей переходных вероятностей**. Мы предполагаем, что она одинакова для всех временных шагов, и соответствующая модель называется **однородной**, **стационарной** или **не зависящей от времени**. Это пример **связывания параметров**, потому что один и тот же параметр разделяется несколькими случайными величинами.

Это предположение позволяет моделировать произвольное число случайных величин с использованием фиксированного числа параметров.

Марковское предположение первого порядка довольно сильное. К счастью, модели первого порядка легко обобщить, так что они будут зависеть от последних M наблюдений, и таким образом создать модель порядка M (сколь угодно большого):

$$p(y_{1:T}) = p(y_{1:M}) \prod_{t=M+1}^T p(y_t | y_{t-M:t-1}). \quad (3.109)$$

Это называется **марковской моделью порядка M** . Например, если $M = 2$, то y_t зависит от y_{t-1} и y_{t-2} , как показано на рис. 3.15b. Это называется **триграммной моделью**, потому что она моделирует распределение троек слов. При $M = 1$ мы получаем **биграммную модель**, которая моделирует распределение пар слов.

Для словарей большого размера число параметров, необходимых для оценки условных распределений в M -граммных моделях с большим M , может оказаться непомерно большим. В таком случае нужно вводить дополнительные предположения вдобавок к условной независимости – например, что $p(y_t | y_{t-M:t-1})$ можно представить матрицей низкого ранга или какой-то нейронной сетью. Это называется **нейронной языковой моделью**. Подробности см. в главе 15.

3.6.2. Вывод

PGM определяет совместное распределение вероятностей. Поэтому можно использовать правила маргинализации и обусловливания для вычисления $p(Y_i | Y_j = y_j)$ для любых переменных i и j . Эффективные алгоритмы такого вычисления обсуждаются во втором томе этой книги.

Например, рассмотрим пример оросительной системы на рис. 3.14. Априорно мы полагаем, что вероятность дождя равна $p(R = 1) = 0.5$. Если мы видим, что трава мокрая, то апостериорная вероятность, что был дождь, становится равной $p(R = 1 | W = 1) = 0.7079$. Допустим, мы еще заметили, что ороситель был включен, тогда вероятность, что был дождь, уменьшается до $p(R = 1 | W = 1, S = 1) = 0.3204$. Это отрицательное взаимное влияние нескольких причин наблюдаемых результатов называется **оправданием** и известно также как **парадокс Берксона** (см. код по адресу code.problm.ai/book1/sprinkler_pgm, который воспроизводит эти вычисления).

3.6.3. Обучение

Если параметры CPD неизвестны, то мы можем рассматривать их как дополнительные случайные величины, добавить соответствующие им вершины в граф, а затем рассматривать как подлежащие выводу **скрытые переменные**. На рис. 3.16а приведен простой пример, в котором имеется N независимых и одинаково распределенных случайных величин y_n , выбранных из одного и того же распределения с параметром θ . (Закрашенные вершины

представляют наблюдаемые значения, а незакрашенные – латентные переменные, или параметры.)

Точнее, модель описывает следующую «историю порождения» данных:

$$\theta \sim p(\theta); \quad (3.110)$$

$$y_n \sim p(y|\theta), \quad (3.111)$$

где $p(\theta)$ – некоторое (неуказанное) априорное распределение параметров, а $p(y|\theta)$ – некоторая заданная функция правдоподобия. Соответствующее совместное распределение имеет вид:

$$p(\mathcal{D}, \theta) = p(\theta)p(\mathcal{D}|\theta), \quad (3.112)$$

где $\mathcal{D} = (y_1, \dots, y_N)$. В силу предположения о независимости и одинаковом распределении правдоподобие можно записать следующим образом:

$$p(\mathcal{D}|\theta) = \prod_{n=1}^N p(y_n|\theta). \quad (3.113)$$

Заметим, что порядок векторов данных для определения этой модели неважен, т. е. можно изменить нумерацию листовых узлов в PGM. Если это свойство имеет место, то данные называются **взаимозаменяемыми**.

3.6.3.1. Блочная нотация

На рис. 3.16а мы видим, что вершины y повторяются N раз. Чтобы избежать загромождения рисунка, часто используется **синтаксический сахар**, называемый **блоками** (plate). Заключается он в том, что мы обводим повторяющиеся переменные прямоугольником, понимая, что при разворачивании модели вершины внутри этого прямоугольника будут повторяться. А в правом нижнем углу блока записывается число повторений. Пример приведен на рис. 3.16б. Эта нотация широко используется для изображения некоторых видов байесовских моделей.

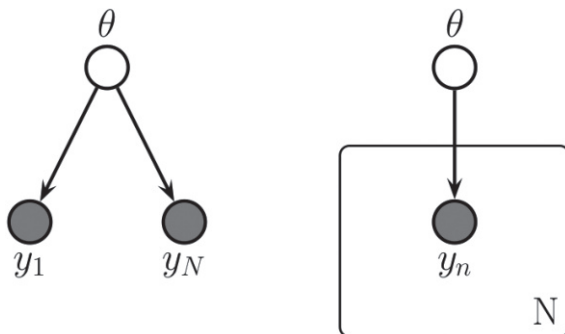


Рис. 3.16 ❖ Слева: точки y_n условно независимы при условии θ . Справа: та же модель в блочной нотации. Представлена та же модель, что слева, но повторяющиеся вершины y_n помещены в блок, а число N в правом нижнем углу блока равно количеству повторений вершины y_n

На рис. 3.17 показан более интересный пример, где GMM (раздел 3.5.1) представлена в виде графовой модели, которая кодирует совместное распределение:

$$p(\mathbf{y}_{1:N}, \mathbf{z}_{1:N}, \boldsymbol{\theta}) = p(\boldsymbol{\pi}) \left[\prod_{k=1}^K p(\boldsymbol{\mu}_k) p(\boldsymbol{\Sigma}_k) \right] \left[\prod_{n=1}^N p(z_n | \boldsymbol{\pi}) p(\mathbf{y}_n | z_n, \boldsymbol{\mu}_{1:K}, \boldsymbol{\Sigma}_{1:K}) \right]. \quad (3.114)$$

Мы видим, что латентные переменные z_n , а равно неизвестные параметры $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{\mu}_{1:K}, \boldsymbol{\Sigma}_{1:K})$, изображены незакрашенными вершинами.

3.7. УПРАЖНЕНИЯ

Упражнение 3.1 [некоррелированность не означает независимости*].

Пусть $X \sim U(-1, 1)$ и $Y = X^2$. Очевидно, что Y зависит от X (более того, Y однозначно определено X). Покажите, что тем не менее $\rho(X, Y) = 0$. *Указание:* если $X \sim U(a, b)$, то $\mathbb{E}[X] = (a + b)/2$ и $\mathbb{V}[X] = (b - a)^2/12$.

Упражнение 3.2 [коэффициент корреляции – число от -1 до $+1$].

Докажите, что $-1 \leq \rho(X, Y) \leq 1$.

Упражнение 3.3 [коэффициент корреляции линейно связанных величин равен ± 1 *].

Покажите, что если $Y = aX + b$ для некоторых $a > 0$ и b , то $\rho(X, Y) = 1$. Покажите также, что если $a < 0$, то $\rho(X, Y) = -1$.

Упражнение 3.4 [линейные комбинации случайных величин].

Пусть \mathbf{x} – случайный вектор со средним \mathbf{m} и ковариационной матрицей $\boldsymbol{\Sigma}$. Пусть \mathbf{A} и \mathbf{B} – матрицы.

- Выведите ковариационную матрицу $\mathbf{A}\mathbf{x}$.
- Покажите, что $\text{tr}(\mathbf{A}\mathbf{B}) = \text{tr}(\mathbf{B}\mathbf{A})$.
- Выведите выражение для $\mathbb{E}[\mathbf{x}^T \mathbf{A} \mathbf{x}]$.

Упражнение 3.5 [гауссово и совместное гауссово распределение].

Пусть $X \sim \mathcal{N}(0, 1)$ и $Y = WX$, где $p(W = -1) = p(W = 1) = 0.5$. Ясно, что X и Y не являются независимыми, поскольку Y – функция от X .

- Покажите, что $Y \sim \mathcal{N}(0, 1)$.
- Покажите, что $\text{Cov}[X, Y] = 0$. Таким образом, X и Y некоррелированы, но зависимы, хотя и являются гауссовыми.

Указание: воспользуйтесь определением ковариации

$$\text{Cov}[X, Y] = \mathbb{E}[XY] - \mathbb{E}[X] \mathbb{E}[Y] \quad (3.115)$$

и правилом **повторного математического ожидания**

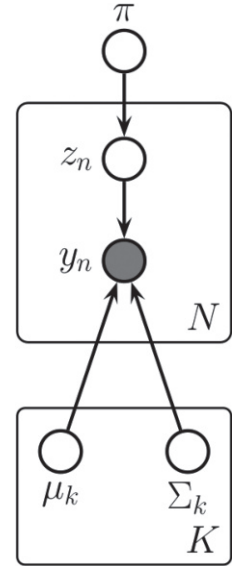


Рис. 3.17 ❖ Модель гауссовой смеси, представленная графовой моделью

$$\mathbb{E}[XY] = \mathbb{E}[\mathbb{E}[XY|W]]. \quad (3.116)$$

Упражнение 3.6 [нормировочная постоянная для многомерного гауссова распределения].

Докажите, что нормировочная постоянная для d -мерного гауссова распределения равна:

$$(2\pi)^{d/2} |\Sigma|^{-1/2} = \int \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1}(\mathbf{x} - \mu)\right) d\mathbf{x}. \quad (3.117)$$

Указание: диагонализуйте Σ и воспользуйтесь тем фактом, что $|\Sigma| = \prod_i \lambda_i$, чтобы записать функцию плотности совместного распределения в виде произведения одномерных гауссиан в преобразованной системе координат. (Нужно будет применить формулу замены переменных.) Наконец, используйте нормировочную постоянную для одномерных гауссовых распределений.

Упражнение 3.7 [слияние датчиков с известными дисперсиями в одномерном случае].

Имеется два датчика с известными (и различными) дисперсиями v_1 и v_2 , но с неизвестным (и одинаковым) средним μ . Предположим, что имеется n_1 наблюдений $y_i^{(1)} \sim \mathcal{N}(\mu, v_1)$ первого датчика и n_2 наблюдений $y_i^{(2)} \sim \mathcal{N}(\mu, v_2)$ второго датчика. (Например, μ – истинная температура на улице, и датчик 1 – точное (с низкой дисперсией) цифровое устройство измерения температуры, а датчик 2 – неточный (с высокой дисперсией) ртутный термометр.) Пусть \mathcal{D} представляет все данные с обоих датчиков. Каково апостериорное распределение $p(\mu|\mathcal{D})$ в предположении, что априорное распределение μ неинформативно (это можно смоделировать гауссовым распределением с точностью 0)? Приведите явное выражение для среднего и дисперсии апостериорного распределения.

Упражнение 3.8 [представление распределения Стьюдента в виде бесконечной смеси гауссовых распределений].

Покажите, что распределение Стьюдента можно записать в виде:

$$p(x|\mu, a, b) = \int_0^\infty \mathcal{N}(x|\mu, \alpha^{-1}) \text{Ga}(\alpha|a, b) d\alpha. \quad (3.118)$$

где Ga – гамма-распределение точности α . Это можно рассматривать как бесконечную **смесь гауссовых распределений** с разными точностями.

Глава 4

Статистика

4.1. ВВЕДЕНИЕ

В главах 2 и 3 мы предполагали, что все параметры θ наших вероятностных моделей известны. В этой главе мы обсудим, как обучить эти параметры на основе данных.

Процесс оценивания θ по \mathcal{D} называется **обучением модели** и составляет содержание машинного обучения. Есть много методов получения таких оценок, но большинство сводится к задаче оптимизации вида

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} L(\theta), \quad (4.1)$$

где $L(\theta)$ – функция потерь, или целевая функция. В этой главе мы обсудим несколько функций потерь. Иногда возможно решение задачи оптимизации в замкнутой форме. Но в общем случае необходим какой-то общий алгоритм оптимизации, это тема главы 8.

Помимо вычисления **точечной оценки** $\hat{\theta}$, мы обсудим, как моделировать неопределенность или достоверность этой оценки. В статистике процесс количественного оценивания неопределенности оценки неизвестной величины по конечной выборке данных называется **выводом**. Мы обсудим байесовский и частотный подходы к выводу¹.

4.2. ОЦЕНКА МАКСИМАЛЬНОГО ПРАВДОПОДОБИЯ (MLE)

Самый распространенный подход к оцениванию параметров – выбрать такие параметры, при которых обучающим данным назначается наибольшая веро-

¹ В глубоком обучении термин «вывод» означает то, что мы называем «предсказанием», т. е. вычисление $p(y|x, \hat{\theta})$.

ятность; это называется **оценкой максимального правдоподобия** (maximum likelihood estimation – **MLE**). Сначала обсудим это понятие подробнее, а затем приведем ряд примеров.

4.2.1. Определение

MLE определяется следующим образом:

$$\hat{\theta}_{\text{mle}} \triangleq \underset{\theta}{\operatorname{argmin}} p(\mathcal{D}|\theta). \quad (4.2)$$

Обычно предполагается, что обучающие примеры независимо выбраны из одного и того же распределения, так что (условное) правдоподобие принимает вид:

$$p(\mathcal{D}|\theta) = \prod_{n=1}^N p(\mathbf{y}_n|\mathbf{x}_n, \theta). \quad (4.3)$$

Это предположение о **независимости и одинаковом распределении** в англоязычной литературе называется iid (independent and identically distributed). Обычно мы имеем дело с **логарифмическим правдоподобием**:

$$LL(\theta) \triangleq \log p(\mathcal{D}|\theta) = \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \theta). \quad (4.4)$$

Оно представляется в виде суммы членов, по одному на каждый пример. Таким образом, MLE принимает вид:

$$\hat{\theta}_{\text{mle}} = \underset{\theta}{\operatorname{argmax}} \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \theta). \quad (4.5)$$

Поскольку большинство алгоритмов оптимизации (в частности, те, что обсуждаются в главе 8) предназначены для **минимизации** функции стоимости, мы можем переопределить **целевую функцию**, выбрав в этом качестве (условное) **отрицательное логарифмическое правдоподобие** (negative log likelihood – **NLL**):

$$\text{NLL}(\theta) \triangleq -\log p(\mathcal{D}|\theta) = -\sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \theta). \quad (4.6)$$

Минимизация этого выражения дает MLE. Если модель безусловная (без учителя), то MLE принимает вид

$$\hat{\theta}_{\text{mle}} = \underset{\theta}{\operatorname{argmin}} -\sum_{n=1}^N \log p(\mathbf{y}_n|\theta), \quad (4.7)$$

так как мы имеем выходы \mathbf{y}_n , но не имеем входов \mathbf{x}_n ¹.

Но бывает так, что нужно максимизировать совместное правдоподобие входов и выходов. Тогда MLE принимает вид:

$$\hat{\boldsymbol{\theta}}_{\text{mle}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} - \sum_{n=1}^N \log p(\mathbf{y}_n, \mathbf{x}_n | \boldsymbol{\theta}). \quad (4.8)$$

4.2.2. Обоснование MLE

Существует несколько обоснований MLE. Одно из них – рассматривать MLE как простую точечную аппроксимацию байесовского апостериорного распределения $p(\boldsymbol{\theta} | \mathcal{D})$, когда априорное распределение равномерное (см. раздел 4.6.7.1).

В частности, предположим, что апостериорное распределение аппроксимируется дельта-функцией $p(\boldsymbol{\theta} | \mathcal{D}) = \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{map}})$, где $\hat{\boldsymbol{\theta}}_{\text{map}}$ – апостериорная мода:

$$\hat{\boldsymbol{\theta}}_{\text{map}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log p(\boldsymbol{\theta} | \mathcal{D}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log p(\mathcal{D} | \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}). \quad (4.9)$$

Если мы пользуемся равномерным априорным распределением $p(\boldsymbol{\theta}) \propto 1$, то оценка MAP оказывается равной MLE, $\hat{\boldsymbol{\theta}}_{\text{map}} = \hat{\boldsymbol{\theta}}_{\text{mle}}$.

Другое обоснование MLE заключается в том, что результирующее прогнозное распределение $p(\mathbf{y} | \hat{\boldsymbol{\theta}}_{\text{mle}})$ максимально близко (в определенном ниже смысле) к **эмпирическому распределению** данных. В безусловном случае эмпирическое распределение определяется так:

$$p_{\mathcal{D}}(\mathbf{y}) \triangleq \frac{1}{N} \sum_{n=1}^N \delta(\mathbf{y} - \mathbf{y}_n). \quad (4.10)$$

Мы видим, что эмпирическое распределение – это ряд дельта-функций, или «пиков», в наблюдаемых обучающих данных. Мы хотим создать модель, распределение которой $q(\mathbf{y}) = p(\mathbf{y} | \boldsymbol{\theta})$ похоже на $p_{\mathcal{D}}(\mathbf{y})$.

Стандартный способ измерения сходства распределений вероятностей p и q – **расхождение Кульбака–Лейблера** (КЛ). Детали мы приведем в разделе 6.2, а пока ограничимся определением:

$$\mathbb{KL}(p || q) = \sum_{\mathbf{y}} p(\mathbf{y}) \log \frac{p(\mathbf{y})}{q(\mathbf{y})} \quad (4.11)$$

¹ В статистике принято использовать букву \mathbf{y} для обозначения величин, для которых моделируется порождающее распределение, а букву \mathbf{x} для обозначения экзогенных данных, которые подаются на вход, но не порождаются. Поэтому предметом обучения с учителем является обучение условных моделей вида $p(\mathbf{y} | \mathbf{x})$, а обучение без учителя – это частный случай, когда $\mathbf{x} = \emptyset$, т. е. мы обучаем безусловное распределение $p(\mathbf{y})$. В литературе по МО при обучении с учителем \mathbf{y} рассматривается как сгенерированные данные, а \mathbf{x} – как заданные, но при обучении с учителем \mathbf{x} , напротив, обозначает сгенерированные переменные.

$$= \underbrace{\sum_{\mathbf{y}} p(\mathbf{y}) \log p(\mathbf{y})}_{-\mathbb{H}(p)} - \underbrace{\sum_{\mathbf{y}} p(\mathbf{y}) \log q(\mathbf{y})}_{\mathbb{H}(p,q)}, \quad (4.12)$$

где $\mathbb{H}(p)$ – энтропия p (см. раздел 6.1), а $\mathbb{H}(p, q)$ – перекрестная энтропия p и q (см. раздел 6.1.2). Можно показать, что $\mathbb{KL}(p||q) \geq 0$, причем равенство достигается тогда и только тогда, когда $p = q$.

Если определить $q(\mathbf{y}) = p(\mathbf{y}|\boldsymbol{\theta})$ и положить $p(\mathbf{y}) = p_{\mathcal{D}}(\mathbf{y})$, то расхождение КЛ принимает вид:

$$\mathbb{KL}(p||q) = \sum_{\mathbf{y}} p_{\mathcal{D}}(\mathbf{y}) \log p_{\mathcal{D}}(\mathbf{y}) - p_{\mathcal{D}}(\mathbf{y}) \log q(\mathbf{y}) \quad (4.13)$$

$$= -H(p_{\mathcal{D}}) - \frac{1}{N} \sum_{n=1}^N \log p(\mathbf{y}_n|\boldsymbol{\theta}) \quad (4.14)$$

$$= \text{const} + \text{NLL}(\boldsymbol{\theta}). \quad (4.15)$$

Здесь первый член – постоянная, которую можно отбросить, оставив только NLL. Следовательно, минимизация расхождения КЛ эквивалентна минимизации NLL, что эквивалентно вычислению MLE по формуле (4.7).

Приведенные выше результаты можно обобщить на обучение с учителем (условное), воспользовавшись следующим эмпирическим распределением:

$$p_{\mathcal{D}}(\mathbf{x}, \mathbf{y}) = p_{\mathcal{D}}(\mathbf{y}|\mathbf{x}) p_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^N \delta(\mathbf{x} - \mathbf{x}_n) \delta(\mathbf{y} - \mathbf{y}_n). \quad (4.16)$$

Тогда математическое ожидание расхождения КЛ равно:

$$\mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})}[\mathbb{KL}(p_{\mathcal{D}}(Y|\mathbf{x})||q(Y|\mathbf{x}))] = \sum_{\mathbf{x}} p_{\mathcal{D}}(\mathbf{x}) \left[\sum_{\mathbf{y}} p_{\mathcal{D}}(\mathbf{y}|\mathbf{x}) \log \frac{p_{\mathcal{D}}(\mathbf{y}|\mathbf{x})}{q(\mathbf{y}|\mathbf{x})} \right] \quad (4.17)$$

$$= \text{const} - \sum_{\mathbf{x}, \mathbf{y}} p_{\mathcal{D}}(\mathbf{x}, \mathbf{y}) \log q(\mathbf{y}|\mathbf{x}) \quad (4.18)$$

$$= \text{const} - \frac{1}{N} \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}). \quad (4.19)$$

Минимизация этого выражения эквивалентна минимизации условного NLL в формуле (4.6).

4.2.3. Пример: MLE для распределения Бернулли

Пусть Y – случайная величина, описывающая подбрасывание монеты: событие $Y = 1$ соответствует выпадению орла, а $Y = 0$ – выпадению решки. Обозначим $\theta = p(Y = 1)$ вероятность выпадения орла. Эта случайная величина имеет распределение Бернулли, с которым мы познакомились в разделе 2.4.

NLL для распределения Бернулли имеет вид:

$$\text{NLL}(\theta) = -\log \prod_{n=1}^N p(y_n|\theta) \quad (4.20)$$

$$= -\log \prod_{n=1}^N \theta^{\mathbb{I}(y_n=1)} (1-\theta)^{\mathbb{I}(y_n=0)} \quad (4.21)$$

$$= -\sum_{n=1}^N \mathbb{I}(y_n = 1) \log \theta + \mathbb{I}(y_n = 0) \log(1 - \theta) \quad (4.22)$$

$$= -[N_1 \log \theta + N_0 \log(1 - \theta)], \quad (4.23)$$

где $N_1 = \sum_{n=1}^N \mathbb{I}(y_n = 1)$ и $N_0 = \sum_{n=1}^N \mathbb{I}(y_n = 0)$ представляют число выпадений орла и решки соответственно. (NLL для биномиального распределения такое же, как для распределения Бернулли, с точностью до несущественного

постоянного члена $\binom{N}{c}$, не зависящего от θ .) Эти два числа называются **достаточными статистиками** данных, так как в них заключено все, что нам необходимо знать о \mathcal{D} . Сумма $N = N_0 + N_1$ называется **размером выборки**.

Для нахождения MLE нужно решить уравнение $\frac{d}{d\theta} \text{NLL}(\theta) = 0$. Производная NLL равна

$$\frac{d}{d\theta} \text{NLL}(\theta) = \frac{-N_1}{\theta} + \frac{N_0}{1-\theta}, \quad (4.24)$$

и значит, MLE равно:

$$\hat{\theta}_{\text{mle}} = \frac{N_1}{N_0 + N_1}. \quad (4.25)$$

Как видим, это просто эмпирическая доля орлов, что вполне согласуется с интуицией.

4.2.4. Пример: MLE для категориального распределения

Допустим, что мы бросаем K -гранную кость N раз. Обозначим $Y_n \in \{1, \dots, K\}$ исход n -го испытания, где $Y_n \sim \text{Cat}(\boldsymbol{\theta})$. Мы хотим оценить вероятности $\boldsymbol{\theta}$ на основе набора данных $\mathcal{D} = \{y_n : n = 1 \dots N\}$. NLL равно

$$\text{NLL}(\boldsymbol{\theta}) = -\sum_k N_k \log \theta_k, \quad (4.26)$$

где N_k – сколько раз наблюдалось событие $Y = k$. (NLL для мультиномиального распределения такое же с точностью до несущественных множителей.)

Для вычисления MLE необходимо минимизировать NLL при ограничении $\sum_{k=1}^K \theta_k = 1$. Для этого воспользуемся методом множителей Лагранжа (см. раздел 8.5.1)¹. Лагранжиан имеет вид:

$$\mathcal{L}(\boldsymbol{\theta}, \lambda) \triangleq -\sum_k N_k \log \theta_k - \lambda \left(1 - \sum_k \theta_k\right). \quad (4.27)$$

Взятие производных по λ дает исходное ограничение:

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 1 - \sum_k \theta_k = 0. \quad (4.28)$$

Взятие производных по θ_k дает

$$\frac{\partial \mathcal{L}}{\partial \theta_k} = -\frac{N_k}{\theta_k} + \lambda = 0 \Rightarrow N_k = \lambda \theta_k. \quad (4.29)$$

Это уравнение можно решить относительно λ , воспользовавшись тем фактом, что сумма вероятностей должна быть равна 1:

$$\sum_k N_k = N = \lambda \sum_k \theta_k = \lambda. \quad (4.30)$$

Таким образом, MLE равна

$$\hat{\theta}_k = \frac{N_k}{\lambda} = \frac{N_k}{N}, \quad (4.31)$$

а это не что иное, как эмпирическая доля событий k .

4.2.5. Пример: MLE для одномерного гауссова распределения

Пусть $Y \sim \mathcal{N}(\mu, \sigma^2)$ и $\mathcal{D} = \{y_n : n = 1 \dots N\}$ – независимая и одинаково распределенная выборка размера N . Параметры $\boldsymbol{\theta} = (\mu, \sigma^2)$ можно оценить с помощью MLE следующим образом. Сначала выведем NLL, равное

$$\text{NLL}(\mu, \sigma^2) = -\sum_{n=1}^N \log \left[\left(\frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp \left(-\frac{1}{2\sigma^2} (y_n - \mu)^2 \right) \right] \quad (4.32)$$

$$= \frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \mu)^2 + \frac{N}{2} \log(2\pi\sigma^2). \quad (4.33)$$

¹ Нет необходимости явно накладывать ограничение $\theta_k \geq 0$, потому что градиент лагранжиана имеет вид $-N_k/\theta_k - \lambda$, поэтому отрицательные значения θ_k увеличивали бы целевую функцию, а не уменьшали ее. (Разумеется, это не запрещает полагать $\theta_k = 0$, и на самом деле такое решение является оптимальным, если $N_k = 0$.)

Минимум этой функции должен удовлетворять следующим условиям, которые мы объясним в разделе 8.1.1.1:

$$\frac{\partial}{\partial \mu} \text{NLL}(\mu, \sigma^2) = 0, \quad \frac{\partial}{\partial \sigma^2} \text{NLL}(\mu, \sigma^2) = 0. \quad (4.34)$$

Так что нам нужно только найти эту стационарную точку. Простое вычисление (упражнение 4.1) показывает, что решение имеет вид:

$$\hat{\mu}_{\text{mle}} = \frac{1}{N} \sum_{n=1}^N y_n = \bar{y}; \quad (4.35)$$

$$\hat{\sigma}_{\text{mle}}^2 = \frac{1}{N} \sum_{n=1}^N (y_n - \hat{\mu}_{\text{mle}})^2 = \frac{1}{N} \left[\sum_{n=1}^N y_n^2 + \hat{\mu}_{\text{mle}}^2 - 2y_n \hat{\mu}_{\text{mle}} \right] = s^2 - \bar{y}^2; \quad (4.36)$$

$$s^2 \triangleq \frac{1}{N} \sum_{n=1}^N y_n^2. \quad (4.37)$$

Величины \bar{y} и s^2 называются **достаточными статистиками** данных, потому что их достаточно для вычисления MLE без потери информации по сравнению с самими исходными данными. Отметим, что вы, возможно, привыкли к такой записи оценки дисперсии:

$$\sigma_{\text{unb}}^2 = \frac{1}{N-1} \sum_{n=1}^N (y_n - \hat{\mu}_{\text{mle}})^2, \quad (4.38)$$

где производится деление не на N , а на $N-1$. Это не MLE, а другая оценка, которая, кстати, является несмещенной (в отличие от MLE); детали см. в разделе 4.7.6.1.

4.2.6. Пример: MLE для многомерного гауссова распределения

В этом разделе мы выведем оценку максимального правдоподобия для параметров многомерного гауссова распределения.

Сначала запишем логарифмическое правдоподобие, опустив несущественные постоянные:

$$LL(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \log p(\mathcal{D}|n, \boldsymbol{\Sigma}) = \frac{N}{2} \log |\boldsymbol{\Lambda}| - \frac{1}{2} \sum_{n=1}^N (\mathbf{y}_n - \boldsymbol{\mu})^\top \boldsymbol{\Lambda} (\mathbf{y}_n - \boldsymbol{\mu}), \quad (4.39)$$

где $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ – **матрица точности** (обратная ковариационная матрица).

4.2.6.1. MLE среднего

После подстановки $\mathbf{z}_n = \mathbf{y}_n - \boldsymbol{\mu}$ применения формулы производной квадратичной формы (7.264) и правила дифференцирования сложной функции имеем

$$\frac{\partial}{\partial \boldsymbol{\mu}} (\mathbf{y}_n - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{y}_n - \boldsymbol{\mu}) = \frac{\partial}{\partial \mathbf{z}_n} \mathbf{z}_n^\top \boldsymbol{\Sigma}^{-1} \mathbf{z}_n \frac{\partial \mathbf{z}_n}{\partial \boldsymbol{\mu}^\top} \quad (4.40)$$

$$= -1(\boldsymbol{\Sigma}^{-1} + \boldsymbol{\Sigma}^{-T}) \mathbf{z}_n, \quad (4.41)$$

так как $\partial \mathbf{z}_n / \partial \boldsymbol{\mu}^\top = -\mathbf{I}$. Отсюда

$$\frac{\partial}{\partial \mathbf{n}} LL(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{1}{2} \sum_{n=1}^N -2\boldsymbol{\Sigma}^{-1} (\mathbf{y}_n - \boldsymbol{\mu}) = \boldsymbol{\Sigma}^{-1} \sum_{n=1}^N (\mathbf{y}_n - \boldsymbol{\mu}) = 0; \quad (4.42)$$

$$\hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_{n=1}^N \mathbf{y}_n = \bar{\mathbf{y}}. \quad (4.43)$$

Таким образом, MLE $\boldsymbol{\mu}$ – это просто эмпирическое среднее.

4.2.6.2. MLE ковариационной матрицы

Мы можем воспользоваться свойством следа (формула (7.36)) и переписать логарифмическое правдоподобие в терминах матрицы точности $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ следующим образом:

$$LL(\hat{\boldsymbol{\mu}}, \boldsymbol{\Lambda}) = \frac{N}{2} \log |\boldsymbol{\Lambda}| - \frac{1}{2} \sum_n \text{tr}[(\mathbf{y}_n - \hat{\boldsymbol{\mu}})(\mathbf{y}_n - \hat{\boldsymbol{\mu}})^\top \boldsymbol{\Lambda}] \quad (4.44)$$

$$= \frac{N}{2} \log |\boldsymbol{\Lambda}| - \frac{1}{2} \text{tr}[\mathbf{S}_{\bar{\mathbf{y}}} \boldsymbol{\Lambda}] \quad (4.45)$$

$$\mathbf{S}_{\bar{\mathbf{y}}} \triangleq \sum_{n=1}^N (\mathbf{y}_n - \bar{\mathbf{y}})(\mathbf{y}_n - \bar{\mathbf{y}})^\top = \left(\sum_n \mathbf{y}_n \mathbf{y}_n^\top \right) - N \bar{\mathbf{y}} \bar{\mathbf{y}}^\top, \quad (4.46)$$

где $\mathbf{S}_{\bar{\mathbf{y}}}$ – **матрица рассеяния** с центром в $\bar{\mathbf{y}}$.

Матрицу рассеяния можно переписать более компактно:

$$\mathbf{S}_{\bar{\mathbf{y}}} = \tilde{\mathbf{Y}}^\top \tilde{\mathbf{Y}} = \mathbf{Y}^\top \mathbf{C}_N^\top \mathbf{C}_N \mathbf{Y} = \mathbf{Y}^\top \mathbf{C}_N \mathbf{Y}, \quad (4.47)$$

где

$$\mathbf{C}_N \triangleq \mathbf{I}_N - \frac{1}{N} \mathbf{1}_N \mathbf{1}_N^\top \quad (4.48)$$

– **центрирующая матрица**, которая преобразует \mathbf{Y} в $\tilde{\mathbf{Y}}$ путем вычитания среднего $\bar{\mathbf{y}} = (1/N) \mathbf{Y}^\top \mathbf{1}_N$ из каждой строки.

Пользуясь результатами из раздела 7.8, мы можем вычислить производные функции потерь по $\boldsymbol{\Lambda}$:

$$\frac{\partial LL(\hat{\boldsymbol{\mu}}, \boldsymbol{\Lambda})}{\partial \boldsymbol{\Lambda}} = \frac{N}{2} \boldsymbol{\Lambda}^{-T} - \frac{1}{2} \mathbf{S}_{\bar{\mathbf{y}}}^\top = 0; \quad (4.49)$$

$$\boldsymbol{\Lambda}^{-T} = \boldsymbol{\Lambda}^{-1} = \boldsymbol{\Sigma} = \frac{1}{N} \mathbf{S}_{\bar{\mathbf{y}}}; \quad (4.50)$$

$$\hat{\Sigma} = \frac{1}{N} \sum_{n=1}^N (y_n - \bar{y})(y_n - \bar{y})^T = \frac{1}{N} \mathbf{Y}^T \mathbf{C}_N \mathbf{Y}. \quad (4.51)$$

Таким образом, MLE ковариационной матрицы – это эмпирическая ковариационная матрица (см. пример на рис. 4.1a).

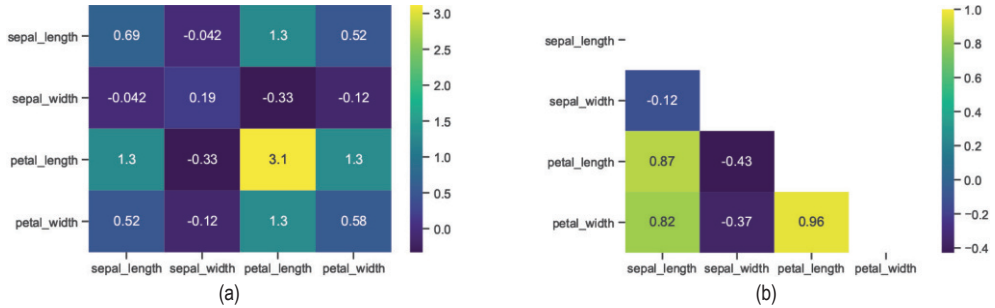


Рис. 4.1 ❖ (a) Ковариационная матрица для признаков в наборе данных об ирисах из раздела 1.2.1.1. (b) Корреляционная матрица. Мы показали только нижний треугольник, потому что матрица симметричная, а на диагонали находятся единицы. Сравните с рис. 1.3. Построено программой по адресу figures.problm.ai/book1/4.1

Иногда удобнее работать с корреляционной матрицей, определенной формулой (3.8). Вычислим ее:

$$\text{corr}(\mathbf{Y}) = (\text{diag}(\Sigma))^{-\frac{1}{2}} \Sigma (\text{diag}(\Sigma))^{-\frac{1}{2}}, \quad (4.52)$$

здесь $\text{diag}(\Sigma)^{-\frac{1}{2}}$ – диагональная матрица элементами $1/\sigma_i$ на диагонали (см. пример на рис. 4.1b).

Заметим, однако, что MLE может оказаться переобученной или численно неустойчивой, особенно когда число примеров N мало по сравнению с числом измерений D . Основная проблема заключается в том, что Σ содержит $O(D^2)$ параметров, поэтому для надежной оценки нужно очень много данных. В частности, как видно из формулы (4.51), MLE полной ковариационной матрицы сингулярна, если $N < D$. И даже если $N > D$, матрица MLE может быть плохо обусловлена, т. е. близка к сингулярной. Мы обсудим решение этой проблемы в разделе 4.5.2.

4.2.7. Пример: MLE для линейной регрессии

Мы кратко упоминали линейную регрессию в разделе 2.6.3. Напомним, что она соответствует следующей модели:

$$p(y|\mathbf{x}; \theta) = \mathcal{N}(y|\mathbf{w}^T \mathbf{x}, \sigma^2), \quad (4.53)$$

где $\theta = (\mathbf{w}, \sigma^2)$. Предположим пока, что σ^2 фиксировано, и займемся оценкой весов \mathbf{w} . Отрицательное логарифмическое правдоподобие (NLL) равно:

$$\text{NLL}(\mathbf{w}) = -\sum_{n=1}^N \log \left[\left(\frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp \left(-\frac{1}{2\sigma^2} (y_n - \mathbf{w}^\top \mathbf{x}_n)^2 \right) \right]. \quad (4.54)$$

Отбрасывая несущественные постоянные слагаемые, получаем следующую упрощенную целевую функцию, которая называется **суммой квадратов невязок** (residual sum of squares – **RSS**):

$$\text{RSS}(\mathbf{w}) \triangleq \sum_{n=1}^N (y_n - \mathbf{w}^\top \mathbf{x}_n)^2 = \sum_{n=1}^N r_n^2, \quad (4.55)$$

где r_n – n -я **невязка**. Умножение на число примеров N дает **среднеквадратическую ошибку** (СКО, англ. MSE):

$$\text{MSE}(\mathbf{w}) = \frac{1}{N} \text{RSS}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N (y_n - \mathbf{w}^\top \mathbf{x}_n)^2. \quad (4.56)$$

Наконец, извлечение квадратного корня дает **корень из среднеквадратической ошибки** (RMSE):

$$\text{RMSE}(\mathbf{w}) = \sqrt{\text{MSE}(\mathbf{w})} = \sqrt{\frac{1}{N} \sum_{n=1}^N (y_n - \mathbf{w}^\top \mathbf{x}_n)^2}. \quad (4.57)$$

Мы можем вычислить MLE, минимизировав NLL, RSS, MSE или RMSE. Во всех случаях результат будет один и тот же, поскольку целевая функция одинакова с точностью до несущественных постоянных.

Сосредоточимся на целевой функции RSS. В матричных обозначениях ее можно записать следующим образом:

$$\text{RSS}(\mathbf{w}) = \sum_{n=1}^N (y_n - \mathbf{w}^\top \mathbf{x}_n)^2 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 = (\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y}). \quad (4.58)$$

В разделе 11.2.2.1 мы докажем, что оптимум, достигаемый, когда $\nabla_{\mathbf{w}} \text{RSS}(\mathbf{w}) = \mathbf{0}$, удовлетворяет следующему уравнению:

$$\hat{\mathbf{w}}_{\text{mle}} \triangleq \underset{\mathbf{w}}{\text{argmin}} \text{RSS}(\mathbf{w}) = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}. \quad (4.59)$$

Эта оценка **обыкновенных наименьших квадратов** (ordinary least squares – **OLS**) эквивалентна MLE.

4.3. Минимизация эмпирического риска (ERM)

Мы можем обобщить MLE, заменив член логарифма (условной) потери в формуле (4.6), $\ell(\mathbf{y}_n, \boldsymbol{\theta}; \mathbf{x}_n) = -\log p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta})$ другой функцией потерь:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^N \ell(y_n, \theta; x_n). \quad (4.60)$$

Такая постановка называется **минимизацией эмпирического риска** (empirical risk minimization – **ERM**), поскольку это ожидаемая потеря, где математическое ожидание берется относительно эмпирического распределения. Детали см. в разделе 5.4.

4.3.1. Пример: минимизации частоты неправильной классификации

При решении задачи классификации иногда используется бинарная функция потерь:

$$\ell_{01}(y_n, \theta; x_n) = \begin{cases} 0, & \text{если } y_n = f(x_n; \theta) \\ 1, & \text{если } y_n \neq f(x_n; \theta) \end{cases} \quad (4.61)$$

где $f(x; \theta)$ – какой-то предиктор. Эмпирический риск принимает вид:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^N \ell_{01}(y_n, \theta; x_n). \quad (4.62)$$

Это просто **частота неправильной классификации** на обучающем наборе.

Отметим, что для бинарных задач частоту неправильной классификации можно переписать в следующих обозначениях. Пусть $\tilde{y} \in \{-1, +1\}$ – истинная метка и $\hat{y} \in \{-1, +1\} = f(x; \theta)$ – наше предсказание. Определим бинарную функцию потерь следующим образом:

$$\ell_{01}(\tilde{y}, \hat{y}) = \mathbb{I}(\tilde{y} \neq \hat{y}) = \mathbb{I}(\tilde{y}\hat{y} < 0). \quad (4.63)$$

Соответствующий эмпирический риск принимает вид:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^N \ell_{01}(y_n, \hat{y}_n) = \frac{1}{N} \sum_{n=1}^N \mathbb{I}(\tilde{y}_n \hat{y}_n < 0), \quad (4.64)$$

где зависимость от x_n и θ неявная.

4.3.2. Суррогатная потеря

К сожалению, бинарная потеря из раздела 4.3.1 – негладкая функция, как показано на рис. 4.2, поэтому оптимизировать ее трудно. (На самом деле это NP-трудная задача [BDEL03].) В этом разделе мы рассмотрим **суррогатную функцию потерь** [BJM06]. Обычно в качестве суррогата выбирается максимально близкая выпуклая верхняя граница, которую минимизировать легко.

Например, рассмотрим вероятностный бинарный классификатор, который порождает следующее распределение меток:

$$p(\tilde{y}|\mathbf{x}, \boldsymbol{\theta}) = \sigma(\tilde{y}\eta) = \frac{1}{1 + e^{-\tilde{y}\eta}}, \quad (4.65)$$

где $\eta = f(\mathbf{x}; \boldsymbol{\theta})$ – логарифм отношения шансов. Тогда логарифмическая потеря равна:

$$\ell_{\text{ll}}(\tilde{y}, \eta) = -\log p(\tilde{y}|\eta) = \log(1 + e^{-\tilde{y}\eta}). \quad (4.66)$$

На рис. 4.2 показано, что это гладкая верхняя граница пороговой потери. Мы построили график зависимости потери от величины $\tilde{y}\eta$, называемой **зазором**, потому что она определяет «безопасный отступ» от порогового значения 0. Таким образом, мы видим, что минимизация отрицательного логарифмического правдоподобия эквивалентна минимизации (довольно точной) верхней границы эмпирической бинарной потери.

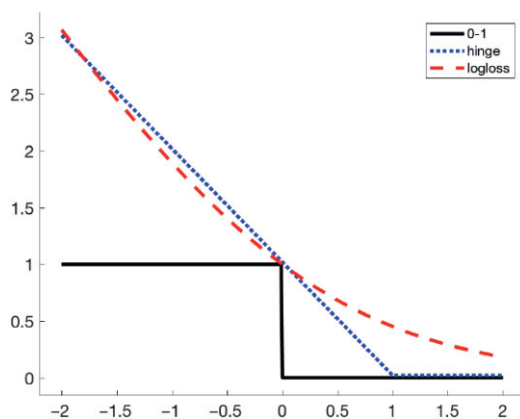


Рис. 4.2 ❖ Различные функции потерь для бинарной классификации. По горизонтальной оси отложен зазор $z = \tilde{y}\eta$, по вертикальной – потеря. В логарифмической потере логарифм берется по основанию 2. Построено программой по адресу figures.problml.ai/book1/4.2

Еще одна выпуклая верхняя граница бинарной функции потерь – **кусочно-линейная потеря** (hinge loss), определенная следующим образом:

$$\ell_{\text{hinge}}(\tilde{y}, \eta) = \max(0, 1 - \tilde{y}\eta) \triangleq (1 - \tilde{y}\eta)_+. \quad (4.67)$$

Ее график показан на рис. 4.2; мы видим, что она выглядит, как полураскрытая дверная петля. Это выпуклая верхняя граница бинарной функции потерь, хотя она дифференцируема не везде.

4.4. ДРУГИЕ МЕТОДЫ ОЦЕНИВАНИЯ*

4.4.1. Метод моментов

Для вычисления MLE необходимо решить уравнение $\nabla_{\theta} \text{NLL}(\theta) = 0$. Иногда это вычислительно трудная задача. В таких случаях можно использовать более простой подход, называемый **методом моментов** (method of moments – **MOM**). Он заключается том, что мы приравняем теоретические моменты распределения эмпирическим моментам и решаем получившуюся систему K уравнений, где K – число параметров. Теоретические моменты по определению равны $\mu_k = \mathbb{E}[Y^k]$, $k = 1 \dots K$, а эмпирические определяются формулой:

$$\hat{\mu}_k = \frac{1}{N} \sum_{n=1}^n y_n^k, \quad (4.68)$$

так что нужно лишь решить уравнение $\mu_k = \hat{\mu}_k$ для каждого k . Ниже мы приведем несколько примеров.

Метод моментов прост, но с точки зрения теории уступает подходу на основе MLE, потому что не позволяет использовать все данные так же эффективно. (Дополнительные сведения о теоретических результатах см., например, в работе [CB02].) Кроме того, он иногда дает несогласованные результаты (см. раздел 4.4.1.2). Однако в тех случаях, когда порождаемые оценки правильны, его можно использовать для инициализации итеративных алгоритмов оптимизации NLL (см., например, [АНК12]) и тем самым объединить вычислительную эффективность MOM со статистической точностью MLE.

4.4.1.1. Пример: MOM для одномерного гауссова распределения

Рассмотрим одномерное гауссово распределение. Из раздела 4.2.5 мы знаем, что

$$\mu_1 = \mu = \bar{y}, \quad (4.69)$$

$$\mu_2 = \sigma^2 + \mu^2 = s^2, \quad (4.70)$$

где \bar{y} – эмпирическое среднее, а s^2 – эмпирическая средняя сумма квадратов, т. е. $\hat{\mu} = \bar{y}$ и $\hat{\sigma}^2 = s^2 - \bar{y}^2$. В этом случае MOM дает такую же оценку, как MLE, но так бывает не всегда.

4.4.1.2. Пример: MOM для равномерного распределения

В этом разделе мы рассмотрим применение MOM к равномерному распределению. В своем изложении мы следуем статье из «Википедии»¹. Пусть $Y \sim \text{Unif}(\theta_1, \theta_2)$ – равномерно распределенная случайная величина, т. е.:

$$p(y|\theta) = \frac{1}{\theta_2 - \theta_1} \mathbb{I}(\theta_1 \leq y \leq \theta_2). \quad (4.71)$$

Первые два момента равны:

$$\mu_1 = \mathbb{E}[Y] = \frac{1}{2}(\theta_1 + \theta_2); \quad (4.72)$$

$$\mu_2 = \mathbb{E}[Y^2] = \frac{1}{3}(\theta_1^2 + \theta_1\theta_2 + \theta_2^2). \quad (4.73)$$

Решая эту систему уравнений, получаем

$$(\theta_1, \theta_2) = \left(\mu_1 - \sqrt{3(\mu_2 - \mu_1^2)}, 2\mu_1 - \theta_1 \right). \quad (4.74)$$

К сожалению, эта оценка иногда неверна. Например, предположим, что $\mathcal{D} = \{0, 0, 0, 0, 1\}$. Эмпирические моменты равны $\hat{\mu}_1 = 1/5$, $\hat{\mu}_2 = 1/5$, поэтому оценки параметров равны $\hat{\theta}_1 = 1/5 - 2\sqrt{3}/5 = -0.493$ и $\hat{\theta}_2 = 1/5 + 2\sqrt{3}/5 = 0.893$. Однако это никак не может быть верно, потому что если $\theta_2 = 0.893$, то любой сгенерированный пример был бы меньше 1.

С другой стороны, рассмотрим MLE. Пусть $y_{(1)} \leq y_{(2)} \leq \dots \leq y_{(N)}$ – **порядковые статистики** данных (т. е. значения, отсортированные в порядке возрастания). Положим $\theta = \theta_2 - \theta_1$. Тогда правдоподобие равно:

$$p(\mathcal{D}|\theta) = (\theta)^{-N} \mathbb{I}(y_{(1)} \geq \theta_1) \mathbb{I}(y_{(N)} \leq \theta_2). \quad (4.75)$$

В допустимом для θ диапазоне производная логарифмического правдоподобия равна:

$$\frac{d}{d\theta} \log p(\mathcal{D}|\theta) = -\frac{N}{\theta} < 0. \quad (4.76)$$

Поэтому правдоподобие – убывающая функция от θ , т. е. мы должны выбрать

$$\hat{\theta}_1 = y_{(1)}; \hat{\theta}_2 = y_{(N)}. \quad (4.77)$$

В примере, рассмотренном выше, получаем $\hat{\theta}_1 = 0$ и $\hat{\theta}_2 = 1$, что и следовало ожидать.

¹ [https://en.wikipedia.org/wiki/Method_of_moments_\(statistics\)](https://en.wikipedia.org/wiki/Method_of_moments_(statistics)).

4.4.2. Онлайновое (рекурсивное) оценивание

Если весь набор данных доступен до начала обучения, то говорят, что производится **пакетное обучение**. Но в некоторых случаях данные поступают последовательно, так что $\mathcal{D} = \{y_1, y_2, \dots\}$ – неограниченный поток. В таком случае необходимо выполнить **онлайновое обучение**.

Пусть $\hat{\theta}_{t-1}$ – наша оценка (например, MLE) при известном отрезке $\mathcal{D}_{1:t-1}$. Чтобы алгоритм обучения гарантированно обрабатывал один пример за постоянное время, мы должны найти правило обучения вида:

$$\theta_t = f(\hat{\theta}_{t-1}; y_t). \quad (4.78)$$

Это называется **рекурсивным обновлением**. Ниже мы приведем примеры таких методов онлайнового обучения.

4.4.2.1. Пример: рекурсивная MLE среднего гауссова распределения

Вернемся к примеру из раздела 4.2.5, где мы вычисляли MLE для одномерного гауссова распределения. Мы знаем, что пакетная оценка среднего равна:

$$\hat{\mu}_t = \frac{1}{t} \sum_{n=1}^t y_n. \quad (4.79)$$

Это просто сумма данных с нарастающим итогом, поэтому ее легко преобразовать в рекурсивную оценку:

$$\hat{\mu}_t = \frac{1}{t} \sum_{n=1}^t y_n = \frac{1}{t} ((t-1)\hat{\mu}_{t-1} + y_t) \quad (4.80)$$

$$= \hat{\mu}_{t-1} + \frac{1}{t} (y_t - \hat{\mu}_{t-1}). \quad (4.81)$$

Это называется **скользящим средним**.

Из формулы (4.81) мы видим, что новая оценка равна сумме старой и поправочного члена. Размер поправки со временем уменьшается (по мере получение новых примеров). Однако если распределение меняется, то нужно придать больший вес недавно полученным примерам. Как это делается, мы обсудим в разделе 4.4.2.2.

4.4.2.2. Экспоненциально взвешенное скользящее среднее

Формула (4.81) показывает, как вычислить скользящее среднее сигнала. В этом разделе мы покажем, как уточнить эту формулу, чтобы придавать больше веса недавним примерам. В частности, мы вычислим следующее **экспоненциально взвешенное скользящее среднее** (exponentially weighted moving average – EWMA), которое иногда называют просто **экспоненциальным скользящим средним** (EMA):

$$\mu_t = \beta \mu_{t-1} + (1 - \beta) y_t, \quad (4.82)$$

где $0 < \beta < 1$. Вклад примера, отстоящего на k шагов в прошлом, умножается на вес $\beta^k(1 - \beta)$. То есть имеем

$$\hat{\mu}_t = \beta \mu_{t-1} + (1 - \beta) y_t \quad (4.83)$$

$$= \beta^2 \mu_{t-2} + \beta(1 - \beta) y_{t-1} + (1 - \beta) y_t \quad (4.84)$$

$$= \beta^t y_0 + (1 - \beta) \beta^{t-1} y_1 + \dots + (1 - \beta) \beta y_{t-1} + (1 - \beta) y_t. \quad (4.85)$$

Сумма **геометрической прогрессии** равна:

$$\beta^t + \beta^{t-1} + \dots + \beta^1 + \beta^0 = \frac{1 - \beta^{t+1}}{1 - \beta}. \quad (4.86)$$

Поэтому

$$(1 - \beta) \sum_{k=0}^t \beta^k = (1 - \beta) \frac{1 - \beta^{t+1}}{1 - \beta} = 1 - \beta^{t+1}. \quad (4.87)$$

Поскольку $0 < \beta < 1$, имеем $\beta^{t+1} \rightarrow 0$ при $t \rightarrow \infty$, поэтому при малых β прошлое забывается быстрее, и, соответственно, быстрее происходит адаптация к новым данным. Это показано на рис. 4.3.

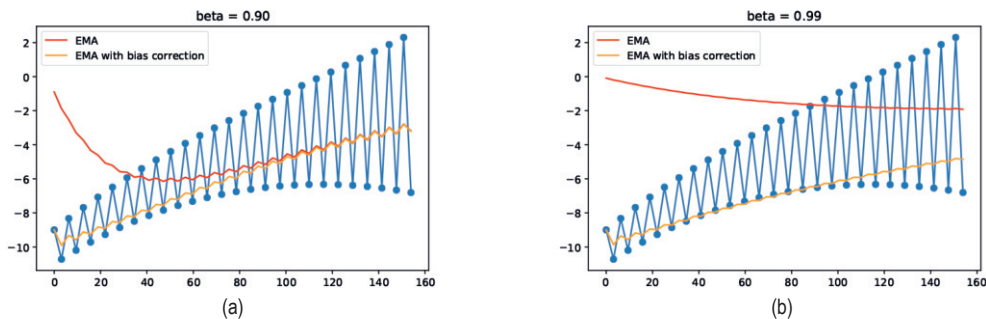


Рис. 4.3 ❖ Экспоненциально взвешенное скользящее среднее с коррекцией и без коррекции смещения. (а) С короткой памятью: $\beta = 0.9$. (б) С долгой памятью: $\beta = 0.99$. Построено программой по адресу figures.problm.ai/book1/4.3

Поскольку начальная оценка равна $\hat{\mu}_0 = 0$, то существует начальное смещение. Это можно исправить, добавив масштабирование (см. [KB15]):

$$\tilde{\mu}_t = \frac{\hat{\mu}_t}{1 - \beta^t}. \quad (4.88)$$

(Отметим, что обновление, описываемое формулой (4.82), по-прежнему применяется к нескорректированному ЕМА, $\hat{\mu}_{t-1}$, т. е. до коррекции на текущем временном шаге.) Достижимое улучшение показано на рис. 4.3.

4.5. РЕГУЛЯРИЗАЦИЯ

Фундаментальная проблема, присущая MLE и ERM, заключается в том, что алгоритм старается выбрать параметры, минимизирующие потерю на обучающем наборе, но в результате может не получиться модель с низкой потерей на будущих данных. Это называется **переобучением**.

Предположим, к примеру, что мы хотим предсказывать вероятность выпадения орла при подбрасывании монеты. Мы подбрасываем $N = 3$ раза и наблюдаем три выпадения орла. MLE равна $\hat{\theta}_{\text{mle}} = N_1/(N_0 + N_1) = 3/(3 + 0) = 1$ (см. раздел 4.2.3). Но если для предсказания использовать распределение $\text{Ber}(y|\hat{\theta}_{\text{mle}})$, то мы будем предсказывать выпадение орла при любом будущем подбрасывании, что крайне маловероятно.

Проблема в том, что параметров модели достаточно для точной аппроксимации наблюдаемых обучающих данных, поэтому она идеально совпадает с эмпирическим распределением. Однако в большинстве случаев эмпирическое распределение не совпадает с истинным, поэтому, сосредоточив всю массу вероятности в наблюдаемом наборе N примеров, мы не оставляем ничего для будущих данных. То есть модель не **обобщается**.

Наиболее распространенным решением проблемы переобучения является **регуляризация**, т. е. включение в NLL (или эмпирический риск) штрафующего члена. Таким образом, требуется оптимизировать целевую функцию вида

$$\mathcal{L}(\theta; \lambda) = \left[\frac{1}{N} \sum_{n=1}^N \ell(y_n, \theta; \mathbf{x}_n) \right] + \lambda C(\theta), \quad (4.89)$$

где $\lambda \geq 0$ – **параметр регуляризации**, а $C(\theta)$ – **штраф за сложность**.

Чаще всего в качестве штрафа за сложность используется $C(\theta) = -\log p(\theta)$, где $p(\theta)$ – априорное распределение θ . Если ℓ – логарифмическая потеря, то регуляризованная целевая функция принимает вид:

$$\mathcal{L}(\theta; \lambda) = -\frac{1}{N} \sum_{n=1}^N \log p(\mathbf{y}_n | \mathbf{x}_n, \theta) - \lambda \log p(\theta). \quad (4.90)$$

Полагая $\lambda = 1$ и соответственно масштабируя $p(\theta)$, мы можем выбрать эквивалентную функцию для минимизации:

$$\mathcal{L}(\theta; \lambda) = -\left[\sum_{n=1}^N \log p(\mathbf{y}_n | \mathbf{x}_n, \theta) + \log p(\theta) \right] = -[\log p(\mathcal{D} | \theta) + \log p(\theta)]. \quad (4.91)$$

Ее минимизация эквивалентна максимизации логарифмического апостериорного распределения:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \log p(\theta | \mathcal{D}) = \underset{\theta}{\operatorname{argmax}} [\log p(\mathcal{D} | \theta) + \log p(\theta) - \text{const}]. \quad (4.92)$$

Это называется **оценкой апостериорного максимума** (maximum a posteriori estimation – **MAP**).

4.5.1. Пример: оценка MAP для распределения Бернулли

Снова рассмотрим подбрасывание монеты. Если наблюдается только один орел, то MLE равна $\theta_{\text{mle}} = 1$. Чтобы избежать такой оценки, мы можем прибавить к θ штраф за «крайние» значения типа $\theta = 0$ или $\theta = 1$. Для этого можно воспользоваться бета-распределением в качестве априорного, $p(\theta) = \text{Beta}(\theta|a, b)$, где $a, b > 1$ поощряет выбор значений θ , близких к $a/(a+b)$ (детали см. в разделе 2.7.4). Сумма логарифмов правдоподобия и априорной вероятности становится равна:

$$LL(\theta) = \log p(\mathcal{D}|\theta) + \log p(\theta) \quad (4.93)$$

$$= [N_1 \log \theta + N_0 \log(1 - \theta)] + [(a - 1) \log(\theta) + (b - 1) \log(1 - \theta)]. \quad (4.94)$$

Применяя метод из раздела 4.2.3, находим оценку MAP:

$$\theta_{\text{map}} = \frac{N_1 + a - 1}{N_1 + N_0 + a + b - 2}. \quad (4.95)$$

Если положить $a = b = 2$ (что отдает слабое предпочтение значениям, близким к 0.5), то оценка примет вид:

$$\theta_{\text{map}} = \frac{N_1 + 1}{N_1 + N_0 + 2}. \quad (4.96)$$

Такое **сглаживание прибавлением единицы** (add-one smoothing) – простая, но широко применяемая техника, позволяющая избежать проблемы **нулевого счетчика** (см. также раздел 4.6.2.9).

Проблема нулевого счетчика и более общая проблема переобучения аналогична тому, что в философии называют **парадоксом черного лебедя**. В его основе лежит старое убеждение Запада, будто все лебеди белые. В этом контексте черный лебедь – метафора того, чего не может быть. (Австралийские черные лебеди были открыты европейскими исследователями в XVII веке.) Термин «парадокс черного лебедя» ввел в обиход знаменитый специалист в области философии науки Карл Поппер; он также вынесен в название недавно вышедшей популярной книги [Tal07]. Этот парадокс использовался для иллюстрации проблемы **индукции**, т. е. формулирования общих выводов о будущем на основе частных наблюдений за прошлым. Чтобы разрешить парадокс, необходимо признать, что в общем случае индукция невозможна и что лучшее, на что можно рассчитывать, – строить правдоподобные гипотезы о том, что сулит будущее, сочетая эмпирические данные с априорным знанием.

4.5.2. Пример: оценка MAP для многомерного гауссова распределения*

В разделе 4.2.6 мы показали, что MLE среднего многомерного гауссова распределения совпадает с эмпирическим средним, $\hat{\mu}_{\text{mle}} = \bar{\mathbf{y}}$. Мы также показали, что MLE ковариации совпадает с эмпирической ковариацией $\hat{\Sigma} = \frac{1}{N} \mathbf{S}_{\bar{\mathbf{y}}}$.

Для большей размерности оценка Σ легко может оказаться сингулярной. Одно из решений этой проблемы – вычислять оценку MAP, как описано ниже.

4.5.2.1. Оценка усадки

В качестве априорного распределения для Σ удобно взять обратное распределение Уишарта. Оно определено на множестве положительно определенных матриц, а его параметры выражаются в терминах априорной матрицы рассеяния $\tilde{\mathbf{S}}$ и размера априорной выборки, или силы \tilde{N} . Можно показать, что результирующая оценка MAP имеет вид

$$\hat{\Sigma}_{\text{map}} = \frac{\tilde{\mathbf{S}} + \mathbf{S}_{\bar{\mathbf{y}}}}{\tilde{N} + N} = \frac{\tilde{N}}{\tilde{N} + N} \frac{\tilde{\mathbf{S}}}{\tilde{N}} + \frac{\tilde{N}}{\tilde{N} + N} \frac{\mathbf{S}_{\bar{\mathbf{y}}}}{N} = \lambda \Sigma_0 + (1 - \lambda) \hat{\Sigma}_{\text{mle}}, \quad (4.97)$$

где $\lambda = \tilde{N}/(\tilde{N} + N)$ контролирует степень регуляризации.

В качестве априорной матрицы рассеяния часто выбирают (см., например, [FR07, стр. 6]) матрицу $\tilde{\mathbf{S}} = \tilde{N} \text{diag}(\hat{\Sigma}_{\text{mle}})$. При таком выборе находим следующую оценку MAP для Σ :

$$\hat{\Sigma}_{\text{map}}(i, j) = \begin{cases} \hat{\Sigma}_{\text{mle}}(i, j), & \text{если } i = j \\ (1 - \lambda) \hat{\Sigma}_{\text{mle}}(i, j) & \text{в противном случае} \end{cases}. \quad (4.98)$$

Таким образом, мы видим, что диагональные элементы равны своим оценкам максимального правдоподобия, а внедиагональные «усаживаются» к 0. Поэтому такой метод называется **оценкой усадки** (shrinkage estimation).

Нам также нужно задать параметр λ , который управляет степенью регуляризации (усадкой к MLE). Обычно его задают с помощью перекрестной проверки (раздел 4.5.5). Можно вместо этого воспользоваться замкнутой формулой, приведенной в работах [LW04a; LW04b; SS05], которая дает оптимальную частотную оценку, если используется квадратичная потеря. Этот способ реализован в библиотеке sklearn (см. <https://scikitlearn.org/stable/modules/generated/sklearn.covariance.LedoitWolf.html>).

Преимущества этого подхода показаны на рис. 4.4. Мы рассматриваем аппроксимацию 50-мерного гауссова распределения по $N = 100$, $N = 50$ и $N = 25$ точкам. Видно, что оценка MAP всегда хорошо обусловлена, в отличие от MLE (обсуждение чисел обусловленности см. в разделе 7.1.4.4). В частности, мы видим, что спектр собственных значений оценки MAP гораздо ближе к спектру истинной матрицы, чем к спектру MLE. Однако на собственные векторы это не влияет.

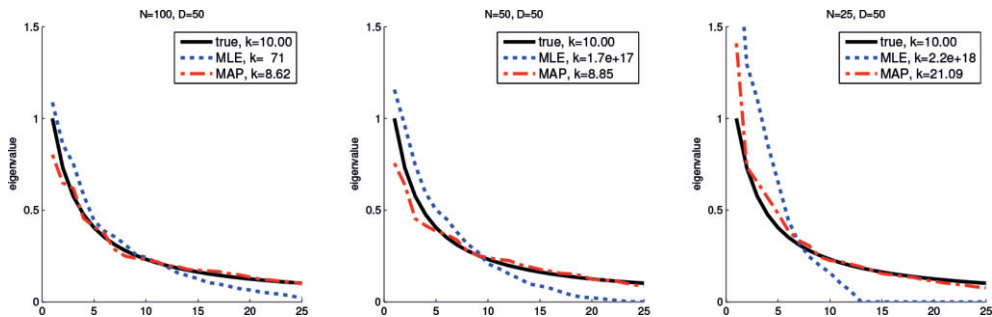


Рис. 4.4 ❖ Оценка ковариационной матрицы для $D = 50$ измерений на основе $N \in \{100, 50, 25\}$ примеров. Мы нанесли на график в порядке убывания собственные значения истинной ковариационной матрицы (сплошная черная линия) и оценки MLE (пунктирная синяя линия) и MAP (пунктирная красная линия), воспользовавшись формулой (4.98) с $\lambda = 0.9$. В надписи мы также указали число обусловленности каждой матрицы. Видно, что MLE часто бывает плохо обусловлена, тогда как оценка MAP численно устойчива. На основе рис. 1 из работы [SS05]. Построено программой по адресу figures.problml.ai/book1/4.4

4.5.3. Пример: уменьшение весов

На рис. 1.7 мы видели, что использование полиномиальной регрессии слишком высокой степени может приводить к переобучению. Для решения проблемы можно уменьшить степень полинома. Однако более общее решение – штрафовать за величину весов (коэффициентов регрессии). Для этого можно воспользоваться гауссовым априорным распределением $p(\mathbf{w})$ с нулевым средним. Получающаяся оценка MAP имеет вид

$$\hat{\mathbf{w}}_{\text{map}} = \underset{\mathbf{w}}{\operatorname{argmin}} NLL(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2, \quad (4.99)$$

где $\|\mathbf{w}\|_2^2 = \sum_{d=1}^D w_d^2$. (Мы используем букву \mathbf{w} , а не $\boldsymbol{\theta}$, потому что штрафовать имеет смысл только за абсолютную величину весовых векторов, а не за другие параметры, например смещение или дисперсию шума.)

Формула (4.99) называется **ℓ_2 -регуляризацией** или **уменьшением весов**. Чем больше λ , тем больше параметры штрафуются за излишнюю величину (отклонение от априорного распределения с нулевым средним) и, следовательно, тем меньшей гибкостью обладает модель.

В случае линейной регрессии такая схема штрафования называется **гребневой регрессией**. Например, рассмотрим пример полиномиальной регрессии из раздела 1.2.2.2, когда предиктор имеет вид:

$$f(x; \mathbf{w}) = \sum_{d=0}^D w_d x^d = \mathbf{w}^T [1, x, x^2, \dots, x^D]. \quad (4.100)$$

Предположим, что используется полином большой степени, например $D = 14$, хотя набор данных мал и содержит всего $N = 21$ пример. Применение

MLE для оценивания параметров даст модель, которая аппроксимирует данные очень хорошо, поскольку веса тщательно подбираются, но результирующая функция чрезмерно «извилиста», т. е. налицо переобучение. На рис. 4.5 показано, как увеличение λ снижает степень переобучения. Дополнительные сведения о гребневой регрессии см. в разделе 11.3.

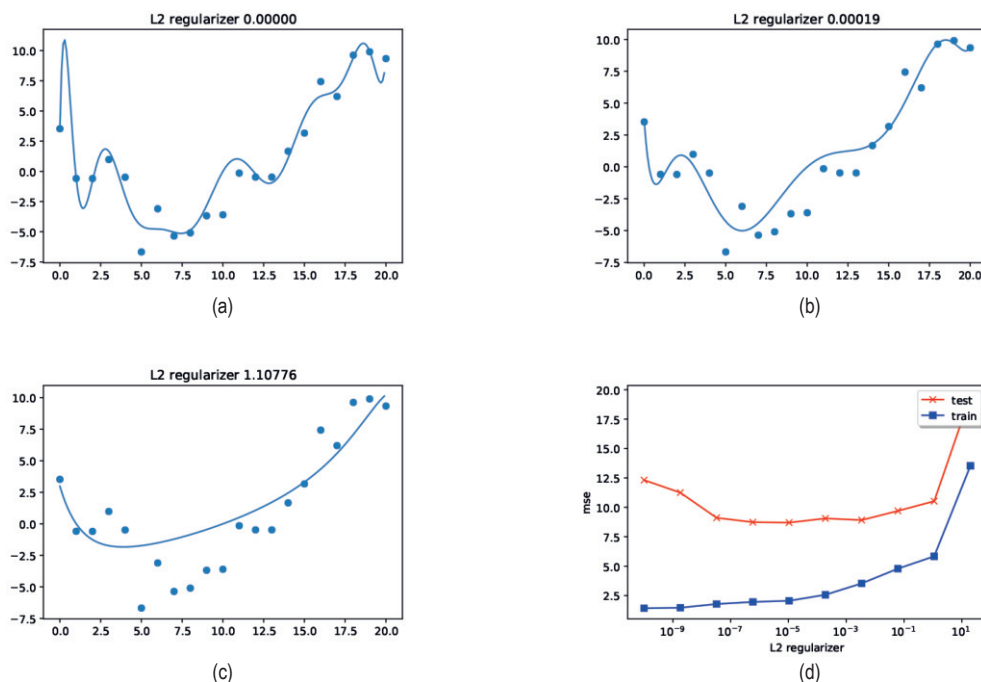


Рис. 4.5 ❖ (а–с) Аппроксимация полинома степени 14 гребневой регрессией по 21 точке. (d) Сравнение MSE и силы регуляризатора. Степень регуляризации увеличивается слева направо, так что сложность модели убывает тоже слева направо. Построено программой по адресу figures.problm.ai/book1/4.5

4.5.4. Подбор регуляризатора с помощью контрольного набора

При использовании регуляризации ключевой вопрос – как выбрать силу регуляризатора λ : малое значение означает, что мы больше заинтересованы в минимизации эмпирического риска, что может привести к переобучению, а большое – что мы хотим оставаться близко к априорному распределению, что может стать причиной **недообученности** модели.

В этом разделе мы опишем простой, но широко применяемый метод выбора λ . Идея в том, чтобы разбить данные на два непересекающихся набора: обучающий $\mathcal{D}_{\text{train}}$ и **контрольный** $\mathcal{D}_{\text{valid}}$. (Часто под обучающий набор отводится примерно 80 % данных, а под контрольный – 20 %.) Мы обучаем модель

на наборе $\mathcal{D}_{\text{train}}$ (для каждого значения λ), а потом оцениваем ее качество на наборе $\mathcal{D}_{\text{valid}}$. Затем берется значение λ , при котором качество на контрольном наборе наилучшее. (Этот метод оптимизации – одномерный пример поиска на сетке, рассматриваемого в разделе 8.8.)

Для более подробного объяснения метода необходимо ввести обозначения. Определим регуляризированный эмпирический риск на наборе данных следующим образом:

$$R_{\lambda}(\boldsymbol{\theta}, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} \ell(y, f(x; \boldsymbol{\theta})) + \lambda C(\boldsymbol{\theta}). \quad (4.101)$$

Для каждого λ вычисляем оценку параметров:

$$\hat{\boldsymbol{\theta}}_{\lambda}(\mathcal{D}_{\text{train}}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} R_{\lambda}(\boldsymbol{\theta}, \mathcal{D}_{\text{train}}). \quad (4.102)$$

Затем вычисляем **риск на контрольном наборе**:

$$R_{\lambda}^{\text{val}} \triangleq R_0(\hat{\boldsymbol{\theta}}_{\lambda}(\mathcal{D}_{\text{train}}), \mathcal{D}_{\text{valid}}). \quad (4.103)$$

Это оценка **риска на генеральной совокупности** – ожидаемая потеря при истинном распределении $p^*(x, y)$. Наконец, выбираем:

$$\lambda^* = \underset{\lambda \in \mathcal{S}}{\operatorname{argmin}} R_{\lambda}^{\text{val}}. \quad (4.104)$$

(Для этого требуется обучить модель по одному разу для каждого значения λ в \mathcal{S} , хотя в некоторых случаях можно поступить и более эффективно.)

Выбрав λ^* , мы можем заново аппроксимировать моделью весь набор данных $\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{valid}}$ и в результате получим:

$$\hat{\boldsymbol{\theta}}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} R_{\lambda^*}(\boldsymbol{\theta}, \mathcal{D}). \quad (4.105)$$

4.5.5. Перекрестная проверка

Описанная в разделе 4.5.4 техника может работать очень хорошо. Однако если размер обучающего набора мал, то при резервировании 20 % данных под контрольный набор может получиться ненадежная оценка параметров модели.

Простое, но популярное решение – воспользоваться **перекрестной проверкой** (cross validation – **CV**). Идея такова: разбиваем обучающий набор на K групп, затем для каждого $k \in \{1, \dots, K\}$ обучаем модель на всех группах, кроме k -й, и проверяем на k -й. И так по кругу, как показано на рис. 4.6. Формально имеем

$$R_{\lambda}^{\text{cv}} \triangleq \frac{1}{K} \sum_{k=1}^K R_0(\hat{\boldsymbol{\theta}}_{\lambda}(D_{-k}), D_k), \quad (4.106)$$

где \mathcal{D}_k – данные, попавшие в k -ю группу, а \mathcal{D}_{-k} – все остальные данные. Эта величина называется **перекрестным риском**. На рис. 4.6 эта процедура показана для $K = 5$. При $K = N$ получается метод **перекрестной проверки с исключением по одному**, поскольку мы всегда проводим обучение на $N - 1$ примере и проверяем на одном оставшемся.

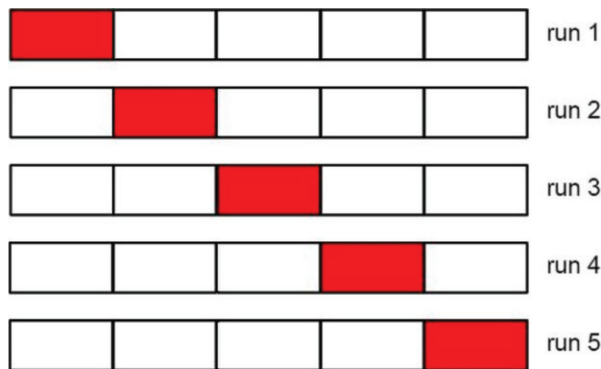


Рис. 4.6 ❖ Схема 5-групповой перекрестной проверки

Оценку CV можно использовать в качестве целевой функции в составе процедуры оптимизации для выбора оптимального гиперпараметра $\hat{\lambda} = \operatorname{argmin}_{\lambda} R_{\lambda}^{\text{cv}}$. Наконец, мы объединяем все имеющиеся данные (обучающие и контрольные) и заново оцениваем параметры модели по формуле $\hat{\theta} = \operatorname{argmin}_{\theta} R_i(\theta, \mathcal{D})$. Дополнительные сведения см. в разделе 5.4.3.

4.5.5.1. Правило одной стандартной ошибки

Перекрестная проверка дает оценку \hat{R}_{λ} , но не дает никаких сведений о неопределенности. Стандартная частотная мера неопределенности оценки – это **стандартная ошибка среднего**, т. е. среднее выборочного распределения оценки (см. раздел 4.7.1). Ее можно вычислить следующим образом. Сначала обозначим $L_n = \ell(\mathbf{y}_n, f(\mathbf{x}_n, \hat{\theta}_{\lambda}(\mathcal{D}_{-n})))$ потерю на n -м примере, используя оценки параметров, полученные на любой обучающей группе, в которую этот пример не входит. (Отметим, что L_n зависит от λ , но эта зависимость в обозначениях не отражена.) Далее обозначим $\hat{\mu} = (1/N) \sum_{n=1}^N L_n$ эмпирическое среднее, а $\hat{\sigma}^2 = (1/N) \sum_{n=1}^N (L_n - \hat{\mu})^2$ эмпирическую дисперсию. Теперь в качестве оценки возьмем $\hat{\mu}$, а в качестве стандартной ошибки этой оценки $\operatorname{se}(\hat{\mu}) = \hat{\sigma}/\sqrt{N}$. Заметим, что σ измеряет внутренне присущую вариативность L_n на выборке, а $\operatorname{se}(\hat{\mu})$ – нашу неуверенность в среднем $\hat{\mu}$.

Предположим, что мы применили перекрестную проверку к набору моделей и вычислили среднее и стандартную ошибку оцененных рисков. Для выбора одной модели часто используется такая эвристика: из этих зашумленных оценок выбрать значение, соответствующее самой простой модели, риск которой превышает риск наилучшей модели не более чем на одну стандартную ошибку. Это называется **правилом одной стандартной ошибки** [HTF01, p216].

4.5.5.2. Пример: гребневая регрессия

В качестве примера рассмотрим выбор силы ℓ_2 -регуляризатора для задачи гребневой регрессии из раздела 4.5.3. На рис. 4.7а приведен график зависимости ошибки от $\log(\lambda)$ на обучающем наборе (синяя линия) и на тестовом наборе (красная линия). Мы видим, что кривая ошибки на тестовом наборе имеет U-образную форму, т. е. сначала уменьшается с ростом степени регуляризации, а затем начинает возрастать, что свидетельствует о недообученности. На рис. 4.7б приведен график зависимости оценки ошибки на тестовом наборе при 5-групповой перекрестной проверке от $\log(\lambda)$. Мы видим, что минимальная ошибка CV близка к оптимальному значению на тестовом наборе (хотя пик ошибки на тестовом наборе для больших λ недооценен вследствие малого размера выборки).

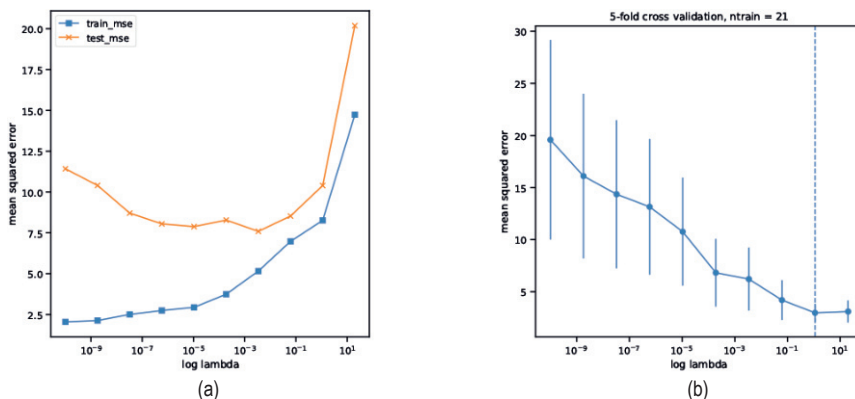


Рис. 4.7 ❖ Аппроксимация полинома степени 14 гребневой регрессией по 21 точке, показанная на рис. 4.5 для разных значений силы регуляризатора λ . Степень регуляризации увеличивается слева направо, так что сложность модели убывает тоже слева направо. (a) Зависимость MSE на обучающем (синий) и тестовом (красный) наборе от $\log(\lambda)$. (b) 5-групповая оценка перекрестной проверки на тестовом наборе; «Усы» показывают стандартную ошибку среднего. Вертикальная прямая соответствует точке, выбранной по правилу одной стандартной ошибки. Построено программой по адресу figures.problml.ai/book1/4.7

4.5.6. Ранняя остановка

Очень простая форма регуляризации, которая на практике часто оказывается весьма эффективной (особенно для сложных моделей) – **ранняя остановка**. При этом используется тот факт, что алгоритмы оптимизации итеративные, поэтому им требуется много шагов, чтобы уйти далеко от начальных оценок параметров. Если мы начинаем замечать признаки переобучения (при наблюдении за качеством на контрольном наборе), то можем остановить процесс оптимизации, чтобы не дать модели запомнить слишком много информации об обучающем наборе (см. иллюстрацию на рис. 4.8).

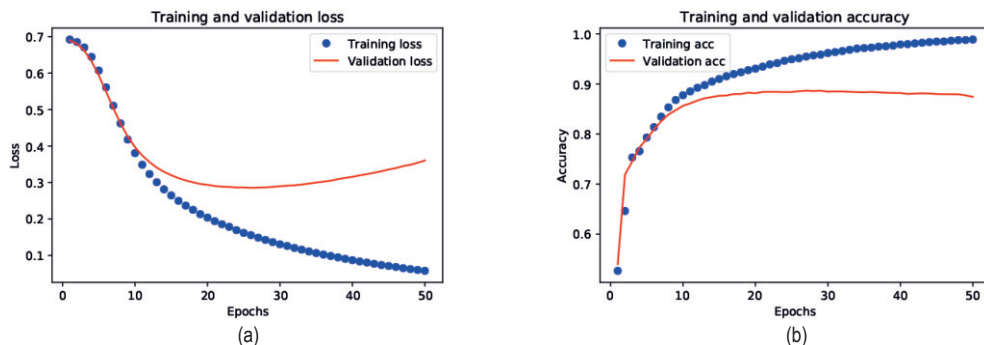


Рис. 4.8 ❖ Качество классификатора текста (нейронной сети, примененной к мешку погружений слов с использованием пулинга усреднением) в зависимости от числа эпох обучения на наборе данных IMDB. Синяя линия = обучение, красная линия = контроль. (а) Потеря перекрестной энтропии. Ранняя остановка происходит примерно на 25-й эпохе. (б) Верность классификации. Построено программой по адресу figures.problml.ai/book1/4.8

4.5.7. Больше данных

По мере того как объем данных растет, шансы переобучения (для модели фиксированной сложности) уменьшаются (в предположении, что данные содержат достаточно информативные примеры и не слишком избыточны). Это показано на рис. 4.9. Приведены СКО на обучающем и тестовом наборах для четырех разных моделей (полиномов возрастающей степени) в виде функции от размера обучающего набора N . (График зависимости ошибки от размера обучающего набора называется **кривой обучения**.) Горизонтальная черная прямая представляет **байесовскую ошибку**, т. е. ошибку оптимального предиктора (истинной модели) вследствие собственных шумов. (В этом примере истинная модель – полином второй степени, а дисперсия шума равна $\sigma^2 = 4$; это называется **минимальным уровнем шума**, потому что уменьшить его невозможно.)

Можно обратить внимание на несколько интересных моментов. Во-первых, ошибка на тестовом наборе для полинома степени 1 остается высокой даже при увеличении N , поскольку модель слишком проста для улавливания истинного положения вещей; это называется **недообучением**. Для других моделей ошибка на тестовом наборе уменьшается до оптимального уровня (минимального уровня шума), но чем проще модель, тем быстрее происходит уменьшение, потому что нужно оценивать меньше параметров. Разрыв между ошибками на тестовом и обучающем наборах больше для сложных моделей, но уменьшается с ростом N .

Еще один интересный момент – тот факт, что ошибка на обучающем наборе (синяя линия) сначала *возрастает* с ростом N , по крайней мере, для достаточно гибких моделей. Причина в следующем: по мере увеличения размера набора данных мы наблюдаем больше различных комбинаций входов и выходов, поэтому задача аппроксимации данных усложняется. Но в конеч-

ном итоге обучающий набор становится больше похож на тестовый, и частота ошибок сходится к пределу, отражающему оптимальное качество модели.

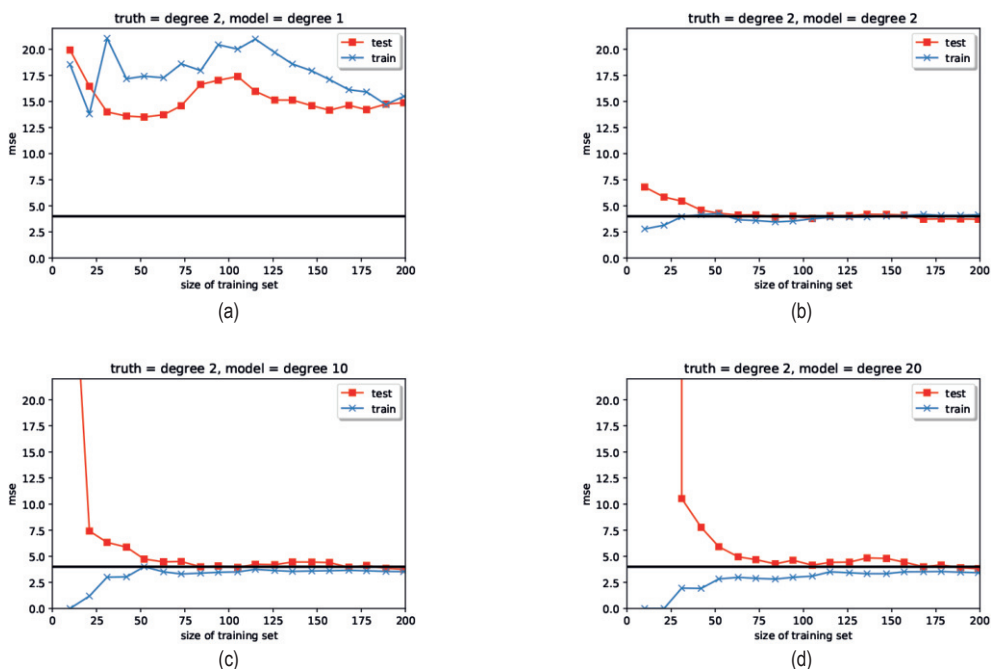


Рис. 4.9 ❖ Зависимость СКО на обучающем и тестовом наборах от размера обучающего набора для данных, сгенерированных из полинома степени 2 с гауссовым шумом, имеющим дисперсию $\sigma^2 = 4$. Мы аппроксимируем эти данные полиномиальными моделями разной степени. Построено программой по адресу figures.problm.ai/book1/4.9

4.6. БАЙЕСОВСКИЕ СТАТИСТИКИ*

До сих пор мы обсуждали способы оценивания параметров по данным. Однако при этом игнорировалась неопределенность оценок, а это может быть важно в таких приложениях, как активное обучение, а также для того чтобы избежать переобучения или просто знать, в какой мере можно доверять оценке некоторой существенной для науки величины. В статистике моделирование неопределенности параметров с помощью распределения вероятностей (в отличие от вычисления простой точечной оценки) называется **выводом**.

В этом разделе мы воспользуемся апостериорным распределением для представления неопределенности. Это подход, применяемый в **байесовской статистике**. Здесь мы приведем лишь краткое введение, а детали можно найти во втором томе этой книги, [Mur22], а также в других хороших книгах, например [McE20; Gel+14].

Для вычисления апостериорного распределения начнем с априорного распределения $p(\theta)$, отражающего наши знания до знакомства с данными. Затем мы определим **функцию правдоподобия** $p(\mathcal{D}|\theta)$, отражающую данные о параметрах, которые мы ожидаем увидеть. И применим формулу Байеса, чтобы обусловить априорное распределение наблюдаемыми данными и вычислить апостериорное распределение $p(\theta|\mathcal{D})$:

$$p(\theta|\mathcal{D}) = \frac{p(\theta)p(\mathcal{D}|\theta)}{p(\mathcal{D})} = \frac{p(\theta)p(\mathcal{D}|\theta)}{\int p(\theta')p(\mathcal{D}|\theta')d\theta'}. \quad (4.107)$$

Знаменатель $p(\mathcal{D})$ называется **маргинальным правдоподобием**, поскольку вычисляется **маргинализацией** неизвестной величины θ (или ее **исключением путем интегрирования**). Его можно интерпретировать как среднюю вероятность данных относительно априорного распределения. Заметим, однако, что $p(\mathcal{D})$ – постоянная, не зависящая от θ , поэтому мы часто будем игнорировать ее, когда требуется только вывести относительные вероятности значений θ .

Формула (4.107) аналогична формуле Байеса для тестирования на COVID-19 из раздела 2.3.1. Разница в том, что неизвестные соответствуют параметрам статистической модели, а не неизвестному состоянию пациента. Кроме того, условием обычно является набор наблюдений \mathcal{D} , а не одно наблюдение (результат одного теста). В частности, для модели обучения с учителем (условной) наблюдаемые данные имеют вид $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n) : n = 1 \dots N\}$, а для модели обучения без учителя (безусловной) – $\mathcal{D} = \{(\mathbf{y}_n) : n = 1 \dots N\}$.

Вычислив апостериорное распределение параметров, мы можем вычислить **апостериорное прогнозное распределение** выходов при известных входах, исключив неизвестные параметры путем **маргинализации**.

В случае обучения с учителем (условном) это выглядит так:

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}) = \int p(\mathbf{y}|\mathbf{x}, \theta)p(\theta|\mathcal{D})d\theta. \quad (4.108)$$

Это можно рассматривать как форму **байесовского усреднения модели** (Bayes model averaging – **ВМА**), потому что мы делаем предсказания, пользуясь бесконечным множеством моделей (значений параметров), каждой из которых назначен вес – ее правдоподобие. ВМА уменьшает шансы получить переобученную модель (раздел 1.2.3), потому что мы не ограничиваемся одной наилучшей моделью.

4.6.1. Сопряженные априорные распределения

В этом разделе мы рассмотрим множество пар (априорное распределение, правдоподобие), для которого апостериорное распределение можно вычислить в замкнутой форме. Конкретно априорные распределения будут «сопряженными» правдоподобию. Говорят, что априорное распределение $p(\theta) \in \mathcal{F}$ является **сопряженным априорным** для функции правдоподобия $p(\mathcal{D}|\theta)$, если апостериорное распределение принадлежит тому же параметрическо-

му семейству, что априорное, т. е. $p(\theta|\mathcal{D}) \in \mathcal{F}$. Иными словами, \mathcal{F} замкнуто относительно байесовского обновления. Если семейство \mathcal{F} является экспоненциальным (см. раздел 3.4), то вычисления можно произвести в замкнутой форме.

В следующих разделах мы приведем примеры такой схемы, которые будут использоваться далее в книге. Для простоты ограничимся безусловными моделями (когда имеются только выходы, или метки y , но нет входных данных, или признаков x); мы ослабим это ограничение в разделе 4.6.7.

4.6.2. Бета-биномиальная модель

Предположим, что монета подбрасывается N раз и требуется вывести вероятность выпадения орла. Обозначим $y_n = 1$ событие, когда на n -м испытании выпал орел, а $y_n = 0$ событие выпадения решки, и пусть $\mathcal{D} = \{y_n : n = 1 \dots N\}$ – все множество данных. Мы предполагаем, что $y_n \sim \text{Ber}(\theta)$, где $\theta \in [0, 1]$ – параметр (вероятность выпадения орла). В этом разделе мы обсудим, как вычислить $p(\theta|\mathcal{D})$.

4.6.2.1. Правдоподобие Бернулли

Мы предполагаем, что данные независимы и одинаково распределены. Поэтому правдоподобие имеет вид

$$p(\mathcal{D}|\theta) = \prod_{n=1}^N \theta^{y_n} (1 - \theta)^{1-y_n} = \theta^{N_1} (1 - \theta)^{N_0}, \quad (4.109)$$

где $N_1 = \sum_{n=1}^N \mathbb{I}(y_n = 1)$ и $N_0 = \sum_{n=1}^N \mathbb{I}(y_n = 0)$ – количество выпадений орла и решки соответственно. Эти числа называются **достаточными статистиками** данных, поскольку это все, что необходимо знать о \mathcal{D} для вывода θ . Сумма $N = N_0 + N_1$ называется размером выборки.

4.6.2.2. Биномиальное правдоподобие

Можно также рассмотреть модель биномиального правдоподобия, в которой мы производим N испытаний и наблюдаем число выпадений орла y , а не последовательность подбрасываний монеты. Тогда правдоподобие имеет вид:

$$p(\mathcal{D}|\theta) = \text{Bin}(y|N, \theta) = \binom{N}{y} \theta^y (1 - \theta)^{N-y}. \quad (4.110)$$

Масштабный коэффициент $\binom{N}{y}$ не зависит от θ , поэтому его можно проигнорировать. Таким образом, это правдоподобие пропорционально правдоподобию Бернулли (4.109), так что вывод θ одинаков для обеих моделей.

4.6.2.3. Априорное распределение

Чтобы упростить вычисления, будем предполагать, что априорное распределение $p(\theta) \in \mathcal{F}$ является сопряженным априорным для функции правдоподобия $p(y|\theta)$. Это означает, что апостериорное распределение принадлежит тому же параметрическому семейству, что априорное, т. е. $p(\theta|\mathcal{D}) \in \mathcal{F}$.

Чтобы это свойство выполнялось при использовании правдоподобия Бернулли (или биномиального), мы должны использовать априорное распределение вида:

$$p(\theta) \propto \theta^{\tilde{\alpha}-1}(1-\theta)^{\tilde{\beta}-1} = \text{Beta}(\theta|\tilde{\alpha}, \tilde{\beta}). \quad (4.111)$$

В этом выражении нетрудно узнать функцию плотности вероятности бета-распределения (см. раздел 2.7.4).

4.6.2.4. Апостериорное распределение

Если умножить правдоподобие Бернулли (4.109) на априорное бета-распределение (2.136), то получится апостериорное бета-распределение:

$$p(\theta|\mathcal{D}) \propto \theta^{N_1}(1-\theta)^{N_0}\theta^{\tilde{\alpha}-1}(1-\theta)^{\tilde{\beta}-1} \quad (4.112)$$

$$\propto \text{Beta}(\theta|\tilde{\alpha} + N_1, \tilde{\beta} + N_0) \quad (4.113)$$

$$= \text{Beta}(\theta|\hat{\alpha}, \hat{\beta}), \quad (4.114)$$

где $\hat{\alpha} \triangleq \tilde{\alpha} + N_1$ и $\hat{\beta} \triangleq \tilde{\beta} + N_0$ – параметры апостериорного распределения. Апостериорное распределение имеет такую же функциональную форму, как априорное, поэтому можно заключить, что бета-распределение является сопряженным априорным правдоподобию Бернулли.

Параметры априорного распределения называются **гиперпараметрами**. Ясно, что (в этом примере) гиперпараметры играют такую же роль, как достаточные статистики, поэтому их называют **псевдосчетчиками**. Мы видим, что для вычисления апостериорного распределения нужно просто сложить наблюдаемые счетчики (из функции правдоподобия) и псевдосчетчики (из априорного распределения).

Сила априорного распределения управляется величиной $\tilde{N} = \tilde{\alpha} + \tilde{\beta}$, которая называется **эквивалентным размером выборки**, потому что играет ту же роль, что наблюдаемый размер выборки $N = N_0 + N_1$.

4.6.2.5. Пример

Пусть, например, $\tilde{\alpha} = \tilde{\beta} = 2$. Это значит, что мы верим, что видели два орла и две решки еще до наблюдения фактических данных; это очень слабо выраженное предпочтение значению $\theta = 0.5$. Что получается при использовании такого априорного распределения, показано на рис. 4.10а. Мы видим, что апостериорное распределение (синяя линия) – «компромисс» между априорным распределением (красная линия) и правдоподобием (черная линия).

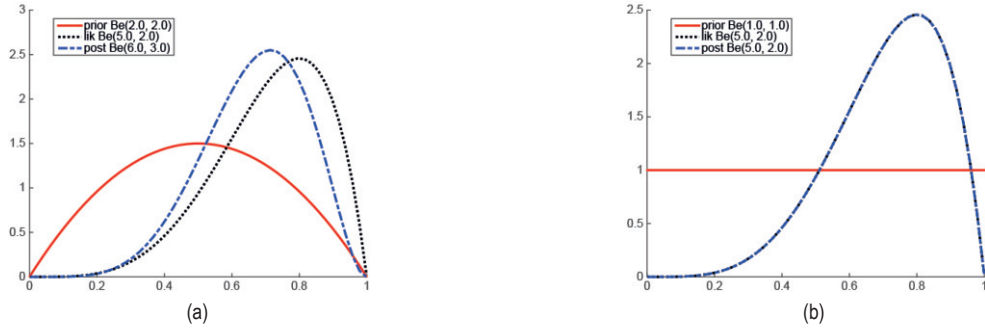


Рис. 4.10 ❖ Обновление априорного бета-распределения в случае правдоподобия Бернулли с достаточными статистиками $N_1 = 4$, $N_0 = 1$. (а) Априорное распределение $\text{Beta}(2,2)$. (б) Равномерное априорное распределение $\text{Beta}(1,1)$. Построено программой по адресу figures.problm.ai/book1/4.10

Если положить $\tilde{\alpha} = \tilde{\beta} = 1$, то соответствующее априорное распределение будет равномерным:

$$p(\theta) = \text{Beta}(\theta|1, 1) \propto \theta^0(1 - \theta)^0 = \text{Unif}(\theta|0, 1). \quad (4.115)$$

Результат выбора такого априорного распределения показан на рис. 4.10b. Мы видим, что апостериорное распределение имеет точно такую же форму, как правдоподобие, так как априорное распределение было «**неинформативным**».

4.6.2.6. Апостериорная мода (оценка MAP)

Наиболее вероятное значение параметра дает оценка MAP:

$$\hat{\theta}_{\text{map}} = \underset{\theta}{\operatorname{argmax}} p(\theta|\mathcal{D}) \quad (4.116)$$

$$= \underset{\theta}{\operatorname{argmax}} \log p(\theta|\mathcal{D}) \quad (4.117)$$

$$= \underset{\theta}{\operatorname{argmax}} \log p(\theta) + \log p(\mathcal{D}). \quad (4.118)$$

Можно показать, что это выражение равно:

$$\hat{\theta}_{\text{map}} = \frac{\tilde{\alpha} + N_1 - 1}{\tilde{\alpha} + N_1 - 1 + \tilde{\beta} + N_0 - 1}. \quad (4.119)$$

Если в качестве априорного распределения взять $\text{Beta}(\theta|2, 2)$, то это сводится к **сглаживанию прибавлением единицы**:

$$\hat{\theta}_{\text{map}} = \frac{N_1 + 1}{N_1 + 1 + N_0 + 1} = \frac{N_1 + 1}{N + 2}. \quad (4.120)$$

Если в качестве априорного распределения взять равномерное $p(\theta) \propto 1$, то оценка MAP совпадает с MLE, потому что $\log p(\theta) = 0$:

$$\hat{\theta}_{\text{mle}} = \underset{\theta}{\operatorname{argmax}} \log p(\mathcal{D}|\theta). \quad (4.121)$$

При использовании в качестве априорного равномерного бета-распределения с $\tilde{\alpha} = \tilde{\beta} = 1$ оценка MAP сводится к MLE:

$$\hat{\theta}_{\text{mle}} = \frac{N_1}{N_1 + N_0} = \frac{N_1}{N}. \quad (4.122)$$

Если $N_1 = 0$, то мы получим оценку $p(Y = 1) = 0.0$, которая означает, что в будущем мы не предсказываем ни одного наблюдения, равного 1. Такая экстремальная оценка вызвана недостаточностью данных. Эту проблему можно решить, воспользовавшись оценкой MAP с более сильным априорным распределением или приняв полностью байесовский подход, при котором θ не оценивается, а исключается путем маргинализации, как было объяснено в разделе 4.6.2.9.

4.6.2.7. Апостериорное среднее

Апостериорная мода может давать плохую сводную информацию о распределении, потому что соответствует всего одной точке. Апостериорное среднее дает более робастную оценку, потому что вычисляется по всему пространству.

Если $p(\theta|\mathcal{D}) = \text{Beta}(\theta|\hat{\alpha}, \hat{\beta})$, то апостериорное среднее равно:

$$\bar{\theta} \triangleq \mathbb{E}[\theta|\mathcal{D}] = \frac{\hat{\alpha}}{\hat{\beta} + \hat{\alpha}} = \frac{\hat{\alpha}}{\hat{N}}, \quad (4.123)$$

где $\hat{N} = \hat{\beta} + \hat{\alpha}$ – сила (эквивалентный размер выборки) апостериорного распределения.

Теперь мы покажем, что апостериорное среднее равно выпуклой комбинации априорного среднего, $m = \tilde{\alpha}/\tilde{N}$ (где $\tilde{N} \triangleq \tilde{\alpha} + \tilde{\beta}$ – сила априорного распределения) и MLE: $\hat{\theta}_{\text{mle}} = N_1/N$:

$$\begin{aligned} \mathbb{E}[\theta|\mathcal{D}] &= \frac{\tilde{\alpha} + N_1}{\tilde{\alpha} + N_1 + \tilde{\beta} + N_0} = \frac{\tilde{N}m + N_1}{N + \tilde{N}} \\ &= \frac{\tilde{N}}{N + \tilde{N}}m + \frac{N}{N + \tilde{N}}\frac{N_1}{N} = \lambda m + (1 - \lambda)\hat{\theta}_{\text{mle}}, \end{aligned} \quad (4.124)$$

где $\lambda = \tilde{N}/\hat{N}$ – отношения априорного эквивалентного размера выборки к апостериорному. Таким образом, чем слабее априорное распределение, тем меньше λ и тем, следовательно, ближе апостериорное среднее к MLE.

4.6.2.8. Апостериорная дисперсия

Чтобы уловить неопределенность оценки, обычно вычисляют **стандартную ошибку** оценки, которая является не чем иным, как стандартным отклонением апостериорного распределения:

$$\text{se}(\theta) = \sqrt{\mathbb{V}[\theta|\mathcal{D}]}. \quad (4.125)$$

Для модели Бернулли мы показали, что апостериорным является бета-распределение. Дисперсия апостериорного бета-распределения равна:

$$\mathbb{V}[\theta|\mathcal{D}] = \frac{\hat{\alpha}\hat{\beta}}{(\hat{\alpha} + \hat{\beta})^2(\hat{\alpha} + \hat{\beta} + 1)} = \mathbb{E}[\theta|\mathcal{D}]^2 \frac{\hat{\beta}}{\hat{\alpha}(1 + \hat{\alpha} + \hat{\beta})}, \quad (4.126)$$

где $\hat{\alpha} = \check{\alpha} + N_1$ и $\hat{\beta} = \check{\beta} + N_0$. Если $N \gg \check{\alpha} + \check{\beta}$, то это выражение упрощается:

$$\mathbb{V}[\theta|\mathcal{D}] = \frac{N_1 N_0}{N^3} = \frac{\hat{\theta}(1 - \hat{\theta})}{N}, \quad (4.127)$$

где $\hat{\theta}$ – оценка максимального правдоподобия, MLE. Следовательно, стандартная ошибка равна:

$$\sigma = \sqrt{\mathbb{V}[\theta|\mathcal{D}]} \approx \sqrt{\frac{\hat{\theta}(1 - \hat{\theta})}{N}}. \quad (4.128)$$

Мы видим, что неопределенность уменьшается со скоростью $1/\sqrt{N}$. Также видно, что неопределенность (дисперсия) достигает максимума, когда $\hat{\theta} = 0.5$, и минимума, когда $\hat{\theta}$ близко к 0 или к 1. Это и понятно, потому что проще удостовериться в том, что монета несимметрична, чем в том, что она симметрична.

4.6.2.9. Апостериорное прогнозное распределение

Предположим, что требуется предсказать будущие наблюдения. Очень распространенный подход – сначала вычислить оценку параметров на основе обучающих данных, $\hat{\theta}(\mathcal{D})$, а затем подставить эти параметры в модель и использовать $p(y|\hat{\theta})$ для предсказания будущего; это называется **подстановочной аппроксимацией** (plug-in approximation). Но иногда при этом возможно переобучение. Например (это крайний случай), предположим, что мы наблюдали подряд $N = 3$ выпадений орла. Тогда MLE равна $\hat{\theta} = 3/3 = 1$. Однако если использовать эту оценку, то следовало бы предсказать, что выпадение решки невозможно.

Возможное решение – вычислить оценку МАР и подставить ее, как обсуждалось в разделе 4.5.1. Здесь мы обсудим полностью байесовское решение, в котором θ исключается путем маргинализации.

Модель Бернулли

Для модели Бернулли результирующее апостериорное прогнозное распределение имеет вид:

$$p(y = 1|\mathcal{D}) = \int_0^1 p(y = 1|\theta)p(\theta|\mathcal{D})d\theta \quad (4.129)$$

$$= \int_0^1 \theta \text{Beta}(\theta|\hat{\alpha}, \hat{\beta})d\theta = \mathbb{E}[\theta|\mathcal{D}] = \frac{\hat{\alpha}}{\hat{\alpha} + \hat{\beta}}. \quad (4.130)$$

В разделе 4.5.1 мы должны были использовать априорное распределение $\text{Beta}(2,2)$, чтобы прийти к сглаживанию прибавлением единицы, но такое априорное распределение неестественно. При байесовском подходе мы можем добиться того же эффекта, воспользовавшись равномерным априорным распределением $p(\theta) = \text{Beta}(\theta|1, 1)$, поскольку прогнозное распределение принимает вид:

$$p(y = 1|\mathcal{D}) = \frac{N_1 + 1}{N_1 + N_0 + 2}. \quad (4.131)$$

Это называется **правилом следования Лапласа**. На рис. 4.11 приведена его иллюстрация в последовательной формулировке.

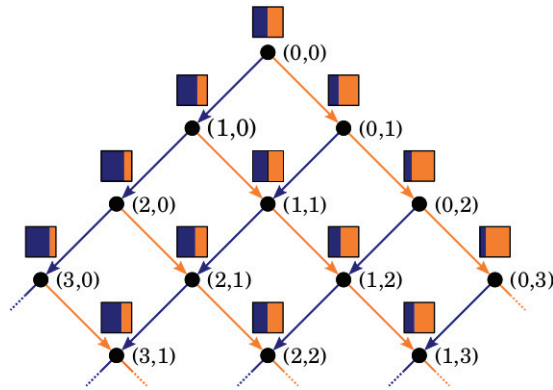


Рис. 4.11 ❖ Последовательное байесовское обновление для модели бета-Бернулли. Каждый цветной квадратик представляет прогнозное распределение $p(x_t|h_t)$, где $h_t = (N_{1,t}, N_{0,t})$ – достаточная статистика, выведенная на основе истории наблюдений до момента t , а именно общего числа выпадений орла и решки. Вероятность выпадения орла (синяя полоса) равна $p(x_t = 1|h_t) = (N_{1,t} + 1) / (t + 2)$ в предположении, что мы начинали с равномерного априорного бета-распределения $\text{Beta}(\theta|1, 1)$. На основе рис. 3 из работы [Ort+19]. Печатается с разрешения Педро Ортега

Биномиальная модель

Теперь предположим, что требуется предсказать, сколько раз выпадет орел при $M > 1$ будущих подбрасываниях монеты, т. е. используется биномиальная модель, а не модель Бернулли. Апостериорное распределение θ такое же, как раньше, но апостериорное прогнозное распределение другое:

$$p(y|\mathcal{D}, M) = \int_0^1 \text{Bin}(y|M, \theta) \text{Beta}(\theta|\hat{\alpha}, \hat{\beta}) d\theta \quad (4.132)$$

$$= \binom{M}{y} \frac{1}{B(\hat{\alpha}, \hat{\beta})} \int_0^1 \theta^y (1 - \theta)^{M-y} \theta^{\hat{\alpha}-1} (1 - \theta)^{\hat{\beta}-1} d\theta. \quad (4.133)$$

В этом интеграле мы узнаем нормировочную постоянную для распределения $\text{Beta}(\hat{\alpha} + y, M - y + \hat{\beta})$. Поэтому

$$\int_0^1 \theta^{y+\hat{\alpha}-1} (1-\theta)^{M-y+\hat{\beta}-1} d\theta = B(y + \hat{\alpha}, M - y + \hat{\beta}). \quad (4.134)$$

Таким образом, мы получаем, что апостериорное прогнозное распределение является (составным) **бета-биномиальным** и описывается следующей формулой:

$$Bb(x|M, \hat{\alpha}, \hat{\beta}) \triangleq \binom{M}{x} \frac{B(x + \hat{\alpha}, M - x + \hat{\beta})}{B(\hat{\alpha}, \hat{\beta})}. \quad (4.135)$$

На рис. 4.12a показана плотность апостериорного прогнозного распределения для $M = 10$ после наблюдения $N_1 = 4$ выпадений орла и $N_0 = 1$ выпадений решки при равномерном априорном распределении $\text{Beta}(1,1)$. На рис. 4.12b показана подстановочная аппроксимация:

$$p(\theta|\mathcal{D}) \approx \delta(\theta - \hat{\theta}); \quad (4.136)$$

$$p(y|\mathcal{D}, M) \approx \int_0^1 \text{Bin}(y|M, \theta) p(\theta|\mathcal{D}) d\theta = \text{Bin}(y|M, \hat{\theta}), \quad (4.137)$$

где $\hat{\theta}$ – оценка MAP. Глядя на рис. 4.12, мы видим, что у байесовского предсказания более длинные хвосты, т. е. масса вероятности размазана по более широкой области, а значит, модель менее подвержена переобучению и парадоксам типа черного лебедя. (Отметим, что в обоих случаях априорное распределение равномерно, так что различие обусловлено не выбором априорного распределения, а тем фактом, что при байесовском подходе неизвестные параметры исключаются путем интегрирования при выполнении предсказаний.)

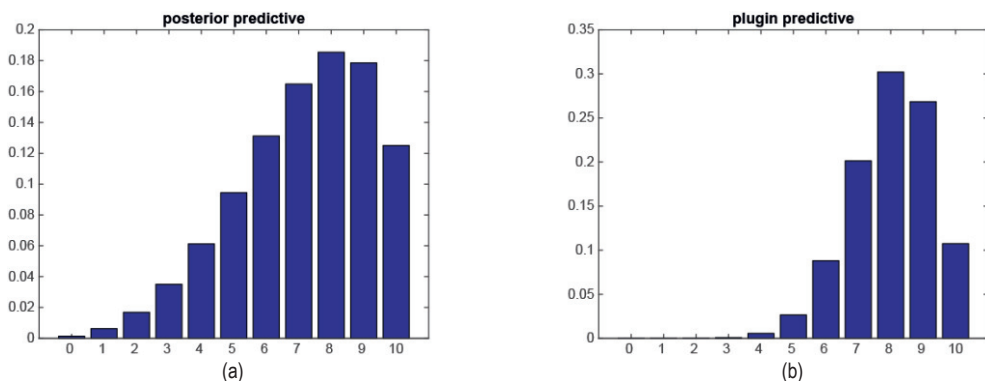


Рис. 4.12 ❖ (a) Апостериорные прогнозные распределения для 10 будущих испытаний после наблюдения $N_1 = 4$ выпадений орла и $N_0 = 1$ выпадений решки. (b) Подстановочная аппроксимация по тем же данным. В обоих случаях используется равномерное априорное распределение. Построено программой по адресу figures.probl.ai/book1/4.12

4.6.2.10. Маргинальное правдоподобие

Маргинальное правдоподобие модели \mathcal{M} определяется как:

$$p(\mathcal{D}|\mathcal{M}) = \int p(\theta|\mathcal{M})p(\mathcal{D}|\theta, \mathcal{M})d\theta. \quad (4.138)$$

При выводе параметров конкретной модели мы можем игнорировать этот член, потому что он не зависит от θ . Однако эта величина играет важную роль при выборе модели, о чем мы будем говорить в разделе 5.2.2. Она также полезна для оценивания гиперпараметров на основе данных (этот подход называется эмпирическим байесовским оцениванием); это тема раздела 4.6.5.3.

В общем случае вычислить маргинальное правдоподобие трудно. Но в случае бета-бернуллиевой модели оно пропорционально отношению апостериорной нормировочной постоянной к априорной. Чтобы убедиться в этом, вспомним, что апостериорное распределение бета-биномиальной модели имеет вид $p(\theta|\mathcal{D}) = \text{Beta}(\theta|a', b')$, где $a' = a + N_1$, $b' = b + N_0$. Мы знаем, что нормировочная постоянная апостериорного распределения равна $B(a', b')$. Следовательно,

$$p(\theta|\mathcal{M}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \quad (4.139)$$

$$= \frac{1}{p(\mathcal{D})} \left[\frac{1}{B(a, b)} \theta^{a-1} (1-\theta)^{b-1} \right] \left[\binom{N}{N_1} \theta^{N_1} (1-\theta)^{N_0} \right] \quad (4.140)$$

$$= \binom{N}{N_1} \frac{1}{p(\mathcal{D})} \frac{1}{B(a, b)} [\theta^{a+N_1-1} (1-\theta)^{b+N_0-1}]. \quad (4.141)$$

Поэтому

$$\frac{1}{B(a + N_1, b + N_0)} = \binom{N}{N_1} \frac{1}{p(\mathcal{D})} \frac{1}{B(a, b)}; \quad (4.142)$$

$$p(\mathcal{D}) = \binom{N}{N_1} \frac{B(a + N_1, b + N_0)}{B(a, b)}. \quad (4.143)$$

Маргинальное правдоподобие бета-бернуллиевой модели такое же, как и раньше, только отсутствует член $\binom{N}{N_1}$.

4.6.2.11. Смеси сопряженных априорных распределений

Бета-распределение является сопряженным априорным для биномиального правдоподобия, что, как мы видели, позволяет легко вычислить апостериорное распределение в замкнутой форме. Однако это априорное распределение довольно ограничительно. Например, предположим, что требуется предсказать исход подбрасывания монеты в казино, и мы полагаем, что монета с равным успехом может быть как симметричной, так и асимметричной,

смещенной в сторону более частого выпадения орла. Это предположение нельзя представить априорным бета-распределением. По счастью, его можно представить **смесью бета-распределений**. Например, можно было бы взять

$$p(\theta) = 0.5 \text{Beta}(\theta|20, 20) + 0.5 \text{Beta}(\theta|30, 10). \quad (4.144)$$

Если θ выбирается из первого распределения, то монета симметрична, а если из второго, то орел выпадает чаще.

Смесь можно представить, введя латентную индикаторную переменную h , где $h = k$ означает, что θ выбирается из k -й компоненты смеси. Априорное распределение имеет вид

$$p(\theta) = \sum_k p(h = k)p(\theta|h = k), \quad (4.145)$$

где каждое $p(\theta|h = k)$ сопряженное, а $p(h = k)$ называются (априорными) весами смеси. Можно показать (упражнение 4.6), что апостериорное распределение также может быть записано в виде смеси сопряженных распределений следующим образом:

$$p(\theta|\mathcal{D}) = \sum_k p(h = k|\mathcal{D})p(\theta|\mathcal{D}, h = k), \quad (4.146)$$

где $p(h = k|\mathcal{D})$ – апостериорные веса смеси:

$$p(h = k|\mathcal{D}) = \frac{p(h = k)p(\mathcal{D}|h = k)}{\sum_{k'} p(h = k')p(\mathcal{D}|h = k')}. \quad (4.147)$$

Здесь величина $p(\mathcal{D}|h = k)$ – маргинальное правдоподобие k -й компоненты смеси (см. раздел 4.6.2.10).

Вернемся к примеру выше. Если имеется априорное распределение (4.144) и мы наблюдаем $N_1 = 20$ выпадений орла и $N_0 = 10$ выпадений решки, то по формуле (4.143) апостериорное распределение имеет вид:

$$p(\theta|\mathcal{D}) = 0.346 \text{Beta}(\theta|40, 30) + 0.654 \text{Beta}(\theta|30, 20). \quad (4.148)$$

Смотрите иллюстрацию на рис. 4.13.

Мы можем вычислить апостериорную вероятность смещения монеты в сторону орла следующим образом:

$$\Pr(\theta > 0.5|\mathcal{D}) = \sum_k \Pr(\theta > 0.5|\mathcal{D}, h = k)p(h = k|\mathcal{D}) = 0.9604. \quad (4.149)$$

В случае единственного априорного распределения $\text{Beta}(20,20)$ мы получили бы чуть меньшее значение $\Pr(\theta > 0.5|\mathcal{D}) = 0.8858$. Так что, если бы мы изначально подозревали, что казино может использовать асимметричную монету, то подтвердить свои опасения могли бы быстрее, чем если бы подходили непредвзято.

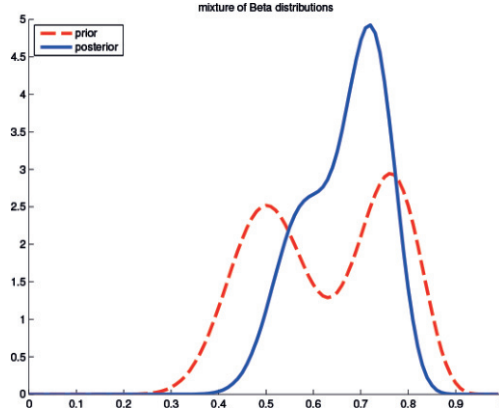


Рис. 4.13 ❖ Смесь двух бета-распределений.
Построено программой по адресу figures.probl.ai/book1/4.13

4.6.3. Дирихле-мультиномиальная модель

В этом разделе мы обобщим результаты из раздела 4.6.2, перейдя от бинарных случайных величин (например, подбрасывание монеты) к K -арным (например, бросание кости).

4.6.3.1. Правдоподобие

Пусть $Y \sim \text{Cat}(\theta)$ – дискретная случайная величина, выбранная из категориального распределения. Правдоподобие описывается формулой:

$$p(\mathcal{D}|\theta) = \prod_{n=1}^N \text{Cat}(y_n|\theta) = \prod_{n=1}^N \prod_{c=1}^C \theta_c^{\mathbb{I}(y_n=c)} = \prod_{c=1}^C \theta_c^{N_c}, \quad (4.150)$$

где $N_c = \sum_n \mathbb{I}(y_n = c)$.

4.6.3.2. Априорное распределение

Сопряженным априорным к категориальному распределению является **распределение Дирихле** – многомерное обобщение бета-распределения. Носителем его является **вероятностный симплекс**, определенный следующим образом:

$$S_K = \left\{ \theta : 0 \leq \theta_k \leq 1, \sum_{k=1}^K \theta_k = 1 \right\}. \quad (4.151)$$

Функция плотности распределения Дирихле имеет вид:

$$\text{Dir}(\theta|\tilde{\alpha}) \triangleq \frac{1}{B(\tilde{\alpha})} \prod_{k=1}^K \theta_k^{\tilde{\alpha}_k-1} \mathbb{I}(\theta \in S_K), \quad (4.152)$$

где $B(\tilde{\alpha})$ – многомерная бета-функция:

$$\frac{1}{B(\tilde{\alpha})} \triangleq \frac{\prod_{k=1}^K \Gamma(\tilde{\alpha}_k)}{\Gamma(\sum_{k=1}^K \tilde{\alpha}_k)}. \quad (4.153)$$

На рис. 4.14 изображено несколько графиков плотности распределения Дирихле при $K = 3$. Мы видим, что $\tilde{\alpha} = \sum_k \tilde{\alpha}_k$ определяет силу распределения (выраженность его пика), а $\tilde{\alpha}_k$ – местоположение пика. Например, $\text{Dir}(1, 1, 1)$ – равномерное распределение, $\text{Dir}(2, 2, 2)$ – широкое распределение с центром в точке $(1/3, 1/3, 1/3)$, а $\text{Dir}(20, 20, 20)$ – узкое распределение с центром в точке $(1/3, 1/3, 1/3)$. $\text{Dir}(3, 3, 20)$ – асимметричное распределение, плотность которого сосредоточена в одном из углов. Если $\tilde{\alpha}_k < 1$ для всех k , то мы получаем пики в узлах симплекса. Выборка из распределения с $\tilde{\alpha}_k < 1$ будет разреженной, как показано на рис. 4.15.

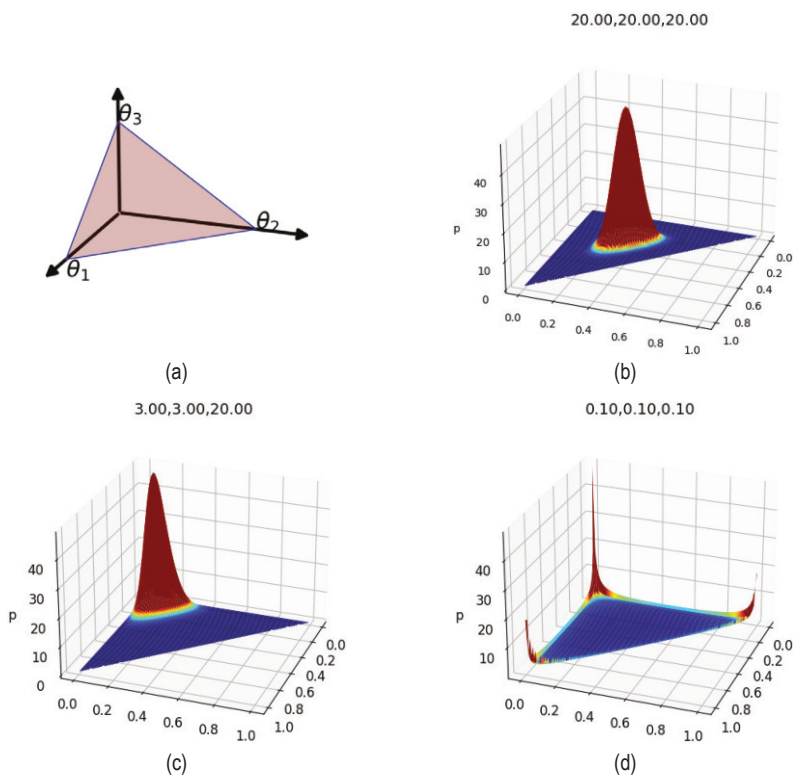


Рис. 4.14 ❖ (a) Распределение Дирихле при $K = 3$ определяет распределение на симплексе, которое можно представить тетраэдром. Точки на поверхности тетраэдра удовлетворяют условиям $0 \leq \theta_k \leq 1$ и $\sum_{k=1}^3 \theta_k = 1$. Построено программой по адресу figures.probml.ai/book1/4.14. (b) График плотности распределения Дирихле для $\tilde{\alpha} = (20, 20, 20)$. (c) График плотности распределения Дирихле для $\tilde{\alpha} = (3, 3, 20)$. (d) График плотности распределения Дирихле для $\tilde{\alpha} = (0.1, 0.1, 0.1)$. Построено программой по адресу figures.probml.ai/book1/4.14

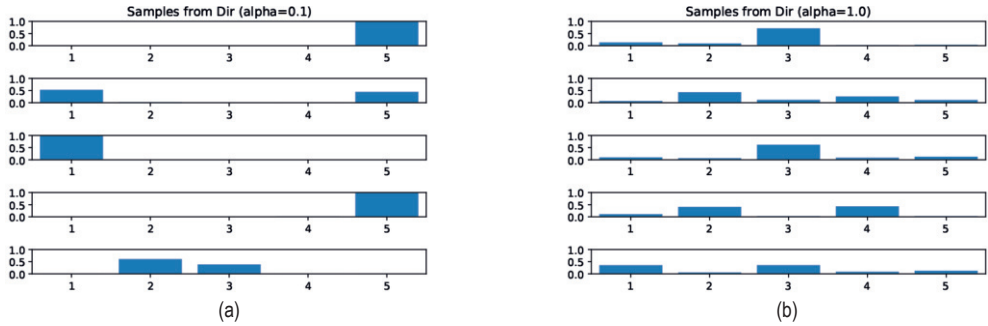


Рис. 4.15 ❖ Выборка из 5-мерного симметричного распределения Дирихле для различных значений параметров. (a) $\tilde{\alpha} = (0.1, 0.1, 0.1)$. Это приводит к очень разреженным распределениям со многими нулями. (b) $\tilde{\alpha} = (1, \dots, 1)$. Это приводит к более равномерным (и плотным) распределениям. Построено программой по адресу figures.probml.ai/book1/4.15

4.6.3.3. Апостериорное распределение

Мы можем объединить мультиномиальное правдоподобие и априорное распределение Дирихле и вычислить апостериорное распределение следующим образом:

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)\text{Dir}(\theta|\tilde{\alpha}) \quad (4.154)$$

$$= \left[\prod_k \theta_k^{N_k} \right] \left[\prod_k \theta_k^{\tilde{\alpha}_k - 1} \right] \quad (4.155)$$

$$= \text{Dir}(\theta|\tilde{\alpha}_1 + N_1, \dots, \tilde{\alpha}_K + N_K) \quad (4.156)$$

$$= \text{Dir}(\theta|\hat{\alpha}), \quad (4.157)$$

где $\hat{\alpha}_k = \tilde{\alpha}_k + N_k$ – параметры апостериорного распределения. Как видно, апостериорное распределение можно вычислить, прибавив эмпирические счетчики к априорным.

Среднее апостериорного распределения равно:

$$\bar{\theta}_k = \frac{\hat{\alpha}_k}{\sum_{k'=1}^K \hat{\alpha}_{k'}}. \quad (4.158)$$

Мода апостериорного распределения, соответствующая оценке MAP, равна:

$$\hat{\theta}_k = \frac{\hat{\alpha}_k - 1}{\sum_{k'=1}^K (\hat{\alpha}_{k'} - 1)}. \quad (4.159)$$

Если взять $\tilde{\alpha}_k$, соответствующие равномерному априорному распределению, то оценка MAP превращается в MLE:

$$\hat{\theta}_k = N_k/N. \quad (4.160)$$

(Более прямой вывод этого результата см. в разделе 4.2.4.)

4.6.3.4. Апостериорное прогнозное распределение

Апостериорное прогнозное распределение описывается формулой:

$$p(y = k|\mathcal{D}) = \int p(y = k|\theta)p(\theta|\mathcal{D})d\theta \quad (4.161)$$

$$= \int \theta_k p(\theta_k|\mathcal{D})d\theta_k = \mathbb{E}[\theta_k|\mathcal{D}] = \frac{\hat{\alpha}_k}{\sum_{k'} \hat{\alpha}_{k'}}. \quad (4.162)$$

Иначе говоря,

$$p(y|\mathcal{D}) = \text{Cat}(y|\bar{\theta}), \quad (4.163)$$

где $\bar{\theta} \triangleq \mathbb{E}[\theta|\mathcal{D}]$ – параметры апостериорного среднего. Если вместо этого подставить оценку MAP, то мы получим неприятную проблему нулевого счетчика. Единственный способ получить такой же эффект, как при сглаживании прибавлением единицы, – использовать оценку MAP с $\tilde{\alpha}_c = 2$.

Формула (4.162) дает вероятность одиночного будущего события при условии прошлых наблюдений $\mathbf{y} = (y_1, \dots, y_N)$. Иногда мы хотим знать вероятность наблюдения группы будущих данных, скажем $\tilde{\mathbf{y}} = (\tilde{y}_1, \dots, \tilde{y}_M)$. Ее можно вычислить следующим образом:

$$p(\tilde{\mathbf{y}}|\mathbf{y}) = \frac{p(\tilde{\mathbf{y}}, \mathbf{y})}{p(\mathbf{y})}. \quad (4.164)$$

В знаменателе находится маргинальное правдоподобие обучающих данных, а в числителе – маргинальное правдоподобие обучающих и будущих тестовых данных. Как вычислить эти маргинальные правдоподобия, мы обсудим в разделе 4.6.3.5.

4.6.3.5. Маргинальное правдоподобие

Рассуждая так же, как в разделе 4.6.2.10, можно показать, что маргинальное правдоподобие для Дирихле-категориальной модели имеет вид:

$$p(\mathcal{D}) = \frac{B(\mathbf{N} + \boldsymbol{\alpha})}{B(\boldsymbol{\alpha})}, \quad (4.165)$$

где

$$B(\boldsymbol{\alpha}) = \frac{\prod_{k=1}^K \Gamma(\alpha_k)}{\Gamma(\sum_k \alpha_k)}. \quad (4.166)$$

Следовательно, приведенный выше результат можно переписать в форме, которая обычно встречается в литературе:

$$p(\mathcal{D}) = \frac{\Gamma(\sum_k \alpha_k)}{\Gamma(N + \sum_k \alpha_k)} \prod_k \frac{\Gamma(N_k + \alpha_k)}{\Gamma(\alpha_k)}. \quad (4.167)$$

4.6.4. Гауссова-гауссова модель

В этом разделе мы выведем апостериорное распределение параметров гауссова распределения. Для простоты будем предполагать, что дисперсия известна (общий случай рассматривается во втором томе этой книги, а также в других стандартных учебниках по байесовской статистике).

4.6.4.1. Одномерный случай

Если σ^2 – известная постоянная, то правдоподобие μ имеет вид:

$$p(\mathcal{D}|\mu) \propto \exp\left[-\frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \mu)^2\right]. \quad (4.168)$$

Можно показать, что сопряженное априорное распределение тоже является гауссовым, $\mathcal{N}(\mu|\tilde{m}, \tilde{\tau}^2)$. Применяя формулу Байеса для гауссовых распределений, как в разделе 4.6.4.1, находим, что соответствующее апостериорное распределение описывается формулой:

$$p(\mu|\mathcal{D}, \sigma^2) = \mathcal{N}(\mu|\hat{m}, \hat{\tau}^2); \quad (4.169)$$

$$\hat{\tau}^2 = \frac{1}{\frac{N}{\sigma^2} + \frac{1}{\tilde{\tau}^2}} = \frac{\sigma^2 \tilde{\tau}^2}{N\sigma^2 + \tilde{\tau}^2}; \quad (4.170)$$

$$\hat{m} = \hat{\tau}^2 \left(\frac{\tilde{m}}{\tilde{\tau}^2} + \frac{N\bar{y}}{\sigma^2} \right) = \frac{\sigma^2}{N\tilde{\tau}^2 + \sigma^2} \tilde{m} + \frac{N\tilde{\tau}^2}{N\tilde{\tau}^2 + \sigma^2} \bar{y}, \quad (4.171)$$

где $\bar{y} \triangleq (1/N) \sum_{n=1}^N y_n$ – эмпирическое среднее.

Этот результат будет проще понять, если работать в терминах параметров точности, т. е. величин, обратных дисперсии. Именно, пусть $\kappa = 1/\sigma^2$ – точность наблюдения, а $\tilde{\lambda} = 1/\tilde{\tau}^2$ – точность априорного распределения. Тогда можно переписать апостериорное распределение следующим образом:

$$p(\mu|\mathcal{D}, \kappa) = \mathcal{N}(\mu|\hat{m}, \hat{\lambda}^{-1}); \quad (4.172)$$

$$\hat{\lambda} = \tilde{\lambda} + N\kappa; \quad (4.173)$$

$$\hat{m} = \frac{N\kappa\bar{y} + \tilde{\lambda}\tilde{m}}{\hat{\lambda}} = \frac{N\kappa}{N\kappa + \tilde{\lambda}} \bar{y} + \frac{\tilde{\lambda}}{N\kappa + \tilde{\lambda}} \tilde{m}. \quad (4.174)$$

Эти формулы интуитивно понятны: апостериорная точность $\hat{\lambda}$ равна априорной точности $\tilde{\lambda}$ плюс N единиц измерения точности κ . А апостериорное среднее является выпуклой комбинацией эмпирического среднего \bar{y} и априорного среднего \tilde{m} . Отсюда ясно, что апостериорное среднее – компромисс между эмпирическим и априорным средним. Если априорное среднее слабое, по сравнению с силой сигнала ($\tilde{\lambda}$ мало по сравнению с κ), то мы назначаем больший вес эмпирическому среднему. А если априорное среднее силь-

ное, по сравнению с силой сигнала ($\tilde{\lambda}$ велико по сравнению с κ), то больший вес назначается априорному среднему. Это показано на рис. 4.16. Заметим также, что апостериорное среднее записывается в терминах $N\kappa\bar{y}$, поэтому наличие N измерений с точностью κ каждое эквивалентно наличию одного измерения со значением \bar{y} точностью $N\kappa$.

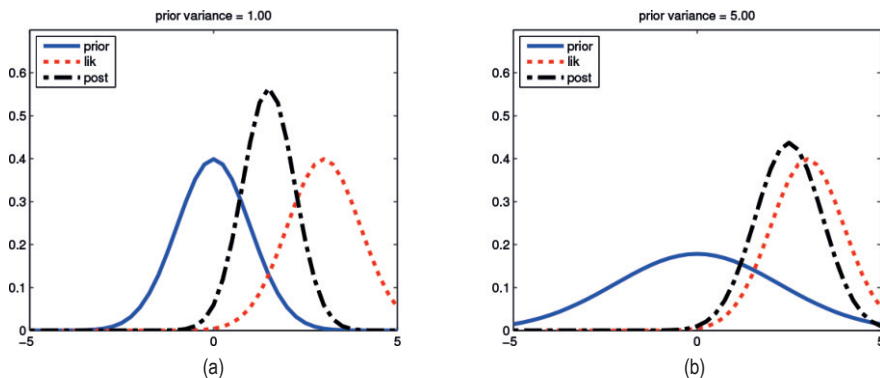


Рис. 4.16 ❖ Вывод среднего одномерного гауссова распределения с известным σ^2 при наличии наблюдения $y = 3$. (а) Используется сильное априорное распределение $p(\mu) = \mathcal{N}(\mu|0, 1)$. (б) Используется слабое априорное распределение $p(\mu) = \mathcal{N}(\mu|0, 5)$. Построено программой по адресу figures.problm.ai/book1/4.16

Апостериорное распределение после наблюдения $N = 1$ примера

Чтобы лучше прочувствовать эти формулы, рассмотрим апостериорное распределение после наблюдения единственной точки y (т. е. $N = 1$). Тогда апостериорное среднее можно записать следующими эквивалентными способами:

$$\hat{m} = \frac{\tilde{\lambda}}{\lambda} \tilde{m} + \frac{\kappa}{\lambda} y \quad (4.175)$$

$$= \tilde{m} + \frac{\kappa}{\lambda} (y - \tilde{m}) \quad (4.176)$$

$$= y - \frac{\tilde{\lambda}}{\lambda} (y - \tilde{m}). \quad (4.177)$$

В первом случае мы имеем выпуклую комбинацию априорного среднего и данных, во втором – априорное среднее, сдвинутое в сторону данных y , а в третьем – данные, сдвинутые к априорному среднему, это называется оценкой **усадки**. Будет проще, если мы определим вес $w = \tilde{\lambda}/\lambda$, т. е. отношение априорной точности к апостериорной. Тогда

$$\hat{m} = y - w(y - \tilde{m}) = (1 - w)y + w\tilde{m}. \quad (4.178)$$

Заметим, что для гауссова распределения апостериорное среднее совпадает с апостериорной модой. Поэтому мы можем использовать эти формулы для вычисления оценки MAP. Простой пример см. в упражнении 4.2.

Апостериорная дисперсия

Помимо апостериорного среднего или моды μ , нас может интересовать апостериорная дисперсия, дающая степень уверенности в нашей оценке. Квадратный корень из нее называется **стандартной ошибкой среднего**:

$$\text{se}(\mu) \triangleq \sqrt{\mathbb{V}[\mu|\mathcal{D}]}. \quad (4.179)$$

Предположим, что мы используем неинформативное априорное распределение для μ , положив $\tilde{\lambda} = 0$ (см. раздел 4.6.5.1). В этом случае апостериорное среднее равно MLE, $\hat{m} = \bar{y}$. Дополнительно предположим, что σ^2 аппроксимируется **выборочной дисперсией**:

$$s^2 \triangleq \frac{1}{N} \sum_{n=1}^N (y_n - \bar{y})^2. \quad (4.180)$$

Отсюда $\hat{\lambda} = N\hat{\kappa} = N/s^2$, так что стандартная ошибка среднего принимает вид:

$$\text{se}(\mu) = \sqrt{\mathbb{V}[\mu|\mathcal{D}]} = \frac{1}{\sqrt{\hat{\lambda}}} = \frac{s}{\sqrt{N}}. \quad (4.181)$$

Таким образом, мы видим, что неопределенность μ убывает со скоростью $1/\sqrt{N}$.

Кроме того, мы можем воспользоваться тем фактом, что 95 % массы гауссова распределения сосредоточено в интервале шириной 2 стандартных отклонения вокруг среднего, который аппроксимирует 95%-ный **байесовский доверительный интервал** μ :

$$I_{.95}(\mu|\mathcal{D}) = \bar{y} \pm 2 \frac{s}{\sqrt{N}}. \quad (4.182)$$

4.6.4.2. Многомерный случай

Для D -мерных данных правдоподобие имеет вид:

$$p(\mathcal{D}|\mu) = \prod_{n=1}^N \mathcal{N}(y_n|\mu, \Sigma) \quad (4.183)$$

$$= \frac{N}{(2\pi)^{D/2} |\Sigma|^{\frac{1}{2}}} \exp \left[-\frac{1}{2} \sum_{n=1}^N (\mathbf{y}_n - \mu)^\top \Sigma^{-1} (\mathbf{y}_n - \mu) \right] \quad (4.184)$$

$$= \mathcal{N}(\bar{\mathbf{y}}|\mu, \frac{1}{N} \Sigma), \quad (4.185)$$

где $\bar{\mathbf{y}} = 1/N \sum_{n=1}^N \mathbf{y}_n$. Таким образом, мы можем заменить множество наблюдений их средним и поделить ковариацию на N .

Для простоты будем использовать сопряженное априорное распределение, которое в данном случае является гауссовым, а именно:

$$p(\boldsymbol{\mu}) = \mathcal{N}(\boldsymbol{\mu}|\tilde{\mathbf{m}}, \tilde{\mathbf{V}}). \quad (4.186)$$

Мы можем вывести апостериорное гауссово распределение $\boldsymbol{\mu}$, опираясь на результаты из раздела 3.3.1. Получаем

$$p(\boldsymbol{\mu}|\mathcal{D}, \boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{\mu}|\hat{\mathbf{m}}, \hat{\mathbf{V}}); \quad (4.187)$$

$$\hat{\mathbf{V}}^{-1} = \tilde{\mathbf{V}}^{-1} + N\boldsymbol{\Sigma}^{-1}; \quad (4.188)$$

$$\hat{\mathbf{m}} = \hat{\mathbf{V}}(\boldsymbol{\Sigma}^{-1}(N\bar{\mathbf{y}}) + \tilde{\mathbf{V}}^{-1}\tilde{\mathbf{m}}). \quad (4.189)$$

На рис. 4.17 приведен пример в двумерном случае.

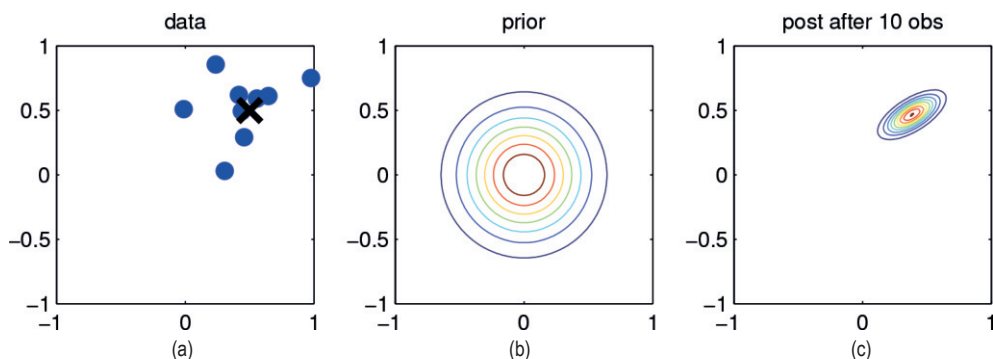


Рис. 4.17 ❖ Байесовский вывод для среднего двумерного гауссова распределения. (а) Данные выбираются из распределения $\mathbf{y}_n \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, где $\boldsymbol{\mu} = [0.5, 0.5]^T$ и $\boldsymbol{\Sigma} = 0.1[2, 1; 1, 1]$. (б) Априорное распределение имеет вид $p(\boldsymbol{\mu}) = \mathcal{N}(\boldsymbol{\mu}|\mathbf{0}, 0.1\mathbf{I}_2)$. (с) Показано апостериорное распределение после наблюдения 10 точек. Построено программой по адресу figures.problai.com/book1/4.17

4.6.5. За пределами сопряженных априорных распределений

Мы видели различные примеры сопряженных априорных распределений, и все они принадлежали экспоненциальному семейству (см. раздел 3.4). Преимуществом этих априорных распределений является простота интерпретации (в терминах достаточных статистик на основе виртуального априорного набора данных) и простота вычислений. Однако для большинства моделей в экспоненциальном семействе не существует априорного распределения, сопряженного правдоподобию. Более того, даже если такое сопряженное априорное распределение существует, предположение о сопряженности может оказаться чрезмерно ограничительным. Поэтому в следующих разделах мы кратко обсудим другие виды априорных распределений.

4.6.5.1. Неинформативные априорные распределения

Если мы ничего или почти ничего не знаем о предметной области, то желательно использовать **неинформативное**, или **объективное** априорное распределение, чтобы «данные говорили сами за себя». Например, если мы хотим вывести вещественную величину, скажем параметр сдвига $\mu \in \mathbb{R}$, то можем использовать **плоское априорное распределение** $p(\mu) \propto 1$. Его можно рассматривать как «бесконечно широкое» гауссово распределение.

К сожалению, неинформативное распределение не единственно, и все они на самом деле кодируют какие-то знания. Поэтому лучше использовать термин **диффузное**, **минимально информативное** или **подразумеваемое по умолчанию** априорное распределение. Дополнительные сведения см. во втором томе этой книги.

4.6.5.2. Иерархические априорные распределения

Байесовские модели требуют априорного распределения параметров $p(\theta)$. Параметры априорного распределения называются **гиперпараметрами** и обозначаются ϕ . Если они неизвестны, то можно задать для них априорное распределение; тем самым определяется **иерархическая байесовская модель**, или **многоуровневая модель**, которую можно визуализировать следующим образом: $\phi \rightarrow \theta \rightarrow \mathcal{D}$. Мы предполагаем, что априорное распределение гиперпараметров фиксировано (т. е. можно использовать какое-то минимально информативное априорное распределение), так что совместное распределение имеет вид:

$$p(\phi, \theta, \mathcal{D}) = p(\phi)p(\theta|\phi)p(\mathcal{D}|\theta). \quad (4.190)$$

Мы надеемся, что сможем обучить гиперпараметры, рассматривая сами параметры как данные. Это полезно, когда имеется несколько взаимосвязанных параметров, подлежащих оцениванию (например, взятых из различных частей генеральной совокупности или из нескольких задач); это подает обучающий сигнал верхнему уровню модели. Детали см. во втором томе, [Mur22].

4.6.5.3. Эмпирические априорные распределения

В разделе 4.6.5.2 мы обсудили иерархические байесовские модели как способ вывода параметров из данных. К сожалению, вывод апостериорного распределения в таких моделях может столкнуться с вычислительными трудностями. В этом разделе мы обсудим вычислительно удобную аппроксимацию, в которой сначала вычисляется точечная оценка гиперпараметров $\hat{\phi}$, а затем вычисляется условное апостериорное распределение, $p(\theta|\hat{\phi}, \mathcal{D})$, а не совместное апостериорное распределение, $p(\theta, \phi|\mathcal{D})$.

Для оценивания гиперпараметров мы можем максимизировать маргинальное правдоподобие:

$$\hat{\phi}_{\text{mml}}(\mathcal{D}) = \underset{\phi}{\operatorname{argmax}} p(\mathcal{D}|\phi) = \underset{\phi}{\operatorname{argmax}} \int p(\mathcal{D}|\theta)p(\theta|\phi)d\theta. \quad (4.191)$$

Этот метод называется оценкой **максимального правдоподобия типа II**, поскольку оптимизируются гиперпараметры, а не сами параметры. Оценив $\hat{\phi}$, мы можем вычислить апостериорное распределение $p(\theta|\hat{\phi}, \mathcal{D})$ обычным образом.

Поскольку параметры априорного распределения оцениваются по данным, этот подход называется **эмпирическим байесовским (ЕВ)** [CL96]. Он нарушает принцип, согласно которому априорное распределение следует выбирать независимо от данных. Однако его можно рассматривать как вычислительно недорогую аппроксимацию вывода в полной иерархической байесовской модели, точно так же как оценка MAP рассматривалась в качестве аппроксимации вывода в одноуровневой модели $\theta \rightarrow \mathcal{D}$. На самом деле можно построить иерархию, при которой чем длиннее интеграл, тем «более байесовской» является модель (см. ниже).

Метод	Определение
Максимальное правдоподобие	$\hat{\theta} = \operatorname{argmax}_{\theta} p(\mathcal{D} \theta)$
Оценка MAP	$\hat{\theta} = \operatorname{argmax}_{\theta} p(\mathcal{D} \theta)p(\theta \phi)$
ML-II (эмпирическая байесовская)	$\hat{\phi} = \operatorname{argmax}_{\phi} \int p(\mathcal{D} \theta)p(\theta \phi)d\theta$
MAP-II	$\hat{\phi} = \operatorname{argmax}_{\phi} \int p(\mathcal{D} \theta)p(\theta \phi)p(\phi)d\theta$
Полная байесовская	$p(\theta, \phi \mathcal{D}) \propto p(\mathcal{D} \theta)p(\theta \phi)p(\phi)$

Заметим, что у ML-II меньше шансов оказаться переобученной, чем у «регулярной» модели максимального правдоподобия, потому что обычно гиперпараметров ϕ меньше, чем параметров θ . Детали см. во втором томе.

4.6.6. Байесовские доверительные интервалы

Апостериорное распределение обычно является многомерным объектом, который трудно визуализировать и с которым нелегко работать. Стандартный способ обобщить такое распределение состоит в том, чтобы вычислить точечную оценку, например апостериорное среднее или моду, а затем вычислить **байесовский доверительный интервал**, который количественно характеризует неопределенность этой оценки. (Байесовский доверительный интервал (credible interval) не то же самое, что доверительный интервал (confidence interval), который является понятием из частотной статистики и обсуждается в разделе 4.7.4.)

Точнее, мы определим $100(1 - \alpha)$ -процентный байесовский доверительный интервал как непрерывную область $C = (\ell, u)$ (нижняя и верхняя границы), в которой сосредоточено $1 - \alpha$ массы апостериорной вероятности, т. е.:

$$C_{\alpha}(\mathcal{D}) = (\ell, u) : P(\ell \leq \theta \leq u|\mathcal{D}) = 1 - \alpha. \quad (4.192)$$

Может существовать много интервалов, удовлетворяющих условию (4.192), поэтому обычно мы выбираем такой, для которого в каждом хвосте сосредоточена масса $(1 - \alpha)/2$; он называется **центральным интервалом**.

Если апостериорное распределение имеет известную функциональную форму, то центральный интервал можно вычислить по формулам $\ell = F^{-1}(\alpha/2)$ и $u = F^{-1}(1 - \alpha/2)$, где F – функция апостериорного распределения, а F^{-1} – обратная к ней. Например, если апостериорное распределение гауссово, $p(\theta|\mathcal{D}) = \mathcal{N}(0, 1)$ и $\alpha = 0.05$, то имеем $\ell = \Phi^{-1}(\alpha/2) = -1.96$ и $u = \Phi^{-1}(1 - \alpha/2) = 1.96$, где Φ обозначает функцию гауссова распределения. Это показано на рис. 2.2b и является обоснованием распространенной практики представления байесовского доверительного интервала в виде $\mu \pm 2\sigma$, где μ – апостериорное среднее, σ – апостериорное стандартное отклонение, а 2 – хорошая аппроксимация 1.96.

В общем случае вычислить обратную функцию апостериорного распределения зачастую бывает трудно. Тогда имеется простая альтернатива – произвести выборку из апостериорного распределения и использовать аппроксимацию Монте-Карло для вычисления апостериорных квантилей: мы просто сортируем S примеров и берем тот, который оказался в позиции α/S отсортированного списка. При $S \rightarrow \infty$ эта процедура сходится к истинному квантилю (см. демонстрацию с помощью программы по адресу code.probl.ai/book1/beta_credible_int_demo).

С центральными интервалами связана одна проблема: вне центрального интервала могут существовать точки, в которых вероятность выше, чем во внутренних точках, как показано на рис. 4.18a. Поэтому вводится альтернативная величина – область **наивысшей апостериорной плотности** (highest posterior density – **HPD**), т. е. множество точек, в которых вероятность превышает некоторый порог. Точнее, мы ищем порог p^* такой, что

$$1 - \alpha = \int_{\theta: p(\theta|\mathcal{D}) > p^*} p(\theta|\mathcal{D}) d\theta, \quad (4.193)$$

а затем определяем HPD:

$$C_\alpha(\mathcal{D}) = \{\theta : p(\theta|\mathcal{D}) \geq p^*\}. \quad (4.194)$$

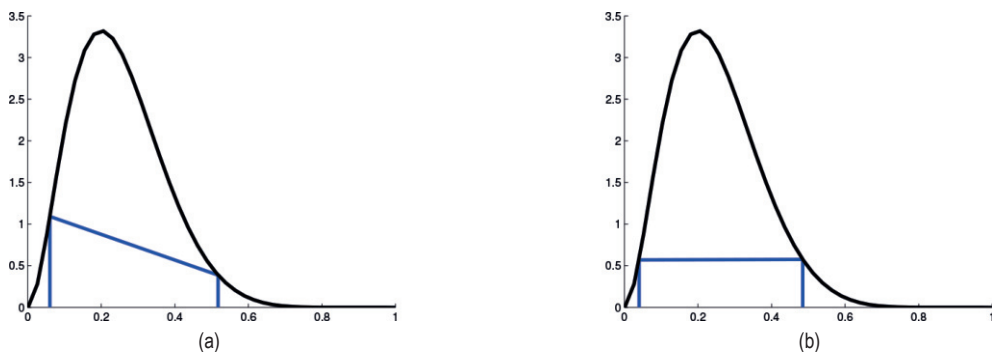


Рис. 4.18 ❖ (a) Центральный интервал (CI) и (b) область наивысшей апостериорной плотности (HPD) для апостериорного распределения $\text{Beta}(3,9)$. CI занимает интервал (0.06, 0.52), а HPD – интервал (0.04, 0.48). На основе рис. 3.6 из работы [Hof09]. Сгенерировано программой по адресу figures.probl.ai/book1/4.18

В одномерном случае область HPD иногда называют **интервалом наивысшей плотности** (highest density interval – **HDI**). Например, на рис. 4.18b показан 95%-ный HDI распределения Beta(3, 9), равный (0.04, 0.48). Мы видим, что он уже центрального интервала, хотя все равно содержит 95 % массы; кроме того, для любой точки внутри него плотность выше, чем для любой внешней точки.

Для унимодального распределения HDI будет самым узким интервалом вокруг моды, содержащим 95 % массы. Чтобы убедиться в этом, представьте «заливку водой» наоборот, когда сначала все находится под водой, а потом мы понижаем уровень, пока 95 % массы не окажется сверху и лишь 5 % под водой. Эта метафора дает простой алгоритм вычисления HDI в одномерном случае; просто найти такие точки, что интервал содержит 95 % массы и имеет минимальную ширину. Это можно сделать с помощью методов одномерной численной оптимизации, если известна обратная функция распределения, или посредством поиска среди отсортированных примеров при наличии выборки (см. код по адресу code.problml.ai/book1/betaHPD).

Если апостериорное распределение мультимодальное, то HDI может даже не быть связной областью, см., например, рис. 4.19b. Однако искать обобщающие характеристики мультимодальных апостериорных распределений всегда трудно.

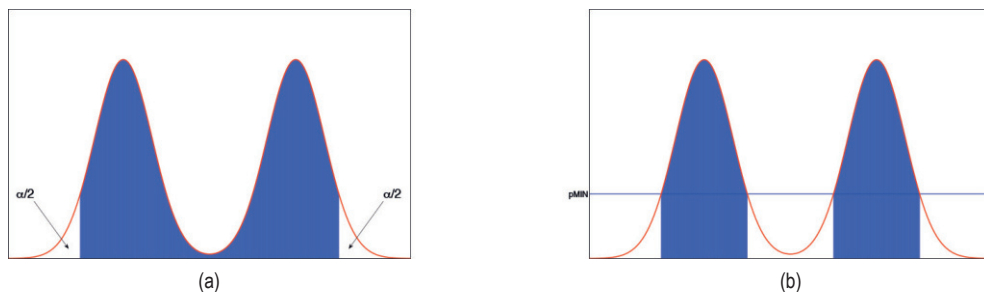


Рис. 4.19 ❖ (a) Центральный интервал и (b) область HPD для гипотетического мультимодального апостериорного распределения. На основе рис. 2.2 из работы [Gel+04]. Построено программой по адресу figures.problml.ai/book1/4.19

4.6.7. Байесовское машинное обучение

До сих пор мы имели дело с безусловными моделями вида $p(\mathbf{y}|\boldsymbol{\theta})$. В машинном обучении с учителем используются условные модели вида $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$. Апостериорное распределение параметров теперь имеет вид $p(\boldsymbol{\theta}|\mathcal{D})$, где $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n) : n = 1 \dots N\}$. Для вычисления этого апостериорного распределения можно воспользоваться рассмотренными выше принципами. Этот подход называется **байесовским машинным обучением**, поскольку мы «по-байесовски» относимся к параметрам модели.

4.6.7.1. Подстановочная аппроксимация

Вычислив апостериорное распределение параметров, мы можем вычислить апостериорное прогнозное распределение выходов при известных входах, исключив неизвестные параметры путем маргинализации:

$$p(y|x, \mathcal{D}) = \int p(y|x, \theta) p(\theta|\mathcal{D}) d\theta. \quad (4.195)$$

Конечно, вычисление этого интеграла чаще всего является вычислительно неразрешимой задачей. Очень простая аппроксимация предполагает, что существует единственная наилучшая модель, $\hat{\theta}$, например MLE. Это эквивалентно аппроксимации апостериорного распределения бесконечно узким и бесконечно высоким пиком в выбранной точке. Это можно записать следующим образом:

$$p(\theta|\mathcal{D}) = \delta(\theta - \hat{\theta}), \quad (4.196)$$

где δ – дельта-функция Дирака (см. раздел 2.6.5). Если воспользоваться этой аппроксимацией, то прогнозное распределение можно получить, просто подставив точечную оценку в правдоподобие:

$$p(y|x, \mathcal{D}) = \int p(y|x, \theta) p(\theta|\mathcal{D}) d\theta \approx \int p(y|x, \theta) \delta(\theta - \hat{\theta}) d\theta = p(y|x, \hat{\theta}). \quad (4.197)$$

Это следует из фильтрующего свойства дельта-функции (формула (2.129)).

Подход, предлагаемый формулой (4.197), называется **подстановочной аппроксимацией** (plug-in approximation). Он эквивалентен стандартному подходу, применяемому в большей части машинного обучения, когда мы сначала обучаем модель (т. е. вычисляем точечную оценку $\hat{\theta}$), а затем используем ее для предсказания. Однако стандартный (подстановочный) подход может быть подвержен переобучению и излишней уверенности, как объяснялось в разделе 1.2.3. Байесовский же подход избегает этих напастей благодаря исключению параметров с помощью маргинализации, но это может дорого стоить. К счастью, даже простые аппроксимации путем усреднения по нескольким правдоподобным значениям параметров могут повысить качество. Ниже будут приведены соответствующие примеры.

4.6.7.2. Пример: скалярный вход, бинарный выход

Пусть требуется выполнить бинарную классификацию, т. е. $y \in \{0, 1\}$. Будем использовать модель вида

$$p(y|x, \theta) = \text{Ber}(y|\sigma(\mathbf{w}^T \mathbf{x} + b)), \quad (4.198)$$

где

$$\sigma(a) \triangleq \frac{e^a}{1 + e^a} \quad (4.199)$$

– **сигмоидная**, или **логистическая**, функция, отображающая $\mathbb{R} \rightarrow [0, 1]$, а $\text{Ber}(y|\mu)$ – распределение Бернулли со средним μ (детали см. в разделе 2.4). Иными словами,

$$p(y = 1|x; \theta) = \sigma(\mathbf{w}^\top \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^\top \mathbf{x} + b)}}. \quad (4.200)$$

Эта модель называется **логистической регрессией**. (Мы обсудим ее подробнее в главе 10.)

Применим эту модель к задаче ответа на вопрос, относится ли ирис к типу *Setosa* или *Versicolor*, $y_n \in \{0, 1\}$, если известна длина чашелистика, x_n . (Описание набора данных об ирисах см. в разделе 1.2.1.1.)

Сначала обучим модель одномерной логистической регрессии вида

$$p(y = 1|x, \theta) = \sigma(b + wx) \quad (4.201)$$

на наборе данных $\mathcal{D} = \{(x_n, y_n)\}$, применив оценку максимального правдоподобия. (О том, как вычисляется MLE для этой модели, см. раздел 10.2.3.) На рис. 4.20а показана подстановочная аппроксимация апостериорного прогнозного распределения $p(y = 1|x, \hat{\theta})$, где $\hat{\theta}$ – MLE параметров. Мы видим, что уверенность в принадлежности цветка типа *Versicolor* возрастает по мере увеличения длины чашелистика; это представлено сигмоидной (S-образной) логистической функцией.

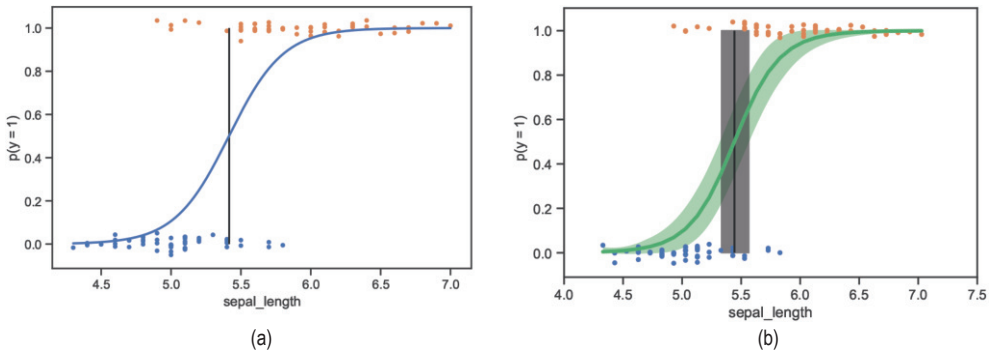


Рис. 4.20 ❖ (а) Логистическая регрессия для классификации ирисов по типу *Versicolor* ($y = 1$) или *Setosa* ($y = 0$) с использованием единственного входного признака x – длины чашелистика. Помеченные точки слегка разнесены по вертикали, чтобы избежать чрезмерного наложения. Вертикальная прямая является решающей границей. Построено программой по адресу figures.probml.ai/book1/4.20. (б) То же, что (а), но показано апостериорное распределение. На основе рис. 4.4 из работы [Mar18]. Построено программой по адресу figures.probml.ai/book1/4.20

Решающей границей по определению называется такое входное значение x^* , для которого $p(y = 1|x^*; \hat{\theta}) = 0.5$. Решая это уравнение, получаем:

$$\sigma(b + wx^*) = \frac{1}{1 + e^{-(b + wx^*)}} = \frac{1}{2}; \quad (4.202)$$

$$b + wx^* = 0; \quad (4.203)$$

$$x^* = -\frac{b}{w}. \quad (4.204)$$

На рис. 4.20a видно, что $x^* \approx 5.5$ см.

Однако описанный подход не позволяет смоделировать неопределенность нашей оценки параметров, а значит, игнорирует индуцированную неопределенность выходных вероятностей и положения решающей границы. Чтобы отразить эту дополнительную неопределенность, мы можем применить байесовский подход и аппроксимировать апостериорное распределение $p(\theta|\mathcal{D})$ (детали см. в разделе 10.5). Зная его, мы можем найти приближенное апостериорное прогнозное распределение с помощью аппроксимации Монте-Карло:

$$p(y = 1|x, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S p(y = 1|x, \theta^s), \quad (4.205)$$

где $\theta^s \sim p(\theta|\mathcal{D})$ – выборка из апостериорного распределения. На рис. 4.20b показаны графики среднего и 95%-го байесовского доверительного интервала этой функции. Мы видим, что теперь для каждого входного значения имеется диапазон предсказанных вероятностей. Применив аппроксимацию Монте-Карло, можно также вычислить распределение местоположения решающей границы:

$$p(x^*|\mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S \delta\left(x^* - \left(-\frac{b^s}{w^s}\right)\right), \quad (4.206)$$

где $(b^s, w^s) = \theta^s$. 95%-ный байесовский доверительный интервал для этого распределения представлен широкой вертикальной полосой на рис. 4.20b.

Хотя такое тщательное моделирование неопределенности в данном приложении не особенно обязательно, оно может быть важно в приложениях, связанных с рисками, например в здравоохранении и финансах. Мы обсудим это в главе 5.

4.6.7.3. Пример: бинарный вход, скалярный выход

Теперь рассмотрим задачу о предсказании времени доставки бандероли, $y \in \mathbb{R}$, компаниями А и В. Идентификатор компании можно представить бинарным признаком $x \in \{0, 1\}$, где $x = 0$ означает компанию А, а $x = 1$ – компанию В. Будем использовать для решения этой задачи следующую дискриминантную модель:

$$p(y|x, \theta) = \mathcal{N}(y|\mu_x, \sigma_x^2), \quad (4.207)$$

где $\mathcal{N}(y|\mu, \sigma^2)$ – гауссово распределение

$$\mathcal{N}(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-\mu)^2}, \quad (4.208)$$

а $\theta = (\mu_0, \mu_1, \sigma_0, \sigma_1)$ – параметры модели. Эту модель можно обучить, применив оценку максимального правдоподобия, как было описано в разделе 4.2.5; альтернативно можно применить байесовский подход, рассмотренный в разделе 4.6.4.

Преимущество байесовского подхода в том, что, улавливая неопределенность параметров θ , мы также улавливаем неопределенность наших предсказаний $p(y|x, \mathcal{D})$, тогда как подстановочная аппроксимация $p(y|x, \hat{\theta})$ не включает эту неопределенность. Например, предположим, что мы воспользовались услугами каждой компании всего по одному разу, так что обучающий набор имеет вид $\mathcal{D} = \{(x_1 = 0, y_1 = 15), (x_2 = 1, y_2 = 20)\}$. Как было показано в разделе 4.2.5, MLE среднего совпадает с эмпирическим средним, $\hat{\mu}_0 = 15$ и $\hat{\mu}_1 = 20$, но MLE стандартного отклонения будет равно нулю, $\hat{\sigma}_0 = \hat{\sigma}_1 = 0$, так как имеется лишь один пример из каждого «класса». В результате подстановочное предсказание вообще не улавливает неопределенность.

Чтобы понять, почему моделирование неопределенности так важно, рассмотрим рис. 4.21. Мы видим, что ожидаемое время доставки (ETA) для компании А меньше, чем для компании В, однако дисперсия распределения А выше, что делает ее выбор рискованным, если нам нужна уверенность в том, что бандероль прибудет вовремя. (Дополнительные сведения о выборе оптимальных действий в условиях неопределенности см. в главе 5.)

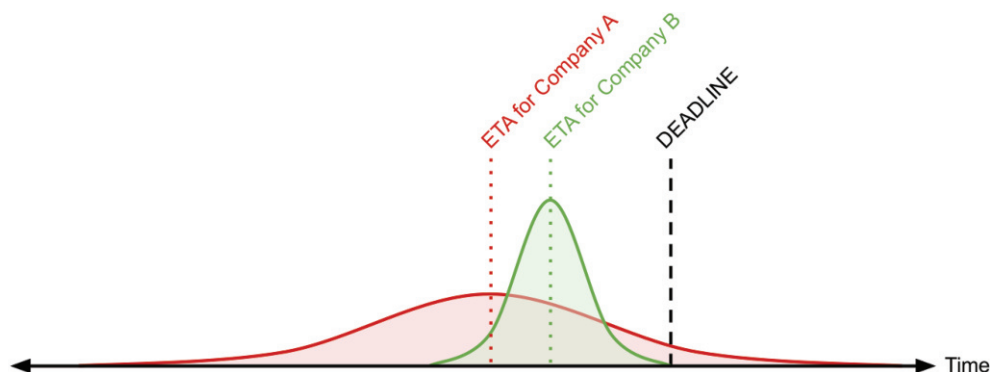


Рис. 4.21 ❖ Распределение времени доставки для двух транспортных компаний. ETA – ожидаемое время прибытия. Распределение для А имеет более высокую неопределенность, так что выбор этой компании сулит большие риски. Взято из статьи <https://bit.ly/39bc4XL>. Печатается с разрешения Брендана Хаса

Конечно, это нереалистичный пример, потому что мы взяли всего по одному примеру для каждой транспортной компании. Но такого рода проблемы встречаются и при наличии нескольких примеров входов определенного

вида, например когда в данных имеется длинный хвост новых образцов, скажем не встречавшихся ранее комбинаций слов или категориальных признаков.

4.6.7.4. Вертикальное масштабирование

Оба описанных выше примера были очень простыми: одномерный вход и выход и всего 2–4 параметра. В большинстве встречающихся на практике задач входные данные многомерные, выходные тоже иногда многомерные, поэтому количество параметров моделей велико. К сожалению, вычисление апостериорного распределения $p(\theta|\mathcal{D})$ и апостериорного прогнозного распределения $p(y|x, \mathcal{D})$ для многих моделей сталкивается с вычислительными трудностями. Мы обсудим эту проблему в разделе 4.6.8.

4.6.8. Вычислительные трудности

Если известны правдоподобие $p(\mathcal{D}|\theta)$ и априорное распределение $p(\theta)$, то апостериорное распределение $p(\theta|\mathcal{D})$ можно вычислить по формуле Байеса. Однако это вычисление обычно практически неосуществимо, если не считать простых частных случаев, например сопряженных моделей (раздел 4.6.1) или моделей, в которых все латентные переменные берутся из небольшого конечного множества допустимых значений. Поэтому апостериорное распределение необходимо аппроксимировать. Существует много методов **приближенного апостериорного вывода**, отличающихся тем, чему отдается предпочтение: верности, простоте или скорости. Ниже мы кратко опишем некоторые алгоритмы, а детали оставим до второго тома, [Mur22] (см. также работу [MFR20], где приведен обзор методов приближенного вывода, начиная с оригинального метода Байеса, датированного 1763 годом).

В качестве сквозного примера будем использовать задачу аппроксимации апостериорного распределения в бета-бернуллиевой модели. Точнее, наша цель – аппроксимировать

$$p(\theta|\mathcal{D}) \propto \left[\prod_{n=1}^N \text{Bin}(y_n|\theta) \right] \text{Beta}(\theta|1,1), \quad (4.209)$$

где \mathcal{D} содержит 10 выпадений орла и 1 выпадение решки (общее число наблюдений $N = 11$), а априорное распределение равномерное. В данном случае апостериорное распределение можно вычислить точно (см. рис. 4.22) с помощью метода из раздела 4.6.2, но у этого примера имеется полезная педагогическая задача – сравнить аппроксимацию с точным ответом. Кроме того, поскольку целевое распределение одномерное, результаты легко визуализировать. (Заметим, однако, что проблема не совсем тривиальна, потому что апостериорное распределение сильно асимметрично вследствие дисбаланса выборки: 10 орлов и 1 решка.)

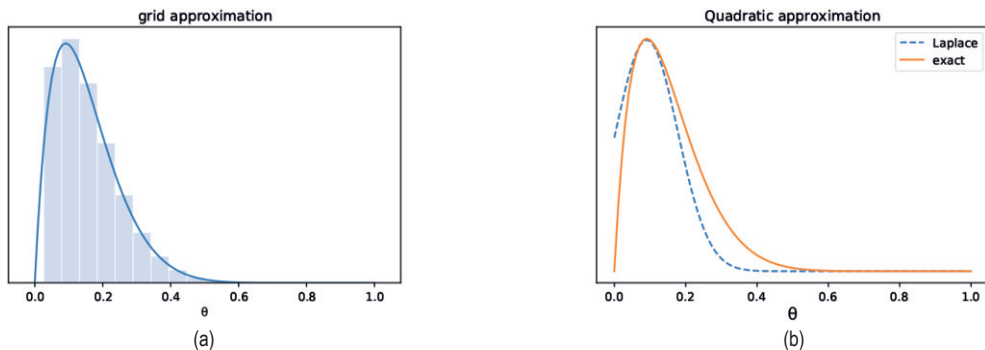


Рис. 4.22 ❖ Аппроксимация апостериорного распределения бета-бернуллиевой модели. (а) Сеточная аппроксимация на сетке с 20 точками. (б) Аппроксимация Лапласа. Построено программой по адресу figures.probl.ai/book1/4.22

4.6.8.1. Сеточная аппроксимация

Простейший подход к аппроксимации вывода апостериорного распределения – разбить пространство возможных значений неизвестных на конечное множество $\theta_1, \dots, \theta_K$, а затем аппроксимировать апостериорное распределение полным перебором:

$$p(\theta = \theta_k | \mathcal{D}) \approx \frac{p(\mathcal{D} | \theta_k) p(\theta_k)}{p(\mathcal{D})} = \frac{p(\mathcal{D} | \theta_k) p(\theta_k)}{\sum_{k'=1}^K p(\mathcal{D}, \theta_{k'})}. \quad (4.210)$$

Это называется **сеточной аппроксимацией**. На рис. 4.22а этот метод показан применительно к нашей одномерной задаче. Как видим, он легко позволяет уловить асимметричность распределения. К сожалению, этот подход не масштабируется на задачи размерности больше 2 или 3, потому что число узлов сетки экспоненциально растет с увеличением числа измерений.

4.6.8.2. Квадратичная аппроксимация (Лапласа)

В этом разделе мы обсудим простой способ аппроксимации апостериорного распределения с применением многомерного гауссова распределения; он называется **аппроксимацией Лапласа** или **квадратичной аппроксимацией** (см., например, [TK86; RMC09]).

Для вывода запишем апостериорное распределение в следующем виде:

$$p(\theta | \mathcal{D}) = \frac{1}{Z} e^{-\Psi(\theta)}, \quad (4.211)$$

где $\Psi(\theta) = -\log p(\theta, \mathcal{D})$ называется **функцией энергии**, а $Z = p(\mathcal{D})$ – нормировочная постоянная. Раскладывая в ряд Тейлора в окрестности моды $\hat{\theta}$ (т. е. состояния с минимальной энергией), получаем

$$\Psi(\boldsymbol{\theta}) \approx \Psi(\hat{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \mathbf{g} + \frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \mathbf{H}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}), \quad (4.212)$$

где \mathbf{g} – градиент в точке моды, а \mathbf{H} – гессиан. Поскольку $\hat{\boldsymbol{\theta}}$ – мода, градиент обращается в ноль. Следовательно,

$$\hat{p}(\boldsymbol{\theta}, \mathcal{D}) = e^{-\Psi(\hat{\boldsymbol{\theta}})} \exp\left[-\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \mathbf{H}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})\right]; \quad (4.213)$$

$$\hat{p}(\boldsymbol{\theta}, \mathcal{D}) = \frac{1}{Z} \hat{p}(\boldsymbol{\theta}, \mathcal{D}) = \mathcal{N}(\boldsymbol{\theta} | \hat{\boldsymbol{\theta}}, \mathbf{H}^{-1}); \quad (4.214)$$

$$Z = e^{-\Psi(\hat{\boldsymbol{\theta}})} (2\pi)^{D/2} |\mathbf{H}|^{-\frac{1}{2}}. \quad (4.215)$$

Последняя строчка вытекает из нормировочной постоянной многомерного гауссова распределения.

Аппроксимацию Лапласа легко применять, потому что можно воспользоваться имеющимися алгоритмами оптимизации для вычисления оценки МАР, а затем останется только вычислить гессиан в точке моды. (В многомерных пространствах можно использовать диагональную аппроксимацию.)

На рис. 4.22b показано применение этого метода к нашей одномерной задаче. К сожалению, аппроксимация получилась не особенно хорошей. Объясняется это тем, что апостериорное распределение асимметрично, а гауссиана симметрична. Кроме того, интересующий нас параметр находится в ограниченном интервале $\theta \in [0, 1]$, тогда как гауссово распределение предполагает неограниченное пространство $\theta \in \mathbb{R}$. По счастью, вторую проблему можно решить с помощью замены переменной. Например, в данном случае можно применить аппроксимацию Лапласа к $\alpha = \text{logit}(\theta)$. Это стандартный прием для упрощения вывода.

4.6.8.3. Вариационная аппроксимация

В разделе 4.6.8.2 мы обсуждали аппроксимацию Лапласа, в которой для нахождения оценки МАР используется процедура оптимизации, а затем кривизна апостериорного распределения в этой точке аппроксимируется с помощью гессиана. В этом разделе мы обсудим **вариационный вывод** (variational inference – **VI**) – другой основанный на оптимизации подход к выводу апостериорного распределения, обладающий гораздо большей гибкостью с точки зрения моделирования (а потому дающий гораздо более точную аппроксимацию).

В VI делается попытка аппроксимировать не поддающееся прямому вычислению распределение вероятностей, например $p(\boldsymbol{\theta} | \mathcal{D})$, распределением, допускающим вычисление, $q(\boldsymbol{\theta})$, так чтобы доставить минимум некоторому расхождению D между распределениями:

$$q^* = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} D(q, p), \quad (4.216)$$

где \mathcal{Q} – вычислимое семейство распределений (например, многомерных гауссовых). Если определить D как расхождение КЛ (см. раздел 6.2), то можно будет вывести нижнюю границу логарифмического маргинального правдоподобия; это величина называется **вариационной нижней оценкой** (evidence lower bound – ELBO). Максимизируя ELBO, мы можем улучшить качество аппроксимации апостериорного распределения. Детали см. во втором томе этой книги, [Mur22].

4.6.8.4. Аппроксимация методом Монте-Карло по схеме марковских цепей

Хотя VI – быстрый метод на основе оптимизации, он может давать смещенную оценку апостериорного распределения, так как ограничен конкретной функциональной формой $q \in \mathcal{Q}$. Более гибкий подход – использовать непараметрическую аппроксимацию в терминах множества примеров, $q(\theta) \approx \frac{1}{S} \sum_{s=1}^S \delta(\theta - \theta^s)$. Это называется **аппроксимацией Монте-Карло** апостериорного распределения. Основная проблема – как эффективно произвести выборку из апостериорного распределения $\theta^s \sim p(\theta|\mathcal{D})$, не прибегая к вычислению нормировочной постоянной $p(\mathcal{D}) = \int p(\theta, \mathcal{D}) d\theta$. Стандартный подход к решению этой проблемы называется **методом Монте-Карло по схеме марковских цепей** (Markov chain Monte Carlo – МСМС). Если дополнить этот алгоритм основанной на градиенте информацией, выведенной из $\nabla \log p(\theta, \mathcal{D})$, то метод можно существенно ускорить; он называется **гамильтоновым методом Монте-Карло** (Hamiltonian Monte Carlo – НМС). Детали см. во втором томе этой книги.

4.7. ЧАСТОТНАЯ СТАТИСТИКА*

Подход к статистическому выводу, описанный в разделе 4.6, называется байесовской статистикой. В нем параметры модели рассматриваются как любые другие неизвестные случайные величины, и для вывода их из данных применяются правила теории вероятностей. Были предприняты попытки придумать такие подходы к статистическому выводу, которые позволили бы избежать трактовки параметров как случайных величин, а значит, избежать использования априорных распределений и формулы Байеса. Этот альтернативный подход называется **частотной статистикой**, **классической** или **ортодоксальной статистикой**.

Основная идея (формализованная в разделе 4.7.1) – представить неопределенность, вычислив, как будет изменяться величина, оцененная на основе данных (например, параметр или предсказанная метка), если изменятся сами данные. Именно эта вариативность при повторных испытаниях лежит в основе моделирования неопределенности в рамках частотного подхода. Что же касается байесовского подхода, то в нем вероятность рассматривается в терминах информации, а не повторных испытаний. Это позволяет вычислить

вероятность однократных событий, о чем было сказано в разделе 2.1.1. Но, быть может, еще важнее, что байесовский подход позволяет избежать некоторых парадоксов, преследующих частотный подход (см. разделы 4.7.5 и 5.5.4).

Эти патологии заставили известного статистика Джорджа Бокса сказать:

Я полагаю, что было бы очень трудно убедить здравомыслящего человека в разумности современной [частотной] статистической практики, но куда меньше трудностей возникло бы с объяснением подхода, основанного на правдоподобию и теореме Байеса (Джордж Бокс, 1962 (цитируется по изданию [Jay76])).

Тем не менее знакомство с частотной статистикой полезно, потому что она широко используется, а некоторые ее ключевые идеи имеет смысл знать даже сторонникам байесовского подхода [Rub84].

4.7.1. Выборочное распределение

В частотной статистике неопределенность представляется не апостериорным распределением случайной величины, а **выборочным распределением оценителя** (оба эти термина мы определим ниже).

Как объясняется в разделе о теории принятия решений 5.1, оценитель – это процедура, описывающая, какое действия предпринять на основе наблюдаемых данных. В контексте оценивания параметров, где действием является возврат вектора параметров, мы будем обозначать его $\theta = \pi(\mathcal{D})$. Например, $\hat{\theta}$ могла бы быть оценкой максимального правдоподобия, оценкой МАР или оценкой метода моментов.

Выборочным распределением оценителя называется распределение результатов, которое мы увидели бы, если бы применили оценитель несколько раз к различным наборам данных, выбранным из некоторого распределения; в контексте оценивания параметров это распределение оценки $\hat{\theta}$, рассматриваемой как случайная величина, зависящая от случайной выборки \mathcal{D} . Раскроем эту мысль. Представьте, что вы производите выборку из S различных наборов данных размера N каждый, порожденных некоторой истинной моделью $p(\mathbf{x}|\theta^*)$, и получаете

$$\tilde{\mathcal{D}}^{(s)} = \{\mathbf{x}_n \sim p(\mathbf{x}_n|\theta^*) : n = 1 : N\}. \quad (4.217)$$

Для краткости обозначим это соотношение $\mathcal{D}^{(s)} \sim \theta^*$. Применив оценитель к каждой выборке $\mathcal{D}^{(s)}$, мы получим множество оценок, $\{\hat{\theta}(\mathcal{D}^{(s)})\}$. Если устремить $S \rightarrow \infty$, то распределение, индуцированное этим множеством, и будет выборочным распределением оценителя. Точнее, имеем

$$p(\pi(\tilde{\mathcal{D}}) = \theta | \tilde{\mathcal{D}} \sim \theta^*) \approx \frac{1}{S} \sum_{s=1}^S \delta(\theta = \pi(\mathcal{D}^{(s)})). \quad (4.218)$$

В некоторых случаях это выражение можно вычислить аналитически, и мы обсудим это в разделе 4.7.2, но чаще приходится применять аппроксимацию Монте-Карло (см. раздел 4.7.3).

4.7.2. Гауссова аппроксимация выборочного распределения MLE

Чаще всего в качестве оценителя применяется MLE. При большом размере выборки выборочное распределение MLE для некоторых моделей становится гауссовым. Это называется **асимптотической нормальностью** выборочного распределения. Формально имеет место следующий результат.

Теорема 4.7.1. *Если параметры допускают идентификацию, то*

$$p(\pi(\tilde{\mathcal{D}}) = \hat{\theta} | \tilde{\mathcal{D}} \sim \theta^*) \rightarrow \mathcal{N}(\hat{\theta} | \theta^*, (N\mathbf{F}(\theta^*))^{-1}), \quad (4.219)$$

где $\mathbf{F}(\theta^*)$ – **информационная матрица Фишера**, определенная формулой (4.220).

Информационная матрица Фишера измеряет кривизну поверхности логарифмического правдоподобия в ее пике, что будет показано ниже.

Более формально **информационная матрица Фишера (FIM)** определяется как ковариация градиента логарифмического правдоподобия (называемого также **функцией вклада**):

$$\mathbf{F} \triangleq \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}|\theta)} [\nabla \log p(\mathbf{x}|\theta) \nabla \log p(\mathbf{x}|\theta)^\top]. \quad (4.220)$$

Следовательно, элемент в позиции (i, j) имеет вид:

$$F_{ij} = \mathbb{E}_{\mathbf{x} \sim \theta} \left[\left(\frac{\partial}{\partial \theta_i} \log p(\mathbf{x}|\theta) \right) \left(\frac{\partial}{\partial \theta_j} \log p(\mathbf{x}|\theta) \right) \right]. \quad (4.221)$$

Можно доказать следующий результат.

Теорема 4.7.2. *Если функция $\log p(\mathbf{x}|\theta)$ дважды дифференцируема, то при некоторых условиях регулярности FIM равна математическому ожиданию гессиана NLL, т. е.*

$$\mathbf{F}_{ij} = -\mathbb{E}_{\mathbf{x} \sim \theta} \left[\frac{\partial^2}{\partial \theta_i \partial \theta_j} \log p(\mathbf{x}|\theta) \right]. \quad (4.222)$$

Таким образом, мы можем интерпретировать FIM как гессиан NLL.

Это поможет понять результат (4.219): функция логарифмического правдоподобия с большой кривизной (большим гессианом) дает оценку с низкой дисперсией, так как параметры «хорошо определены» данными и, следовательно, робастную относительно повторной выборки.

4.7.3. Бутстрэпная аппроксимация выборочного распределения любого оценителя

В случаях, когда оценитель является сложной функцией от данных (не просто MLE) или размер выборки мал, мы можем аппроксимировать его выборочное распределение с помощью метода Монте-Карло, называемого **бутстрэп**.

Идея проста. Если бы мы знали истинные параметры θ^* , то могли бы сгенерировать (скажем, S) фиктивных наборов данных размера N каждый с истинным распределением: $\tilde{\mathcal{D}}^{(s)} = \{\mathbf{x}_n \sim p(\mathbf{x}_n|\theta^*) : n = 1 \dots N\}$. Затем мы могли бы вычислить оценитель на каждой выборке $\hat{\theta}^s = \pi(\tilde{\mathcal{D}}^{(s)})$ и использовать эмпирическое распределение результирующих $\hat{\theta}^s$ как оценку выборочного распределения, как в формуле (4.218). Поскольку θ^* неизвестно, то идея **параметрического бутстрэпа** заключается в том, чтобы генерировать каждый выборочный набор данных с помощью $\hat{\theta} = \pi(\mathcal{D})$, а не θ^* , т. е. мы используем $\tilde{\mathcal{D}}^{(s)} = \{\mathbf{x}_n \sim p(\mathbf{x}_n|\hat{\theta}) : n = 1 \dots N\}$ в формуле (4.218). Это подстановочная аппроксимация выборочного распределения.

Альтернатива, называемая **непараметрическим бутстрэпом**, заключается в том, чтобы произвести выборку с возвращением N примеров из исходных данных. В результате создается новое распределение $\mathcal{D}^{(s)}$ того же размера, что и исходное. Однако количество уникальных точек в бутстрэпной выборке в среднем будет равно всего $0.632 \times N$ (чтобы убедиться в этом, заметим, что вероятность выбрать элемент хотя бы один раз равна $(1 - (1 - 1/N)^N)$, что приближенно равно $1 - e^{-1} \approx 0.632$ для больших N).

На рис. 4.23а–b показан пример, когда с помощью параметрического бутстрэпа вычисляется выборочное распределение MLE для распределения Бернулли. (Непараметрический бутстрэп дает примерно такие же результаты.) При $N = 10$ мы видим, что выборочное распределение асимметрично, а потому довольно далеко от гауссова, но при $N = 100$ распределение больше напоминает гауссово, как и предсказывает теория (см. раздел 4.7.2).

4.7.3.1. Бутстрэп – апостериорное распределение «для бедных»

Возникает естественный вопрос: есть ли связь между оценками параметров $\hat{\theta}^s \pi(\tilde{\mathcal{D}}^{(s)})$, вычисленными методом бутстрэпа, и значениями параметров, выбранными из апостериорного распределения, $\theta^s \sim p(\cdot|\mathcal{D})$? Концептуально они абсолютно различны. Но в часто встречающемся случае, когда в роли оценителя выступает MLE, а априорное распределение не очень сильное, они могут быть весьма похожи. Например, на рис. 4.23с–d показан пример, когда апостериорное распределение вычисляется для равномерного априорного распределения $\text{Beta}(1,1)$, а затем из него производится выборка. Как видно, апостериорное и выборочное распределения похожи. Так что можно считать, что бутстрэпное распределение – это апостериорное распределение «для бедных» [HTF01, стр. 235].

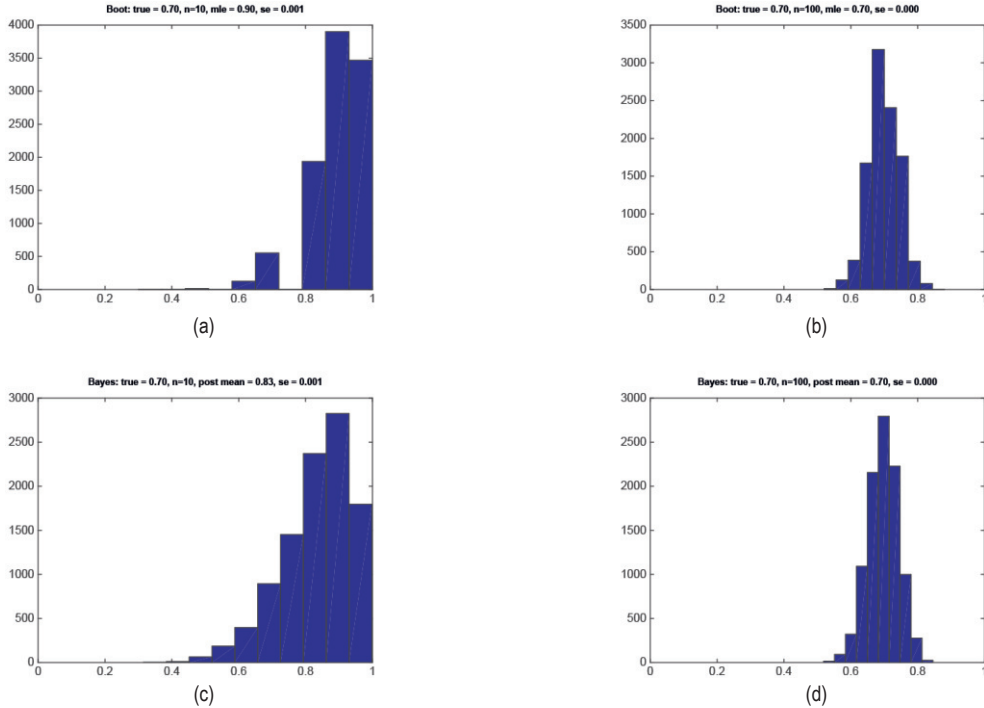


Рис. 4.23 ❖ Сравнение бутстрэпа (верхний ряд) и байесовского подхода (нижний ряд). N примеров данных были выбраны из распределения $\text{Ber}(\theta = 0.7)$. Левый столбец: $N = 10$. Правый столбец: $N = 100$. (a–b) Бутстрэпная аппроксимация выборочного распределения MLE для распределения Бернулли. Показана гистограмма, построенная по $B = 10\,000$ бутстрэпных примеров. (c–d) Гистограмма, построенная по 10 000 примеров из апостериорного распределения с равномерным априорным распределением. Построено программой по адресу figures.problai.ai/book1/4.23

Однако, как это ни кажется удивительным, бутстрэп может оказаться медленнее, чем выборка из апостериорного распределения. Причина в том, что бутстрэп должен сгенерировать S выборочных наборов, а затем обучить модель на каждом из них. (В работе [Kle+11] обсуждаются некоторые методы ускорения бутстрэпа в случае применения к большим наборам данных.)

4.7.4. Доверительные интервалы

В частотной статистике вариативность, индуцированная выборочным распределением, используется с целью оценки неопределенности оценок параметров. Точнее, мы определяем $100(1 - \alpha)\%$ -ный **доверительный интервал** (confidence interval) для оценки параметра θ как любой интервал $I(\tilde{\mathcal{D}}) = (\ell(\tilde{\mathcal{D}}), u(\tilde{\mathcal{D}}))$, выведенный на основе гипотетического набора данных $\tilde{\mathcal{D}}$, такой что

$$\Pr(\theta \in I(\tilde{\mathcal{D}}) | \tilde{\mathcal{D}} \sim \theta) = 1 - \alpha. \quad (4.223)$$

Обычно полагают $\theta = 0.05$, что дает 95%-ный доверительный интервал. Это означает, что если многократно производить выборку данных и для каждого построенного таким образом набора данных вычислять $I(\tilde{\mathcal{D}})$, то примерно 95 % таких интервалов будут содержать истинное значение параметра θ .

Заметим, однако, что формула (4.223) не означает, что для любого набора данных $\theta \in I(\mathcal{D})$ с вероятностью 95 %; такие доверительные интервалы вычисляются в байесовской статистике и называются байесовскими (credible interval) (см. раздел 4.6.6), а не в частотной статистике. Дополнительные сведения об этом важном различии см. в разделе 4.7.5.

Отложим в сторону «философские» споры и обсудим, как вычислить доверительный интервал. Предположим, что $\hat{\theta}$ – оценка параметра θ . Обозначим θ^* его истинное, но неизвестное значение. Также предположим, что выборочное распределение $\Delta = \hat{\theta} - \theta^*$ известно. Обозначим $\underline{\delta}$ и $\bar{\delta}$ его квантили $\alpha/2$ и $1 - \alpha/2$. Тогда

$$\Pr(\underline{\delta} \leq \hat{\theta} - \theta^* \leq \bar{\delta}) = 1 - \alpha. \quad (4.224)$$

Изменяя порядок членов, получаем

$$\Pr(\hat{\theta} - \bar{\delta} \leq \theta^* \leq \hat{\theta} - \underline{\delta}) = 1 - \alpha. \quad (4.225)$$

И следовательно,

$$I(\tilde{\mathcal{D}}) = (\hat{\theta}(\tilde{\mathcal{D}}) - \bar{\delta}(\tilde{\mathcal{D}}), \hat{\theta}(\tilde{\mathcal{D}}) + \underline{\delta}(\tilde{\mathcal{D}})) \quad (4.226)$$

является $100(1 - \alpha)\%$ -ным доверительным интервалом.

В некоторых случаях распределение $\Delta = \hat{\theta} - \theta^*$ можно вычислить аналитически. Это можно использовать для вывода точных доверительных интервалов. Однако чаще предполагают гауссово приближение к выборочному распределению, как в разделе 4.7.2. В этом случае имеем $\sqrt{NF(\hat{\theta})}(\hat{\theta} - \theta^*) \sim \mathcal{N}(0,1)$. Поэтому приближенный доверительный интервал можно вычислить по формуле:

$$\hat{\theta} \pm z_{\alpha/2} \hat{s}\hat{e}, \quad (4.227)$$

где $z_{\alpha/2}$ квантиль $\alpha/2$ функции гауссова распределения, а $\hat{s}\hat{e} = 1/\sqrt{NF(\hat{\theta})}$ – оценка стандартной ошибки. Если положить $\alpha = 0.05$, то будем иметь $z_{\alpha/2} = 1.96$, что оправдывает часто применяемую аппроксимацию $\hat{\theta} \pm 2\hat{s}\hat{e}$.

Если гауссова аппроксимация не подходит, то можно использовать бутстрэпную аппроксимацию (см. раздел 4.7.3). Конкретно мы производим выборку S наборов данных из $\hat{\theta}(\mathcal{D})$ и применяем к каждому оценщику, чтобы получить $\hat{\theta}(\mathcal{D}^{(s)})$; затем используем эмпирическое распределение $\hat{\theta}(\mathcal{D}) - \hat{\theta}(\mathcal{D}^{(s)})$ как аппроксимацию выборочного распределения Δ .

4.7.5. Предостережения: доверительные интервалы и байесовские доверительные интервалы не одно и то же

95%-ный частотный доверительный интервал для параметра θ определяется как любой интервал $I(\tilde{\mathcal{D}})$ такой, что $\Pr(\theta \in I(\tilde{\mathcal{D}}) | \tilde{\mathcal{D}} \sim \theta) = 0.95$ (см. раздел 4.7.4). Это не означает, что параметр с вероятностью 95 % находится в таком интервале при условии наблюдаемых данных. Эта величина – которую мы обычно и хотим вычислить – описывается байесовским доверительным интервалом $p(\theta \in I | \mathcal{D})$, как было объяснено в разделе 4.6.6. Это совершенно разные понятия: при частотном подходе θ рассматривается как неизвестная фиксированная постоянная, а данные трактуются как случайные величины. При байесовском подходе данные считаются фиксированными (поскольку известны), а параметр – случайной величиной (поскольку неизвестен).

Это противоречащее интуиции определение доверительных интервалов может приводить к странным результатам. Рассмотрим следующий пример из работы [Ber85, стр. 11]. Пусть имеется два целых числа $\mathcal{D} = (y_1, y_2)$, выбранных из распределения:

$$p(y|\theta) = \begin{cases} 0.5, & \text{если } y = \theta \\ 0.5, & \text{если } y = \theta + 1 \\ 0 & \text{в противном случае} \end{cases}. \quad (4.228)$$

Если $\theta = 39$, то мы ожидаем следующих исходов с вероятностью 0.25 каждый:

$$(39, 39), (39, 40), (40, 39), (40, 40). \quad (4.229)$$

Обозначим $m = \min(y_1, y_2)$ и определим следующий интервал:

$$[\ell(\mathcal{D}), u(\mathcal{D})] = [m, m]. \quad (4.230)$$

Для приведенной выше выборки это дает

$$[39, 39], [39, 39], [39, 39], [40, 40]. \quad (4.231)$$

Следовательно, (4.230), очевидно, является 75%-ным доверительным интервалом, поскольку 39 содержится в 3/4 всех интервалов. Однако если мы наблюдаем $\mathcal{D} = (39, 40)$, то $p(\theta = 39 | \mathcal{D}) = 1/0$, т. е. мы точно знаем, что θ должно быть равно 39, но «уверенность» в этом факте составляет всего 75 %. Мы видим, что доверительный интервал «покрывает» истинный параметр только в 75 % случаев, если вычислять несколько ДИ по различным случайно выбранным наборам данных, но если взять всего один наблюдаемый набор данных, а значит, единственный ДИ, то частотная вероятность «покрытия» может полностью сбить с толку.

Приведем еще один, не такой искусственный, пример. Пусть требуется оценить параметр θ распределения Бернулли. Обозначим $\bar{y} = \frac{1}{N} \sum_{n=1}^N y_n$ выборочное среднее. MLE равна $\hat{\theta} = \bar{y}$. Приближенный 95%-ный доверительный интервал для параметра распределения Бернулли равен $\bar{y} \pm 1.96 \sqrt{\bar{y}(1 - \bar{y})/N}$ (он называется **интервалом Вальда** и основан на гауссовой аппроксимации распределения Бернулли; сравните с формулой (4.128)). Теперь рассмотрим одиночное испытание: $N = 1$ и $y_1 = 0$. MLE равна 0, что означает переобучение, как мы видели в разделе 4.5.1. Но наш 95%-ный доверительный интервал тоже равен $(0, 0)$, что выглядит еще хуже. Можно возразить, что этот дефект вызван тем, что мы аппроксимировали истинное выборочное распределение гауссовым, или тем, что размер выборки слишком мал, или тем, что параметр «слишком экстремальный». Однако интервал Вальда может вести себя плохо даже для больших N и совсем не экстремальных параметров [BCD01]. С другой стороны, байесовский доверительный интервал с неинформативным априорным распределением Джеффриса ведет себя именно так, как мы ожидаем.

Еще несколько интересных примеров вместе с кодом на Python можно найти в работе [Van14]. Смотрите также работы [Ное+14; Мор+16; Лю+20; Ча+19b], из которых следует, что многие люди, включая профессиональных статистиков, неправильно понимают и используют частотные доверительные интервалы на практике, тогда как байесовские доверительные интервалы не подвержены этим проблемам.

4.7.6. Компромисс между смещением и дисперсией

Оцениватель – это применяемая к данным процедура, которая возвращает оцениваемую величину. Пусть $\hat{\theta}(\cdot)$ – оцениватель, а $\hat{\theta}(\mathcal{D})$ – оцениваемая величина. В частотной статистике мы рассматриваем данные как случайную величину, выбираемую из истинного, но неизвестного распределения, $p^*(\mathcal{D})$; оно индуцирует распределение оцениваемой величины, $p^*(\hat{\theta}(\mathcal{D}))$, называемое выборочным распределением (см. раздел 4.7.1). В этом разделе мы обсудим два важных свойства этого распределения: смещение и дисперсию.

4.7.6.1. Смещение оценки

Смещение оценивателя определяется следующим образом:

$$\text{bias}(\hat{\theta}(\cdot)) \triangleq \mathbb{E}[\hat{\theta}(\mathcal{D})] - \theta^*, \quad (4.232)$$

где θ^* – истинное значение параметра, а математическое ожидание берется относительно «природного распределения» $p(\mathcal{D}|\theta^*)$. Если смещение равно нулю, то оценка называется **несмещенной**. Например, оценка MLE среднего гауссова распределения несмещенная:

$$\text{bias}(\hat{\mu}) = \mathbb{E}[\bar{x}] - \mu = \mathbb{E}\left[\frac{1}{N} \sum_{n=1}^N x_n\right] - \mu = \frac{N\mu}{N} - \mu = 0, \quad (4.233)$$

где \bar{x} – выборочное среднее.

Однако MLE дисперсии гауссова распределения $\sigma_{\text{mle}}^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \bar{x})^2$ не является несмещенной оценкой σ^2 . Действительно, можно показать (упражнение 4.7), что

$$\mathbb{E}[\sigma_{\text{mle}}^2] = \frac{N-1}{N} \sigma^2, \quad (4.234)$$

так что оценка максимального правдоподобия немного меньше истинной дисперсии. Интуитивно это можно объяснить тем, что мы «истратили» одну из точек на оценку среднего, поэтому для выборки размера 1 оценка дисперсии будет равна 0. Но если μ известно, то оценка MLE будет несмещенной (см. упражнение 4.8).

Теперь рассмотрим следующую оценку:

$$\sigma_{\text{unb}}^2 \triangleq \frac{1}{N-1} \sum_{n=1}^N (x_n - \bar{x})^2 = \frac{N}{N-1} \sigma_{\text{mle}}^2. \quad (4.235)$$

То, что она несмещенная, легко доказывается:

$$\mathbb{E}[\sigma_{\text{unb}}^2] = \frac{N}{N-1} \mathbb{E}[\sigma_{\text{mle}}^2] = \frac{N}{N-1} \frac{N-1}{N} \sigma^2 = \sigma^2. \quad (4.236)$$

4.7.6.2. Дисперсия оценки

Интуитивно понятно, что мы хотим, чтобы оценка была несмещенной. Однако этого недостаточно. Например, пусть требуется оценить среднее гауссова распределения по выборке $\mathcal{D} = \{x_1, \dots, x_N\}$. Оценка по одной только первой точке данных, $\hat{\theta}(\mathcal{D}) = x_1$, несмещенная, но обычно отстоит от θ^* дальше, чем эмпирическое среднее \bar{x} (тоже несмещенная оценка). Так что важна еще и дисперсия.

Дисперсия оценки определяется следующим образом:

$$\mathbb{V}[\hat{\theta}] \triangleq \mathbb{E}[\hat{\theta}^2] - (\mathbb{E}[\hat{\theta}])^2, \quad (4.237)$$

где математическое ожидание берется относительно $p(\mathcal{D}|\theta^*)$. Она измеряет, как сильно будет изменяться оценка при изменении данных. Это понятие можно обобщить на ковариационную матрицу для векторных оценок.

4.7.6.3. Компромисс между смещением и дисперсией

В этом разделе мы обсудим фундаментальный компромисс, на который приходится идти при выборе метода оценивания параметров, в предположении,

что наша цель – минимизировать среднеквадратическую ошибку (СКО, англ. MSE) оценки. Обозначим $\hat{\theta} = \hat{\theta}(\mathcal{D})$ оценку, а $\bar{\theta} = \mathbb{E}[\hat{\theta}]$ математическое ожидание оценки (при изменении \mathcal{D}). (Все математические ожидания и дисперсии берутся относительно $p(\mathcal{D}|\theta^*)$, но, чтобы не загромождать формулы, явное обусловливание мы опускаем.) Тогда имеем

$$\mathbb{E}[(\hat{\theta} - \theta^*)^2] = \mathbb{E}[(\hat{\theta} - \bar{\theta}) + (\bar{\theta} - \theta^*)]^2 \quad (4.238)$$

$$= \mathbb{E}[(\hat{\theta} - \bar{\theta})^2] + 2(\bar{\theta} - \theta^*)\mathbb{E}[\hat{\theta} - \bar{\theta}] + (\bar{\theta} - \theta^*)^2 \quad (4.239)$$

$$= \mathbb{E}[(\hat{\theta} - \bar{\theta})^2] + (\bar{\theta} - \theta^*)^2 \quad (4.240)$$

$$= \mathbb{V}[\hat{\theta}] + \text{bias}^2(\hat{\theta}). \quad (4.241)$$

Словами:

СКО = дисперсия + смещение².

(4.242)

Это называется **компромиссом между смещением и дисперсией** (см., например, [GBD92]). А означает он, что иногда разумнее использовать смещенную оценку, коль скоро она уменьшает дисперсию на величину, большую квадрата смещения, – в предположении, что целью является минимизация среднеквадратической ошибки.

4.7.6.4. Пример: оценка МАР среднего гауссова распределения

Приведем пример, взятый из работы [Hof09, стр. 79]. Пусть требуется оценить среднее гауссова распределения по выборке $\mathbf{x} = (x_1, \dots, x_N)$. Мы предполагаем, что данные выбраны из распределения $x_n \sim \mathcal{N}(\theta^* = 1, \sigma^2)$. Очевидной оценкой является MLE. У нее нулевое смещение, а дисперсия равна:

$$\mathbb{V}[\bar{x}|\theta^*] = \frac{\sigma^2}{N}. \quad (4.243)$$

Но можно было бы использовать также оценку МАР. В разделе 4.6.4.2 было показано, что оценка МАР при априорном гауссовом распределении вида $\mathcal{N}(\theta_0, \sigma^2/\kappa_0)$ имеет вид:

$$\tilde{x} \triangleq \frac{N}{N + \kappa_0} \bar{x} + \frac{\kappa_0}{N + \kappa_0} \theta_0 = w\bar{x} + (1 - w)\theta_0, \quad (4.244)$$

где $0 \leq w \leq 1$ определяет, в какой мере мы доверяем MLE по сравнению с априорным распределением. Смещение и дисперсия, соответственно, равны:

$$\mathbb{E}[\tilde{x}] - \theta^* = w\theta^* + (1 - w)\theta_0 - \theta^* = (1 - w)(\theta_0 - \theta^*); \quad (4.245)$$

$$\mathbb{V}[\tilde{x}] = w^2 \frac{\sigma^2}{N}. \quad (4.246)$$

Поскольку оценка MAP смещена (в предположении, что $w < 1$), ее дисперсия меньше.

Предположим, что наше априорное распределение задано не совсем верно, а именно что мы используем $\theta_0 = 0$, тогда как в действительности $\theta^* = 1$.

На рис. 4.24a мы видим, что выборочное распределение оценки MAP для $\theta_0 > 0$ смещено в сторону от истины, но имеет меньшую дисперсию (уже), чем для оценки MLE.

На рис. 4.24b показана зависимость $\text{mse}(\bar{x})/\text{mse}(\bar{x})$ от N . Мы видим, что для оценки MAP среднеквадратическая ошибка меньше, чем для MLE при $\kappa_0 \in \{1, 2\}$. Случай $\kappa_0 = 0$ соответствует MLE, а случай $\kappa_0 = 3$ – сильному априорному распределению, которое ухудшает качество, потому что априорное среднее неверно. Как видим, при условии, что сила априорного распределения хорошо «настроена», оценка MAP может превосходить по качеству оценку MLE с точки зрения минимизации СКО.

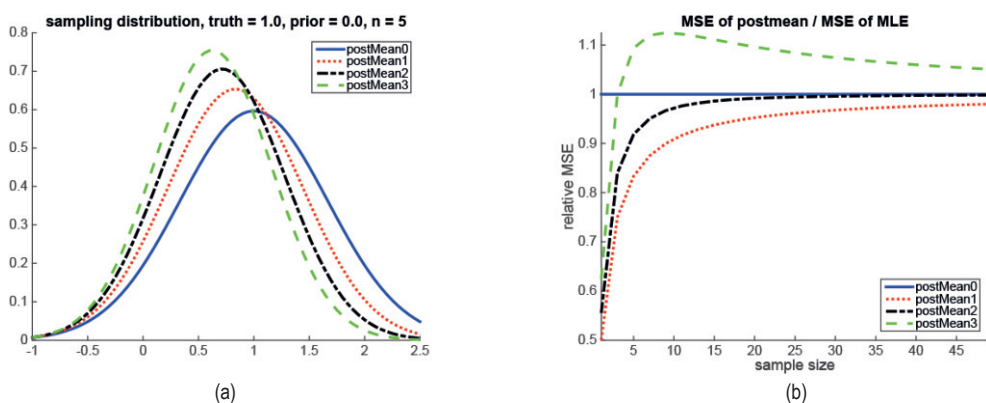


Рис. 4.24 ❖ Слева: выборочное распределение оценки MAP (эквивалентной апостериорному среднему) при априорных распределениях $\mathcal{N}(\theta_0 = 0, \sigma^2/\kappa_0)$ с различной силой κ_0 . (Если положить $\kappa = 0$, то оценка MAP сведется к MLE.) В качестве данных взято $n = 5$ примеров из распределения $\mathcal{N}(\theta^* = 1, \sigma^2 = 1)$. Справа: зависимость отношения СКО апостериорного среднего относительно к СКО MLE от размера выборки. На основе рис. 5.6 из работы [Hof09]. Построено программой по адресу figures.probml.ai/book1/4.24

4.7.6.5. Пример: оценка MAP для линейной регрессии

Еще один важный пример компромисса между смещением и дисперсией дает гребневая регрессия, рассматриваемая в разделе 11.3. Вкратце это соответствует оценке MAP для линейной регрессии при гауссовом априорном распределении, $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \lambda^{-1}\mathbf{I})$. Априорное распределение с нулевым средним поощряет выбор малых весов, что уменьшает переобученность модели; член точности λ управляет силой априорного распределения. При $\lambda = 0$

получаем MLE; при $\lambda > 0$ – смещенную оценку. Чтобы проиллюстрировать влияние на дисперсию, рассмотрим простой пример обучения одномерной модели гребневой регрессии с двумя разными значениями λ . На рис. 4.25 слева показаны отдельные аппроксимирующие кривые, а справа – средняя кривая. Как видим, при увеличении силы регуляризатора дисперсия уменьшается, но смещение растет. Смотрите также рис. 4.26, где схематически изображен компромисс между смещением и дисперсией в терминах сложности модели.

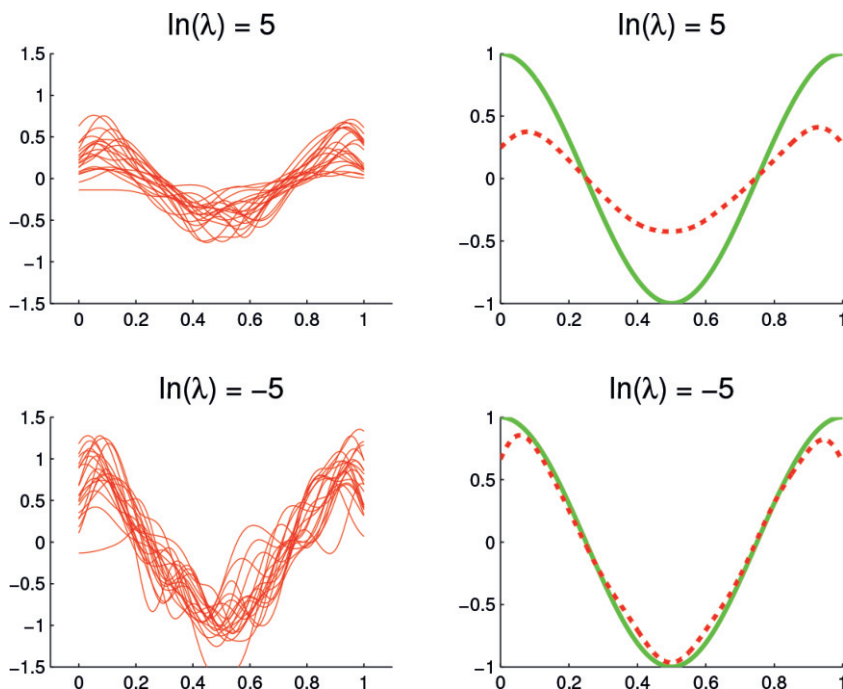


Рис. 4.25 ❖ Иллюстрация компромисса между смещением и дисперсией для гребневой регрессии. Мы генерируем 100 наборов данных на основе истинной функции, показанной сплошной зеленой линией. Слева: показаны графики регуляризированной аппроксимации для 20 разных наборов данных. Используется линейная регрессия с разложением по гауссовым радиально-базисным функциям с 25 центрами, равномерно распределенными в интервале $[0, 1]$. Справа: показан график, усредненный по всем 100 наборам данных. Верхний ряд: сильная регуляризация; отдельные кривые похожи друг на друга (низкая дисперсия), но средняя далека от истинной (высокое смещение). Нижний ряд: слабая регуляризация; отдельные кривые заметно различаются (высокая дисперсия), но средняя близка к истинной (низкое смещение). На основе рис. 3.5 из работы [Bis06]. Построено программно по адресу figures.problai/book1/4.25

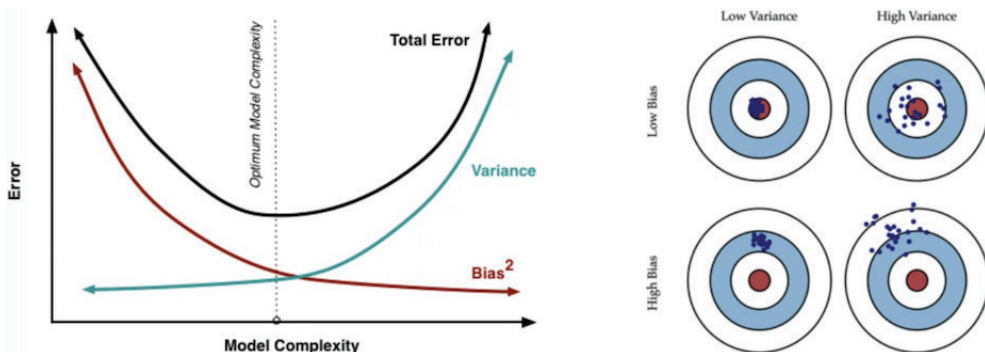


Рис. 4.26 ❖ Схематическая иллюстрация компромисса между смещением и дисперсией. Взято со страницы <http://scott.fortmann-roe.com/docs/BiasVariance.html>. Печатается с разрешения Скотта Фортманна-Роу

4.7.6.6. Применение компромисса между смещением и дисперсией для классификации

Если вместо квадратичной ошибки использовать пороговую потерю, то частотный риск уже нельзя будет выразить в виде суммы квадрата смещения и дисперсии. На самом деле можно показать (упражнение 7.2 в книге [HTF09]), что смещение и дисперсия комбинируются мультипликативно. Если оценка расположена по правильную сторону решающей границы, то смещение отрицательно, а убывание дисперсии приводит к уменьшению частоты неправильной классификации. Но если оценка расположена по другую сторону, то смещение положительно, поэтому имеет смысл увеличить дисперсию [Fri97a]. Этот малоизвестный факт ясно показывает, что компромисс между смещением и дисперсией не особенно полезен для классификации. Лучше сконцентрировать внимание на ожидаемой потере, а не на самих смещении и дисперсии. Ожидаемую потерю можно аппроксимировать с помощью перекрестной проверки, как описано в разделе 4.5.5.

4.8. УПРАЖНЕНИЯ

Упражнение 4.1 [MLE для одномерного гауссова распределения*].

Покажите, что MLE для одномерного гауссова распределения описывается формулами:

$$\hat{\mu} = \frac{1}{N} \sum_{n=1}^N y_n; \quad (4.247)$$

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{n=1}^N (y_n - \hat{\mu})^2. \quad (4.248)$$

Упражнение 4.2 [Оценка MAP для одномерного гауссова распределения*].
(Источник: Яккола.)

Рассмотрим выборку x_1, \dots, x_n из гауссова распределения с известной дисперсией σ^2 и неизвестным средним μ . Предположим, что априорное распределение среднего тоже гауссово и имеет вид $\mu \sim \mathcal{N}(t, s^2)$ с фиксированным средним t и фиксированной дисперсией s^2 . Таким образом, единственное неизвестное – это μ .

- Вычислите оценку MAP $\hat{\mu}_{\text{MAP}}$. Можете привести только результат, без доказательства. Или, проделав куда больше работы, вычислите производные логарифма апостериорного распределения, приравняйте нулю и решите систему уравнений.
- Покажите, что при увеличении числа примеров n оценка MAP сходится к оценке максимального правдоподобия.
- Предположим, что n мало и фиксировано. К чему сходится оценка MAP при увеличении дисперсии априорного распределения s^2 ?
- Предположим, что n мало и фиксировано. К чему сходится оценка MAP при уменьшении дисперсии априорного распределения s^2 ?

Упражнение 4.3 [байесовский доверительный интервал гауссова апостериорного распределения].

(Источник: Де Гроот.) Пусть $X \sim \mathcal{N}(\mu, \sigma^2 = 4)$, где μ неизвестно, но имеет априорное распределение $\mathcal{N}(\mu_0, \sigma_0^2 = 9)$. Апостериорное распределение после наблюдения n примеров имеет вид $\mathcal{N}(\mu_n, \sigma_n^2)$ (это так называемый байесовский доверительный интервал). Насколько велико должно быть n , чтобы гарантированно выполнялось условие:

$$p(\ell \leq \mu_n \leq u | \mathcal{D}) \geq 0.95, \quad (4.249)$$

где (ℓ, u) – интервал (с центром в μ_n) шириной 1, а \mathcal{D} – данные? *Указание:* вспомните, что 95 % массы вероятности гауссова распределения находится на расстоянии $\pm 1.96\sigma$ от среднего.

Упражнение 4.4 [BIC для гауссовых распределений*].

(Источник: Яккола.)

Байесовский информационный критерий (Bayesian information criterion – BIC) – это функция логарифмического правдоподобия со штрафом, которую можно использовать для выбора модели. Определяется он следующим образом:

$$\text{BIC} = \log p(\mathcal{D} | \hat{\theta}_{ML}) - \frac{d}{2} \log(N), \quad (4.250)$$

где d – число параметров модели, а N – число примеров. В этом упражнении мы увидим, как использовать BIC для выбора между гауссовым распределением с полной и с диагональной ковариационной матрицей. Очевидно, что у гауссова распределения с полной ковариационной матрицей правдоподобие больше, но, быть может, увеличение числа параметров не окупается, если улучшение, по сравнению с диагональной ковариационной матрицей, слишком мало. Поэтому мы используем критерий BIC для выбора модели.

Можно написать:

$$\log p(\mathcal{D}|\hat{\Sigma}, \hat{\mu}) = -\frac{N}{2} \text{tr}(\hat{\Sigma}^{-1}\hat{\mathbf{S}}) - \frac{N}{2} \log(|\hat{\Sigma}|); \quad (4.251)$$

$$\hat{\mathbf{S}} = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T, \quad (4.252)$$

где $\hat{\mathbf{S}}$ – матрица рассеяния (эмпирическая ковариация), след матрицы (tr) – это сумма ее диагональных элементов, и мы воспользовались свойством следа.

- Выведите BIC для D -мерного гауссова распределения с полной ковариационной матрицей. Максимально упростите свой ответ, воспользовавшись формой MLE. Не забудьте задать число свободных параметров d .
- Выведите BIC для D -мерного гауссова распределения с *диагональной* ковариационной матрицей. Не забудьте задать число свободных параметров d . *Указание:* в диагональном случае оценка максимального правдоподобия Σ совпадает с $\hat{\Sigma}_{ML}$ с тем отличием, что внедиагональные члены равны нулю:

$$\hat{\Sigma}_{diag} = \text{diag}(\hat{\Sigma}_{ML}(1,1), \dots, \hat{\Sigma}_{ML}(D,D)). \quad (4.253)$$

Упражнение 4.5 [BIC для двумерно дискретного распределения].

(Источник: Яккола.)

Обозначим $x \in \{0, 1\}$ результат подбрасывания монеты ($x = 0$ – орел, $x = 1$ – решка). Монета может быть асимметричной, так что орел выпадает с вероятностью θ_1 . Предположим, что кто-то еще наблюдает за подбрасыванием монеты и сообщает вам исход, y . Но этот человек ненадежен и правильно сообщает результат только с вероятностью θ_2 , т. е. распределение $p(y|x, \theta_2)$ описывается таблицей:

	$y = 0$	$y = 1$
$x = 0$	θ_2	$1 - \theta_2$
$x = 1$	$1 - \theta_2$	θ_2

Предположим, что θ_2 не зависит от x и θ_1 .

- Выпишите совместное распределение вероятностей $p(x, y|\theta)$ в виде таблицы 2×2 в терминах $\theta = (\theta_1, \theta_2)$.
- Предположим, что имеется следующий набор данных: $\mathbf{x} = (1, 1, 0, 1, 1, 0, 0)$, $\mathbf{y} = (1, 0, 0, 0, 1, 0, 1)$. Каковы оценки MLE для θ_1 и θ_2 ? Поясните свой ответ. *Указание:* обратите внимание, что функция правдоподобия распадается в произведение:

$$p(x, y|\theta) = p(y|x, \theta_2)p(x|\theta_1). \quad (4.254)$$

Что представляет собой $p(\mathcal{D}|\hat{\theta}, M_2)$, где M_2 обозначает эту модель с 2 параметрами? (Если хотите, можете оставить ответ в дробной форме.)

- с. Теперь рассмотрим модель с 4 параметрами, $\theta = (\theta_{0,0}, \theta_{0,1}, \theta_{1,0}, \theta_{1,1})$, представляющую $p(x, y|\theta) = \theta_{x,y}$. (Только три из этих параметров свободны, так как их сумма должна быть равна 1.) Какова MLE θ ? Что представляет собой $p(\mathcal{D}|\hat{\theta}, M_4)$, где M_4 обозначает эту модель с 4 параметрами?
- д. Предположим, что мы не уверены, какая модель правильна. Мы вычисляем логарифмическое правдоподобие с перекрестной проверкой и исключением по одному для двухпараметрической и четырехпараметрической модели следующим образом:

$$L(m) = \sum_{i=1}^n \log p(x_i, y_i | m, \hat{\theta}(\mathcal{D}_{-i})), \quad (4.255)$$

где $\hat{\theta}(\mathcal{D}_{-i})$ обозначает MLE, вычисленную для \mathcal{D} с исключенной i -й строкой. Какая модель будет выбрана в результате перекрестной проверки и почему? *Указание:* обратите внимание, как изменяется таблица счетчиков, когда обучающие примеры исключаются по одному.

- е. Напомним, что альтернативой перекрестной проверке является критерий BIC, определенный следующим образом:

$$\text{BIC}(M, \mathcal{D}) \triangleq \log p(\mathcal{D}|\hat{\theta}_{MLE}) - \frac{\text{dof}(M)}{2} \log N, \quad (4.256)$$

где $\text{dof}(M)$ – число свободных параметров модели. Вычислите BIC для обеих моделей (используя логарифм по основанию e). Какую модель предпочитает BIC?

Упражнение 4.6 [смесь сопряженных априорных распределений является сопряженным распределением*].

Рассмотрим смесовое априорное распределение:

$$p(\theta) = \sum_k p(h = k) p(\theta | z = k), \quad (4.257)$$

где каждое $p(\theta | z = k)$ является сопряженным правдоподобию. Докажите, что это сопряженное априорное распределение.

Упражнение 4.7 [оценка максимального правдоподобия σ_{MLE}^2 смещенная].

Покажите, что $\hat{\sigma}_{MLE}^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \hat{\mu})^2$ – смещенная оценка σ^2 , т. е. покажите, что

$$\mathbb{E}_{X_1, \dots, X_n \sim \mathcal{N}(n, \sigma)} [\hat{\sigma}^2(X_1, \dots, X_n)] \neq \sigma^2.$$

Указание: заметьте, что X_1, \dots, X_N независимы, и воспользуйтесь тем фактом, что математическое ожидание произведения независимых случайных величин равно произведению их математических ожиданий.

Упражнение 4.8 [оценивание σ^2 , когда μ известно*].

Пусть имеется выборка $x_1, \dots, x_N \sim \mathcal{N}(\mu, \sigma^2)$, где μ – известная постоянная. Выведите выражение оценки максимального правдоподобия σ^2 в этом случае? Является ли оценка несмещенной?

Упражнение 4.9 [дисперсия и СКО оценок дисперсии гауссова распределения*].

Докажите, что стандартная ошибка MLE дисперсии гауссова распределения равна:

$$\sqrt{\mathbb{V}[\sigma_{\text{mle}}^2]} = \sqrt{\frac{2(N-1)}{N^2}} \sigma^2. \quad (4.258)$$

Указание: воспользуйтесь тем фактом, что

$$\frac{N-1}{\sigma^2} \sigma_{\text{unb}}^2 \sim \chi_{N-1}^2 \quad (4.259)$$

и что $\mathbb{V}[\chi_{N-1}^2] = 2(N-1)$. Наконец, покажите, что $\text{MSE}(\sigma_{\text{unb}}^2) = \frac{2N-1}{N^2} \sigma^4$

и $\text{MSE}(\sigma_{\text{mle}}^2) = \frac{2}{N-1} \sigma^4$.

Глава 5

Теория принятия решений

5.1. БАЙЕСОВСКАЯ ТЕОРИЯ ПРИНЯТИЯ РЕШЕНИЙ

Байесовский вывод предлагает оптимальный способ обновить наши предположения о скрытых величинах H на основе наблюдаемых данных $\mathbf{X} = \mathbf{x}$ путем вычисления апостериорного распределения $p(H|\mathbf{x})$. Однако рано или поздно необходимо перейти от предположений к **действиям**, которые можно выполнить в реальном мире. И как решить, какое действие лучшее? Тут-то и вступает в игру **байесовская теория принятия решений**. Эта глава является кратким введением в нее. Подробнее см., например, работы [DeG70; KWW22].

5.1.1. Основы

В теории принятия решений мы предполагаем, что принимающий решения **агент** располагает набором возможных действий, \mathcal{A} , из которых производит выбор. Например, рассмотрим случай, когда гипотетический врач лечит пациента, который, возможно, подхватил COVID-19. Предположим, что есть два возможных действия: не делать ничего или выписать пациенту дорогое лекарство, которое имеет тяжелые побочные эффекты, но может спасти ему жизнь.

С каждым из этих действий ассоциированы затраты и выгоды, которые зависят от **состояния природы** $H \in \mathcal{H}$. Эту информацию можно закодировать в виде **функции потерь** $\ell(h, a)$, описывающей, какие мы понесем потери, если предпримем действие $a \in \mathcal{A}$, когда природа находится в состоянии $h \in \mathcal{H}$.

Например, предположим, что состояние определяется возрастом пациента (молодой или старый) и тем, болен он COVID-19 или нет. Заметим, что возраст можно наблюдать непосредственно, но состояние заболевания следует выводиться из зашумленных наблюдений, как обсуждалось в разделе 2.3. Таким образом, это состояние **частично наблюдаемо**.

Предположим, что затраты, ассоциированные с приемом лекарства, одинаковы и не зависят от состояния пациента. Но вот выгоды различаются. Если пациент молодой, то мы ожидаем, что он проживет долго, поэтому затраты, которые придется понести, если не выписать ему лекарство, когда он болен, высоки. Если же пациент старый, то жить ему осталось меньше, поэтому стоимость отказа от лекарства в случае заболевания ниже (особенно ввиду побочных эффектов), хотя с этим можно спорить. В медицинских кругах общепринятой единицей стоимости является **год жизни с поправкой на качество** (quality-adjusted life year – QALY). Предположим, что ожидаемый индекс QALY для молодого человека равен 60, а для старого 10.

Предположим, что затраты, ассоциированные с приемом лекарства, составляют 8 QALY вследствие боли и страданий, вызванных побочными эффектами. Получающаяся матрица потерь показана в табл. 5.1.

Таблица 5.1. Гипотетическая матрица потерь для лица, принимающего решения; имеется 4 состояния природы и 2 возможных действия

Состояние	Ничего	Лекарство
Нет COVID-19, молодой	0	8
COVID-19, молодой	60	8
Нет COVID-19, старый	0	8
COVID-19, старый	10	8

Эти числа отражают относительные затраты и выгоды и зависят от многих факторов. Их можно вывести, задавая лицу, принимающему решение, вопросы о **предпочтениях**, связанных с возможными исходами. В теории принятия решений есть теорема, утверждающая, что любое непротиворечивое множество предпочтений можно преобразовать к порядковой шкале затрат (см., например, [https://en.wikipedia.org/wiki/Preference_\(economics\)](https://en.wikipedia.org/wiki/Preference_(economics))).

Задав функцию потерь, мы можем вычислить **апостериорную ожидаемую потерю** или **риск** для каждого возможного действия:

$$R(a|\mathbf{x}) \triangleq \mathbb{E}_{p(h|\mathbf{x})}[\ell(h, a)] = \sum_{h \in \mathcal{H}} \ell(h, a) p(h|\mathbf{x}). \quad (5.1)$$

Оптимальная политика (называемая также **байесовской оценкой**) описывает, какое действие предпринять для каждого возможного наблюдения, чтобы минимизировать риск:

$$\pi^*(\mathbf{x}) = \operatorname{argmin}_{a \in \mathcal{A}} \mathbb{E}_{p(h|\mathbf{x})}[\ell(h, a)]. \quad (5.2)$$

Существует альтернативная, но эквивалентная формулировка этого результата. Определим **функцию полезности** $U(h, a)$ как желательность каждого возможного действия в каждом возможном состоянии. Если положить $U(h, a) = -\ell(h, a)$, то оптимальная политика выглядит следующим образом:

$$\pi^*(\mathbf{x}) = \operatorname{argmin}_{a \in \mathcal{A}} \mathbb{E}_h[U(h, a)]. \quad (5.3)$$

Это называется **принципом максимальной ожидаемой полезности**.

Вернемся к примеру с COVID-19. Наблюдение x включает возраст (молодой или старый) и результат тестирования (положительный или отрицательный). Пользуясь результатами из раздела 2.3.1 о формуле Байеса для диагностики COVID-19, мы можем преобразовать результат тестирования в распределение состояний заболевания (т. е. вычислить вероятность того, что пациент болен или не болен). Зная эти предположения и матрицу потерь в табл. 5.1, мы можем вычислить оптимальную политику для каждого возможного наблюдения; она показана в табл. 5.2.

Таблица 5.2. Оптимальная политика лечения пациентов с COVID-19 для каждого из возможных наблюдений

тест	возраст	pr(covid)	затраты-ничего	затраты-лекарство	действие
0	0	0.01	0.84	8.00	0
0	1	0.01	0.14	8.00	0
1	0	0.80	47.73	8.00	1
1	1	0.80	7.95	8.00	0

Из табл. 5.2 видно, что лекарство следует выписывать только молодым людям с положительным тестом. Но если снизить затраты, ассоциированные с лекарством, с 8 до 5 единиц, то оптимальная политика изменится: лекарство следует давать всем имеющим положительный тест. Политика может измениться также в зависимости от надежности теста. Например, если повысить чувствительность с 0.875 до 0.975, то вероятность заболевания COVID-19 при положительном тесте увеличится с 0.80 до 0.81, и в результате оптимальной станет политика, предписывающая давать лекарство всем пациентам с положительным тестом, даже при затратах 8 QALY. (Соответствующий код находится по адресу code.problml.ai/book1/dtheory).

До сих пор мы предполагали, что агент **безразличен к риску**. Это означает, что на его решение не влияет степень уверенности в множестве исходов. Например, такому агенту не важно, получит ли он 50 долл. наверняка, 100 долл. с вероятностью 50 % или ничего не получит. Напротив, **избегающий риска** агент выберет первый вариант. Байесовскую теорию принятия решений можно обобщить на приложения, **чувствительные к риску**, но здесь мы этим вопросом заниматься не будем (см., например, [Cho+15]).

5.1.2. Проблемы классификации

В этом разделе мы воспользуемся байесовской теорией принятия решений для предсказания оптимальной метки класса при наличии наблюдаемых входных данных $\mathbf{x} \in \mathcal{X}$.

5.1.2.1. Бинарная потеря

Предположим, что состояниям природы соответствуют метки классов, т. е. $\mathcal{H} = \mathcal{Y} = \{1, \dots, C\}$. Еще предположим, что действиям также соответствуют метки классов, т. е. $\mathcal{A} = \mathcal{Y}$. В такой формулировке часто применяется **бинарная функция потерь** $\ell_{01}(y^*, \hat{y})$, определенная следующим образом:

	$\hat{y} = 0$	$\hat{y} = 1$	(5.4)
$y^* = 0$	0	1	
$y^* = 1$	1	0	

Это можно записать короче:

$$\ell_{01}(y^*; \hat{y}) = \mathbb{I}(y^* \neq \hat{y}). \quad (5.5)$$

В таком случае апостериорная ожидаемая потеря равна:

$$R(\hat{y}|\mathbf{x}) = p(\hat{y} \neq y^*|\mathbf{x}) = 1 - p(y^* = \hat{y}|\mathbf{x}). \quad (5.6)$$

Поэтому чтобы минимизировать ожидаемую потерю, нужно выбрать наиболее вероятную метку:

$$\pi(\mathbf{x}) = \operatorname{argmin}_{y \in \mathcal{Y}} p(y|\mathbf{x}). \quad (5.7)$$

Это решение соответствует моде апостериорного распределения и называется **оценкой апостериорного максимума**, или **оценкой MAP**.

5.1.2.2. Классификация с учетом стоимости

Рассмотрим проблему бинарной классификации со следующей функцией потерь $\ell(y^*, \hat{y})$:

$$\begin{pmatrix} \ell_{00} & \ell_{01} \\ \ell_{10} & \ell_{11} \end{pmatrix}. \quad (5.8)$$

Обозначим $p_0 = p(y^* = 0|\mathbf{x})$ и $p_1 = 1 - p_0$. Таким образом, мы должны выбирать метку $\hat{y} = 0$ тогда и только тогда, когда

$$\ell_{00}p_0 + \ell_{10}p_1 < \ell_{01}p_0 + \ell_{11}p_1. \quad (5.9)$$

Если $\ell_{00} = \ell_{11} = 0$, то это условие упрощается:

$$p_1 < \frac{\ell_{01}}{\ell_{01} + \ell_{10}}. \quad (5.10)$$

Теперь предположим, что $\ell_{10} = c\ell_{01}$, т. е. ложноотрицательный результат обходится в c раз дороже, чем ложноположительный. Тогда решающее правило упрощается еще сильнее: выбирать $a = 0$ тогда и только тогда, когда $p_1 < 1/(1 + c)$. Например, если стоимость ложноотрицательного результата

в два раза больше стоимости ложноположительного, т. е. $c = 2$, то порог объявления положительного результата равен $1/3$.

5.1.2.3. Классификация с возможностью отклонения примера

Иногда мы можем сказать «не знаю», вместо того чтобы возвращать ответ, в котором не уверены; это называется **отклонением** (см., например, [BW08]). Это особенно важно в таких областях, как медицина и финансы, когда следует избегать риска.

Формализовать возможность отклонения можно следующим образом. Предположим, что имеются состояния природы $\mathcal{H} = \mathcal{Y} = \{1, \dots, C\}$ и множество действий $\mathcal{A} = \mathcal{Y} \cup \{0\}$, где действие 0 представляет отклонение. Определим функцию потерь:

$$\ell(y^*, a) = \begin{cases} 0, & \text{если } y^* = a \text{ и } a \in \{1, \dots, C\} \\ \lambda_r, & \text{если } a = 0 \\ \lambda_e & \text{в противном случае} \end{cases}, \quad (5.11)$$

где λ_r – стоимость отклонения, а λ_e – стоимость ошибки классификации. В упражнении 5.1 вам будет предложено доказать, что оптимальное действие заключается в том, чтобы выбрать отклонение, если вероятность наиболее вероятного класса меньше $\lambda^* = 1 - \lambda_r/\lambda_e$; в противном случае следует выбрать наиболее вероятный класс. Иными словами, оптимальная политика выглядит так:

$$a^* = \begin{cases} y^*, & \text{если } p^* > \lambda^* \\ \text{отклонить} & \text{в противном случае} \end{cases}, \quad (5.12)$$

где

$$y^* = \operatorname{argmax}_{y \in \{1, \dots, C\}} p(y|x); \quad (5.13)$$

$$p^* = p(y^*|x) = \max_{y \in \{1, \dots, C\}} p(y|x); \quad (5.14)$$

$$\lambda^* = 1 - \frac{\lambda_r}{\lambda_e}. \quad (5.15)$$

Смотрите иллюстрацию на рис. 5.1.

Интересное применение отклонения возникает в телеигре Jeopardy¹. В этой игре участники должны отвечать на различные вопросы, но, дав неверный ответ, теряют деньги. В 2011 году IBM продемонстрировала компьютерную систему **Watson**, которая победила чемпиона-человека. В Watson использу-

¹ Российский аналог называется «Своя игра». – Прим. перев.

ется целый ряд интересных методов [Fer+10], но к теме нашего обсуждения относится модуль, который оценивает степень уверенности в ответе. Система «нажимает на кнопку», только если достаточно уверена в правильности ответа.

Описание других методов и приложений см., например, в работах [Cor+16; GEY19].

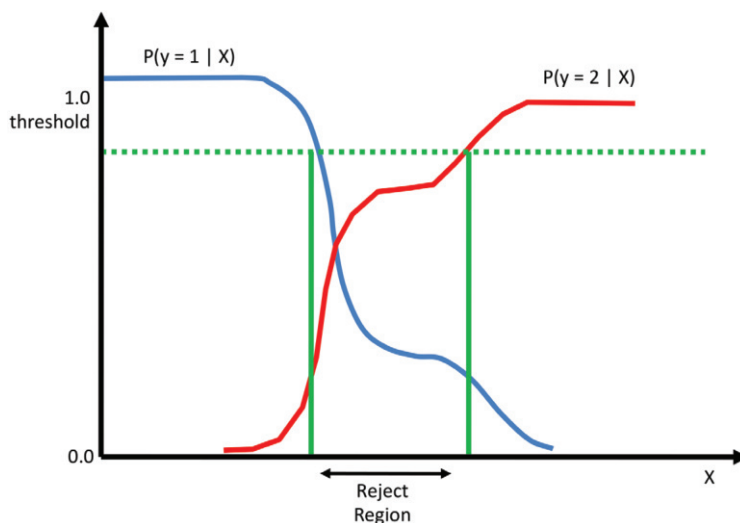


Рис. 5.1 ❖ Для некоторых областей пространства входов, где степень определенности апостериорного класса невысока, мы можем предпочесть не выбирать класс 1 или 2, а отклонить пример и отказаться от действия. На основе рис. 1.26 из работы [Bis06]

5.1.3. ROC-кривые

В разделе 5.1.2.2 мы показали, что можем выбрать оптимальную метку в проблеме бинарной классификации, задав порог вероятности τ , зависящий от относительной стоимости ложноположительных и ложноотрицательных результатов. Но можно выбрать не один, а несколько порогов и сравнить получающее качество решения. Этот подход мы и обсудим ниже.

5.1.3.1. Матрицы неточностей классификации

Для любого фиксированного порога τ рассмотрим решающее правило:

$$\hat{y}_\tau(\mathbf{x}) = \mathbb{I}(p(y = 1|\mathbf{x}) \geq 1 - \tau). \quad (5.16)$$

Мы можем следующим образом вычислить эмпирическое количество ложноположительных результатов (FP), возникающих при применении этой политики к множеству N помеченных примеров:

$$FP_{\tau} = \sum_{n=1}^N \mathbb{I}(\hat{y}_{\tau}(\mathbf{x}_n) = 1, y_n = 0). \quad (5.17)$$

Аналогично можно вычислить эмпирическое количество ложноотрицательных (FN), истинно положительных (TP) и истинно отрицательных (TN) результатов. Эти величины можно сохранить в **матрице неточностей классификации** C размера 2×2 , элемент C_{ij} которой показывает, сколько раз пример с истинной меткой класса i был классифицирован как принадлежащий классу с меткой j . В случае бинарной классификации получающая матрица выглядит, как показано в табл. 5.3.

Таблица 5.3. Матрица неточностей для проблемы бинарной классификации.

TP – количество истинно положительных результатов, FP – ложноположительных, TN – истинно отрицательных, FN – ложноотрицательных, P – истинное количество положительных результатов, \hat{P} – предсказанное количество положительных результатов, N – истинное количество отрицательных результатов, \hat{N} – предсказанное количество отрицательных результатов

		Оценка		Сумма по строке
		0	1	
Истина	0	TN	FP	N
	1	FN	TP	P
Сумма по столбцу		\hat{N}	\hat{P}	

Имея такую таблицу, мы можем вычислить $p(\hat{y}|y)$ или $p(y|\hat{y})$ в зависимости от того, что нормируется: строки или столбцы. Для этих распределений можно подсчитать различные сводные статистики, показанные в табл. 5.4 и 5.5. Например, **частота истинно положительных результатов** (TPR), она же **чувствительность**, **полнота** или **частота попаданий**, определяется как

$$TPR_{\tau} = p(\hat{y} = 1|y = 1, \tau) = \frac{TP_{\tau}}{TP_{\tau} + FN_{\tau}}, \quad (5.18)$$

а **частота ложноположительных результатов** (FPR), она же **частота ложных тревог** или **частота ошибок первого рода**, определяется как

$$FPR_{\tau} = p(\hat{y} = 1|y = 0, \tau) = \frac{FP_{\tau}}{FP_{\tau} + TN_{\tau}}. \quad (5.19)$$

Теперь можно построить график зависимости TPR от FPR в виде неявной функции от τ . Он называется **рабочей характеристикой приемника** (receiver operating characteristic), или **ROC-кривой** (см. пример на рис. 5.2а).

Таблица 5.4. Матрица неточностей для проблемы бинарной классификации, нормированная по строкам для получения $p(\hat{y}|y)$. Сокращения: TNR = частота истинно отрицательных результатов, Spec = специфичность, FPR = частота ложноположительных результатов, FNR = частота ложноотрицательных результатов, Miss = частота непопаданий, TPR = частота истинно положительных результатов, Sens = чувствительность.

Отметим, что $FNR=1-TPR$ и $FPR=1-TNR$

		Оценка	
		0	1
Истина	0	$TN/N = TNR = \text{Spec}$	$FP/N = FPR = \text{Тип I}$
	1	$FN/P = FNR = \text{Miss} = \text{Тип II}$	$TP/P = TPR = \text{Sens} = \text{полнота}$

Таблица 5.5. Матрица неточностей для проблемы бинарной классификации, нормированная по столбцам для получения $p(y|\hat{y})$.

Сокращения: NPV = прогностическая ценность отрицательного результата, FDR = частота ложных обнаружений, FOR = частота ложных пропусков, PPV = прогностическая ценность положительного результата, Prec = точность.

Отметим, что $FOR=1-NPV$ и $FDR=1-PPV$

		Оценка	
		0	1
Истина	0	$TN/\hat{N} = NPV$	$FP/\hat{P} = FDR$
	1	$FN/\hat{N} = FOR$	$TP/\hat{P} = \text{Prec} = PPV$

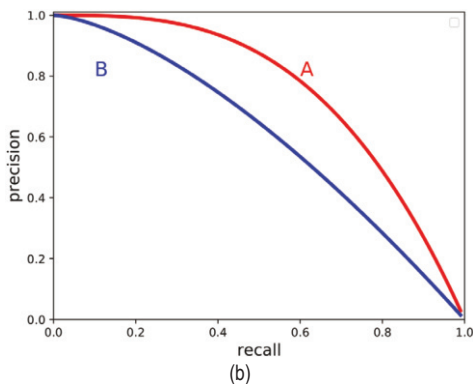
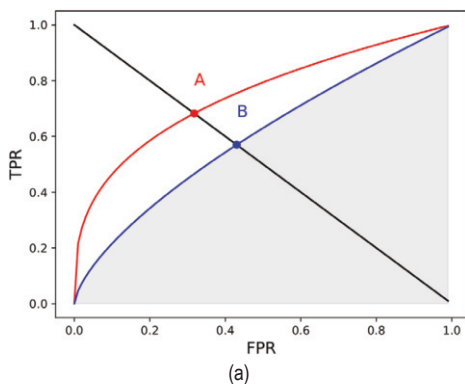


Рис. 5.2 ❖ (a) ROC-кривые для двух гипотетических систем классификации. Красная кривая для системы А лучше синей кривой для системы В. Изображен график зависимости частоты истинно положительных результатов (TPR) от частоты ложноположительных результатов (FPR) при изменении порога τ . Отмечен также коэффициент равных ошибок (EER) – красная и синяя точки. Серым цветом для классификатора В показана площадь под кривой (area under the curve – AUC). Построено программой по адресу figures.problml.ai/book1/5.2. (b) Кривая точность–полнота для двух гипотетических систем классификации. Красная кривая для системы А лучше синей кривой для системы В. Построено программой по адресу figures.problml.ai/book1/5.2

5.1.3.2. Обобщение ROC-кривой в виде скаляра

Качество ROC-кривой часто выражают одним числом – **площадью под кривой**, или **AUC**. Чем больше AUC, тем лучше; максимальное значение, очевидно, равно 1. Еще одна часто используемая сводная статистика – **коэффициент равных ошибок** (equal error rate – **EER**), определяется как значение, удовлетворяющее условию $FPR = FNR$. Поскольку $FNR = 1 - TPR$, мы можем вычислить EER, проведя прямую из левого верхнего в правый нижний угол и найдя ее точки пересечения с ROC-кривой (см. точки A и B на рис. 5.2a). Чем меньше EER, тем лучше; минимальное значение, очевидно, равно 0 (соответствует левому верхнему углу).

5.1.3.3. Несбалансированность классов

В некоторых задачах имеется значительная **несбалансированность классов**. Например, в системах информационного поиска множество отрицательных (нерелевантных) результатов обычно гораздо больше множества положительных (релевантных) результатов. На ROC-кривую несбалансированность классов не влияет, поскольку TPR и FPR – доли положительных и отрицательных результатов соответственно. Однако в таких случаях полезность ROC-кривой снижается, потому что значительное изменение абсолютного числа ложноположительных результатов не приводит к существенному изменению *частоты* ложноположительных результатов, так как FPR делится на $FP + TN$ (см., например, обсуждение в [SR15]). Таким образом, все «действие» сосредоточено в крайней левой части кривой. При таких обстоятельствах можно выбрать другие способы обобщения матрицы неточностей классификации, например кривые *точность–полнота*, которые мы обсудим в разделе 5.1.4.

5.1.4. Кривые *точность–полнота*

В некоторых задачах понятие «отрицательный» не имеет четкого определения. Рассмотрим, к примеру, обнаружение объектов в изображениях: если в основу детектора положена классификация участков изображения (патчей), то количество исследованных патчей, а значит, и число истинно отрицательных результатов, является параметром алгоритма, а не частью постановки задачи. Аналогично в системах информационного поиска обычно выбирают начальное множество потенциальных результатов, которые затем ранжируются по релевантности; задав порог отсечения, мы можем выделить множество положительных и отрицательных результатов, но заметим, что размер множества отрицательных результатов зависит от общего числа отобранных кандидатов, а это параметр алгоритма, а не часть постановки задачи.

В таких случаях мы можем использовать в качестве обобщенной характеристики качества системы **кривую *точность–полнота***, описанную ниже. (Более подробное обсуждение вопроса о связи между ROC-кривыми и кривыми *точность–полнота* см. в работе [DG06]).

5.1.4.1. Вычисление точности и полноты

Ключевая идея заключается в том, чтобы заменить FPR величиной, вычисленной только по положительным результатам, а именно:

$$\mathcal{P}(\tau) \triangleq p(y = 1 | \hat{y} = 1, \tau) = \frac{TP_\tau}{TP_\tau + FP_\tau}. \quad (5.20)$$

Точность измеряет, какая доля полученных результатов действительно положительна. Ее можно сравнить с полнотой (то же самое, что TPR), которая измеряет долю положительных результатов среди полученных:

$$\mathcal{R}(\tau) \triangleq p(\hat{y} = 1 | y = 1, \tau) = \frac{TP_\tau}{TP_\tau + FN_\tau}. \quad (5.21)$$

Если $\hat{y}_n \in \{0, 1\}$ – предсказанная метка, а $y_n \in \{0, 1\}$ – истинная метка, то точность и полноту можно оценить так:

$$\mathcal{P}(\tau) = \frac{\sum_n y_n \hat{y}_n}{\sum_n \hat{y}_n}; \quad (5.22)$$

$$\mathcal{R}(\tau) = \frac{\sum_n y_n \hat{y}_n}{\sum_n y_n}. \quad (5.23)$$

Теперь можно построить график зависимости точности от полноты при изменении порога τ (см. рис. 5.2b). Держаться ближе к правому верхнему углу – лучшее, что можно сделать.

5.1.4.2. Обобщение кривых точность–полнота в виде скаляра

Кривую точность–полнота (PR-кривая) можно обобщенно охарактеризовать одним числом несколькими способами. Во-первых, можно взять точность для фиксированного уровня полноты, например точность на $K = 10$ элементах с наибольшей полнотой. Это называется **точностью на K элементах**. Можно вместо этого вычислить площадь под PR-кривой. Однако точность необязательно монотонно убывает при возрастании полноты. Например, предположим, что классификатор имеет точность 90 % при полноте 10 % и точность 96 % при полноте 20 %. В таком случае измерять нужно не точность при полноте 10 %, а максимальную точность, которой можно достичь при полноте *не менее* 10 % (она должна быть равна 96 %). Это называется **интерполированной точностью**. Результат усреднения интерполированных точностей называется **усредненной точностью** (average precision – AP); она равна площади под интерполированной PR-кривой, но может быть не равна площади под исходной PR-кривой¹. **Средняя усредненная точность** (mean

¹ Детали см. в статье по адресу <https://sanchom.wordpress.com/tag/average-precision/>.

average precision – **mAP**) – это средняя AP, вычисленная по множеству различных PR-кривых.

5.1.4.3. F-мера

Для фиксированного порога, соответствующего одной точке на PR-кривой, мы можем вычислить значения точности и полноты, которые обозначим \mathcal{P} и \mathcal{R} . Часто они объединяются в одну статистику F_β , которая придает полноте в $\beta > 0$ раз больший вес, чем точности:

$$\frac{1}{F_\beta} = \frac{1}{1 + \beta^2} \frac{1}{\mathcal{P}} = \frac{\beta^2}{1 + \beta^2} \frac{1}{\mathcal{R}} \quad (5.24)$$

или эквивалентно

$$F_\beta \triangleq (1 + \beta^2) \frac{\mathcal{P} \cdot \mathcal{R}}{\beta^2 \mathcal{P} + \mathcal{R}} = \frac{(1 + \beta^2) \text{TP}}{(1 + \beta^2) \text{TP} + \beta^2 \text{FN} + \text{FP}}. \quad (5.25)$$

При $\beta = 1$ мы получаем среднее гармоническое полноты и точности:

$$\frac{1}{F_1} = \frac{1}{2} \left(\frac{1}{\mathcal{P}} + \frac{1}{\mathcal{R}} \right); \quad (5.26)$$

$$F_1 = \frac{1}{1/\mathcal{R} + 1/\mathcal{P}} = 2 \frac{\mathcal{P} \cdot \mathcal{R}}{\mathcal{P} + \mathcal{R}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}. \quad (5.27)$$

Чтобы понять, почему используется среднее гармоническое, а не среднее арифметическое $(\mathcal{P} + \mathcal{R}) / 2$, рассмотрим следующий сценарий. Предположим, что отобраны все потенциальные элементы, т. е. $\hat{y}_n = 1$ для всех n и $\mathcal{R} = 1$. В таком случае точность \mathcal{P} определяется **распространенностью**, $p(y = 1) = \frac{\sum_n \mathbb{I}(y_n = 1)}{N}$. Предположим, что распространенность низкая,

скажем $p(y = 1) = 10^{-4}$. Среднее арифметическое \mathcal{P} и \mathcal{R} равно $(\mathcal{P} + \mathcal{R})/2 = (10^{-4} + 1)/2 \approx 50\%$. С другой стороны, среднее гармоническое этой стратегии равно только $(2 \times 10^{-4} \times 1)/(1 + 10^{-4}) \approx 0.2\%$. В общем случае среднее гармоническое более консервативно и требует, чтобы большими были как точность, так и полнота.

При использовании в качестве оценки F_1 точности и полноте назначается одинаковый вес. Но если полнота важнее, то можно брать $\beta = 2$, а если важнее точность, то $\beta = 0.5$.

5.1.4.4. Несбалансированность классов

ROC-кривые чувствительны к несбалансированности классов, а PR-кривые – нет, как отмечено в работе [Wil20]. Чтобы убедиться в этом, обозначим долю положительных примеров в наборе данных $\pi = P/(P + N)$ и определим отношение $r = P/N = \pi/(1 - \pi)$. Пусть $n = P + N$ – размер генеральной совокуп-

ности. На ROC-кривые изменение r не влияет, потому что TPR определена как отношение внутри множества положительных примеров, а FPR как отношение внутри множества отрицательных примеров. Это означает, что неважно, какой класс мы считаем положительным, а какой отрицательным.

Теперь рассмотрим PR-кривые. Точность можно записать в виде:

$$\text{Prec} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{P \cdot \text{TPR}}{P \cdot \text{TPR} + N \cdot \text{FPR}} = \frac{\text{TPR}}{\text{TPR} + \frac{1}{r} \text{FPR}}. \quad (5.28)$$

Таким образом, $\text{Prec} \rightarrow 1$ при $\pi \rightarrow 1$ и $r \rightarrow \infty$ и $\text{Prec} \rightarrow 0$ при $\pi \rightarrow 0$ и $r \rightarrow 0$. Например, если перейти от сбалансированной задачи, в которой $r = 0.5$, к несбалансированной, в которой $r = 0.1$ (т. е. положительные примеры встречаются реже), то точность при любом пороге снизится, а полнота (она же TPR) останется неизменной, так что вся PR-кривая пройдет ниже. Стало быть, если имеется несколько бинарных задач с разной распространенностью (например, обнаружение часто или редко встречающихся объектов), то следует быть осторожнее при усреднении их точностей [HCD12].

На F-меру также влияет несбалансированность классов. Чтобы убедиться в этом, перепишем F-меру следующим образом:

$$\frac{1}{F_\beta} = \frac{1}{1 + \beta^2} \frac{1}{P} + \frac{\beta^2}{1 + \beta^2} \frac{1}{R} \quad (5.29)$$

$$= \frac{1}{1 + \beta^2} \frac{\text{TPR} + \frac{N}{P} \text{FPR}}{\text{TPR}} + \frac{\beta^2}{1 + \beta^2} \frac{1}{\text{TPR}}; \quad (5.30)$$

$$F_\beta = \frac{(1 + \beta^2) \text{TPR}}{\text{TPR} + \frac{1}{r} \text{FPR} + \beta^2}. \quad (5.31)$$

5.1.5. Задачи регрессии

До сих пор мы считали, что множества действий \mathcal{A} и состояний природы \mathcal{H} конечны. В этом разделе мы рассмотрим случай, когда оба множества совпадают с вещественной прямой, $\mathcal{A} = \mathcal{H} = \mathbb{R}$, и различные функции потерь, часто используемые в этом случае (который обобщается на \mathbb{R}^D , если вычислять потерю поэлементно). Получающиеся в результате решающие правила можно применить для вычисления оптимальных параметров оценки, оптимального действия робота и т. д.

5.1.5.1. ℓ_2 -потеря

Чаще всего для непрерывных состояний и действий используется ℓ_2 -потеря, называемая также **квадратической ошибкой** или **квадратичной потерей**:

$$\ell_2(h, a) = (h - a)^2. \quad (5.32)$$

В этом случае риск равен:

$$R(a|\mathbf{x}) = \mathbb{E}[(h - a)^2|\mathbf{x}] = \mathbb{E}[h^2|\mathbf{x}] - 2a\mathbb{E}[h|\mathbf{x}] + a^2. \quad (5.33)$$

Оптимальное действие должно быть таким, чтобы производная риска (в соответствующей точке) обращалась в нуль (как объясняется в главе 8). Следовательно, оптимальное действие заключается в выборе апостериорного среднего:

$$\frac{\partial}{\partial a} R(a|\mathbf{x}) = -2\mathbb{E}[h|\mathbf{x}] + 2a = 0 \Rightarrow \pi(\mathbf{x}) = \mathbb{E}[h|\mathbf{x}] = \int h p(h|\mathbf{x}) dh. \quad (5.34)$$

Это часто называется оценкой **минимума среднеквадратической ошибки**, или оценкой **MMSE**.

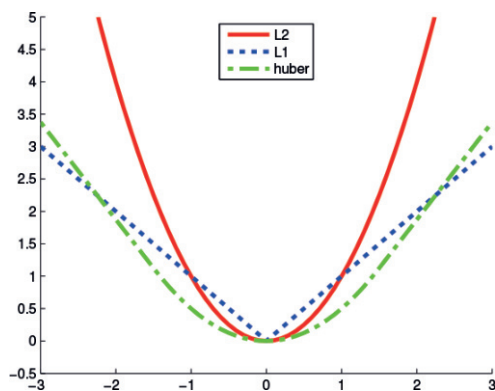


Рис. 5.3 ❖ Функций потерь ℓ_2 , ℓ_1 и Хьюбера с $\delta = 1.5$. Построено программой по адресу figures.probl.ai/book1/5.3

5.1.5.2 ℓ_1 -потеря

ℓ_2 -потеря штрафует за отклонение от истины квадратично и потому чувствительна к **выбросам**. Более робастной является **ℓ_1 -потеря**, штрафующая за абсолютную величину отклонения от истины:

$$\ell_1(h, a) = |h - a|. \quad (5.35)$$

Это схематически изображено на рис. 5.3. В упражнении 5.4 вам будет предложено доказать, что оптимальная оценка совпадает с апостериорной **медианой**, т. е. таким значением, что $\Pr(h < a|\mathbf{x}) = \Pr(h \geq a|\mathbf{x}) = 0.5$. Мы можем использовать эту потерю для робастной регрессии, как обсуждается в разделе 11.6.1.

5.1.5.3. Функция потерь Хьюбера

Также робастной является **функция потерь Хьюбера** [Hub64], определенная следующим образом:

$$\ell_\delta(h, a) = \begin{cases} r^2/2, & \text{если } |r| \leq \delta \\ \delta|r| - \delta^2/2, & \text{если } |r| > \delta \end{cases} \quad (5.36)$$

где $r = h - a$. Она эквивалентна ℓ_2 , когда ошибка меньше δ , и ℓ_1 в противном случае. Ее график изображен на рис. 5.3. В разделе 11.6.3 обсуждается, как использовать такую потерю для робастной регрессии.

5.1.6. Задачи вероятностного предсказания

В разделе 5.1.2 мы предполагали, что множество возможных действий сводится к выбору одной метки класса (или, возможно, действия «не знаю»). В разделе 5.1.5 предполагалось, что действие заключается в выборе вещественного скалярного значения. В этом разделе мы будем предполагать, что действие – это выбор распределения вероятностей интересующего нас значения. То есть мы хотим произвести **вероятностное предсказание**, или **вероятностный прогноз**, а не просто предсказать конкретное значение. Точнее, мы предполагаем, что истинное «состояние природы» – *распределение*, $h = p(Y|x)$, действие – тоже распределение, $a = q(Y|x)$, и мы хотим выбрать q , доставляющее минимум математическому ожиданию $\mathbb{E}[\ell(p, q)]$ для заданного x . Ниже мы обсудим возможные функции потерь для этого случая.

5.1.6.1. Расхождение КЛ, перекрестная энтропия и логарифмическая потеря

Для сравнения двух распределений часто применяется **расхождение Кульбака–Лейблера (КЛ)**, определяемое следующим образом:

$$\mathbb{KL}(p||q) \triangleq \sum_{y \in \mathcal{Y}} p(y) \log \frac{p(y)}{q(y)}. \quad (5.37)$$

(Ради упрощения обозначений мы предположили, что величина u дискретная, но определение обобщается и на вещественные величины.) В разделе 6.2 мы покажем, что расхождение КЛ обладает следующим свойством: $\mathbb{KL}(p||q) \geq 0$, и равенство достигается только в случае $p = q$. Заметим, что это несимметричная функция своих аргументов.

Расхождение КЛ можно преобразовать следующим образом:

$$\mathbb{KL}(p||q) = \sum_{y \in \mathcal{Y}} p(y) \log p(y) - \sum_{y \in \mathcal{Y}} p(y) \log q(y) \quad (5.38)$$

$$= -\mathbb{H}(p) + \mathbb{H}(p, q); \quad (5.39)$$

$$\mathbb{H}(p) \triangleq -\sum_y p(y) \log p(y); \quad (5.40)$$

$$\mathbb{H}(p, q) \triangleq -\sum_y p(y) \log q(y). \quad (5.41)$$

Член $\mathbb{H}(p)$ называется **энтропией**. Это мера неопределенности или дисперсии p ; она достигает максимума для равномерного распределения p и равна 0, если p вырождено, т. е. является детерминированной дельта-функцией. Энтропия часто применяется в теории информации, занимающейся оптимальными способами сжатия и передачи данных (см. главу 6). В схеме оптимального кодирования под часто встречающиеся символы (например, значения из Y с большой вероятностью $p(y)$) отводится меньше битов, а под встречающиеся реже – больше битов. Ключевой результат заключается в том, что количество битов, необходимых для сжатия набора данных, выбранных из распределения p , не может быть меньше $\mathbb{H}(p)$; поэтому энтропия дает нижнюю границу степени сжатия данных без потери информации. Член $\mathbb{H}(p, q)$ называется **перекрестной энтропией**. Он измеряет ожидаемое количество битов, необходимых для сжатия набора данных, выбранных из распределения p , если при проектировании кода используется распределение q . Таким образом, расхождение КЛ – это дополнительное число битов, необходимых потому, что при проектировании кода использовалось неправильное распределение q . Если расхождение КЛ равно нулю, значит, мы можем правильно предсказывать вероятности всех возможных будущих событий, т. е. обучились предсказывать будущее так же хорошо, как «оракул», имеющий доступ к истинному распределению p .

Чтобы найти оптимальное распределение для предсказания будущих данных, необходимо минимизировать $\mathbb{KL}(p|q)$. Поскольку $\mathbb{H}(p)$ постоянна относительно q , ее можно проигнорировать и минимизировать только перекрестную энтропию:

$$q^*(Y|x) = \operatorname{argmin}_q \mathbb{H}(q(Y|x), p(Y|x)). \quad (5.42)$$

Теперь рассмотрим частный случай, когда истинное состояние природы описывается вырожденным распределением, вся масса которого сосредоточена в одном исходе, скажем c , т. е. $h = p(Y|x) = \mathbb{I}(Y = c)$. Такое распределение часто называют **унитарным** (one-hot), потому что оно «делает горячим» (hot) только c -й элемент вектора, а все остальные оставляет «холодными», как показано на рис. 2.1. В этом случае энтропия принимает вид:

$$\mathbb{H}(\delta(Y = c), q) = -\sum_{y \in \mathcal{Y}} \delta(y = c) \log q(y) = -\log q(c). \quad (5.43)$$

Это называется **логарифмической потерей** прогнозного распределения q при заданной целевой метке c .

5.1.6.2. Правила верной оценки

Потеря перекрестной энтропии очень часто применяется для вероятностного прогнозирования, но это не единственная возможная метрика. Желательное для нас свойство заключается в том, чтобы функция потерь достигала минимума тогда и только тогда, когда лицо, принимающее решение, выбирает распределение q , совпадающее с истинным распределением p , т. е.

$\ell(p, p) \leq \ell(p, q)$ и равенство имеет место тогда и только тогда, когда $p = q$. Такая функция потерь ℓ называется **правилом верной оценки** (proper scoring rule) [GR07].

Можно показать, что потеря перекрестной энтропии является правилом верной оценки в силу того, что $\mathbb{KL}(p||p) \leq \mathbb{KL}(p||q)$. Однако член $\log p(y)/q(y)$ может оказаться очень чувствительным к ошибкам из-за маловероятных событий [QC+06]. Широко применяемой альтернативой является **оценка Бриера** [Bri50], определенная следующим образом (для дискретного распределения с множеством значений C):

$$\ell(p, q) \triangleq \frac{1}{C} \sum_{c=1}^C (q(y = c|x) - p(y = c|x))^2. \quad (5.44)$$

Это просто квадратическая ошибка прогнозного распределения относительного истинного, когда оба распределения рассматриваются как векторы. Будучи основана на квадратической ошибке, оценка Бриера менее чувствительна к чрезвычайно редко или чрезвычайно часто встречающимся классам. К счастью, она также является правилом верной оценки.

5.2. БАЙЕСОВСКАЯ ПРОВЕРКА ГИПОТЕЗ

Предположим, что имеется две гипотезы или модели, которые обычно называются **нулевой гипотезой**, M_0 , и **альтернативной гипотезой**, M_1 , а мы хотим узнать, какая из них имеет больше шансов оказаться верной. Это называется **проверкой гипотез**.

Если мы пользуемся бинарной потерей, то оптимальным будет решение выбрать альтернативную гипотезу тогда и только тогда, когда $p(M_1|\mathcal{D}) > p(M_0|\mathcal{D})$, или, эквивалентно, когда $p(M_1|\mathcal{D})/p(M_0|\mathcal{D}) > 1$. Если априорное распределение равномерное, $p(M_0) = p(M_1) = 0.5$, то решающее правило принимает вид: выбрать M_1 тогда и только тогда, когда $p(\mathcal{D}|M_1)/p(\mathcal{D}|M_0) > 1$. Эта величина, являющаяся отношением маргинальных правдоподобий обеих моделей, называется **коэффициентом Байеса**:

$$B_{1,0} \triangleq \frac{p(\mathcal{D}|M_1)}{p(\mathcal{D}|M_0)}. \quad (5.45)$$

Она похожа на **отношение правдоподобия**, но параметры исключены в результате интегрирования, что позволяет сравнивать модели разной сложности в силу байесовской интерпретации принципа бритвы Оккама, обсуждаемой в разделе 5.2.3.

Если $B_{1,0} > 1$, то мы предпочитаем модель 1, иначе модель 0. Конечно, может случиться, что $B_{1,0}$ лишь немного больше 1. В таком случае мы не вполне уверены, что модель 1 лучше. Джеффрис [Jef61] предложил шкалу силы свидетельства для интерпретации величины коэффициента Байеса, показанную в табл. 5.6. Это байесовская альтернатива частотной концепции p -значения (см. раздел 5.5.3).

Пример вычисления коэффициентов Байеса приведен в разделе 5.2.1.

Таблица 5.6. Шкала Джеффриса силы свидетельства для интерпретации коэффициента Байеса

Коэффициент Байеса $BF(1,0)$	Интерпретация
$BF < 1/100$	Решительное свидетельство в пользу M_0
$BF < 1/10$	Сильное свидетельство в пользу M_0
$1/10 < BF < 1/3$	Умеренное свидетельство в пользу M_0
$1/3 < BF < 1$	Слабое свидетельство в пользу M_0
$1 < BF < 3$	Слабое свидетельство в пользу M_1
$3 < BF < 10$	Умеренное свидетельство в пользу M_1
$BF > 10$	Сильное свидетельство в пользу M_1
$BF > 100$	Решительное свидетельство в пользу M_1

5.2.1. Пример: проверка симметричности монеты

Предположим, что мы несколько раз подбрасываем монету и хотим понять, были ли наблюдаемые данные порождены симметричной монетой, $\theta = 0.5$, или несимметричной, для которой θ может быть любым значением в интервале $[0, 1]$. Обозначим первую модель M_0 , а вторую M_1 . Маргинальное правдоподобие в случае истинности модели M_0 равно просто

$$p(\mathcal{D}|M_0) = \left(\frac{1}{2}\right)^N, \quad (5.46)$$

где N – число подбрасываний монеты. Из формулы (4.143) получаем, что маргинальное правдоподобие в случае истинности модели M_1 при априорном бета-распределении равно:

$$p(\mathcal{D}|M_1) = \int p(\mathcal{D}|\theta)p(\theta)d\theta = \frac{B(\alpha_1 + N_1, \alpha_0 + N_0)}{B(\alpha_1, \alpha_0)}. \quad (5.47)$$

На рис. 5.4а показан график зависимости $\log p(\mathcal{D}|M_1)$ от числа выпадений орла N_1 в предположении, что $N = 5$ и априорное распределение равномерное, $\alpha_1 = \alpha_0 = 1$. (Форма кривой не особенно чувствительна к α_1 и α_0 при условии, что априорное распределение симметрично, т. е. $\alpha_0 = \alpha_1$.) Если наблюдается 2 или 3 орла, то гипотеза M_0 о симметричной монете более вероятна, чем M_1 , потому что модель M_0 проще (в ней нет свободных параметров) – было бы подозрительным совпадением, если бы при подбрасывании асимметричной монеты орел и решка выпали в точности поровну. Однако, по мере того как счетчики смещаются в ту или другую сторону, мы начинаем отдавать предпочтение гипотезе об асимметричной монете. Заметим, что график коэффициента Байеса, $\log B_{1,0}$, имел бы точно такую же форму, потому что $\log p(\mathcal{D}|M_0)$ – постоянная.

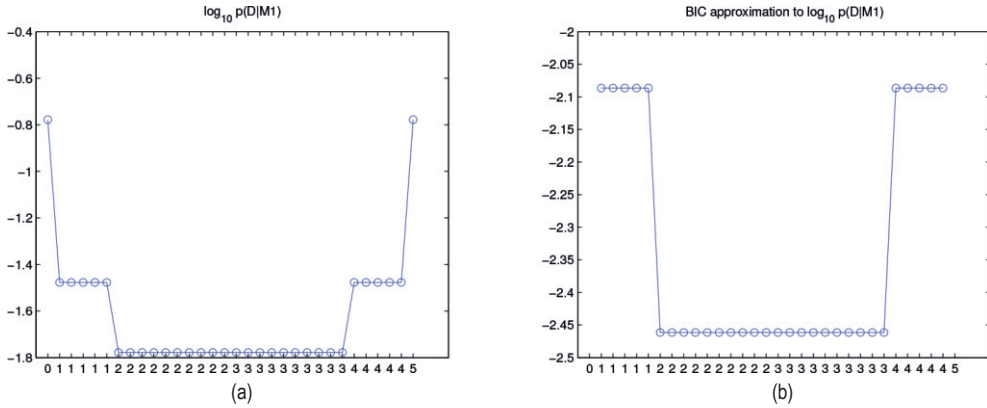


Рис. 5.4 ❖ (а) Зависимость логарифмического маргинального правдоподобия от количества выпадений орла при подбрасывании монеты. (б) Аппроксимация BIC. (Шкала по вертикали произвольна, потому что N фиксировано.) Построено программой по адресу figures.probml.ai/book1/5.4

5.2.2. Байесовский выбор модели

Теперь предположим, что имеется множество \mathcal{M} , содержащее больше 2 моделей, а нам нужно выбрать наиболее вероятную. Это называется **выбором модели**. Можно считать, что это задача на принятие решений, в котором действие состоит в выборе одной модели $m \in \mathcal{M}$. В случае пороговой потери оптимальное действие – выбор самой вероятной модели:

$$\hat{m} = \operatorname{argmax}_{m \in \mathcal{M}} p(m|\mathcal{D}), \quad (5.48)$$

где

$$p(m|\mathcal{D}) = \frac{p(\mathcal{D}|m)p(m)}{\sum_{m \in \mathcal{M}} p(\mathcal{D}|m)p(m)} \quad (5.49)$$

– апостериорное распределение моделей. Если априорное распределение моделей равномерное, $p(m) = 1/|\mathcal{M}|$, то модель MAP описывается формулой:

$$\hat{m} = \operatorname{argmax}_{m \in \mathcal{M}} p(\mathcal{D}|m). \quad (5.50)$$

Величина $p(\mathcal{D}|m)$ равна:

$$p(\mathcal{D}|m) = \int p(\mathcal{D}|\theta, m)p(\theta|m)d\theta. \quad (5.51)$$

Она называется **маргинальным правдоподобием**, или **свидетельством** в пользу модели m . Интуитивно понятно, что это правдоподобие данных, усредненное по всем возможным значениям параметров с весами, определяемыми априорным распределением $p(\theta|m)$. Если при любой комбинации параметров θ вероятность данных высока, то, наверное, это хорошая модель.

5.2.2.1. Пример: полиномиальная регрессия

В качестве примера байесовского выбора модели рассмотрим одномерную полиномиальную регрессию. На рис. 5.5 показано апостериорное распределение трех разных моделей, соответствующих полиномам степени 1, 2 и 3, для аппроксимации $N = 5$ точек. Мы будем считать, что априорное распределение моделей равномерное, и воспользуемся эмпирической байесовской оценкой априорного распределения весов регрессии (см. раздел 11.7.7). Затем мы вычислим свидетельство в пользу каждой модели (о том, как это делается, см. раздел 11.7). Мы видим, что данных для обоснованного выбора сложной модели недостаточно, поэтому в качестве модели MAP выбирается $m = 1$. На рис. 5.6 показан аналогичный график для $N = 30$ точек. Теперь моделью MAP будет $m = 2$; имея выборку большего размера, мы можем без опаски выбрать более сложную модель.

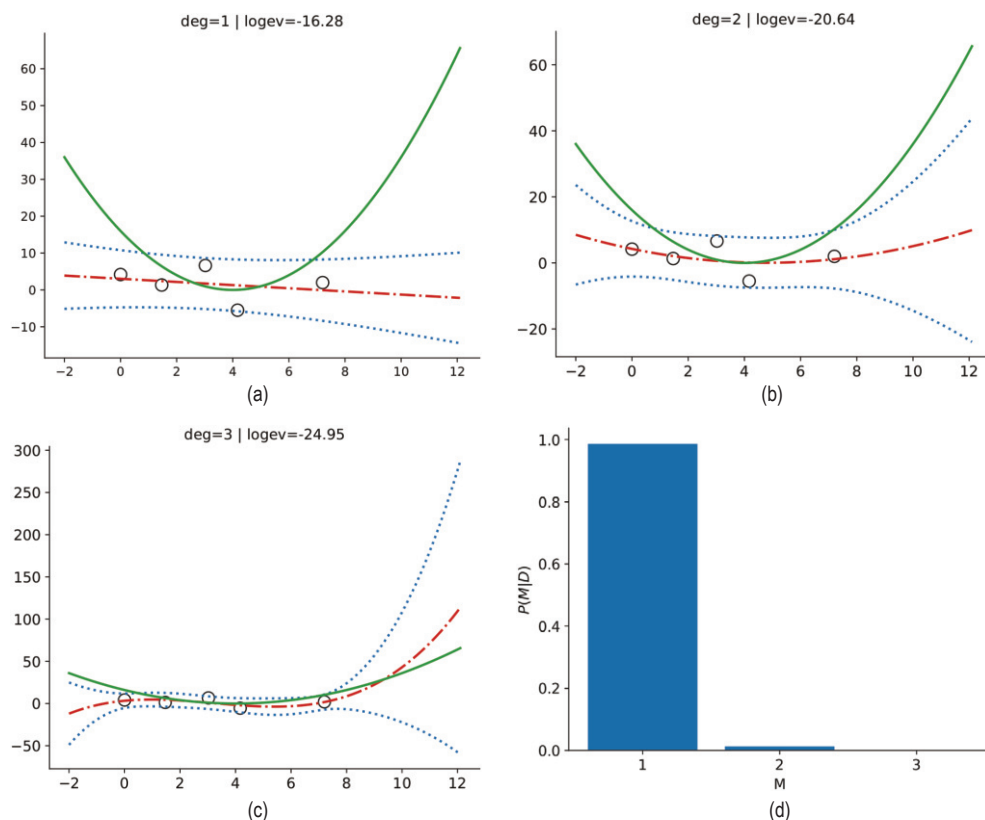
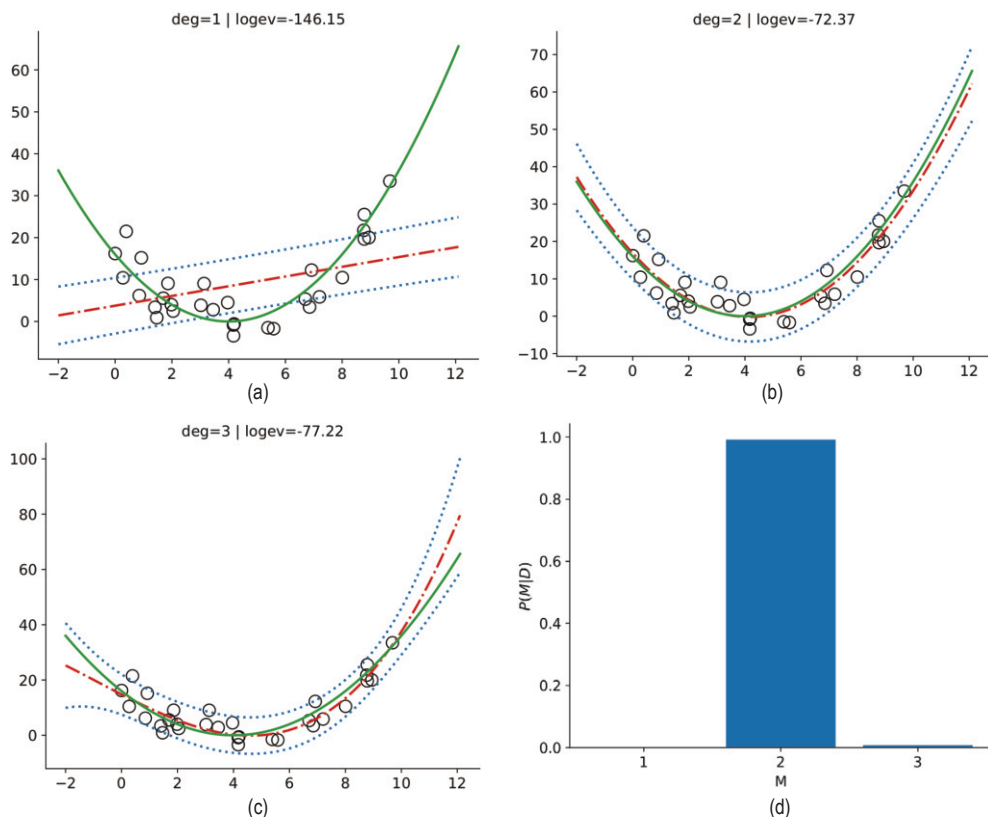


Рис. 5.5 ❖ Байесовский выбор модели полиномиальной регрессии. (а–с) $N = 5$ точек аппроксимируются полиномами степени 1, 2 и 3. Сплошная зеленая кривая – истинная функция, штриховая красная кривая – предсказания (пунктирные синие линии представляют интервал $\pm \sigma$ вокруг среднего). (d) Построен график апостериорного распределения моделей $p(m|D)$ в предположении, что априорное распределение равномерно, $p(m) \propto 1$. Построено программой по адресу figures.problm.lai/book1/5.5

Рис. 5.6 ❖ То же, что на рис. 5.5, но теперь $N = 30$.Построено программой по адресу figures.problml.ai/book1/5.6

5.2.3. Бритва Оккама

Рассмотрим две модели: простую m_1 и более сложную m_2 . Предположим, что обе могут объяснить данные, если подобрать параметры правильно, т. е. так, что обе вероятности $p(D|\hat{\theta}_1, m_1)$ и $p(D|\hat{\theta}_2, m_2)$ велики. Интуитивно понятно, что следует предпочесть m_1 , поскольку она проще, но не хуже m_2 . Этот принцип называется **бритвой Оккама**.

Теперь посмотрим, как ранжирование моделей на основе маргинального правдоподобия, подразумевающего усреднение правдоподобия относительно априорного распределения, приводит именно к такому поведению. Сложная модель придает меньше априорной вероятности «хорошим» параметрам, объясняющим данные, $\hat{\theta}_2$, потому что интеграл априорного распределения по всему пространству параметров должен быть равен 1. Таким образом, при усреднении она рассматривает части пространства параметров с низким правдоподобием. С другой стороны, у простой модели меньше параметров, поэтому априорное распределение сосредоточено в меньшем объеме, сле-

довательно, усреднение производится преимущественно по хорошей части пространства параметров, в окрестности θ_1 . Поэтому маргинальное правдоподобие отдает предпочтение более простой модели. Это называется **байесовской интерпретацией бритвы Оккама** [Mac95; MG05].

По-другому понять байесовскую интерпретацию бритвы Оккама можно, сравнив относительную прогностическую способность простой и сложной модели. Поскольку сумма вероятностей должна быть равна 1, имеем $\sum_{\mathcal{D}} p(\mathcal{D}'|m) = 1$, где суммирование производится по всем возможным наборам данных. Сложные модели, способные предсказывать много исходов, вынуждены размазывать массу вероятности тонким слоем, а потому не могут назначить никакому набору такой высокой вероятности, как модели попроще. Иногда это называют законом **сохранения массы вероятности**, он иллюстрируется на рис. 5.7. По горизонтальной оси откладываются все возможные наборы данных в порядке возрастания сложности (измеряемой как-то абстрактно), а по вертикальной – предсказания трех различных моделей: простой, M_1 , средней, M_2 , и сложной, M_3 . Вертикальная прямая обозначает фактически наблюдаемые данные. Модель 1 слишком проста и назначает \mathcal{D}_0 низкую вероятность. Модель 3 также назначает \mathcal{D}_0 сравнительно низкую вероятность, потому что умеет предсказывать много наборов данных и, следовательно, размазывает доступную вероятность тонким и широким слоем. Модель 2 – как раз «то, что надо»: она предсказывает наблюдаемые данные с разумной степенью уверенности, но не предсказывает слишком много лишнего. Поэтому модель 2 является наиболее вероятной.

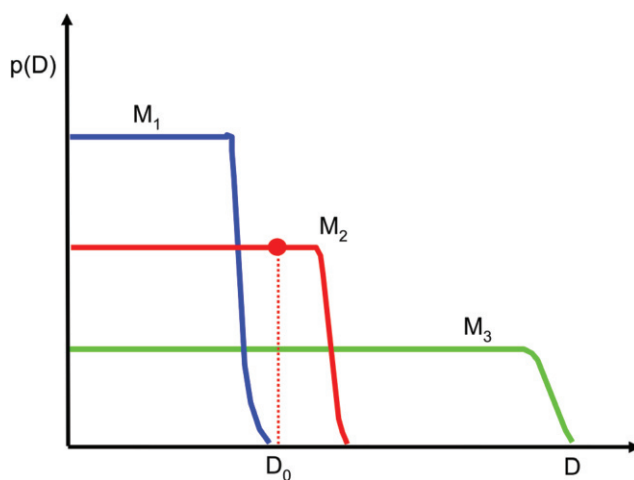


Рис. 5.7 ❖ Схематическая иллюстрация байесовской интерпретации бритвы Оккама. Широкая (зеленая) кривая соответствует сложной модели, узкая (синяя) – простой модели, а средняя (красная) – «правильной» модели. На основе рис. 3.13 из работы [Bis06]. Смотрите также [MG05, рис. 2], где похожий график приведен для реальных данных

5.2.4. Связь между перекрестной проверкой и маргинальным правдоподобием

Мы видели, как маргинальное правдоподобие помогает выбрать модель «правильной» сложности. В небайесовских подходах к выбору модели стандартным приемом для достижения той же цели является перекрестная проверка (раздел 4.5.5).

Оказывается – мы это сейчас покажем, – что маргинальное правдоподобие тесно связано с перекрестной проверкой с исключением по одному (leave-one-out cross-validation – LOO-CV). Начнем с маргинального правдоподобия, которое запишем в последовательной форме:

$$p(\mathcal{D}|m) = \prod_{n=1}^N p(y_n|y_{1:n-1}, \mathbf{x}_{1:N}, m) = \prod_{n=1}^N p(y_n|\mathbf{x}_{1:N}, \mathcal{D}_{1:n-1}, m), \quad (5.52)$$

где

$$p(y|\mathbf{x}, \mathcal{D}_{1:n-1}, m) = \int p(y|\mathbf{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}_{1:n-1}, m) d\boldsymbol{\theta}. \quad (5.53)$$

Применим к этому распределению подстановочную аппроксимацию:

$$p(y|\mathbf{x}, \mathcal{D}_{1:n-1}, m) \approx \int p(y|\mathbf{x}, \boldsymbol{\theta}) \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_m(\mathcal{D}_{1:n-1})) d\boldsymbol{\theta} = p(y|\mathbf{x}, \hat{\boldsymbol{\theta}}_m(\mathcal{D}_{1:n-1})). \quad (5.54)$$

Тогда получаем:

$$\log p(\mathcal{D}|m) \approx \sum_{n=1}^N \log p(y_n|\mathbf{x}_n, \hat{\boldsymbol{\theta}}_m(\mathcal{D}_{1:n-1})). \quad (5.55)$$

Это очень похоже на оценку правдоподобия методом перекрестной проверки с исключением по одному, только вычисляемой последовательно. Сложная модель переобучится на «ранних» примерах и будет плохо предсказывать остальные, поэтому мы получим низкое маргинальное правдоподобие и низкую оценку перекрестной проверки. Дальнейшее обсуждение см. в работе [FH20].

5.2.5. Информационные критерии

Маргинальное правдоподобие, $p(\mathcal{D}|m) = \int p(\mathcal{D}|\boldsymbol{\theta}, m) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$, необходимое для байесовского выбора модели, рассмотренного в разделе 5.2.2, бывает трудно вычислить, потому что требуется интегрирование по всему пространству параметров. К тому же, результат может сильно зависеть от выбора априорного распределения. В этом разделе мы обсудим другие метрики для выбора модели, называемые **информационными критериями**. Наше обсуждение будет кратким, дополнительные сведения см., например, в [GHV14].

5.2.5.1. Байесовский информационный критерий (BIC)

Байесовский информационный критерий (BIC) [Sch78] можно представлять себе как простую аппроксимацию логарифмического маргинального правдоподобия. В частности, если взять гауссову аппроксимацию апостериорного распределения, обсуждавшуюся в разделе 4.6.8.2, то из формулы (4.215) получим:

$$\log p(\mathcal{D}|m) \approx \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}_{\text{map}}) + \log p(\hat{\boldsymbol{\theta}}_{\text{map}}) - \frac{1}{2} \log |\mathbf{H}|, \quad (5.56)$$

где \mathbf{H} – гессиан отрицательного логарифмического совместного распределения $\log p(\mathcal{D}, \boldsymbol{\theta})$, вычисленного в точке $\hat{\boldsymbol{\theta}}_{\text{map}}$, соответствующей оценке MAP. В формуле (5.56) мы видим логарифмическое правдоподобие плюс штрафующие члены. Если априорное распределение равномерное, $p(\boldsymbol{\theta}) \propto 1$, то соответствующий член можно отбросить и заменить оценку MAP оценкой MLE, $\hat{\boldsymbol{\theta}}$, что дает

$$\log p(\mathcal{D}|m) \approx \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}) - \frac{1}{2} \log |\mathbf{H}|. \quad (5.57)$$

Теперь сосредоточимся на члене $\log |\mathbf{H}|$, который иногда называют **коэффициентом Оккама**, поскольку он является мерой P сложности модели (объем апостериорного распределения). Имеем $\mathbf{H} = \sum_{i=1}^N \mathbf{H}_i$, где $\mathbf{H}_i = \nabla \nabla \log p(\mathcal{D}|\boldsymbol{\theta})$. Аппроксимируем каждое \mathbf{H}_i фиксированной матрицей $\hat{\mathbf{H}}$. Тогда будем иметь

$$\log |\mathbf{H}| = \log |N\hat{\mathbf{H}}| = \log(N^D |\hat{\mathbf{H}}|) = D \log N + \log |\hat{\mathbf{H}}|, \quad (5.58)$$

где $D = \dim(\boldsymbol{\theta})$, и мы предположили, что матрица \mathbf{H} полного ранга. Член $\log |\hat{\mathbf{H}}|$ можно отбросить, потому что он не зависит от N и потому будет пренебрежимо мал по сравнению с правдоподобием. Собирая все вместе, получаем **оценку BIC**, которую нужно максимизировать:

$$J_{\text{BIC}}(m) = \log p(\mathcal{D}|m) \approx \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) - \frac{D_m}{2} \log N. \quad (5.59)$$

Можно также определить **потерю BIC**, подлежащую минимизации, умножив на -2 :

$$\mathcal{L}_{\text{BIC}}(m) = -2 \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) + D_m \log N. \quad (5.60)$$

(Масштабный коэффициент 2 выбран, чтобы упростить выражение при использовании модели с гауссовым правдоподобием.)

5.2.5.2. Информационный критерий Акаике

Информационный критерий Акаике [Aka74] тесно связан с BIC. Он имеет вид:

$$\mathcal{L}_{\text{AIC}}(m) = -2\log p(\mathcal{D}|\hat{\theta}, m) + 2D. \quad (5.61)$$

Эта метрика штрафует сложные модели не так сильно, как BIC, потому что член регуляризации не зависит от N . Эту оценку можно вывести, встав на частотную точку зрения.

5.2.5.3. Минимальная длина описания (MDL)

Проблему сравнительной оценки различных моделей можно рассматривать в терминах теории информации (глава 6). Цель отправителя – передать данные получателю. Сначала отправитель должен определить, какую модель m будет использовать; это требует $C(m) = -\log p(m)$ битов (см. раздел 6.1). Затем получатель может обучить модель, вычислив $\hat{\theta}_m$, и таким образом приближенно реконструировать данные. Для точной реконструкции данных отправитель должен передать невязки, которые нельзя объяснить моделью; это требует $-L(m) = -\log p(\mathcal{D}|\hat{\theta}, m) = -\sum_n \log p(y_n|\hat{\theta}, m)$ битов. Полная стоимость равна:

$$\mathcal{L}_{\text{MDL}}(m) = -\log p(\mathcal{D}|\hat{\theta}, m) + C(m). \quad (5.62)$$

Как видим, общая форма такая же, как в критериях BIC/AIC. Выбор модели, доставляющей минимум $J(m)$, называется **принципом минимальной длины описания** (minimum description length – MDL). Детали см., например, в работе [HY01].

5.3. ЧАСТОТНАЯ ТЕОРИЯ ПРИНЯТИЙ РЕШЕНИЙ

В этом разделе мы обсудим **частотную теорию принятия решений**. Она похожа на байесовскую, обсуждавшуюся в разделе 5.1, но отличается отсутствием априорного, а значит, и апостериорного распределения неизвестных состояний природы. Следовательно, мы не можем определить риск как апостериорную ожидаемую потерю. В разделе 5.3.1 мы рассмотрим другие определения.

5.3.1. Вычисление риска оценки

Определим частотный **риск** оценки π при условии неизвестного состояния природы θ как математическое ожидание потери в случае применения этой оценки к данным \mathbf{x} , выбранным из функции правдоподобия $p(\mathbf{x}|\theta)$:

$$R(\theta, \pi) \triangleq \mathbb{E}_{p(\mathbf{x}|\theta)}[\ell(\theta, \pi(\mathbf{x}))]. \quad (5.63)$$

Пример приведен в разделе 5.3.1.1.

5.3.1.1. Пример

Наш пример заимствован из работы [BS94]. Рассмотрим задачу оценки среднего гауссова распределения. Предполагается, что данные выбраны из распределения $x_n \sim \mathcal{N}(\theta^*, \sigma^2 = 1)$. Если использовать квадратичную потерю $\ell_2(\theta, \hat{\theta}) = (\theta - \hat{\theta})^2$, то соответствующей функцией риска будет СКО.

Рассмотрим пять оценок для вычисления θ :

- $\pi_1(\mathcal{D}) = \bar{x}$, выборочное среднее;
- $\pi_2(\mathcal{D}) = \text{median}(\mathcal{D})$, выборочная медиана;
- $\pi_3(\mathcal{D}) = \theta_0$, фиксированное значение;
- $\pi_\kappa(\mathcal{D})$, апостериорное среднее при априорном распределении $\mathcal{N}(\theta|\theta_0, \sigma^2/\kappa)$:

$$\pi_\kappa(\mathcal{D}) = \frac{N}{N + \kappa} \bar{x} + \frac{\kappa}{N + \kappa} \theta_0 = w\bar{x} + (1 - w)\theta_0. \quad (5.64)$$

Для π_κ мы берем $\theta_0 = 0$ и рассматриваем слабое априорное распределение с $\kappa = 1$ и более сильное с $\kappa = 5$.

Пусть $\hat{\theta} = \hat{\theta}(\mathbf{x}) = \pi(\mathbf{x})$ – оцениваемый параметр. Риск этой оценки определяется СКО. В разделе 4.7.6.3 мы показали, что СКО можно представить в виде суммы квадрата смещения и дисперсии:

$$\text{MSE}(\hat{\theta}|\theta^*) = \mathbb{V}[\hat{\theta}] + \text{bias}^2(\hat{\theta}), \quad (5.65)$$

где смещение определено как $\text{bias}(\hat{\theta}) = \mathbb{E}[\hat{\theta} - \theta^*]$. Теперь воспользуемся этим выражением, чтобы вывести риск каждой оценки.

π_1 – выборочное среднее. Это несмещенная оценка, поэтому ее риск равен:

$$\text{MSE}(\pi_1|\theta^*) = \mathbb{V}[\bar{x}] = \frac{\sigma^2}{N}. \quad (5.66)$$

π_2 – выборочная медиана. Это также несмещенная оценка. Кроме того, можно показать, что ее дисперсия приближенно равна $\pi/(2N)$, так что риск равен:

$$\text{MSE}(\pi_2|\theta^*) = \frac{\pi}{2N}. \quad (5.67)$$

π_3 возвращает постоянную θ_0 , так что ее смещение равно $(\theta^* - \theta_0)$, а дисперсия нулевая. Поэтому риск равен:

$$\text{MSE}(\pi_3|\theta^*) = (\theta^* - \theta_0)^2. \quad (5.68)$$

Наконец, π_4 – апостериорное среднее при априорном гауссовом распределении. Его СКО можно вывести следующим образом:

$$\text{MSE}(\pi_\kappa|\theta^*) = \mathbb{E}[(w\bar{x} + (1 - w)\theta_0 - \theta^*)^2] \quad (5.69)$$

$$= \mathbb{E}[(w(\bar{x} - \theta^*) + (1 - w)(\theta_0 - \theta^*))^2] \quad (5.70)$$

$$= w^2 \frac{\sigma^2}{N} + (1-w)^2 (\theta_0 - \theta^*)^2 \quad (5.71)$$

$$= \frac{1}{(N + \kappa)^2} (N\sigma^2 + \kappa^2 (\theta_0 - \theta^*)^2). \quad (5.72)$$

Графики этих функций показаны на рис. 5.8 для $N \in \{5, 20\}$. Мы видим, что в общем случае наилучшая оценка зависит от значения θ^* , которое нам неизвестно. Если θ^* очень близко к θ_0 , то π_3 (которая просто предсказывает θ_0) наилучшая. Если θ^* принадлежит разумному диапазону вокруг θ_0 , то наилучшей оценкой будет апостериорное среднее, которое сочетает априорную гипотезу θ_0 с фактическими данными. Если θ^* далеко от θ_0 , то наилучшей оценкой будет MLE.

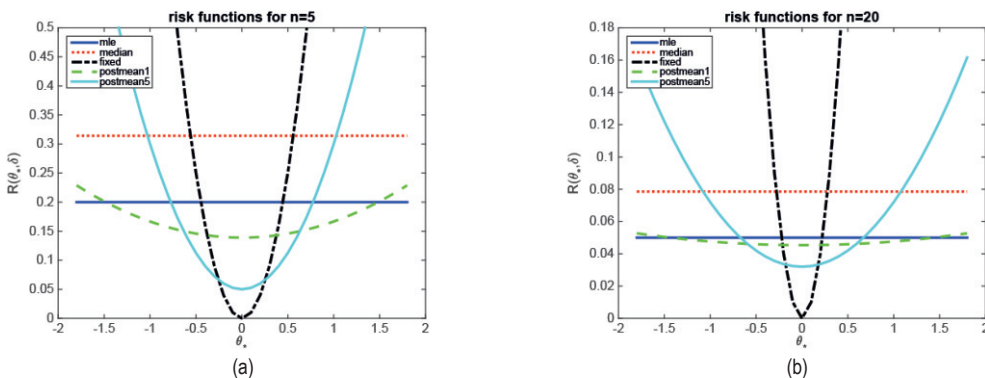


Рис. 5.8 ❖ Функции риска для оценивания среднего гауссова распределения. Каждая кривая представляет график зависимости $R(\hat{\theta}_i(\cdot), \theta^*)$ от θ^* , где i – индекс оценки. Каждая оценка применяется к N примерам, выбранным из $\mathcal{N}(\theta^*, \sigma^2 = 1)$. Синяя горизонтальная прямая – выборочное среднее (MLE); красная горизонтальная прямая – выборочная медиана, черная кривая – оценка $\hat{\theta} = \theta_0 = 0$; зеленая кривая – апостериорное среднее при $\kappa = 1$; голубая кривая – апостериорное среднее при $\kappa = 5$. (a) $N = 5$ примеров. (b) $N = 20$ примеров. На основе рис. В.1 из работы [BS94]. Построено программой по адресу figures.problml.ai/book1/5.8

5.3.1.2. Байесовский риск

В общем случае истинное состояние природы θ , порождающее данные \mathbf{x} , неизвестно, поэтому мы не можем вычислить риск в формуле (5.63). Возможное решение – высказать гипотезу об априорном распределении ρ для θ , а затем произвести по нему усреднение. Это дает нам **байесовский риск**, который называют также **интегральным риском**:

$$R(\rho, \pi) \triangleq \mathbb{E}_{\rho(\theta)}[R(\theta, \pi)] = \int d\theta d\mathbf{x} \rho(\theta) p(\mathbf{x}|\theta) \ell(\theta, \pi(\mathbf{x})). \quad (5.73)$$

Решающее правило, заключающееся в минимизации байесовского риска, называется **байесовской оценкой**. Это эквивалент оптимальной политики, рекомендуемой байесовской теорией принятия решений (5.2), поскольку

$$\pi(\mathbf{x}) = \operatorname{argmin}_a \int d\theta p(\theta) p(\mathbf{x}|\theta) \ell(\theta, a) = \operatorname{argmin}_a \int d\theta p(\theta|\mathbf{x}) \ell(\theta, a). \quad (5.74)$$

Таким образом, мы видим, что выбор оптимального действия отдельно для каждого случая (как при байесовском подходе) является оптимальным в среднем (как при частотном подходе). Иными словами, байесовский подход – это хороший способ достижения частотных целей. Дальнейшее обсуждение этого вопроса см. в работе [BS94, стр. 448].

5.3.1.3. Максимальный риск

Разумеется, использование априорного распределения может показаться нежелательным в контексте частотной статистики. Поэтому мы можем следующим образом определить **максимальный риск**:

$$R_{\max}(\pi) \triangleq \sup_{\theta} R(\theta, \pi). \quad (5.75)$$

Решающее правило, которое требует минимизировать максимальный риск, называется **минимаксной оценкой** и обозначается $\pi_{\text{ММ}}$. Например, на рис. 5.9 показано, что для π_1 риск в худшем случае меньше, чем для π_2 , на всех возможных значениях θ , так что это минимаксная оценка.

Минимаксные оценки в чем-то привлекательны. Но вычислять их трудно. И, кроме того, они очень пессимистичны. На самом деле можно показать, что все минимаксные оценки эквивалентны байесовским оценкам при **наименее выгодном априорном распределении**. В большинстве статистических задач (за исключением теоретико-игровых) предположение о враждебном поведении природы неразумно.

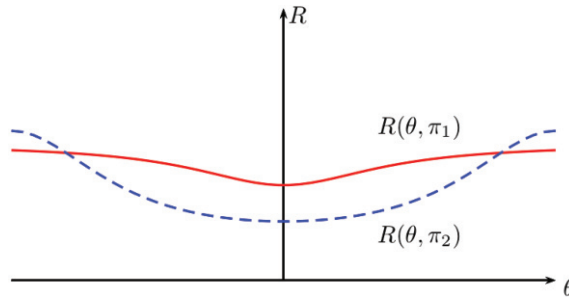


Рис. 5.9 ❖ Функции риска для двух решающих процедур, π_1 и π_2 . Поскольку риск π_1 в худшем случае меньше, она является минимаксной оценкой, пусть даже для π_2 риск меньше для большинства значений θ . Таким образом, минимаксные оценки чрезмерно консервативны

5.3.2. Состоятельные оценки

Пусть имеется набор данных $\mathcal{D} = \{\mathbf{x}_n : n = 1 \dots N\}$, где примеры $\mathbf{x}_n \in \mathcal{X}$ порождены распределением $p(\mathbf{x}|\theta^*)$, а $\theta^* \in \Theta$ – истинные параметры. Предположим

еще, что параметры **идентифицируемые**, т. е. $p(\mathcal{D}|\theta) = p(\mathcal{D}|\theta')$ тогда и только тогда, когда $\theta = \theta'$ для любого набора данных \mathcal{D} . Тогда говорят, что оценка $\pi: \mathcal{X}^N \rightarrow \Theta$ **состоятельная**, если $\hat{\theta}(\mathcal{D}) \rightarrow \theta$, когда $N \rightarrow \infty$ (где стрелка означает сходимость вероятностей). Иными словами, процедура π восстанавливает истинные параметры (или их подмножество) в пределе, когда данных становится бесконечно много. Это эквивалентно минимизации пороговой потери, $\mathcal{L}(\theta^*, \hat{\theta}) = \mathbb{I}(\theta^* \neq \hat{\theta})$. Пример состоятельной оценки дает оценка максимального правдоподобия (MLE).

Отметим, что оценка может быть несмещенной, но при этом несостоятельной. Например, рассмотрим оценку $\pi(\{\mathbf{x}_1, \dots, \mathbf{x}_N\}) = \mathbf{x}_N$. Это несмещенная оценка среднего, потому что $\mathbb{E}[\pi(\mathcal{D})] = \mathbb{E}[\mathbf{x}]$. Но выборочное распределение $\pi(\mathcal{D})$ не сходится к фиксированному значению, поэтому не может сходиться к точке θ^* .

Хотя состоятельность – желательное свойство, ее полезность на практике ограничена, потому что большинство реальных наборов данных не происходит из выбранного нами семейства моделей (т. е. не существует такого θ^* , что $p(\cdot|\theta^*)$ порождает наблюдаемые данные \mathcal{D}). На практике полезнее искать оценки, которые минимизируют некоторую меру различия между эмпирическим распределением $p_{\mathcal{D}}(\mathbf{x}|\mathcal{D})$ и оценкой распределения $p(\mathbf{x}|\theta)$. Если в качестве меры различия взять расхождение КЛ, то оценкой оказывается MLE.

5.3.3. Допустимые оценки

На рис. 5.8 видно, что выборочная медиана (пунктирная красная линия) всегда имеет больший риск, чем выборочное среднее (сплошная черная линия), по крайней мере, при гауссовой модели правдоподобия. Поэтому говорят, что выборочное среднее **мажорирует** выборочную медиану в качестве оценки.

Говорят, что π_1 **мажорирует** π_2 , если $R(\theta, \pi_1) \leq R(\theta, \pi_2)$ для всех θ . Мажорирование называется строгим, если это неравенство строгое для какого-то θ . Оценка называется **допустимой**, если не существует другой строго мажорирующей ее оценки.

На рис. 5.8 видно, что выборочная медиана (пунктирная красная линия) всегда имеет больший риск, чем выборочное среднее (сплошная черная линия). Поэтому выборочная медиана не является допустимой оценкой среднего. Однако это заключение применимо только к гауссовой модели правдоподобия. Если истинная модель $p(\mathbf{x}|\theta)$ характеризуется тяжелыми хвостами, как распределение Лапласа или Стьюдента, или является смесью гауссовых распределений, то выборочная медиана, скорее всего, будет иметь меньший риск, потому что более робастна к выбросам.

Удивительнее другой факт – можно показать, что выборочное среднее тоже не всегда является допустимой оценкой, даже в случае гауссовой модели правдоподобия с квадратичной ошибкой в качестве функции потерь (это называется **парадоксом Штейна** [Ste56]).

Естественно сосредоточить внимание только на допустимых оценках. В работе [Wal47] доказан следующий важный результат (известный под названием **теоремы о полном классе**), который по существу означает, что тем самым мы ограничиваемся только байесовскими оценками.

Теорема 5.3.1 (Вальд). *Любое допустимое частотное решающее правило является байесовским решающим правилом при некотором, возможно несобственном, априорном распределении.*

Однако полезность понятия допустимости ограничена. Например, в следующем примере показано, как легко построить допустимые, но «глупые» оценки.

Теорема 5.3.2. *Пусть $X \sim \mathcal{N}(\theta, 1)$. Рассмотрим оценку θ с квадратичной функцией потерь. Пусть $\pi_1(x) = \theta_0$ – постоянная, не зависящая от данных. Эта оценка допустима.*

Доказательство. Предположим, что это не так. Тогда существует другая оценка π_2 с меньшим риском такая, что $R(\theta^*, \pi_2) \leq R(\theta^*, \pi_1)$, причем неравенство должно быть строгим хотя бы для одного θ^* . Рассмотрим риск при $\theta^* = \theta_0$. Имеем $R(\theta_0, \pi_1) = 0$:

$$R(\theta_0, \pi_2) = \int (\pi_2(x) - \theta_0)^2 p(x|\theta_0) dx. \quad (5.76)$$

Поскольку $0 \leq R(\theta^*, \pi_2) \leq R(\theta^*, \pi_1)$ для всех θ^* и $R(\theta_0, \pi_1) = 0$, имеем $R(\theta_0, \pi_2) = 0$, а значит, $\pi_2(x) = \theta_0 = \pi_1(x)$. Таким образом, есть лишь одна возможность, при которой риск π_2 в точке θ_0 не выше, чем риск π_1 , – когда π_2 совпадает с π_1 . Следовательно, не существует другой оценки π_2 со строго меньшим риском, и значит, π_1 допустима. ■

Заметим, что оценка $\pi_1(x) = \theta_0$ эквивалентна байесовскому решающему правилу относительно априорного распределения, достоверно сосредоточенного в θ_0 , именно поэтому она глупая.

5.4. МИНИМИЗАЦИЯ ЭМПИРИЧЕСКОГО РИСКА

В этом разделе мы рассмотрим, как применять частотную теорию принятия решений в контексте обучения с учителем.

5.4.1. Эмпирический риск

В стандартных изложениях частотной теории принятия решений в учебниках математической статистики существует единственное неизвестное «состояние природы», соответствующее неизвестным параметрам θ^* некоторой модели, а риск определяется, как в формуле (5.63): $R(\pi, \theta^*) = \mathbb{E}_{p(\mathcal{D}|\theta^*)}[\ell(\theta^*, \pi(\mathcal{D}))]$.

В обучении с учителем у нас имеется другое неизвестное состояние природы (именно, выход y) для каждого входа x , а оценкой π является прогнозная функция $\hat{y} = f(x)$ и состояние природы – это истинное распределение $p^*(x, y)$. Таким образом, риск оценки равен:

$$R(f, p^*) = R(f) \triangleq \mathbb{E}_{p^*(\mathbf{x})p^*(\mathbf{y}|\mathbf{x})}[\ell(\mathbf{y}, f(\mathbf{x}))]. \quad (5.77)$$

Эта величина называется **риском на генеральной совокупности** (population risk), потому математическое ожидание берется относительно истинного совместного распределения $p^*(\mathbf{x}, \mathbf{y})$. Конечно, p^* неизвестно, но мы можем аппроксимировать его эмпирическим распределением N примеров:

$$p_{\mathcal{D}}(\mathbf{x}, \mathbf{y}|\mathcal{D}) \triangleq \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}_n, \mathbf{y}_n) \in \mathcal{D}} \delta(\mathbf{x} - \mathbf{x}_n) \delta(\mathbf{y} - \mathbf{y}_n), \quad (5.78)$$

где $p_{\mathcal{D}}(\mathbf{x}, \mathbf{y}) = p_{\text{train}}(\mathbf{x}, \mathbf{y})$. Подстановка этого распределения дает **эмпирический риск**:

$$R(f, \mathcal{D}) \triangleq \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x}, \mathbf{y})}[\ell(\mathbf{y}, f(\mathbf{x}))] = \frac{1}{N} \sum_{n=1}^N \ell(\mathbf{y}_n, f(\mathbf{x}_n)). \quad (5.79)$$

Отметим, что здесь $R(f, \mathcal{D})$ – случайная величина, т. к. она зависит от обучающего набора. Естественно выбрать предиктор следующим образом:

$$\hat{f}_{\text{ERM}} = \underset{f \in \mathcal{H}}{\operatorname{argmin}} R(f, \mathcal{D}) = \underset{f \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^N \ell(\mathbf{y}_n, f(\mathbf{x}_n)), \quad (5.80)$$

где оптимизация производится по конкретному функциональному **пространству гипотез** \mathcal{H} . Это называется **минимизацией эмпирического риска** (empirical risk minimization – ERM).

5.4.1.1. Ошибка аппроксимации и ошибка оценивания

В этом разделе мы проанализируем теоретическое качество аппроксимирующих функций, выбираемых в соответствии с принципом ERM. Пусть $f^{**} = \underset{f}{\operatorname{argmin}} R(f)$ – функция, доставляющая минимум риску на генеральной совокупности при оптимизации на множестве всех возможных функций. Конечно, мы не можем рассмотреть все вообще функции, поэтому определим $f^* = \underset{f \in \mathcal{H}}{\operatorname{argmin}} R(f)$ как наилучшую функцию в нашем пространстве гипотез \mathcal{H} . К сожалению, мы не можем вычислить f^* , потому что не можем вычислить риск на генеральной совокупности, поэтому окончательно остановимся на определении функции-предиктора, которая минимизирует эмпирический риск на пространстве гипотез:

$$f_N^* = \underset{f \in \mathcal{H}}{\operatorname{argmin}} R(f, \mathcal{D}) = \underset{f \in \mathcal{H}}{\operatorname{argmin}} \mathbb{E}[\ell(\mathbf{y}, f(\mathbf{x}))]. \quad (5.81)$$

Можно показать [BB08], что риск разности между выбранным нами и наилучшим возможным предиктором можно разложить в сумму двух членов:

$$\mathbb{E}_p[R(f_N^*) - R(f^{**})] = \underbrace{R(f^*) - R(f^{**})}_{\mathbb{E}_{\text{app}}(\mathcal{H})} + \underbrace{\mathbb{E}_p[R(f_N^*) - R(f^*)]}_{\mathbb{E}_{\text{est}}(\mathcal{H}, N)}. \quad (5.82)$$

Первый член, $\mathcal{E}_{\text{app}}(\mathcal{H})$, – **ошибка аппроксимации**, которая показывает, насколько точно \mathcal{H} может смоделировать истинную оптимальную функцию f^* . Второй член, $\mathcal{E}_{\text{est}}(\mathcal{H}, N)$, – **ошибка оценивания**, или **ошибка обобщения**, она измеряет разность между оценками рисков в силу конечности обучающего набора. Ее можно аппроксимировать разностью между ошибкой на обучающем наборе и ошибкой на тестовом наборе, используя два эмпирических распределения, выбранных из p^* :

$$\mathbb{E}_p[R(f_N^*) - R(f^*)] \approx \mathbb{E}_{p_{\text{train}}}[\ell(\mathbf{y}, f_N^*(\mathbf{x}))] - \mathbb{E}_{p_{\text{test}}}[\ell(\mathbf{y}, f_N^*(\mathbf{x}))]. \quad (5.83)$$

Эту разность часто называют **разрывом обобщения**.

Мы можем уменьшить ошибку аппроксимации, воспользовавшись более выразительным семейством функций \mathcal{H} , но обычно при этом увеличивается ошибка обобщения из-за переобучения. Как разрешить эту дилемму, мы обсудим ниже.

5.4.1.2. Регуляризованный риск

Чтобы избежать переобучения, обычно прибавляют к целевой функции штраф за сложность, получая в результате **регуляризованный эмпирический риск**:

$$R_\lambda(f, \mathcal{D}) = R(f, \mathcal{D}) + \lambda C(f), \quad (5.84)$$

где $C(f)$ измеряет сложность функции-предиктора $f(\mathbf{x}, \boldsymbol{\theta})$, а величина $\lambda \geq 0$, называемая **гиперпараметром**, управляет величиной штрафа за сложность (как выбирать λ , мы обсудим в разделе 5.4.2).

На практике обычно работают с параметрическими функциями и применяют регуляризатор к самим параметрам. В результате получается целевая функция такого вида:

$$R_\lambda(\boldsymbol{\theta}, \mathcal{D}) = R(\boldsymbol{\theta}, \mathcal{D}) + \lambda C(\boldsymbol{\theta}). \quad (5.85)$$

Отметим, что если используется логарифмическая потеря, а регуляризатором является отрицательное логарифмическое априорное распределение, то регуляризованный риск имеет вид:

$$R_\lambda(\boldsymbol{\theta}, \mathcal{D}) = -\frac{1}{N} \sum_{n=1}^N \log p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}) - \lambda \log p(\boldsymbol{\theta}). \quad (5.86)$$

Его минимизация эквивалентна вычислению оценки МАР.

5.4.2. Структурный риск

Естественный способ оценить гиперпараметры – минимизировать наименьший достижимый эмпирический риск:

$$\hat{\lambda} = \underset{\lambda}{\operatorname{argmin}} \min_{\boldsymbol{\theta}} R_\lambda(\boldsymbol{\theta}, \mathcal{D}). \quad (5.87)$$

Это пример **двухуровневой**, или **вложенной, оптимизации**. К сожалению, такой метод работать не будет, потому что он всегда выбирает наименьшую степень регуляризации, т. е. $\hat{\lambda} = 0$. Чтобы убедиться в этом, заметим, что

$$\operatorname{argmin}_{\lambda} \min_{\theta} R_{\lambda}(\theta, \mathcal{D}) = \operatorname{argmin}_{\lambda} \min_{\theta} R(\theta, \mathcal{D}) + \lambda C(\theta), \quad (5.88)$$

а эта величина достигает минимума, когда $\lambda = 0$. Проблема в том, что эмпирический риск является заниженной оценкой риска на генеральной совокупности, что приводит к переобучению при выборе λ . Это называется **оптимистичностью ошибки на обучающем наборе**.

Если бы мы знали регуляризированный риск на генеральной совокупности $R_{\lambda}(\theta)$ вместо регуляризированного эмпирического риска $R_{\lambda}(\theta, \mathcal{D})$, то могли бы использовать его для выбора модели правильной сложности (например, значения λ). Это называется **минимизацией структурного риска** [Vap98]. Существует два основных способа оценки риска на генеральной совокупности для данной модели (значения λ): перекрестная проверка (раздел 5.4.3) и статистическая теория обучения (раздел 5.4.4).

5.4.3. Перекрестная проверка

В этом разделе мы обсудим простой способ оценки риска на генеральной совокупности в контексте обучения с учителем. Мы просто разбиваем набор данных на две части, одна используется для обучения модели, а другая, называемая **контрольным набором** или **зарезервированным набором**, используется для оценки риска. Мы можем обучить модель на обучающем наборе, а затем использовать ее результаты на контрольном наборе в качестве приближения к риску на генеральной совокупности.

Для более подробного описания метода нам понадобится ввести некоторые обозначения. Сначала выразим явно зависимость эмпирического риска от набора данных:

$$R_{\lambda}(\theta, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \ell(\mathbf{y}, f(\mathbf{x}; \theta)) + \lambda C(\theta). \quad (5.89)$$

Определим также $\hat{\theta}_{\lambda}(\mathcal{D}) = \operatorname{argmin}_{\theta} R_{\lambda}(\mathcal{D}, \theta)$. Наконец, обозначим $\mathcal{D}_{\text{train}}$ и $\mathcal{D}_{\text{valid}}$ части набора \mathcal{D} . (Часто под обучающий набор отводится 80 % данных, а под контрольный 20 %.)

Любую модель λ мы сначала обучаем на обучающем наборе и вычисляем $\hat{\theta}_{\lambda}(\mathcal{D}_{\text{train}})$. Затем мы используем нерегуляризированный эмпирический **риск на контрольном наборе** в качестве оценки риска на генеральной совокупности:

$$R_{\lambda}^{\text{val}} \triangleq R_0(\hat{\theta}_{\lambda}(\mathcal{D}_{\text{train}}), \mathcal{D}_{\text{valid}}). \quad (5.90)$$

Отметим, что для обучения и оценки модели используются разные данные.

Описанная техника может работать очень хорошо. Однако если количество обучающих примеров мало, то возникают проблемы, потому что модели

не хватает данных для обучения, и объем данных для получения надежной оценки качества работы на будущих данных тоже недостаточен.

Простое, но популярное решение этой проблемы – воспользоваться **перекрестной проверкой** (cross validation – CV). Идея такова: обучающие данные разбиваются на K групп; затем для каждой группы $k \in \{1, \dots, K\}$ мы обучаем модель на всех данных, кроме k -й группы, и проверяем на k -й группе, как показано на рис. 4.6. Формально имеем

$$R_{\lambda}^{\text{cv}} \triangleq \frac{1}{K} \sum_{k=1}^K R_0(\hat{\theta}_{\lambda}(\mathcal{D}_{-k}), \mathcal{D}_k), \quad (5.91)$$

где \mathcal{D}_k – данные в k -й группе, а \mathcal{D}_{-k} – все остальные данные. Это называется **перекрестным риском** (cross-validated risk). На рис. 4.6 эта процедура показана для $K = 5$. Для $K = N$ метод называется **перекрестной проверкой с исключением по одному**, потому что мы всегда обучаем на $N - 1$ примере и проверяем на оставшемся.

Оценку CV можно использовать в качестве целевой функции внутри процедуры оптимизации для выбора оптимального гиперпараметра $\hat{\lambda} = \operatorname{argmin}_{\lambda} R_{\lambda}^{\text{cv}}$. В самом конце мы можем объединить все имеющиеся данные (обучающие и контрольные) и еще раз оценить параметры модели $\hat{\theta} = \operatorname{argmin}_{\theta} R_{\lambda}(\theta, \mathcal{D})$.

5.4.4. Статистическая теория обучения*

Принципиальная проблема перекрестной проверки – низкая скорость, поскольку модель приходится обучать несколько раз. Поэтому возникает желание аналитически вычислить аппроксимации или границы риска на генеральной совокупности. Это предмет **статистической теории обучения** (СТО, англ. SLT) (см., например, [Var98]).

Точнее, целью СТО является вычисление верхней границы ошибки обобщения с некоторой вероятностью. Если эта граница удовлетворяется, то есть уверенность, что для гипотезы, выбранной методом минимизации эмпирического риска, риск на генеральной совокупности тоже будет низким. В случае бинарных классификаторов это означает, что гипотеза будет давать правильные предсказания; в таком случае мы говорим, что она **вероятно приближенно корректна** (probably approximately correct – PAC), а класс гипотез называется **PAC-обучаемым** (детали см., например, в [KV94]).

5.4.4.1. Нахождение границы ошибки обобщения

В этом разделе мы выведем условия, при которых можно доказать, что класс гипотез является PAC-обучаемым. Сначала рассмотрим случай конечного пространства гипотез размером $\dim(\mathcal{H}) = |\mathcal{H}|$. Иными словами, мы выбираем гипотезу из конечного списка, а не оптимизируем вещественные параметры. Тогда можно доказать следующую теорему.

Теорема 5.4.1. Для любого распределения данных p^* и любого набора данных \mathcal{D} размера N , выбранного из p^* вероятность того, что ошибка обобщения бинарного классификатора будет больше ε в худшем случае, ограничена сверху значением

$$P\left(\max_{h \in \mathcal{H}} |R(h) - R(h, \mathcal{D})| > \varepsilon\right) \leq 2 \dim(\mathcal{H}) e^{-2N\varepsilon^2}, \quad (5.92)$$

где $R(h, \mathcal{D}) = (1/N) \sum_{i=1}^N \mathbb{I}(f(\mathbf{x}_i) \neq y_i^*)$ – эмпирический риск, а $R(h) = \mathbb{E}[\mathbb{I}(f(\mathbf{x}) \neq y^*)]$ – риск на генеральной совокупности.

Доказательство. Прежде чем приступить к доказательству, приведем два полезных результата. Первый – **неравенство Хёфдинга**, утверждающее, что если $E_1, \dots, E_N \sim \text{Ver}(\theta)$, то для любого $\varepsilon > 0$

$$P(|\bar{E} - \theta| > \varepsilon) \leq 2e^{-2N\varepsilon^2}, \quad (5.93)$$

где $\bar{E} = (1/N) \sum_{i=1}^N E_i$ – эмпирическая частота ошибок, а θ – истинная частота ошибок. Второй – **граница объединения** – утверждает, что если A_1, \dots, A_d – множество событий, то $P(\bigcup_{i=1}^d A_i) \leq \sum_{i=1}^d P(A_i)$. Применяя эти результаты, имеем:

$$P\left(\max_{h \in \mathcal{H}} |R(h) - R(h, \mathcal{D})| > \varepsilon\right) = P\left(\bigcup_{h \in \mathcal{H}} |R(h) - R(h, \mathcal{D})| > \varepsilon\right) \quad (5.94)$$

$$\leq \sum_{h \in \mathcal{H}} P(|R(h) - R(h, \mathcal{D})| > \varepsilon) \quad (5.95)$$

$$\leq \sum_{h \in \mathcal{H}} 2e^{-2N\varepsilon^2} = 2 \dim(\mathcal{H}) e^{-2N\varepsilon^2}. \quad (5.96)$$

■

Эта граница говорит, что оптимистичность оценки на обучающем наборе возрастает с увеличением $\dim(\mathcal{H})$, но убывает с увеличением $N = |\mathcal{D}|$, как и следовало ожидать.

5.4.4.2. VC-размерность

Если пространство гипотез \mathcal{H} бесконечно (т. е. параметры вещественные), то использовать $\dim(\mathcal{H}) = |\mathcal{H}|$ невозможно. Вместо этого используется **VC-размерность** класса гипотез, названная так в честь Вапника и Червоненкиса; она измеряет количество степеней свободы (эффективное число параметров) класса гипотез. Детали см., например, в работе [Var98].

К сожалению, вычислить VC-размерность многих интересных моделей трудно, а ее верхние границы обычно очень нестрогие, так что практическая ценность этого подхода ограничена. Однако недавно были предложены другие, более практичные, оценки ошибки обобщения, особенно для глубоких нейронных сетей (см., например, [Jia+20]).

5.5. ЧАСТОТНАЯ ПРОВЕРКА ГИПОТЕЗ*

Пусть имеется две гипотезы, **нулевая** H_0 и **альтернативная** H_1 , и на основе набора данных \mathcal{D} требуется выбрать предположительно правильную. Можно было бы воспользоваться байесовским подходом и вычислить коэффициент

Байеса $p(H_0|\mathcal{D})/p(H_1|\mathcal{D})$, как было описано в разделе 5.2. Однако для этого необходимо проинтегрировать по всем возможным параметризациям моделей H_0 и H_1 , что может быть трудно с вычислительной точки зрения, а кроме того, результат может сильно зависеть от выбора априорного распределения. В этом разделе мы рассмотрим частотный подход к решению этой проблемы.

5.5.1. Критерий отношения правдоподобия

Если использовать бинарную потерю и предположить, что $p(H_0) = p(H_1)$, то оптимальным будет решение принять H_0 тогда и только тогда, когда $p(\mathcal{D}|H_0)/p(\mathcal{D}|H_1) > 1$. Это называется **критерием отношения правдоподобия**. Ниже приводятся примеры.

5.5.1.1. Пример: сравнение гауссовых средних

Предположим, мы хотим знать, стоит ли за данными гауссово распределение со средним μ_0 или μ_1 . (Предполагается, что дисперсия σ^2 одинакова и известна.) Это показано на рис. 5.10а, где изображены графики $p(x|H_0)$ и $p(x|H_1)$. Отношение правдоподобия вычисляется следующим образом:

$$\frac{p(\mathcal{D}|H_0)}{p(\mathcal{D}|H_1)} = \frac{\exp\left(-\frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu_0)^2\right)}{\exp\left(-\frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu_1)^2\right)} \quad (5.97)$$

$$= \exp\left(\frac{1}{2\sigma^2} (2N\bar{x}(\mu_0 - \mu_1) + N\mu_1^2 - N\mu_0^2)\right). \quad (5.98)$$

Как видим, это отношение зависит только от наблюдаемых данных через их среднее \bar{x} . Это пример **критериальной статистики** $\tau(\mathcal{D})$ – достаточной скалярной статистики для проверки гипотез. Из рис. 5.10а видно, что $p(\mathcal{D}|H_0)/p(\mathcal{D}|H_1) > 1$ тогда и только тогда, когда $\bar{x} < x^*$, где x^* – точка пересечения двух функций плотности вероятности (предполагается, что такая точка единственна).

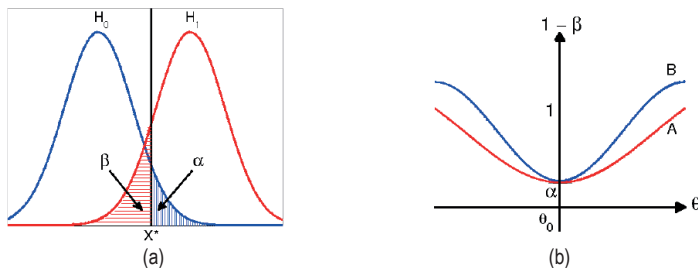


Рис. 5.10 ❖ (а) Парадигма проверки гипотез Неймана–Пирсона. (б) Две гипотетически двусторонние кривые мощности критерия. На основе рис. 6.3.5 из работы [LM86]. Построено программой по адресу figures.problm.ai/book1/5.10

5.5.1.2. Простые и сложные гипотезы

В разделе 5.5.1.1 параметры нулевой и альтернативной гипотез либо задавались полностью (μ_0 и μ_1), либо были общими (σ^2). Это называется проверкой **простых гипотез**. В общем случае могут быть заданы не все параметры гипотезы; это называется **сложной гипотезой**. В таком случае неизвестные параметры следует исключить путем интегрирования, как в байесовском подходе, потому что гипотеза с большим числом параметров всегда будет иметь более высокое правдоподобие. В качестве приближения можно исключить их путем взятия максимума, что дает критерий максимального отношения правдоподобия:

$$\frac{p(H_0|\mathcal{D})}{p(H_1|\mathcal{D})} = \frac{\int_{\theta \in H_0} p(\theta)p_{\theta}(\mathcal{D})}{\int_{\theta \in H_1} p(\theta)p_{\theta}(\mathcal{D})} \approx \frac{\max_{\theta \in H_0} p_{\theta}(\mathcal{D})}{\max_{\theta \in H_1} p_{\theta}(\mathcal{D})}. \quad (5.99)$$

5.5.2. Проверка значимости нулевой гипотезы

Вместо того чтобы предполагать пороговую потерю, обычно проектируют решающее правило, так чтобы **частота ошибок первого рода** (вероятность по ошибке отвергнуть нулевую гипотезу H_0) была равна α . (Подробнее о частотах ошибок бинарных решающих правил см. раздел 5.1.3.) Частота ошибок α называется **значимостью** критерия. А подход в целом называется **проверкой значимости нулевой гипотезы** (null hypothesis significance testing – NHST).

В примере с гауссовым средним (рис. 5.10а) частота ошибок первого рода обозначена заштрихованной синим цветом областью:

$$\alpha(\mu_0) = p(\text{отвергнуть } H_0 | H_0 \text{ истинна}) \quad (5.100)$$

$$= p(\bar{X}(\tilde{\mathcal{D}}) > x^* | \tilde{\mathcal{D}} \sim H_0) \quad (5.101)$$

$$= p\left(\frac{\bar{X} - \mu_0}{\sigma/\sqrt{N}} > \frac{x^* - \mu_0}{\sigma/\sqrt{N}}\right). \quad (5.102)$$

Отсюда $x^* = z_{\alpha}\sigma/\sqrt{N} + \mu_0$, где z_{α} – верхний α -квантиль стандартного нормального распределения.

Частота ошибок второго рода – это вероятность ошибочно принять нулевую гипотезу, хотя истинна альтернативная:

$$\begin{aligned} \beta(\mu_1) &= p(\text{ошибка второго рода}) \\ &= p(\text{принять } H_0 | H_1 \text{ истинна}) = p(\tau(\tilde{\mathcal{D}}) < \tau^* | \tilde{\mathcal{D}} \sim H_1). \end{aligned} \quad (5.103)$$

Эта величина показана заштрихованной красным цветом области на рис. 5.10а. **Мощность** (power) критерия по определению равна $1 - \beta(\mu_1)$; это вероятность отвергнуть гипотезу H_0 при условии, что истинна H_1 . Иными словами, это способность критерия правильно распознавать, что нулевая гипотеза ложна. Очевидно, что наименьшая мощность имеет место, когда $\mu_1 = \mu_0$ (кривые налагаются друг на друга); в этом случае имеем $1 - \beta(\mu_1) = \alpha(\mu_0)$.

Чем дальше отстоят друг от друга μ_1 и μ_0 , тем ближе мощность к 1 (поскольку красная область уменьшается, $\beta \rightarrow 0$). Если имеется два критерия, A и B , где $\text{power}(B) \geq \text{power}(A)$ при одной и той же частоте ошибок первого рода, то говорят, что B **мажорирует** A (см. рис. 5.10b). Критерий самой большой мощности в предположении справедливости H_1 среди всех критериев с уровнем значимости α называется **наиболее мощным критерием**. Оказывается, что наиболее мощным является критерий отношения правдоподобия; этот результат известен как **теорема Неймана–Пирсона**.

5.5.3. р-значения

Когда H_0 отвергается, часто говорят, что результат **статистически значим** при уровне α . Однако результат может быть статистически, но не практически значимым, все зависит от того, насколько далеко от решающей границы находится критериальная статистика.

Вместо того чтобы объявлять результат значимым или незначимым, лучше указать **р-значение**. Так называется вероятность, в предположении справедливости нулевой гипотезы, наблюдать критериальную статистику, большую или равную фактически наблюдаемой:

$$\text{pval}(\tau(\mathcal{D})) \triangleq \Pr(\tau(\tilde{\mathcal{D}}) \geq \tau(\mathcal{D}) | \tilde{\mathcal{D}} \sim H_0). \quad (5.104)$$

Иначе говоря, $\text{pval}(\tau_{\text{obs}}) \triangleq \Pr(\tau_{\text{null}} \geq \tau_{\text{obs}})$, где $\tau_{\text{obs}} = \tau(\mathcal{D})$, $\tau_{\text{null}} = \tau(\tilde{\mathcal{D}})$, а $\tilde{\mathcal{D}} \sim H_0$ – гипотетические будущие данные. Чтобы стала ясна связь с проверкой гипотез, предположим, что выбран такой порог принятия решения t^* , что $\Pr(\tau(\tilde{\mathcal{D}}) \geq t^* | H_0) = \alpha$. Если положить $t^* = \tau(\mathcal{D})$, то $\alpha = \text{pval}(\tau(\mathcal{D}))$.

Следовательно, принимая только гипотезы, для которых р-значение меньше $\alpha = 0.05$, мы в 95 % случаев будем правильно отвергать нулевую гипотезу. Однако это не значит, что альтернативная гипотеза H_1 истинна с вероятностью 0.95. На самом деле даже специалисты в большинстве своем неправильно интерпретируют р-значения¹. Величина, которую чаще всего хотелось бы вычислить, – это байесовская апостериорная вероятность $p(H_1 | \mathcal{D}) = 0.95$. Подробнее об этом важном различии см. раздел 5.5.4.

5.5.4. О вреде р-значений

р-значение часто интерпретируют как правдоподобие данных в предположении справедливости нулевой гипотезы, считая, что малые значения якобы означают, что H_0 маловероятна, а H_1 , следовательно, вероятна. При этом рассуждение строится так:

Если H_0 истинна, то такая критериальная статистика, скорее всего, не имела бы места. Но она имеет место. Следовательно, H_0 , скорее всего, ложна.

¹ См., например, <https://fivethirtyeight.com/features/not-even-scientists-can-easily-explain-p-values/>.

Однако это рассуждение неверно. Чтобы понять, почему, рассмотрим следующий пример (взят из работы [Coh94]):

Если человек американец, то, скорее всего, он не является членом Конгресса. Этот человек член Конгресса. Следовательно, он, скорее всего, не американец.

Очевидно, что это рассуждение ошибочно. Для сравнения приведем логически верное рассуждение:

Если человек марсианин, то он не является членом Конгресса. Этот человек член Конгресса. Следовательно, он не марсианин.

Разница между этими двумя случаями в том, что в примере с марсианином используется **дедукция**, т. е. рассуждение от логического определения к его следствиям. Точнее, в этом примере используется логическое правило **modus tollens**, когда мы начинаем с определения вида $P \Rightarrow Q$ и, наблюдая $\neg Q$, заключаем, что $\neg P$. Наоборот, в примере с американцем используется **индукция**, т. е. рассуждение в обратном направлении – от наблюдаемого факта к вероятным (но необязательно истинным) причинам – с применением статистических закономерностей, а не логически точных определений.

Для выполнения индукции необходимо использовать вероятностный вывод (детали подробно объясняются в работе [Jay03]). В частности, чтобы вычислить вероятность нулевой гипотезы, нужно применить формулу Байеса:

$$p(H_0|\mathcal{D}) = \frac{p(\mathcal{D}|H_0)p(H_0)}{p(\mathcal{D}|H_0)p(H_0) + p(\mathcal{D}|H_1)p(H_1)}. \quad (5.105)$$

Если априорное распределение равномерное, т. е. $p(H_0) = p(H_1) = 0.5$, то ее можно переписать в терминах **отношения правдоподобия** $LR = p(\mathcal{D}|H_0)/p(\mathcal{D}|H_1)$ следующим образом:

$$p(H_0|\mathcal{D}) = \frac{LR}{LR + 1}. \quad (5.106)$$

В примере с американским Конгрессом \mathcal{D} является наблюдением, согласно которому человек является членом Конгресса. Нулевая гипотеза H_0 утверждает, что человек американец, а альтернативная гипотеза H_1 – что он не американец. Мы предполагаем, что вероятность $p(\mathcal{D}|H_0)$ мала, потому что большинство американцев не являются членами Конгресса. Однако и вероятность $p(\mathcal{D}|H_1)$ тоже мала – на самом деле в этом примере она равна 0, потому что только американцы могут быть членами Конгресса. Поэтому $LR = \infty$, так что $p(H_0|\mathcal{D}) = 1.0$, как и подсказывает интуиция. Заметим, однако, что при проверке значимости нулевой гипотезы $p(\mathcal{D}|H_1)$ игнорируется, равно как и априорная вероятность $p(H_0)$, поэтому результаты получаются неверными – и не только в этой, а и во многих других задачах.

В общем случае различие между р-значениями и $p(H_0|\mathcal{D})$ может быть очень велико. Так, в работе [SBB01] показано, что, даже если р-значение равно всего

0.05, апостериорная вероятность H_0 может достигать 30 % и более, даже если априорное распределение равномерное.

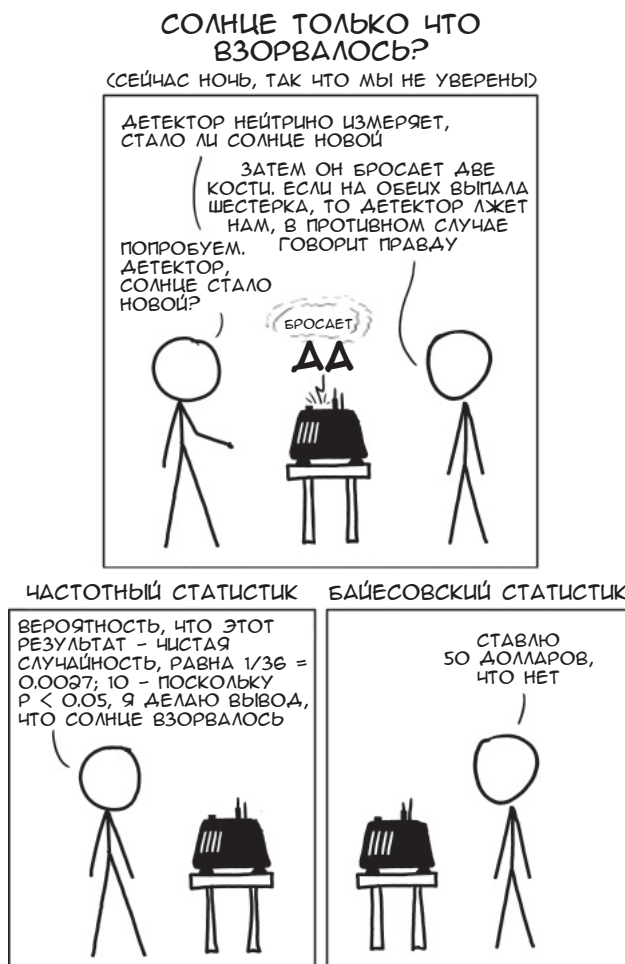


Рис. 5.11 ❖ Иллюстрация различия между частотным и байесовским подходом. (Слова $p < 0.05$ объясняются в разделе 5.5.4. Слова о «ставке» – отсылка к теореме о голландской книге, которая по существу доказывает, что байесовский подход к азартным играм (и другим задачам теории принятия решений) является оптимальным, как объясняется, например, в работе [Háj08].) Взято со страницы <https://xkcd.com/1132/>. Печатается с разрешения Рэндалла Монро (автора xkcd)

Рассмотрим конкретный пример из [SAM04, стр. 74]. Предположим, что произведено 200 клинических испытаний некоторого лекарства. И мы проверяем статистический критерий, чтобы понять, имеет лекарство значимый эффект или нет. В этом критерии частота ошибок первого рода $\alpha = 0.05$,

а частота ошибок второго рода $\beta = 0.2$. Получившиеся данные показаны в табл. 5.7.

Таблица 5.7. Некоторые статистики гипотетического клинического испытания [SAM04, стр. 74]

	Неэффективно	Эффективно	
«Не значимо»	171	4	175
«Значимо»	9	16	25
	180	20	200

Мы можем следующим образом вычислить вероятность неэффективности лекарства при условии, что результаты предположительно «значимы» (significant):

$$p(H_0 | \text{'significant'}) = \frac{p(\text{'significant'} | H_0)p(H_0)}{p(\text{'significant'} | H_0)p(H_0) + p(\text{'significant'} | H_1)p(H_1)} \quad (5.107)$$

$$= \frac{p(\text{ошибка первого рода})p(H_0)}{p(\text{ошибка первого рода})p(H_0) + (1 - p(\text{ошибка второго рода}))p(H_1)} \quad (5.108)$$

$$= \frac{\alpha p(H_0)}{\alpha p(H_0) + (1 - \beta)p(H_1)}. \quad (5.109)$$

Если у нас имеются априорные знания, основанные на прошлом опыте, что большинство (скажем, 90 %) лекарств неэффективно, то находим $p(H_0 | \text{'significant'}) = 0.36$, что гораздо больше, чем вероятность 5 %, которую люди обычно ассоциируют с p -значением $\alpha = 0.05$.

Таким образом, мы не должны доверять заявлениям о статистической значимости, если они идут вразрез с нашим прошлым опытом.

5.5.5. Почему же не все исповедуют байесовский подход?

В разделах 4.7.5 и 5.5.4 мы видели, что вывод, основанный на частотных принципах, может демонстрировать интуитивно неочевидное поведение, которое иногда вступает в противоречие со здравым смыслом. На это обращено внимание в многочисленных статьях (см., например, [Mat98; MS11; Kru13; Gel16; Ное+14; Lyu+20; Cha+19b]).

Фундаментальная причина заключается в том, что частотный вывод нарушает **принцип правдоподобия** [BW88], который гласит, что вывод должен быть основан на правдоподобии наблюдаемых данных, а не на гипотетических будущих данных, которые мы не наблюдали. Байесовский подход, очевидно, удовлетворяет принципу правдоподобия и, следовательно, не подвержен таким патологиям.

Учитывая фундаментальные изъяны частотной статистики и тот факт, что байесовские методы такими изъянами не страдают, возникает естественный вопрос: «Почему же не все исповедуют байесовский подход»? Статистик (приверженец частотного подхода) Брэдли Эфрон написал статью, которую так и назвал [Efr86]. Она коротенькая, и ее стоит прочитать всем интересующимся этим вопросом. Ниже процитировано ее начало:

Вопрос, вынесенный в заголовок, имеет смысл по меньшей мере по двум причинам. Прежде всего, все когда-то были «байесовцами». Лаплас всей душой поддерживал байесовскую постановку проблемы вывода, и большинство ученых XIX века были с ним солидарны. Включая Гаусса, работы которого по статистике обычно изложены в частотных терминах.

Второй, более важный, момент – убедительность байесовской аргументации. Современные статистики, идя по стопам Сэвиджа и де Финетти, выдвинули мощные теоретические аргументы в пользу байесовского вывода. Побочным продуктом этой работы стал обескураживающий перечень противоречий, свойственных частотной точке зрения.

И тем не менее не все исповедуют байесовский подход. Наше время (1986 год) – первое столетие, в котором статистика широко используется в научных работах, и так сложилось, что статистика XX века преимущественно не байесовская. Однако Линдли (1975) предсказывает, что в XXI веке это положение изменится.

Время покажет, был ли Линдли прав. Однако похоже, все идет к тому. Например, некоторые журналы запретили использовать p -значения [TM15; AGM19], а журнал *The American Statistician* (издаваемый Американской статистической ассоциацией) посвятил целый выпуск предупреждениям относительно использования p -значений и NHST [WSL19].

Традиционно барьером на пути использования байесовских методов была сложность вычислений, но в наши дни это уже не такая серьезная проблема, поскольку компьютеры стали быстрее, а алгоритмы качественнее (мы обсудим это во втором томе книги, [Mur22]). Другое, более фундаментальное, затруднение заключается в том, что байесовский подход правилен лишь настолько, насколько правильны лежащие в основе модели предположения. Однако это замечание в равной мере относится и к частотным методам, так как выборочное распределение оценщика должно выводиться с использованием предположений о механизме порождения данных. (На самом деле в работе [BT73] показано, что выборочные распределения для оценки MLE в типичных моделях идентичны апостериорным распределениям при неинформативном априорном.) По счастью, мы можем проверить правильность предположений эмпирически с помощью перекрестной проверки (раздел 4.5.5), калибровки и байесовской проверки моделей. Мы обсудим эти вопросы во втором томе.

Подводя итоги, стоит процитировать Дональда Рубина, который написал статью [Rub84] под названием «Bayesianly Justifiable and Relevant Frequency Calculations for the Applied Statistician»¹. В ней он пишет:

Прикладной статистик должен иметь байесовский склад ума в принципе и поверять его реальным миром на практике. Он должен стремиться использовать такие спецификации, которые приводят к приближенно калиброванным процедурам при разумных отклонениях [от исходных предположений]. Он должен избегать моделей, существенно противоречащих наблюдаемым данным, – частотные вычисления для гипотетических репликаций могут смоделировать адекватность модели и оказать содействие в выборе более подходящих моделей.

5.6. УПРАЖНЕНИЯ

Упражнение 5.1 [бинарные классификаторы с возможностью отклонения]. (Источник: [DHS01, Q2.13].)

Во многих задачах классификации есть возможность либо отнести \mathbf{x} к классу j , либо, если нет уверенности, **отклонить** пример. Если стоимость отклонения меньше, чем стоимость неправильной классификации объекта, то такое действие может оказаться оптимальным. Пусть α_i означает, что выбрано действие i из диапазона $1 \dots C + 1$, где C – число классов, а $C + 1$ – отклонение. Пусть $Y = j$ – истинное (но неизвестное) состояние природы. Определим следующую функцию потерь:

$$\lambda(\alpha_i | Y = j) = \begin{cases} 0, & \text{если } i = j \text{ и } i, j \in \{1, \dots, C\} \\ \lambda_r, & \text{если } i = C + 1 \\ \lambda_s & \text{в противном случае} \end{cases}. \quad (5.110)$$

Иными словами, мы несем нулевую потерю 0 в случае правильной классификации, потерю λ_r , если пример отклонен, и потерю λ_s в случае неправильной классификации.

- Покажите, что минимальный риск достигается, когда мы принимаем решение $Y = j$, если $p(Y = j | \mathbf{x}) \geq p(Y = k | \mathbf{x})$ для всех k (т. е. j – самый вероятный класс) и если $p(Y = j | \mathbf{x}) \geq 1 - \lambda_r / \lambda_s$, а в противном случае отклоняем пример.
- Качественно опишите, что происходит, когда λ_r / λ_s увеличивается от 0 до 1 (т. е. относительная стоимость отклонения растет).

Упражнение 5.2 [проблема продавца газет*].

Рассмотрим следующую классическую задачу теории принятия решений и экономики. Требуется решить, какое количество Q некоторого продукта (например, газет) купить, чтобы максимизировать прибыль. Оптимальное

¹ Байесовское обоснование и релевантные частотные вычисления для прикладного статистика.

количество зависит от предполагаемого спроса D на продукт, от ваших затрат на его приобретение C и назначенной вами продажной цены P . Предположим, что D – неизвестная величина, но известны ее функция плотности вероятности $f(D)$ и функция распределения $F(D)$. Мы можем вычислить ожидаемую прибыль, рассмотрев два случая: если $D > Q$, то мы продаем все Q экземпляров и получаем прибыль $\pi = (P - C)Q$; если же $D < Q$, то мы продаем только D экземпляров, получаем прибыль $(P - C)D$, но несем убытки в размере $C(Q - D)$ на нераспроданных экземплярах. Таким образом, ожидаемая прибыль при закупке Q единиц продукта равна:

$$E\pi(Q) = \int_Q^\infty (P - C)Qf(D)dD + \int_0^Q (P - C)Df(D)dD - \int_0^Q C(Q - D)f(D)dD. \quad (5.111)$$

Упростите это выражение, затем возьмите производные по Q и покажите, что оптимальная величина Q^* (которая доставляет максимум ожидаемой прибыли) удовлетворяет условию:

$$F(Q^*) = \frac{P - C}{P}. \quad (5.112)$$

Упражнение 5.3 [коэффициенты Байеса и ROC-кривые*].

Пусть $B = p(D|H_1)/p(D|H_0)$ – коэффициент Байеса в пользу модели 1. Предположим, что мы построили графики двух ROC-кривых, одна из которых вычислена путем задания порога B , а другая – путем задания порога $p(H_1|D)$. Они будут одинаковыми или разными? Объясните свой ответ.

Упражнение 5.4 [апостериорная медиана – оптимальная оценка при ℓ_1 -потере].

Докажите, что апостериорная медиана является оптимальной оценкой в случае ℓ_1 -потери.

Глава 6

Теория информации

В этой главе мы познакомимся с основами теории информации. Дополнительные сведения можно найти в других книгах, например [Mac03; СТ06], а также во втором томе этой книги, [Mur22].

6.1. Энтропия

Энтропию распределения вероятностей можно интерпретировать как меру неопределенности или непредсказуемости случайной величины, выбранной из этого распределения.

Мы можем воспользоваться энтропией, чтобы определить **собственную информацию** источника данных. Например, предположим, что наблюдается последовательность символов $X_n \sim p$, порожденная распределением p . Если энтропия p высока, то предсказать значение каждого наблюдения X_n будет трудно. Поэтому говорят, что набор данных $\mathcal{D} = (X_1, \dots, X_n)$ имеет высокую собственную информацию. Напротив, если p – вырожденное распределение с нулевой энтропией (минимально возможное значение), то все X_n будут одинаковы, поэтому \mathcal{D} содержит мало информации. (Все сказанное можно формализовать в терминах сжатия данных, но это будет обсуждаться во втором томе.)

6.1.1. Энтропия дискретных случайных величин

Энтропия дискретной случайной величины X , имеющей K состояний с распределением p , определяется формулой:

$$\mathbb{H}(X) \triangleq \sum_{k=1}^K p(X = k) \log_2 p(X = k) = -\mathbb{E}_X[\log p(X)]. \quad (6.1)$$

(Отметим, что $\mathbb{H}(X)$ обозначает энтропию случайной величины с распределением p , точно так же как $\mathbb{V}[X]$ обозначает дисперсию распределения, ассоциированного с X ; можно было бы вместо этого писать $\mathbb{H}(p)$.) Обычно логарифм берется по основанию 2, и тогда единицы энтропии называются **битами** (сокращение от binary digit – двоичная цифра). Например, если

$X \in \{1, \dots, 5\}$, а гистограмма распределения имеет вид $p = [0.25, 0.25, 0.2, 0.15, 0.15]$, то $H = 2.29$ бит. Если в качестве основания берется e , то единица измерения называется **нат**.

Дискретным распределением с **максимальной энтропией** является равномерное распределение. Следовательно, для K -арной случайной величины максимум энтропии достигается, когда $p(x = k) = 1/K$; в этом случае, $\mathbb{H}(X) = \log_2 K$. Чтобы убедиться в этом, заметим, что

$$\mathbb{H}(X) = -\sum_{k=1}^K \frac{1}{K} \log(1/K) = -\log(1/K) = \log(K). \quad (6.2)$$

Наоборот, распределением с минимальной (нулевой) энтропией является любая дельта-функция, для которой вся масса вероятности сосредоточена в одном состоянии. У такого распределения нет никакой неопределенности.

В частном случае бинарных случайных величин $X \in \{0, 1\}$ мы можем написать $p(X = 1) = \theta$ и $p(X = 0) = 1 - \theta$. Поэтому энтропия принимает вид:

$$\mathbb{H}(X) = -[p(X = 1)\log_2 p(X = 1) + p(X = 0)\log_2 p(X = 0)] \quad (6.3)$$

$$= -[\theta \log_2 \theta + (1 - \theta) \log_2 (1 - \theta)]. \quad (6.4)$$

Это называется **функцией двоичной энтропии** и также обозначается $\mathbb{H}(\theta)$. График этой функции показан на рис. 6.1. Как видим, максимальное значение, равное 1 биту, достигается, когда распределение равномерное, $\theta = 0.5$. Чтобы определить состояние симметричной монеты, достаточно одного вопроса, предполагающего ответ «да» или «нет».

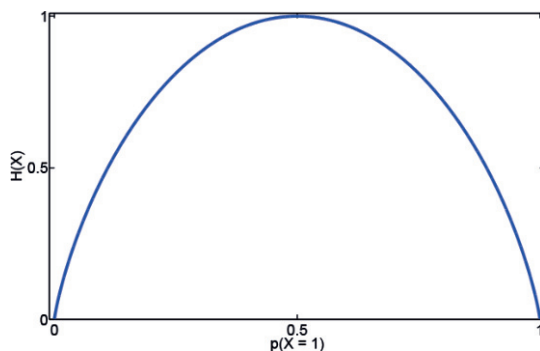


Рис. 6.1 ❖ Энтропия бернуллиевой случайной величины как функция от θ . Максимальная энтропия равна $\log_2 2 = 1$. Построено программой по адресу figures.problm.ai/book1/6.1

В качестве интересного применения энтропии рассмотрим задачу о представлении **мотива последовательности ДНК** – распределения коротких строк ДНК. Это распределение можно оценить, выровняв множество последовательностей ДНК (например, разных видов), а затем оценив эмпирическое распределение возможных нуклеотидов, представленных алфавитом из четырех букв, – $X \sim \{A, C, G, T\}$, в каждой позиции t i -й последовательности:

$$\mathbf{N}_t = \left(\sum_{i=1}^N \mathbb{I}(X_{it} = A), \sum_{i=1}^N \mathbb{I}(X_{it} = C), \sum_{i=1}^N \mathbb{I}(X_{it} = G), \sum_{i=1}^N \mathbb{I}(X_{it} = T) \right); \quad (6.5)$$

$$\hat{\theta}_t = \mathbf{N}_t / N. \quad (6.6)$$

Здесь \mathbf{N}_t – вектор длины 4, содержащий число вхождений каждой из четырех букв в t -й позиции последовательностей из данного множества. Это распределение $\hat{\theta}_t$ называется **мотивом**. Мы также можем вычислить самую вероятную букву в каждой позиции; это называется **консенсусной последовательностью**.

Один из способов визуального обобщения этих данных – **логотип последовательности**, показанный на рис. 6.2b. Мы рисуем буквы А, С, G и Т, так что самая вероятная буква оказывается сверху; высота t -го столбца равна $2 - H_t$, где H_t – энтропия $\hat{\theta}_t$ (отметим, что 2 – максимально возможная энтропия распределения 4 букв). Таким образом, высокие столбцы соответствуют почти детерминированным распределениям, т. е. позициям, оказавшимся консервативными в ходе эволюции (например, потому что они части кодирующей области гена). В этом примере мы видим, что столбец 13 состоит только из букв G и, значит, имеет высоту 2.

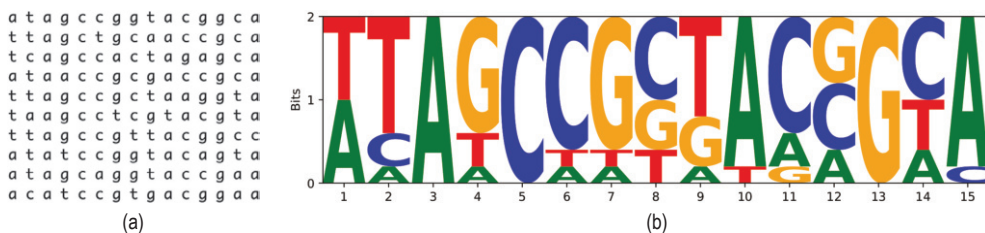


Рис. 6.2 ❖ (а) Несколько выровненных последовательностей ДНК. Каждая строка – последовательность, а столбец – позиция внутри последовательности. (б) Соответствующая **матрица весов позиций**, представленная в виде логотипа последовательности. Каждый столбец представляет распределение вероятностей алфавита {A, C, G, T} для соответствующей позиции последовательности. Размер буквы пропорционален вероятности. Высота столбца t равна $2 - H_t$, где $0 \leq H_t \leq 2$ – энтропия (в битах) распределения p_t . Таким образом, детерминированные распределения (с нулевой энтропией, соответствующие консервативным позициям) имеют высоту 2, а равномерные распределения (с энтропией 2) – высоту 0. Построено программой по адресу figures.probml.ai/book1/6.2

Для оценки энтропии случайной величины, имеющей много возможных состояний, нужно оценить ее распределение, для чего может понадобиться много данных. Например, пусть X представляет идентификатор слова в английском документе. Поскольку имеется длинный хвост редких слов, да еще постоянно возникают новые слова, трудно дать надежную оценку $p(X)$, а значит, и $H(X)$. Одно из возможных решений этой проблемы приведено в работе [VV13].

6.1.2. Перекрестная энтропия

Перекрестной энтропией распределений p и q называется величина:

$$\mathbb{H}(p, q) \triangleq - \sum_{k=1}^K p_k \log q_k. \quad (6.7)$$

Можно показать, что перекрестная энтропия равна математическому ожиданию числа битов, необходимых для сжатия примеров данных, выбранных из распределения p , с помощью кода, основанного на распределении q . Она достигает минимума, когда $q = p$, и в этом случае ожидаемое число битов оптимального кода равно $\mathbb{H}(p, p) = \mathbb{H}(p)$ – это утверждение называется **теоремой Шеннона о кодировании источника** (см., например, [CT06]).

6.1.3. Совместная энтропия

Совместной энтропией двух случайных величин X и Y называется

$$\mathbb{H}(X, Y) = - \sum_{x, y} p(x, y) \log_2 p(x, y). \quad (6.8)$$

Например, рассмотрим выбор целого числа от 1 до 8, $n \in \{1, \dots, 8\}$. Пусть $X(n) = 1$, если n четно, а $Y(n) = 1$, если n – простое число:

n	1	2	3	4	5	6	7	8
X	0	1	0	1	0	1	0	1
Y	0	1	1	0	1	0	1	0

Совместное распределение имеет вид

$p(X, Y)$	$Y=0$	$Y=1$
$X=0$	1/8	3/8
$X=1$	3/8	1/8

поэтому совместная энтропия равна:

$$\mathbb{H}(X, Y) = - \left[\frac{1}{8} \log_2 \frac{1}{8} + \frac{3}{8} \log_2 \frac{3}{8} + \frac{3}{8} \log_2 \frac{3}{8} + \frac{1}{8} \log_2 \frac{1}{8} \right] = 1.81 \text{ бита}. \quad (6.9)$$

Очевидно, что маргинальные вероятности распределены равномерно: $p(X=1) = p(X=0) = p(Y=0) = p(Y=1) = 0.5$, поэтому $\mathbb{H}(X) = \mathbb{H}(Y) = 1$. Отсюда $\mathbb{H}(X, Y) = 1.81 \text{ бит} < \mathbb{H}(X) + \mathbb{H}(Y) = 2 \text{ бита}$. На самом деле эта верхняя граница совместной энтропии имеет место и в общем случае. Если X и Y независимы, то $\mathbb{H}(X, Y) = \mathbb{H}(X) + \mathbb{H}(Y)$, так что оценка неулучшаемая. Это интуитивно понятно: когда части каким-то образом коррелированы, число «степеней свободы» в системе уменьшается, поэтому снижается и суммарная энтропия.

А какова нижняя граница $\mathbb{H}(X, Y)$? Если Y – детерминированная функция от X , то $\mathbb{H}(X, Y) = \mathbb{H}(X)$. Поэтому

$$\mathbb{H}(X, Y) \geq \max\{\mathbb{H}(X), \mathbb{H}(Y)\} \geq 0. \quad (6.10)$$

Интуитивно это означает, что комбинирование величин не уменьшает энтропию: нельзя уменьшить неопределенность, просто добавляя в задачу новые неизвестные, необходимо наблюдать какие-то данные; этот вопрос мы обсудим в разделе 6.1.4.

Определение совместной энтропии очевидным образом обобщается на n случайных величин.

6.1.4. Условная энтропия

Условной энтропией Y при условии X называется неопределенность Y , остающаяся после наблюдения X , усредненная по всем возможным значениям X :

$$\mathbb{H}(X, Y) \triangleq \mathbb{E}_{p(X)}[\mathbb{H}(p(Y|X))] \quad (6.11)$$

$$= \sum_x p(x) \mathbb{H}(p(Y|X=x)) = - \sum_x p(x) \sum_y p(y|x) \log p(y|x) \quad (6.12)$$

$$= - \sum_{x,y} p(x,y) \log p(y|x) = - \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)} \quad (6.13)$$

$$= - \sum_{x,y} p(x,y) \log p(x,y) - \sum_x p(x) \log \frac{1}{p(x)} \quad (6.14)$$

$$= \mathbb{H}(X, Y) - \mathbb{H}(X). \quad (6.15)$$

Если Y – детерминированная функция от X , то знание X полностью определяет Y , так что $\mathbb{H}(Y|X) = 0$. Если X и Y независимы, то знание X ничего не говорит о Y , поэтому $\mathbb{H}(Y|X) = \mathbb{H}(Y)$. Так как $\mathbb{H}(X, Y) \leq \mathbb{H}(Y) + \mathbb{H}(X)$, имеем

$$\mathbb{H}(Y|X) \leq \mathbb{H}(Y), \quad (6.16)$$

причем равенство достигается тогда и только тогда, когда X и Y независимы. Это значит, что в среднем обусловливание данными никогда не увеличивает неопределенность. Оговорка «в среднем» необходима, потому что после любого *конкретного* наблюдения (значения X) «неожиданность» может возрасти (т. е. $\mathbb{H}(Y|x) > \mathbb{H}(Y)$). Однако с точки зрения математического ожидания ознакомление с данными – вещь хорошая (см. также раздел 6.3.8).

Формулу (6.15) можно переписать следующим образом:

$$\mathbb{H}(X_1, X_2) = \mathbb{H}(X_1) + \mathbb{H}(X_2|X_1). \quad (6.17)$$

Она обобщается в **цепное правило для энтропии**:

$$\mathbb{H}(X_1, X_2, \dots, X_n) = \sum_{i=1}^n \mathbb{H}(X_i|X_1, \dots, X_{i-1}). \quad (6.18)$$

6.1.5. Перплексия

Перплексией дискретного распределения вероятностей p называется величина

$$\text{perplexity}(p) \triangleq 2^{\mathbb{H}(p)}. \quad (6.19)$$

Часто ее интерпретируют как меру предсказуемости. Например, пусть p – равномерное распределение K состояний. В этом случае перплексия равна K . Очевидно, что нижняя граница перплексии равна $2^0 = 1$, и достигается она, когда распределение позволяет точно предсказывать исходы.

Теперь предположим, что имеется эмпирическое распределение, основанное на данных \mathcal{D} :

$$p_{\mathcal{D}}(x|\mathcal{D}) = \frac{1}{N} \sum_{n=1}^N \delta_{x_n}(x). \quad (6.20)$$

Мы можем измерить, насколько хорошо p предсказывает \mathcal{D} , вычислив

$$\text{perplexity}(p_{\mathcal{D}}, p) \triangleq 2^{\mathbb{H}(p_{\mathcal{D}}, p)}. \quad (6.21)$$

Перплексия часто используется для оценки качества статистических языковых моделей, т. е. моделей, порождающих последовательности лексем. Предположим, что в роли данных выступает один большой документ x длины N , а p – простая униграммная модель. В этом случае член перекрестной энтропии имеет вид:

$$H = -\frac{1}{N} \sum_{n=1}^N \log p(x_n), \quad (6.22)$$

а значит, перплексия равна:

$$\text{perplexity}(p_{\mathcal{D}}, p) = 2^H = \sqrt[N]{\prod_{n=1}^N \frac{1}{p(x_n)}}. \quad (6.23)$$

Иногда это выражение называют **экспоненциальной перекрестной энтропией**. Как видим, это среднее геометрическое обратных вероятностей.

В случае языковых моделей при предсказании следующего слова мы обычно обуславливаем вероятность предыдущими словами. Например, в биграммной модели используется марковская модель второго порядка вида $p(x_i|x_{i-1})$. Определим **коэффициент ветвления** языковой модели как число допустимых слов, которые могут следовать за данным словом. Таким образом, перплексия интерпретируется как средневзвешенный коэффициент ветвления. Например, предположим, что модель предсказывает все слова с равной вероятностью независимо от контекста, т. е. $p(x_i|x_{i-1}) = 1/K$. Тогда перплексия равна $((1/K)^N)^{-1/N} = K$. Если одни символы вероятнее других

и модель правильно отражает этот факт, то ее перплексия будет меньше K . Однако, как будет показано в разделе 6.2, имеет место неравенство $\mathbb{H}(p^*) \leq \mathbb{H}(p^*, p)$, так что мы никогда не сможем сделать перплексию меньше энтропии порождающего случайного процесса p^* .

Продолжение обсуждения перплексии и ее использования в языковых моделях см. в работе [JM08, стр. 96].

6.1.6. Дифференциальная энтропия непрерывных случайных величин*

Если X – непрерывная случайная величина и $p(x)$ – ее функция плотности распределения, то **дифференциальная энтропия** определяется следующим образом:

$$h(X) \triangleq - \int_{\mathcal{X}} dx p(x) \log p(x) \quad (6.24)$$

в предположении, что этот интеграл существует. Например, предположим, что $X \sim U(0, a)$. Тогда

$$h(X) = - \int_0^a dx \frac{1}{a} \log \frac{1}{a} = \log a. \quad (6.25)$$

Заметим, что, в отличие от дискретного случая, дифференциальная энтропия может быть отрицательной, поскольку функция плотности вероятности может быть больше 1. Например, если $X \sim U(0, 1/8)$, то $h(X) = \log_2(1/8) = -3$.

Чтобы лучше понять, что такое дифференциальная энтропия, вспомните, что вещественные величины представимы лишь с конечной точностью. Можно показать [СТ91, стр. 228], что энтропия n -битовой дискретизации непрерывной случайной величины X приближенно равна $h(X) + n$. Например, предположим, что $X \sim U(0, 1/8)$. Тогда в двоичном представлении X первые 3 бита справа от двоичной точки должны быть равны 0 (потому что число $\leq 1/8$). Поэтому, чтобы описать X с точностью n битов, необходимо только $n - 3$ бита, что согласуется с приведенным выше результатом $h(X) = -3$.

6.1.6.1. Пример: энтропия гауссова распределения

Энтропия d -мерного гауссова распределения равна:

$$h(\mathcal{N}(\mu, \Sigma)) = \frac{1}{2} \ln |2\pi e \Sigma| = \frac{1}{2} \ln [(2\pi e)^d |\Sigma|] = \frac{d}{2} + \frac{d}{2} \ln(2\pi) + \frac{1}{2} \ln |\Sigma|. \quad (6.26)$$

В одномерном случае это выражение принимает вид:

$$h(\mathcal{N}(\mu, \sigma^2)) = \frac{1}{2} \ln [2\pi e \sigma^2]. \quad (6.27)$$

6.1.6.2. Связь с дисперсией

Энтропия гауссова распределения монотонно возрастает с ростом дисперсии. Однако так бывает не всегда. Например, рассмотрим смесь двух одномерных гауссовых распределений с центрами в точках -1 и $+1$. При раздвигании средних, например в точки -10 и $+10$, дисперсия увеличивается (потому что среднее расстояние до полного среднего становится больше). Однако энтропия остается более-менее постоянной, так как мы по-прежнему не уверены, куда попадет пример, даже если знаем, что он будет вблизи -10 или $+10$. (Точную энтропию смеси гауссовых распределений вычислить трудно, но способ вычисления верхней и нижней границы представлен в работе [Hub+08].)

6.1.6.3. Дискретизация

В общем случае вычислить дифференциальную энтропию непрерывной случайной величины трудно. Для получения простой аппроксимации можно произвести **дискретизацию**, или **квантование** величины. Есть разные методы (см., например, [DKS95; KK06], где приведен обзор), но самый простой – раскидать распределение по интервалам на основе эмпирических квантилей. Важнейший вопрос – сколько брать интервалов [LM04]. В работе [Sco79] предложена следующая эвристика:

$$B = N^{1/3} \frac{\max(\mathcal{D}) - \min(\mathcal{D})}{3.5\sigma(\mathcal{D})}, \quad (6.28)$$

где $\sigma(\mathcal{D})$ – эмпирическое стандартное отклонение данных, а $N = |\mathcal{D}|$ – количество точек в эмпирическом распределении. Однако такая техника дискретизации плохо масштабируется, когда X – многомерный случайный вектор, из-за проклятия размерности.

6.2. ОТНОСИТЕЛЬНАЯ ЭНТРОПИЯ (РАСХОЖДЕНИЕ KL)*

Если даны два распределения p и q , то часто бывает полезно определить **метрику**, измеряющую их «близость» или «похожесть». На самом деле мы рассмотрим более общий случай и определим меру расхождения $D(p, q)$, которая количественно выражает расстояние между q и p , не требуя, чтобы D была метрикой в строгом смысле слова. Точнее, будем говорить, что D является расхождением, если $D(p, q) \geq 0$, причем равенство достигается тогда и только тогда, когда $p = q$; напомним, что определение метрики требует, чтобы функция D была симметричной и удовлетворяла **неравенству треугольника** $D(p, r) \leq D(p, q) + D(q, r)$. Существует много разных мер расхождения. В этом разделе мы будем рассматривать **расхождение Кульбака–Лейблера (КЛ)**, называемое также **информационным выигрышем** или **относительной энтропией** распределений p и q .

6.2.1. Определение

Для дискретных распределений расхождение КЛ определяется следующим образом:

$$\mathbb{KL}(p||q) \triangleq \sum_{k=1}^K p_k \log \frac{p_k}{q_k}. \quad (6.29)$$

Это определение естественно обобщается на непрерывные распределения:

$$\mathbb{KL}(p||q) \triangleq \int dx p(x) \log \frac{p(x)}{q(x)}. \quad (6.30)$$

6.2.2. Интерпретация

Мы можем переписать расхождение КЛ в таком виде:

$$\mathbb{KL}(p||q) = \underbrace{\sum_{k=1}^K p_k \log p_k}_{-\mathbb{H}(p)} - \underbrace{\sum_{k=1}^K p_k \log q_k}_{\mathbb{H}(p,q)}. \quad (6.31)$$

В первом члене мы узнаем отрицательную энтропию, а во втором перекрестную энтропию. Можно показать, что перекрестная энтропия $\mathbb{H}(p, q)$ является нижней границей числа битов, необходимых для сжатия данных, порожденных распределением p , с помощью кода, основанного на распределении q ; поэтому расхождение КЛ можно интерпретировать как «дополнительное число битов», т. е. штраф, который придется уплатить, если в качестве основы для схемы кодирования выбрано неправильное распределение q вместо истинного распределения p .

Существуют и другие интерпретации расхождения КЛ (см. второй том этой книги, [Mur22]).

6.2.3. Пример: расхождение КЛ между двумя гауссовыми распределениями

Можно показать, что расхождение КЛ между двумя многомерными гауссовыми распределениями равно:

$$\begin{aligned} & \mathbb{KL}(\mathcal{N}(\mathbf{x}|\mu_1, \Sigma_1) || \mathcal{N}(\mathbf{x}|\mu_2, \Sigma_2)) \\ &= \frac{1}{2} \left[\text{tr}(\Sigma_2^{-1} \Sigma_1) + (\mu_2 - \mu_1)^T \Sigma_2^{-1} (\mu_2 - \mu_1) - D + \log \left(\frac{\det(\Sigma_2)}{\det(\Sigma_1)} \right) \right]. \end{aligned} \quad (6.32)$$

В скалярном случае это выражение принимает вид:

$$\mathbb{KL}(\mathcal{N}(x|\mu_1, \sigma_1) || \mathcal{N}(x|\mu_2, \sigma_2)) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}. \quad (6.33)$$

6.2.4. Неотрицательность расхождения КЛ

В этом разделе мы докажем, что расхождение КЛ всегда неотрицательно.

Для этого нам понадобится **неравенство Йенсена**, которое утверждает, что для любой выпуклой функции f

$$f\left(\sum_{i=1}^n \lambda_i \mathbf{x}_i\right) \leq \sum_{i=1}^n \lambda_i f(\mathbf{x}_i), \quad (6.34)$$

где $\lambda_i \geq 0$ и $\sum_{i=1}^n \lambda_i = 1$. Иначе говоря, функция f от среднего меньше, чем среднее функций f . Это очевидно для $n = 2$, потому что выпуклая кривая расположена выше прямой, соединяющей ее крайние точки (см. раздел 8.1.3). Доказательство для общего случая можно провести по индукции.

Например, если $f(x) = \log(x)$ (это вогнутая функция), то имеем:

$$\log(\mathbb{E}_x g(x)) \geq \mathbb{E}_x \log(g(x)). \quad (6.35)$$

Ниже мы воспользуемся этим результатом.

Теорема 6.2.1 (информационное неравенство). $\mathbb{KL}(p||q) \geq 0$, причем равенство достигается тогда и только тогда, когда $p = q$.

Доказательство. Мы проведем доказательство, следуя работе [СТ06, стр. 28]. Пусть $A = \{x : p(x) > 0\}$ – опора $p(x)$. Пользуясь вогнутостью логарифма и неравенством Йенсена (раздел 6.2.4), получаем:

$$-\mathbb{KL}(p||q) = -\sum_{x \in A} p(x) \log \frac{p(x)}{q(x)} = \sum_{x \in A} p(x) \log \frac{q(x)}{p(x)} \quad (6.36)$$

$$\leq \log \sum_{x \in A} p(x) \frac{p(x)}{q(x)} = \log \sum_{x \in A} q(x) \quad (6.37)$$

$$\leq \log \sum_{x \in \mathcal{X}} q(x) = \log 1 = 0. \quad (6.38)$$

Поскольку $\log(x)$ – строго вогнутая функция ($-\log(x)$ выпуклая), равенство в (6.37) достигается тогда и только тогда, когда $p(x) = cq(x)$ для некоторого c , что описывает часть всего пространства \mathcal{X} , содержащуюся в A . Равенство в (6.38) достигается тогда и только тогда, когда $\sum_{x \in A} q(x) = \sum_{x \in \mathcal{X}} q(x) = 1$, откуда следует, что $c = 1$. Поэтому $\mathbb{KL}(p||q) = 0$ тогда и только тогда, когда $p(x) = q(x)$ для всех x . ■

У этой теоремы много важных следствий, с которыми мы не раз встретимся в этой книге. Например, можно показать, что равномерное распределение доставляет максимум энтропии.

Следствие 6.2.1 (равномерное распределение доставляет максимум энтропии). $\mathbb{H}(X) \leq \log |\mathcal{X}|$, где $|\mathcal{X}|$ – число состояний \mathcal{X} , причем равенство достигается тогда и только тогда, когда распределение $p(x)$ равномерное.

Доказательство. Обозначим $u(x) = 1/|X|$. Тогда

$$0 \leq \mathbb{KL}(p||u) = \sum_x p(x) \log \frac{p(x)}{u(x)} = \log|X| - \mathbb{H}(X). \quad (6.39)$$

■

6.2.5. Расхождение КЛ и оценка максимального правдоподобия

Пусть требуется найти распределение q , максимально близкое к p в терминах расхождения КЛ:

$$q^* = \operatorname{argmin}_q \mathbb{KL}(p||q) = \operatorname{argmin}_q \int p(x) \log p(x) dx - \int p(x) \log q(x) dx. \quad (6.40)$$

Теперь предположим, что p – эмпирическое распределение, вся масса которого равномерно сосредоточена в наблюдаемых обучающих данных:

$$p_D(x) = \frac{1}{N} \sum_{n=1}^N \delta(x - x_n). \quad (6.41)$$

Применив фильтрующее свойство дельта-функции, получаем:

$$\mathbb{KL}(p_D||q) = - \int p_D(x) \log q(x) dx + C \quad (6.42)$$

$$= - \int \left[\frac{1}{N} \sum_n \delta(x - x_n) \right] \log q(x) dx + C \quad (6.43)$$

$$= - \frac{1}{N} \sum_n \log q(x_n) + C, \quad (6.44)$$

где $C = \int p(x) \log p(x) dx$ – постоянная, не зависящая от q . Это так называемая целевая функция **перекрестной энтропии**, которая равна среднему отрицательному логарифмическому правдоподобию q на обучающем наборе. Таким образом, мы видим, что минимизация расхождения КЛ с эмпирическим распределением эквивалентна максимизации правдоподобия.

Эта точка зрения делает явным дефект обучения на основе правдоподобия, а именно то, что оно наделяет обучающий набор слишком большим весом. В большинстве приложений мы на самом деле не верим, что эмпирическое распределение является хорошим представлением истинного распределения, поскольку оно всего лишь представляет «пики» в конечном множестве точек и оставляет плотность нулевой в других местах. Даже если набор данных велик (скажем, миллион изображений), генеральная совокупность, из которой выбираются данные, обычно еще больше (например, мощность множества «всех естественных изображений» куда больше миллиона). Можно было бы сгладить эмпирическое распределение, воспользовавшись ядерной оценкой плотности (раздел 16.3), но тогда потребовалось бы похожее ядро на

пространстве изображений. Альтернативный алгоритмический подход – воспользоваться **приращением данных**, применяя к наблюдаемым примерам возмущение, которое, как мы полагаем, имитирует «естественную вариацию». Применение оценки MLE к такому пополненному набору данных часто дает прекрасные результаты, особенно при обучении моделей с большим числом параметров (см. раздел 19.1).

6.2.6. Прямое и обратное расхождение КЛ

Пусть требуется аппроксимировать распределение p более простым распределением q . Для этого можно минимизировать либо $\mathbb{KL}(q||p)$, либо $\mathbb{KL}(p||q)$. При этом получается разное поведение.

Сначала рассмотрим **прямое расхождение КЛ**, называемое также **включающим**, которое определяется следующим образом:

$$\mathbb{KL}(p||q) = \int p(x) \log \frac{p(x)}{q(x)} dx. \quad (6.45)$$

Его минимизация относительно q называется **М-проекцией** или **проекцией момента**.

Получить представление об оптимальном q можно, рассмотрев входные данные x , для которых $p(x) > 0$, но $q(x) = 0$. В этом случае член $\log p(x)/q(x)$ бесконечен. Следовательно, минимизация расхождения КЛ вынуждает q включить все области пространства, где вероятность p ненулевая. По-другому можно сказать, что q **избегает нулей** или **покрывает моды**, и обычно оно дает завышенную оценку носителя p . На рис. 6.3а показано покрытие моды, когда p – бимодальное, а q – унимодальное распределение.

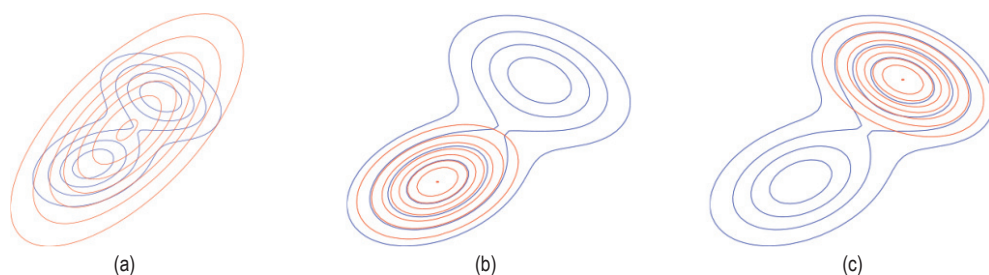


Рис. 6.3 ❖ Прямое и обратное расхождение КЛ для бимодального распределения. Синие кривые – линии уровня истинного распределения p , красные – линии уровня унимодальной аппроксимации q . (а) Минимизация прямого расхождения КЛ, $\mathbb{KL}(p||q)$, относительно q заставляет q «покрывать» p . (b-с) Минимизация обратного расхождения КЛ, $\mathbb{KL}(q||p)$, относительно q заставляет q «фиксироваться» на одной из двух мод p . На основе рис. 10.3 из работы [Bis06].

Построено программой по адресу figures.problml.ai/book1/6.3

Теперь рассмотрим **обратное расхождение КЛ**, называемое также **исключающим**:

$$\mathbb{KL}(q||p) = \int q(x) \log \frac{q(x)}{p(x)} dx. \quad (6.46)$$

Его минимизация относительно q называется **I-проекцией** или **информационной проекцией**.

Чтобы лучше понять природу оптимального q , рассмотрим входные данные, для которых $p(x) = 0$, но $q(x) > 0$. В этом случае член $\log q(x)/p(x)$ бесконечен. Следовательно, минимизация расхождения КЛ вынуждает q исключить все области пространства, где вероятность p нулевая. Сделать это можно, например, сосредоточив всю массу вероятности q в очень немногих частях пространства; это называется **форсированием нулей** или **поиском мод**. В таком случае q обычно занижает опору p . Поиск мод в случае, когда p бимодальное, а q унимодальное, показан на рис. 6.3b–с.

6.3. Взаимная информация*

Расхождение КЛ дает способ измерения схожести двух распределений. А как измерить, насколько зависимы две случайных величины? Можно свести вопрос об измерении зависимости случайных величин к вопросу о схожести их распределений. Это приводит к понятию **взаимной информации** (mutual information – MI) двух случайных величин.

6.3.1. Определение

Взаимной информации случайных величин X и Y называется следующая функция:

$$\mathbb{I}(X; Y) \triangleq \mathbb{KL}(p(x, y)||p(x)p(y)) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}. \quad (6.47)$$

(Мы пишем $\mathbb{I}(X; Y)$, а не $\mathbb{I}(X, Y)$ на случай, если X и/или Y представляют наборы переменных; например, можно написать $\mathbb{I}(X; Y, Z)$, чтобы представить взаимную информацию X и (Y, Z) .) Для непрерывных случайных величин нужно просто заменить суммы интегралами.

Легко видеть, что MI всегда неотрицательна, даже для непрерывных случайных величин, потому что

$$\mathbb{I}(X; Y) = \mathbb{KL}(p(x, y)||p(x)p(y)) \geq 0. \quad (6.48)$$

6.3.2. Интерпретация

Зная, что взаимная информация – это расхождение КЛ между совместным и разложенным на множители маргинальным распределением, мы можем сказать, что MI измеряет информационный выигрыш от замены модели,

которая рассматривает обе величины как независимые, $p(x)p(y)$, моделью, в которой используется их истинное совместное распределение $p(x, y)$.

Чтобы лучше понять смысл MI, перепишем его в терминах совместной и условной энтропий:

$$\mathbb{I}(X; Y) = \mathbb{H}(X) - \mathbb{H}(X|Y) = \mathbb{H}(Y) - \mathbb{H}(Y|X). \quad (6.49)$$

Таким образом, мы можем интерпретировать взаимную информацию X и Y как уменьшение неопределенности X после наблюдения Y или, в силу симметрии, уменьшение неопределенности Y после наблюдения X . Кстати говоря, это дает другое доказательство того, что обусловливание в среднем уменьшает энтропию. Именно, имеем $0 \leq \mathbb{I}(X; Y) = \mathbb{H}(X) - \mathbb{H}(X|Y)$, откуда $\mathbb{H}(X|Y) \leq \mathbb{H}(X)$.

Возможна и другая интерпретация. Можно показать, что

$$\mathbb{I}(X; Y) = \mathbb{H}(X, Y) - \mathbb{H}(X|Y) - \mathbb{H}(Y|X). \quad (6.50)$$

И наконец, можно показать, что

$$\mathbb{I}(X; Y) = \mathbb{H}(X) + \mathbb{H}(Y) - \mathbb{H}(X, Y). \quad (6.51)$$

На рис. 6.4 эти формулы выражены в терминах **информационных диаграмм**. (Формально это мера – положительная или отрицательная – сопоставляющая теоретико-множественным выражениям их теоретико-информационные аналоги [Yeu91].)

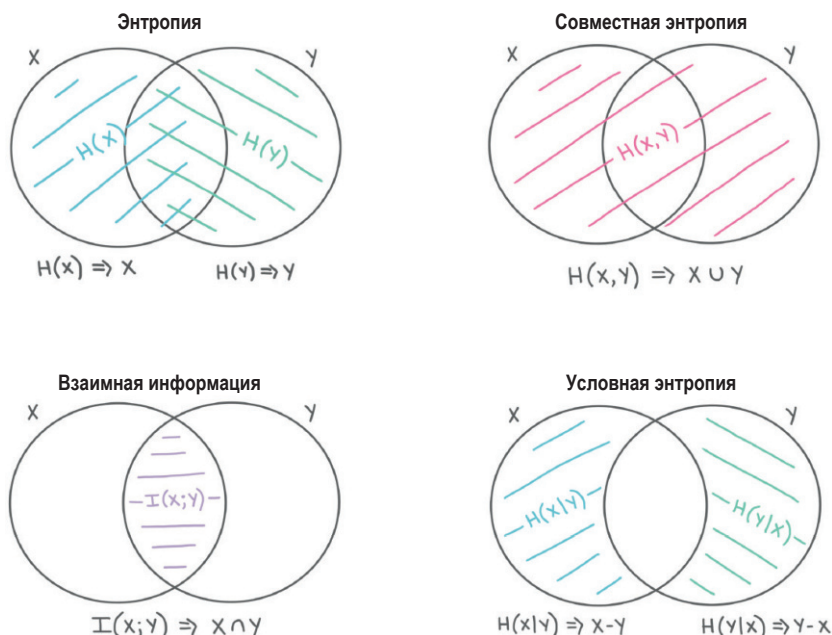


Рис. 6.4 ❖ Маргинальная энтропия, условная энтропия и взаимная информация, представленные с помощью информационных диаграмм. Печатается с разрешения Кэти Эверетт

6.3.3. Пример

Вернемся к примеру простых и четных чисел из раздела 6.1.3. Напомним, что $\mathbb{H}(X) = \mathbb{H}(Y) = 1$. Условное распределение $p(Y|X)$ получается нормировкой каждой строки:

	$Y = 0$	$Y = 1$
$X = 0$	1/4	3/4
$X = 1$	3/4	1/4

Отсюда условная энтропия равна:

$$\mathbb{H}(Y|X) = -\left[\frac{1}{8}\log_2\frac{1}{4} + \frac{3}{8}\log_2\frac{3}{4} + \frac{3}{8}\log_2\frac{3}{4} + \frac{1}{8}\log_2\frac{1}{4}\right] = 0.81 \text{ бита}, \quad (6.52)$$

а взаимная информация равна:

$$\mathbb{I}(X; Y) = \mathbb{H}(Y) - \mathbb{H}(Y|X) = (1 - 0.81) \text{ бита} = 0.19 \text{ бита}. \quad (6.53)$$

Легко проверить, что

$$\mathbb{H}(X; Y) = \mathbb{H}(X|Y) + \mathbb{I}(X; Y) + \mathbb{H}(Y|X) \quad (6.54)$$

$$= (0.81 + 0.19 + 0.81) \text{ бита} = 1.81 \text{ бита}. \quad (6.55)$$

6.3.4. Условная взаимная информация

Определение **условной взаимной информации** очевидно:

$$\mathbb{I}(X; Y|Z) \triangleq \mathbb{E}_{p(Z)}[\mathbb{I}(X; Y)|Z] \quad (6.56)$$

$$= \mathbb{E}_{p(x,y,z)}\left[\log\frac{p(x,y|z)}{p(x|z)p(y|z)}\right] \quad (6.57)$$

$$= \mathbb{H}(X|Z) + \mathbb{H}(Y|Z) - \mathbb{H}(X; Y|Z) \quad (6.58)$$

$$= \mathbb{H}(X|Z) - \mathbb{H}(X|Y, Z) = \mathbb{H}(Y|Z) - \mathbb{H}(Y|X, Z) \quad (6.59)$$

$$= \mathbb{H}(X, Z) + \mathbb{H}(Y, Z) - \mathbb{H}(Z) - \mathbb{H}(X, Y, Z) \quad (6.60)$$

$$= \mathbb{I}(Y; X, Z) - \mathbb{I}(Y; Z). \quad (6.61)$$

Последнее равенство означает, что условная взаимная информация – это избыточная (остаточная) информация, которую X сообщает о Y , за вычетом того, что мы уже знали о Y , имея только Z .

Формулу (6.61) можно переписать следующим образом:

$$\mathbb{I}(Z, Y; X) = \mathbb{I}(Z; X) + \mathbb{I}(Y; X|Z). \quad (6.62)$$

Обобщая на N случайных величин, получаем **цепное правило для взаимной информации**:

$$\mathbb{I}(Z_1, \dots, Z_N; X) = \sum_{n=1}^N \mathbb{I}(Z_n; X | Z_1, \dots, Z_{n-1}). \quad (6.63)$$

6.3.5. Взаимная информация как «обобщенный коэффициент корреляции»

Пусть пара (x, y) имеет совместное гауссово распределение:

$$\begin{pmatrix} x \\ y \end{pmatrix} \sim N \left(\mathbf{0}, \begin{pmatrix} \sigma^2 & \rho\sigma^2 \\ \rho\sigma^2 & \sigma^2 \end{pmatrix} \right). \quad (6.64)$$

Сейчас мы покажем, как вычислить взаимную информацию между X и Y . Из формулы (6.26) находим, что энтропия равна:

$$h(X, Y) = \frac{1}{2} \log[(2\pi e)^2 \det \Sigma] = \frac{1}{2} \log[(2\pi e)^2 \sigma^4 (1 - \rho^2)]. \quad (6.65)$$

Так как X и Y – по отдельности нормально распределенные случайные величины с дисперсией σ^2 , имеем:

$$h(X) = h(Y) = \frac{1}{2} \log[2\pi e \sigma^2]. \quad (6.66)$$

Отсюда

$$I(X, Y) = h(X) + h(Y) - h(X, Y) \quad (6.67)$$

$$= \log[2\pi e \sigma^2] - \frac{1}{2} \log[(2\pi e)^2 \sigma^4 (1 - \rho^2)] \quad (6.68)$$

$$= \frac{1}{2} \log[(2\pi e \sigma^2)^2] - \frac{1}{2} \log[(2\pi e \sigma^2)^2 (1 - \rho^2)] \quad (6.69)$$

$$= \frac{1}{2} \log \frac{1}{1 - \rho^2} = -\frac{1}{2} \log[1 - \rho^2]. \quad (6.70)$$

Теперь обсудим некоторые интересные частные случаи.

1. $\rho = 1$. В этом случае $X = Y$ и $\mathbb{I}(X, Y) = 1$, что имеет смысл. Наблюдение Y сообщает нам бесконечный объем информации о X (поскольку мы точно знаем ее истинное значение).
2. $\rho = 0$. В этом случае X и Y независимы и $\mathbb{I}(X, Y) = 0$, что тоже имеет смысл. Наблюдение Y ничего не сообщает нам о X .
3. $\rho = -1$. В этом случае $X = -Y$ и $\mathbb{I}(X, Y) = 1$, что опять-таки имеет смысл. Наблюдение Y позволяет предсказать X с бесконечной точностью.

Теперь рассмотрим случай, когда X и Y – скалярные величины, но их совместное распределение не гауссово. В общем случае трудно вычислить взаимную информацию между непрерывными случайными величинами, потому что необходимо оценить совместную плотность $p(X, Y)$. Для скалярных

величин можно в качестве простой аппроксимации применить **дискретизацию**, или **квантование**, разделив области значений каждой величины на интервалы и вычислив, сколько значений попадает в каждый интервал гистограммы [Sco79]. Затем можно легко вычислить взаимную информацию, пользуясь эмпирической функцией вероятности.

К сожалению, результат может существенно зависеть от количества интервалов и положения их границ. Избежать этого можно, воспользовавшись расстояниями до K ближайших соседей, чтобы оценить плотности непараметрическим адаптивным способом. Это основа **оценки KSG** взаимной информации, предложенной в работе [KSG04]. Она реализована в функции `sklearn.feature_selection.mutual_info_regression`. Подробнее об этой оценке см. в работах [GOV18; HN19].

6.3.6. Нормированная взаимная информация

В некоторых приложениях удобно иметь нормированную меру зависимости, изменяющуюся от 0 до 1. Сейчас мы обсудим один из способов построения такой меры.

Прежде всего, отметим, что

$$\mathbb{I}(X; Y) = \mathbb{H}(X) - \mathbb{H}(X|Y) \leq \mathbb{H}(X), \quad (6.71)$$

$$= \mathbb{H}(Y) - \mathbb{H}(Y|X) \leq \mathbb{H}(Y), \quad (6.72)$$

так что

$$0 \leq \mathbb{I}(X; Y) \leq \min(\mathbb{H}(X), \mathbb{H}(Y)). \quad (6.73)$$

Поэтому определить **нормированную взаимную информацию** можно следующим образом:

$$NMI(X, Y) = \frac{\mathbb{I}(X; Y)}{\min(\mathbb{H}(X), \mathbb{H}(Y))} \leq 1. \quad (6.74)$$

Эта величина принимает значения от 0 до 1. Если $NMI(X, Y) = 0$, то имеем $\mathbb{I}(X; Y) = 0$, т. е. X и Y независимы. Если $NMI(X; Y) = 1$ и $\mathbb{H}(X) < \mathbb{H}(Y)$, то

$$\mathbb{I}(X; Y) = \mathbb{H}(X) - \mathbb{H}(X|Y) = \mathbb{H}(X) \Rightarrow \mathbb{H}(X|Y) = 0, \quad (6.75)$$

т. е. X – детерминированная функция от Y . Например, предположим, что X – дискретная случайная величина с функцией вероятности $[0.5, 0.25, 0.25]$. Имеем $MI(X, X) = 1.5$ (мы используем логарифм по основанию 2) и $\mathbb{H}(X) = 1.5$, т. е. нормированная MI равна 1, как и следовало ожидать.

Для непрерывных случайных величин нормировать взаимную информацию труднее, потому что необходимо оценить дифференциальную энтропию, а это сильно зависит от уровня квантования. Дальнейшее обсуждение см. в разделе 6.3.7.

6.3.7. Максимальный коэффициент информации

Как было сказано в разделе 6.3.6, полезно иметь нормированную оценку взаимной информации, но для вещественных данных вычислить ее трудно. Возможный подход, известный под названием **максимальный коэффициент информации** (maximal information coefficient – **MIC**) [Res+11], заключается в том, чтобы определить следующую величину:

$$\text{MIC}(X, Y) = \max_G \frac{\mathbb{I}((X; Y)|_G)}{\log \|G\|}, \quad (6.76)$$

где G – множество двумерных сеток, $(X, Y)|_G$ представляет дискретизацию величин на сетке G , $\|G\|$ равно $\min(G_x, G_y)$, где G_x – число ячеек сетки в направлении x , а G_y – число ячеек сетки в направлении y . (Максимальное разрешение сетки зависит от размера выборки n ; авторы предлагают ограничить сетки, так чтобы $G_x G_y \leq B(n)$, где $B(n) = n^\alpha$, а $\alpha = 0.6$.) В знаменателе находится энтропия равномерного совместного распределения; деление на нее гарантирует, что $0 \leq \text{MIC} \leq 1$.

Интуитивно эта статистика означает следующее: если существует связь между X и Y , то должно существовать какое-то разбиение двумерного пространства входов, улавливающее эту связь. Поскольку мы не знаем, какую именно сетку использовать, MIC производит поиск на сетках с разным разрешением (например, 2×2 , 2×3 и т. д.), а также с разным расположением границ. Зная сетку, легко дискретизировать данные и вычислить MI. Мы определим **характеристическую матрицу** $M(k, l)$ как максимальную взаимную информацию, достижимую на любой сетке размера (k, l) , нормированную на $\log(\min(k, l))$. Тогда MIC – максимальный элемент этой матрицы, $\max_{kl \leq B(n)} M(k, l)$. Этот процесс наглядно изображен на рис. 6.5.

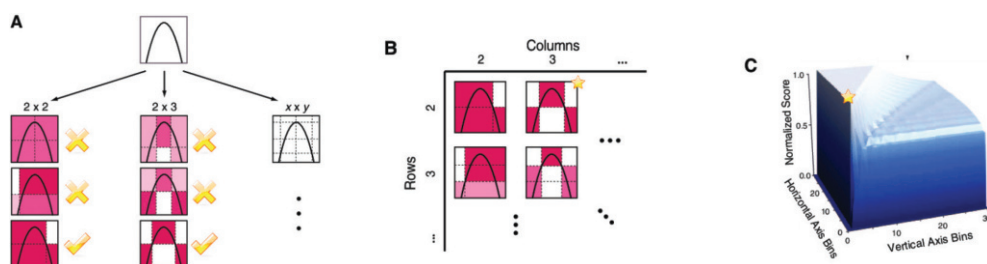


Рис. 6.5 ❖ Как вычисляется максимальный коэффициент информации (MIC). (а) Производится поиск на сетках с разным разрешением и местоположением ячеек и для каждой сетки вычисляется MI. (б) Для каждого разрешения сетки (k, l) определяем матрицу $M(k, l)$ как максимальную MI для любой сетки этого размера, нормированную на $\log(\min(k, l))$. (с) Визуализируем матрицу M . Максимальный элемент (обозначенный звездочкой) по определению и есть MIC. На основе рис. 1 из работы [Res+11]. Печатается с разрешения Дэвида Решефа

В работе [Res+11] показано, что у этой величины имеется свойство **равноценности** (equitability), т. е. она дает похожие оценки одинаково зашумленным связям независимо от типа связи (линейная, нелинейная, нефункциональная и т. п.).

В работе [Res+16] предложена улучшенная оценка, названная **MICe**, которую можно вычислить более эффективно и которая требует оптимизации только по одномерным сеткам, что можно сделать за время $O(n)$, применив динамическое программирование.

Авторы предлагают также величину **TICe** (total information content – полная собственная информация), которая обладает большей способностью обнаруживать связи по выборкам небольшого размера, но меньшей равноценностью. Эффективную реализацию обеих метрик см. в работе [Alb+18].

Нулевой MIC можно интерпретировать как отсутствие связи между величинами, а MIC, равный 1, как незашумленную связь любого вида. Это показано на рис. 6.6. В отличие от коэффициентов корреляции, MIC способен находить не только линейные связи. Поэтому MIC называли «корреляцией XXI века» [Spe11].

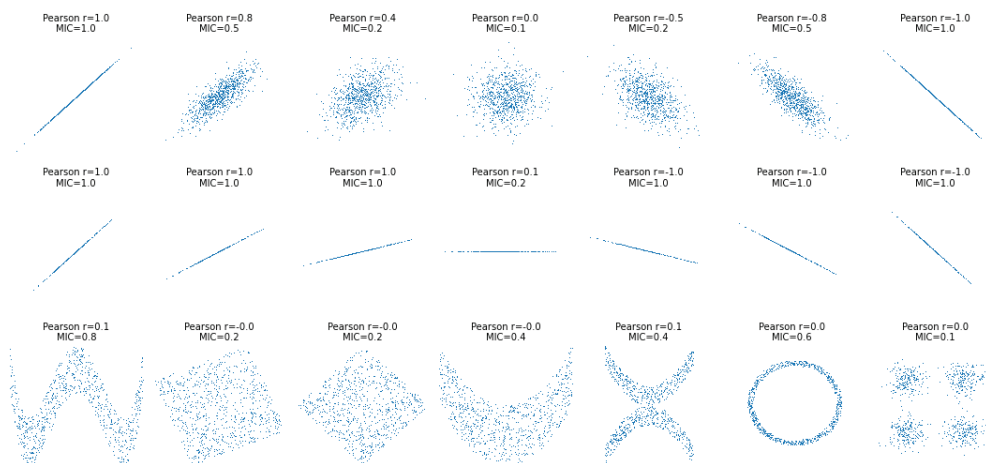


Рис. 6.6 ❖ Диаграммы некоторых двумерных распределений и соответствующие оценки коэффициента корреляции R^2 и максимального коэффициента информации (MIC). Сравните с рис. 3.1. Построено программой по адресу figures.problm.ai/book1/6.6

На рис. 6.7 приведен более интересный пример, взятый из работы [Res+11]. Данные состоят из 357 величин, измеряющих различные показатели в области социального устройства, экономики, здравоохранения и политики, собранные Всемирной организацией здравоохранения (ВОЗ). В левой части рисунка мы видим зависимость коэффициента корреляции (CC) от MIC для всех 63 546 пар величин. В правой части приведены диаграммы рассеяния для конкретных пар величин, которые мы сейчас и обсудим.

- В точке **С** (рядом с 0,0 на графике) коэффициенты CC и MIC малы. Из соответствующей диаграммы рассеяния видно, что между этими ве-

личинами (процент смертей из-за травм и плотность зубных врачей в популяции) нет никакой связи.

- В точках **D** и **H** коэффициенты СС (по абсолютной величине) и МИС велики, потому что они представляют почти линейные связи.
- В точках **E**, **F** и **G** СС низкий, а МИС высокий. Это говорит о том, что связи между величинами нелинейные (а иногда, как в точках **E** и **F**, и нефункциональные, т. е. типа один ко многим).

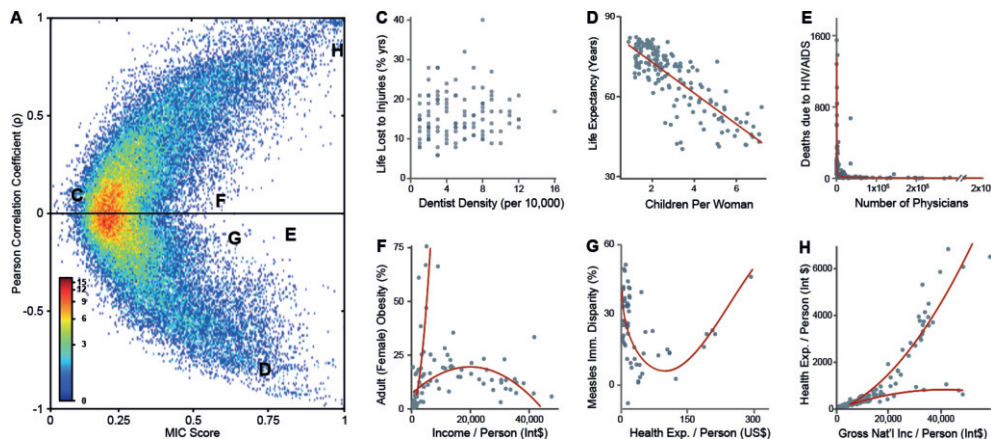


Рис. 6.7 ❖ Слева: зависимость коэффициента корреляции от максимального коэффициента информации (МИС) для всех попарных связей в данных ВОЗ. Справа: диаграммы рассеяния для некоторых пар величин. Красные линии – кривые непараметрической сглаживающей регрессии, построенные отдельно для каждого тренда. На основе рис. 4 из работы [Res+11]. Печатается с разрешения Дэвида Решеффа

6.3.8. Неравенство обработки данных

Пусть имеется неизвестная величина X и мы наблюдаем зашумленную функцию от нее, которую назовем Y . Если в результате обработки зашумленных наблюдений создается новая величина Z , то интуитивно очевидно, что мы не можем таким образом увеличить количество информации о неизвестной величине X . Этот факт называется **неравенством обработки данных**. Сформулируем его формально, а затем и докажем.

Теорема 6.3.1. Пусть $X \rightarrow Y \rightarrow Z$ – марковская цепь, т. е. $X \perp Z | Y$. Тогда $\mathbb{I}(X; Y) \geq \mathbb{I}(X; Z)$.

Доказательство. В силу цепного правила для взаимной информации (формула (6.62)) взаимную информацию можно раскрыть двумя способами:

$$\mathbb{I}(X; Y, Z) = \mathbb{I}(X; Z) + \mathbb{I}(X; Y | Z) \quad (6.77)$$

$$= \mathbb{I}(X; Y) + \mathbb{I}(X; Z | Y). \quad (6.78)$$

Поскольку $X \perp Z|Y$, имеем $\mathbb{I}(X; Z|Y) = 0$, так что

$$\mathbb{I}(X; Z) + \mathbb{I}(X; Y|Z) = \mathbb{I}(X; Y). \quad (6.79)$$

Поскольку $\mathbb{I}(X; Y|Z) \geq 0$, имеем $\mathbb{I}(X; Y) \geq \mathbb{I}(X; Z)$. Аналогично можно доказать, что $\mathbb{I}(Y; Z) \geq \mathbb{I}(X; Z)$. ■

6.3.9. Достаточные статистики

Из неравенства обработки данных вытекает важное следствие. Пусть имеется цепь $\theta \rightarrow \mathcal{D} \rightarrow s(\mathcal{D})$. Тогда

$$\mathbb{I}(\theta; s(\mathcal{D})) \leq \mathbb{I}(\theta; \mathcal{D}). \quad (6.80)$$

Если имеет место равенство, то говорят, что $s(\mathcal{D})$ – **достаточная статистика** данных \mathcal{D} для вывода θ . В этом случае можно эквивалентно написать $\theta \rightarrow s(\mathcal{D}) \rightarrow \mathcal{D}$, потому что мы можем реконструировать данные, зная $s(\mathcal{D})$, так же точно, как зная θ .

Примером достаточной статистики являются сами данные, $s(\mathcal{D}) = \mathcal{D}$, но это не очень полезно, т. к. в этом случае нет никакого обобщения данных. Поэтому мы определим **минимальную достаточную статистику** $s(\mathcal{D})$ как такую, которая не содержит избыточной информации о θ ; таким образом, $s(\mathcal{D})$ максимально сжимает данные \mathcal{D} без потери информации, существенной для предсказания θ . Формально говорят, что s – минимальная достаточная статистика для \mathcal{D} , если для любой достаточной статистики $s'(\mathcal{D})$ существует функция f такая, что $s(\mathcal{D}) = f(s'(\mathcal{D}))$. Эту ситуацию можно кратко выразить следующим образом:

$$\theta \rightarrow s(\mathcal{D}) \rightarrow s'(\mathcal{D}) \rightarrow \mathcal{D}. \quad (6.81)$$

Здесь $s'(\mathcal{D})$ берет $s(\mathcal{D})$ и добавляет к ней избыточную информацию, создавая тем самым отображение один ко многим. Например, минимальной достаточной статистикой для множества N испытаний Бернулли является просто N и $N_1 = \sum_n \mathbb{I}(X_n = 1)$, т. е. число успехов. Иными словами, не нужно запоминать всю последовательность орлов и решек, достаточно знать лишь общее число тех и других. Аналогично для вывода среднего гауссова распределения с известной дисперсией достаточно эмпирического среднего и количества примеров.

6.3.10. Неравенство Фано*

Популярный метод **отбора признаков** – взять входные признаки X_d , имеющие высокую взаимную информацию с выходной величиной Y . Ниже мы объясним, почему этот подход разумен. В частности, мы докажем **неравенство Фано**, которое дает оценку сверху вероятности неправильной классификации (любым методом) в терминах взаимной информации между признаками X и меткой класса Y .

Теорема 6.3.2 (неравенство Фано). Рассмотрим оценку $\hat{Y} = f(X)$ такую, что $Y \rightarrow X \rightarrow \hat{Y}$ – марковская цепь. Обозначим E событие $\hat{Y} \neq Y$, означающее, что произошла ошибка, и пусть $P_e = P(Y \neq \hat{Y})$ – вероятность ошибки. Тогда

$$\mathbb{H}(Y|X) \leq \mathbb{H}(Y|\hat{Y}) \leq \mathbb{H}(E) + P_e \log|\mathcal{Y}|. \quad (6.82)$$

Поскольку $\mathbb{H}(E) \leq 1$, как мы видели на рис. 6.1, этот результат можно ослабить:

$$1 + P_e \log|\mathcal{Y}| \geq \mathbb{H}(Y|X), \quad (6.83)$$

откуда

$$P_e \geq \frac{\mathbb{H}(Y|X) - 1}{\log|\mathcal{Y}|}. \quad (6.84)$$

Таким образом, минимизация $\mathbb{H}(Y|X)$ (эквивалентная максимизации $\mathbb{I}(X; Y)$) дает также минимизацию нижней границы P_e .

Доказательство. (Займствовано из [СТ06, стр. 38].) Применяя цепное правило для энтропии, получаем:

$$\mathbb{H}(E, Y|\hat{Y}) = \mathbb{H}(Y|\hat{Y}) + \underbrace{\mathbb{H}(E, Y|\hat{Y})}_{=0} \quad (6.85)$$

$$= \mathbb{H}(E|\hat{Y}) + \mathbb{H}(Y|E, \hat{Y}). \quad (6.86)$$

Поскольку обусловливание уменьшает энтропию (см. раздел 6.2.4), имеем $\mathbb{H}(E|\hat{Y}) \leq \mathbb{H}(E)$. Последний член можно ограничить следующим образом:

$$\mathbb{H}(Y|E, \hat{Y}) = P(E = 0)\mathbb{H}(Y|\hat{Y}, E = 0) + P(E = 1)\mathbb{H}(Y|\hat{Y}, E = 1) \quad (6.87)$$

$$\leq (1 - P_e)0 + P_e \log|\mathcal{Y}|. \quad (6.88)$$

Отсюда

$$\mathbb{H}(Y|\hat{Y}) \leq \underbrace{\mathbb{H}(E|\hat{Y})}_{\leq \mathbb{H}(E)} + \underbrace{\mathbb{H}(Y|E, \hat{Y})}_{P_e \log|\mathcal{Y}|}. \quad (6.89)$$

Наконец, в силу неравенства обработки данных, имеем $\mathbb{I}(Y; \hat{Y}) \leq \mathbb{I}(Y; X)$, поэтому $\mathbb{H}(Y|X) \leq \mathbb{H}(Y|\hat{Y})$, откуда и вытекает неравенство (6.82). ■

6.4. УПРАЖНЕНИЯ

Упражнение 6.1 [выражений взаимной информации в терминах энтропий*].

Докажите следующие тождества:

$$I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \quad (6.90)$$

и

$$H(X, Y) = H(X|Y) + H(Y|X) + I(X; Y). \quad (6.91)$$

Упражнение 6.2 [связь между $D(p||q)$ и статистикой χ^2].
(Источник: [СТ91, Q12.2].)

Покажите, что если $p(x) \approx q(x)$, то

$$\mathbb{KL}(p||q) \approx \frac{1}{2} \chi^2, \quad (6.92)$$

где

$$\chi^2 = \sum_x \frac{(p(x) - q(x))^2}{q(x)}. \quad (6.93)$$

Указание: напишите

$$p(x) = \Delta(x) + q(x); \quad (6.94)$$

$$\frac{p(x)}{q(x)} = 1 + \frac{\Delta(x)}{q(x)} \quad (6.95)$$

и воспользуйтесь разложением $\log(1 + x)$ в ряд Тейлора:

$$\log(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} \dots \quad (6.96)$$

для $-1 < x \leq 1$.

Упражнение 6.3 [ох уж эти энтропии*].

(Источник: Маккей.) Рассмотрим совместное распределение $p(X, Y)$.

		x			
		1	2	3	4
y	1	1/8	1/16	1/32	1/32
	2	1/16	1/8	1/32	1/32
	3	1/16	1/16	1/16	1/16
	4	1/4	0	0	0

- Чему равна совместная энтропия $H(X, Y)$?
- Чему равны маргинальные энтропии $H(X)$ и $H(Y)$?
- Энтропия X при условии конкретного значения y определяется как

$$H(X|Y = y) = -\sum_x p(x|y) \log p(x|y). \quad (6.97)$$

Вычислите $H(X|y)$ для каждого значения y . Возрастает ли когда-нибудь апостериорная энтропия X при условии наблюдения Y ?

d. Условная энтропия определяется как

$$H(X|Y) = \sum_x p(y) H(X|Y = y). \quad (6.98)$$

Вычислите ее. Возрастает или убывает апостериорная энтропия X при усреднении по всем возможным значениям Y ?

e. Чему равна взаимная информация между X и Y ?

Упражнение 6.4 [прямое и обратное расхождение КЛ].

(Источник: упражнение 33.7 из [Mac03].)

Рассмотрим аппроксимацию совместного распределения $p(x, y)$ произведением $q(x, y) = q(x)q(y)$. Покажите, что для минимизации прямого расхождения КЛ $\mathbb{KL}(p||q)$ следует положить $q(x) = p(x)$ и $q(y) = p(y)$, т. е. оптимальная аппроксимация является произведением маргинальных распределений.

Теперь рассмотрим следующее совместное распределение, где строки представляют y , а столбцы – x .

	1	2	3	4
1	1/8	1/8	0	0
2	1/8	1/8	0	0
3	0	0	1/4	0
4	0	0	0	1/4

Покажите, что обратное расхождение КЛ $\mathbb{KL}(q||p)$ для этого p имеет три разных точки минимума. Найдите эти точки минимума и вычислите $\mathbb{KL}(q||p)$ в каждой из них. Чему будет равно значение $\mathbb{KL}(q||p)$, если положить $q(x, y) = p(x)p(y)$?

Глава 7

Линейная алгебра

Эта глава написана в соавторстве с Зико Колтером.

7.1. ВВЕДЕНИЕ

Линейная алгебра изучает векторы и матрицы. В этой главе собран лишь материал, который понадобится нам далее в книге. Гораздо больше информации можно найти в других источниках, например [Str09; Kle13; Mol04; TB97; Axl15; Tho17; Agg20].

7.1.1. Обозначения

В этом разделе мы введем некоторые обозначения.

7.1.1.1. Векторы

Вектором $\mathbf{x} \in \mathbb{R}^n$ называется список n чисел, обычно записываемый в виде столбца:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}. \quad (7.1)$$

Вектор, состоящий из одних единиц, обозначается $\mathbf{1}$. Вектор, состоящий из одних нулей, обозначается $\mathbf{0}$.

Единичным вектором \mathbf{e}_i называется вектор, все элементы которого равны 0, за исключением элемента i , равного 1:

$$\mathbf{e}_i = (0, \dots, 0, 1, 0, \dots, 0). \quad (7.2)$$

Такой вектор также называется **унитарным**.

7.1.1.2. Матрицы

Матрицей $\mathbf{A} \in \mathbb{R}^{m \times n}$ с m строками и n столбцами называется двумерный массив чисел, организованный следующим образом:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}. \quad (7.3)$$

Если $m = n$, то матрица называется **квадратной**.

Мы используем нотацию A_{ij} или $A_{i,j}$ для обозначения элемента \mathbf{A} , находящегося на пересечении i -й строки и j -го столбца. $\mathbf{A}_{i,:}$ обозначает всю i -ю строку, а $\mathbf{A}_{:,j}$ – весь j -й столбец. По умолчанию рассматриваются векторы-столбцы (т. е. $\mathbf{A}_{i,:}$ трактуется как вектор-столбец с n элементами). Заглавными полужирными буквам обозначаются матрицы, строчными полужирными буквами – векторы, а простыми буквами – скаляры.

Матрицу можно рассматривать как множество столбцов, расположенных вдоль горизонтальной оси:

$$\mathbf{A} = \begin{bmatrix} | & | & & | \\ \mathbf{A}_{:,1} & \mathbf{A}_{:,2} & \cdots & \mathbf{A}_{:,n} \\ | & | & & | \end{bmatrix}. \quad (7.4)$$

Для краткости это записывается в виде:

$$\mathbf{A} = [\mathbf{A}_{:,1}, \mathbf{A}_{:,2}, \dots, \mathbf{A}_{:,n}]. \quad (7.5)$$

Можно также рассматривать матрицу как множество строк, расположенных вдоль вертикальной оси:

$$\mathbf{A} = \begin{bmatrix} - & \mathbf{A}_{1,:}^\top & - \\ - & \mathbf{A}_{2,:}^\top & - \\ & \vdots & \\ - & \mathbf{A}_{m,:}^\top & - \end{bmatrix}. \quad (7.6)$$

Для краткости это записывается в виде:

$$\mathbf{A} = [\mathbf{A}_{1,:}, \mathbf{A}_{2,:}, \dots, \mathbf{A}_{m,:}]. \quad (7.7)$$

(Обратите внимание на использование точки с запятой.)

Транспонированием матрицы называется операция замены строк на столбцы. Если дана матрица $\mathbf{A} \in \mathbb{R}^{m \times n}$, то транспонированная в ней матрица $\mathbf{A}^\top \in \mathbb{R}^{n \times m}$ имеет вид:

$$(\mathbf{A}^\top)_{ij} = A_{ji}. \quad (7.8)$$

Легко проверить следующие свойства операции транспонирования:

$$(A^T)^T = A; \quad (7.9)$$

$$(AB)^T = B^T A^T; \quad (7.10)$$

$$(A + B)^T = A^T + B^T. \quad (7.11)$$

Квадратная матрица, обладающая свойством $A = A^T$, называется **симметричной**. Множество всех симметричных матриц размера n обозначается \mathbb{S}^n .

7.1.1.3. Тензоры

Тензор (в машинном обучении) – это просто обобщение двумерного массива на большее число измерений, как показано на рис. 7.1. Элементы трехмерного тензора обозначаются A_{ijk} . Число измерений называется **порядком**, или **рангом** тензора¹. В математике тензоры можно рассматривать как способ определения мультилинейных отображений – точно так же, как матрицы можно использовать для определения линейных функций, хотя нам такая интерпретация не понадобится.

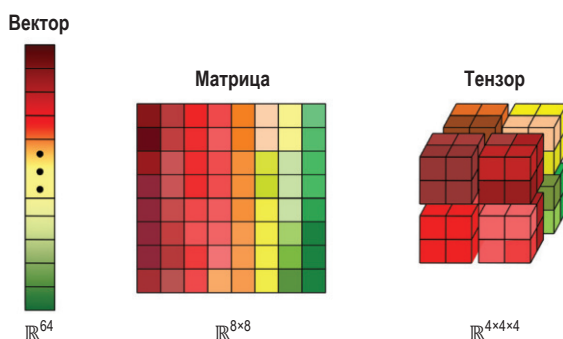


Рис. 7.1 ❖ Одномерный вектор, двумерная матрица, трехмерный тензор. Цветом обозначены отдельные элементы вектора; этот список чисел можно также сохранить в виде двумерной матрицы, как показано на среднем рисунке. (В этом примере матрица организована по столбцам, а не по строкам, как в Python.) Можно также представить вектор в виде трехмерного тензора, показанного на правом рисунке

Матрицу можно преобразовать в вектор, расположив столбцы друг под другом, как показано на рис. 7.1. Это обозначается так:

$$\text{vec}(A) = [A_{:,1}; \dots; A_{:,n}] \in \mathbb{R}^{mn \times 1}. \quad (7.12)$$

Наоборот, вектор можно преобразовать в матрицу, причем двумя способами: по строкам (как в языках Python и C++) или по столбцам (как в языках Julia, Matlab, R и Fortran). Различие показано на рис. 7.2.

¹ Заметим, однако, что ранг двумерной матрицы – это другое понятие, оно обсуждается в разделе 7.1.4.3.

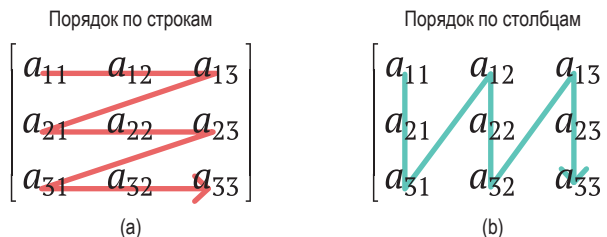


Рис. 7.2 ❖ (a) Порядок по строкам и (b) порядок по столбцам. Взято со страницы https://commons.wikimedia.org/wiki/File:Row_and_column_major_order.svg. Печатается с разрешения автора «Википедии» Cmglee.

7.1.2. Векторные пространства

В этом разделе мы обсудим ряд фундаментальных понятий линейной алгебры.

7.1.2.1. Сложение векторов и умножение вектора на скаляр

Мы можем считать, что вектор $\mathbf{x} \in \mathbb{R}^n$ определяет точку в n -мерном евклидовом пространстве. **Векторным пространством** называется множество таких векторов, на котором определены операции сложения и умножения на скаляр (вещественное число), позволяющие создавать новые точки. Эти операции определяются поэлементно очевидным образом: $\mathbf{x} + \mathbf{y} = (x_1 + y_1, \dots, x_n + y_n)$ и $c\mathbf{x} = (cx_1, \dots, cx_n)$, где $c \in \mathbb{R}$ (см. иллюстрацию на рис. 7.3а).

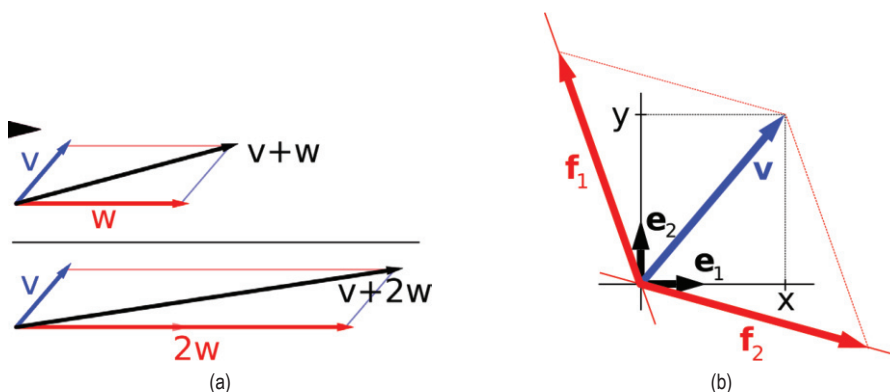


Рис. 7.3 ❖ (a) Сверху: вектор \mathbf{v} (синий) складывается с вектором \mathbf{w} (красным). Снизу: \mathbf{w} умножается на 2, что дает сумму $\mathbf{v} + 2\mathbf{w}$. Взято со страницы https://en.wikipedia.org/wiki/Vector_space. Печатается с разрешения автора «Википедии» IkamusumeFan. (b) Вектор \mathbf{v} в \mathbb{R}^2 (синий) выражен в двух разных базисах: в стандартном базисе \mathbb{R}^2 , $\mathbf{v} = x\mathbf{e}_1 + y\mathbf{e}_2$ (черные), и в другом, неортогональном базисе: $\mathbf{v} = \mathbf{f}_1 + \mathbf{f}_2$ (красные). Взято со страницы https://en.wikipedia.org/wiki/Vector_space. Печатается с разрешения автора «Википедии» Jakob.scholbach

7.1.2.2. Линейная независимость, линейная оболочка и базисы

Множество векторов $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ называется **линейно независимым**, если никакой вектор нельзя представить в виде линейной комбинации остальных. Вектор, который *можно* представить в виде линейной комбинации других векторов, называется **линейно зависимым** от них. Например, если

$$\mathbf{x}_n = \sum_{i=1}^{n-1} \alpha_i \mathbf{x}_i \quad (7.13)$$

для каких-то $\{\alpha_1, \dots, \alpha_{n-1}\}$, то \mathbf{x}_n линейно зависит от $\{\mathbf{x}_1, \dots, \mathbf{x}_{n-1}\}$; в противном случае он линейно независим от $\{\mathbf{x}_1, \dots, \mathbf{x}_{n-1}\}$.

Линейной оболочкой множества векторов $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ называется множество всех векторов, которые можно представить линейными комбинациями $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$. То есть:

$$\text{span}(\{\mathbf{x}_1, \dots, \mathbf{x}_n\}) \triangleq \left\{ \mathbf{v} : \mathbf{v} = \sum_{i=1}^n \alpha_i \mathbf{x}_i, \alpha_i \in \mathbb{R} \right\}. \quad (7.14)$$

Можно показать, что если $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ – множество n линейно независимых векторов, где все $\mathbf{x}_i \in \mathbb{R}^n$, то $\text{span}(\{\mathbf{x}_1, \dots, \mathbf{x}_n\}) = \mathbb{R}^n$. Иными словами, любой вектор $\mathbf{v} \in \mathbb{R}^n$ можно записать в виде линейной комбинации векторов $\mathbf{x}_1, \dots, \mathbf{x}_n$.

Базисом \mathcal{B} называется множество линейно независимых векторов, линейной оболочкой которых является все пространство, т. е. $\text{span}(\mathcal{B}) = \mathbb{R}^n$. Часто можно выбирать один из нескольких базисов, как показано на рис. 7.3b. **Стандартный базис** образован координатными векторами $\mathbf{e}_1 = (1, 0, \dots, 0)$, \dots $\mathbf{e}_n = (0, 0, \dots, 0, 1)$. Это позволяет переходить от трактовки вектора в \mathbb{R}^2 как «стрелки на плоскости», исходящей из начала координат, к трактовке как упорядоченного списка чисел (коэффициентов в некотором базисе).

7.1.2.3. Линейные отображения и матрицы

Линейным отображением или **линейным преобразованием** называется функция $f: \mathcal{V} \rightarrow \mathcal{W}$ такая, что $f(\mathbf{v} + \mathbf{w}) = f(\mathbf{v}) + f(\mathbf{w})$ и $f(a\mathbf{v}) = af(\mathbf{v})$ для любых $\mathbf{v}, \mathbf{w} \in \mathcal{V}$. После выбора базиса \mathcal{V} линейное отображение $f: \mathcal{V} \rightarrow \mathcal{W}$ полностью определяется заданием образов базисных векторов, потому что любой элемент \mathcal{V} однозначно представляется в виде их линейной комбинации.

Пусть $\mathcal{V} = \mathbb{R}^n$ и $\mathcal{W} = \mathbb{R}^m$. Мы можем вычислить $f(\mathbf{v}_i) \in \mathbb{R}^m$ для любого вектора из базиса \mathcal{V} и сохранить результаты в столбцах матрицы \mathbf{A} размера $m \times n$. Затем мы можем вычислить $\mathbf{y} = f(\mathbf{x}) \in \mathbb{R}^m$ для любого вектора $\mathbf{x} \in \mathbb{R}^n$ следующим образом:

$$\mathbf{y} = \left\{ \sum_{j=1}^n a_{1j} \mathbf{x}_j, \dots, \sum_{j=1}^n a_{mj} \mathbf{x}_j \right\}. \quad (7.15)$$

Это соответствует умножению вектора \mathbf{x} на матрицу \mathbf{A} :

$$\mathbf{y} = \mathbf{A}\mathbf{x}. \quad (7.16)$$

Детали см. в разделе 7.2.

Если функция обратима, то можно записать:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}. \quad (7.17)$$

Детали см. в разделе 7.3.

7.1.2.4. Образ и ядро матрицы

Рассмотрим матрицу $\mathbf{A} \in \mathbb{R}^{m \times n}$ как множество m векторов в \mathbb{R}^n . **Образом** (или **пространством столбцов**)¹ этой матрицы называется линейная оболочка столбцов \mathbf{A} . Иначе говоря,

$$\text{range}(\mathbf{A}) \triangleq \{\mathbf{v} \in \mathbb{R}^m : \mathbf{v} = \mathbf{A}\mathbf{x}, \mathbf{x} \in \mathbb{R}^n\}. \quad (7.18)$$

Образ можно рассматривать как множество векторов, которые можно «получить» или которых можно «достичь» с помощью \mathbf{A} ; это подпространство \mathbb{R}^m , размерность которого равна рангу \mathbf{A} (см. раздел 7.1.4.3). **Ядром** (или нулевым пространством) матрицы $\mathbf{A} \in \mathbb{R}^{m \times n}$ называется множество векторов, которые при умножении на \mathbf{A} переходят в нулевой вектор, т. е.

$$\text{nullspace}(\mathbf{A}) \triangleq \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} = \mathbf{0}\}. \quad (7.19)$$

Линейная оболочка строк \mathbf{A} является дополнением к ядру \mathbf{A} .

На рис. 7.4 иллюстрируется образ и ядро матрицы. Как вычислять их, мы обсудим в разделе 7.5.4 ниже.

7.1.2.5. Линейная проекция

Проекцией $\mathbf{y} \in \mathbb{R}^m$ на линейную оболочку $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ (здесь предполагается, что $\mathbf{x}_i \in \mathbb{R}^m$) называется вектор $\mathbf{v} \in \text{span}(\{\mathbf{x}_1, \dots, \mathbf{x}_n\})$ такой, что \mathbf{v} максимально близко к \mathbf{y} в смысле евклидовой нормы $\|\mathbf{v} - \mathbf{y}\|^2$. Мы будем обозначать проекцию $\text{Proj}(\mathbf{y}; \{\mathbf{x}_1, \dots, \mathbf{x}_n\})$ и можем определить ее формально как

$$\text{Proj}(\mathbf{y}; \{\mathbf{x}_1, \dots, \mathbf{x}_n\}) = \underset{\mathbf{v} \in \text{span}(\{\mathbf{x}_1, \dots, \mathbf{x}_n\})}{\text{argmin}} \|\mathbf{y} - \mathbf{v}\|_2. \quad (7.20)$$

Для матрицы (полного ранга) $\mathbf{A} \in \mathbb{R}^{m \times n}$, где $m \geq n$, можно определить проекцию вектора $\mathbf{y} \in \mathbb{R}^m$ на образ \mathbf{A} :

$$\text{Proj}(\mathbf{y}; \mathbf{A}) = \underset{\mathbf{v} \in \mathcal{R}(\mathbf{A})}{\text{argmin}} \|\mathbf{v} - \mathbf{y}\|_2 = \mathbf{A}(\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{y}. \quad (7.21)$$

Это то же самое, что нормальные уравнения из раздела 11.2.2.2.

¹ Автор употребляет термины *range* (область значения) и *nullspace* (нулевое пространство), тогда как в русскоязычной литературе обычно говорят об *образе* и *ядре* матрицы и обозначают их соответственно *Im* и *Ker*. Мы оставили обозначения автора, но употребляем русскоязычную терминологию. – *Прим. перев.*

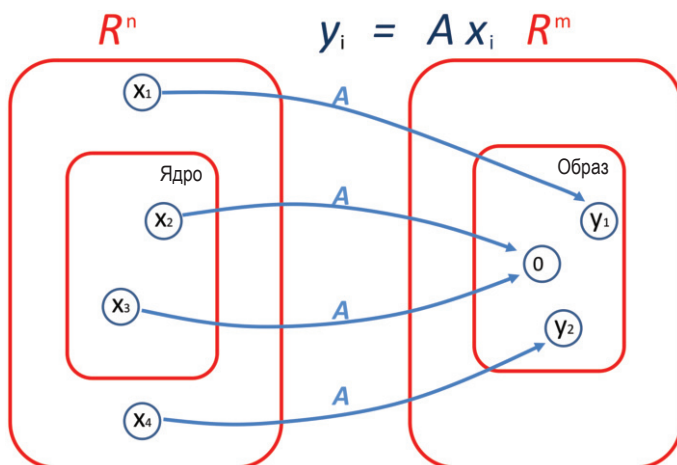


Рис. 7.4 ❖ Визуализация ядра и образа матрицы A размера $m \times n$. Здесь $y_1 = Ax_1$ и $y_2 = Ax_4$, так что y_1 и y_2 принадлежат образу A (достижимы из некоторого x). Кроме того, $Ax_2 = 0$ и $Ax_3 = 0$, так что x_2 и x_3 принадлежат ядру A (отображаются в 0). Как видим, образ часто является подмножеством области определения отображения

7.1.3. Нормы вектора и матрицы

В этом разделе мы обсудим, как можно измерить «размер» вектора и матрицы.

7.1.3.1. Нормы вектора

Нормой вектора $\|x\|$ неформально называется мера его «длины». Более формально норма – это любая функция $f: \mathbb{R}^n \rightarrow \mathbb{R}$, обладающая четырьмя свойствами:

- для любого $x \in \mathbb{R}^n$ $f(x) \geq 0$ (неотрицательность);
- $f(x) = 0$ тогда и только тогда, когда $x = 0$ (определенность);
- для любых $x \in \mathbb{R}^n$, $t \in \mathbb{R}$ $f(tx) = |t|f(x)$ (однородность по абсолютной величине);
- для любых $x, y \in \mathbb{R}^n$ $f(x + y) \leq f(x) + f(y)$ (неравенство треугольника).

Приведем несколько примеров:

p-норма $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$, $p \geq 1$.

2-норма $\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$, называемая также евклидовой нормой. Отметим, что $\|x\|_2^2 = x^T x$.

1-норма $\|x\|_1 = \sum_{i=1}^n |x_i|$

Мах-норма $\|x\|_\infty = \max_i |x_i|$.

0-норма $\|x\|_0 = \sum_{i=1}^n \mathbb{I}(|x_i| > 0)$. Это **псевдонорма**, потому что она не удовлетворяет условию однородности. Она равна количеству ненулевых элементов x . Если определить $0^0 = 0$, то ее можно записать в виде $\|x\|_0 = \sum_{i=1}^n x_i^0$.

7.1.3.2. Нормы матрицы

Будем считать, что матрица $\mathbf{A} \in \mathbb{R}^{m \times n}$ определяет линейную функцию $f(\mathbf{x}) = \mathbf{Ax}$. Определим **индуцированную норму** \mathbf{A} как максимальное удлинение входного вектора единичной нормы при воздействии f :

$$\|\mathbf{Ax}\|_p = \max_{\mathbf{x} \neq \mathbf{0}} \frac{\|\mathbf{Ax}\|_p}{\|\mathbf{x}\|_p} = \max_{\|\mathbf{x}\|=1} \|\mathbf{Ax}\|_p. \quad (7.22)$$

Обычно $p = 2$ и в этом случае

$$\|\mathbf{A}\|_2 = \sqrt{\lambda_{\max}(\mathbf{A}^T \mathbf{A})} = \max_i \sigma_i, \quad (7.23)$$

где σ_i – i -е сингулярное число.

Ядерная, или следовая норма определяется следующим образом:

$$\|\mathbf{A}\|_* = \text{tr}(\sqrt{\mathbf{A}^T \mathbf{A}}) = \sum_i \sigma_i, \quad (7.24)$$

где $\sqrt{\mathbf{A}^T \mathbf{A}}$ – квадратный корень из матрицы. Поскольку сингулярные числа всегда неотрицательны, имеем:

$$\|\mathbf{A}\|_* = \sum_i |\sigma_i| = \|\boldsymbol{\sigma}\|_1. \quad (7.25)$$

Использование этой нормы в качестве регуляризатора поощряет обращение многих сингулярных чисел в ноль, что дает матрицу низкого ранга. В общем случае можно определить **p -норму Шаттена**:

$$\|\mathbf{A}\|_p = \left(\sum_i \sigma_i^p(\mathbf{A}) \right)^{1/p}. \quad (7.26)$$

Если рассматривать матрицу как вектор, то можно определить норму матрицы в терминах нормы вектора, $\|\mathbf{A}\| = \|\text{vec}(\mathbf{A})\|$. Если в качестве нормы вектора берется 2-норма, то соответствующая норма матрицы называется **нормой Фробениуса**:

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2} = \sqrt{\text{tr}(\mathbf{A}^T \mathbf{A})} = \|\text{vec}(\mathbf{A})\|_2. \quad (7.27)$$

Если вычисление \mathbf{A} обходится дорого, а вычисление \mathbf{Av} дешево (для случайного вектора \mathbf{v}), то можно предложить стохастическую аппроксимацию нормы Фробениуса, воспользовавшись оценкой следа Хатчинсона из формулы (7.37):

$$\|\mathbf{A}\|_F^2 = \text{tr}(\mathbf{A}^T \mathbf{A}) = \mathbb{E}[\mathbf{v}^T \mathbf{A}^T \mathbf{A} \mathbf{v}] = \mathbb{E}[\|\mathbf{Av}\|_2^2], \quad (7.28)$$

где $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

7.1.4. Свойства матриц

В этом разделе мы обсудим различные скалярные свойства матриц.

7.1.4.1. След квадратной матрицы

Следом квадратной матрицы $A \in \mathbb{R}^{n \times n}$, обозначаемым $\text{tr}(A)$, называется сумма ее диагональных элементов:

$$\text{tr}(A) \triangleq \sum_{i=1}^n A_{ii}. \quad (7.29)$$

След обладает следующими свойствами, где $c \in \mathbb{R}$ – скаляр, а $A, B \in \mathbb{R}^{n \times n}$ – квадратные матрицы:

$$\text{tr}(A) = \text{tr}(A^T); \quad (7.30)$$

$$\text{tr}(A + B) = \text{tr}(A) + \text{tr}(B); \quad (7.31)$$

$$\text{tr}(cA) = c \text{tr}(A); \quad (7.32)$$

$$\text{tr}(AB) = \text{tr}(BA); \quad (7.33)$$

$$\text{tr}(A) = \sum_{i=1}^n \lambda_i, \text{ где } \lambda_i \text{ – собственные значения } A. \quad (7.34)$$

Имеет место также следующее **свойство циклической перестановки**: если матрицы A, B, C таковы, что произведение ABC – квадратная матрица, то

$$\text{tr}(ABC) = \text{tr}(BCA) = \text{tr}(CAB). \quad (7.35)$$

Отсюда следует **свойство следа**, позволяющее записать скалярное произведение $x^T Ax$ в виде:

$$x^T Ax = \text{tr}(x^T Ax) = \text{tr}(xx^T A). \quad (7.36)$$

Иногда вычислять матрицу A дорого, но можно дешево вычислить произведение матрицы на вектор Av . Пусть v – случайный вектор такой, что $\mathbb{E}[vv^T] = I$. В таком случае мы можем создать аппроксимацию Монте-Карло $\text{tr}(A)$, воспользовавшись следующим тождеством:

$$\text{tr}(A) = \text{tr}(A \mathbb{E}[vv^T]) = \mathbb{E}[\text{tr}(Avv^T)] = \mathbb{E}[\text{tr}(v^T Av)]. \quad (7.37)$$

Это называется **оценкой следа Хатчинсона** [Hut90].

7.1.4.2. Определитель квадратной матрицы

Определитель квадратной матрицы, $\det(A)$ или $|A|$, измеряет, как изменяется единичный объем под воздействием линейного преобразования, опи-

сываемого данной матрицей. (Формальное определение довольно сложное и нам здесь не нужно.)

Определитель обладает следующими свойствами ($\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$):

$$|\mathbf{A}| = |\mathbf{A}^T|; \quad (7.38)$$

$$|c\mathbf{A}| = c^n |\mathbf{A}|; \quad (7.39)$$

$$|\mathbf{AB}| = |\mathbf{A}||\mathbf{B}|; \quad (7.40)$$

$$|\mathbf{A}| = 0 \text{ тогда и только тогда, когда } \mathbf{A} \text{ сингулярная}; \quad (7.41)$$

$$|\mathbf{A}^{-1}| = 1/|\mathbf{A}|, \text{ если } \mathbf{A} \text{ не сингулярная}; \quad (7.42)$$

$$|\mathbf{A}| = \prod_{i=1}^n \lambda_i, \text{ где } \lambda_i \text{ – собственные значения } \mathbf{A}. \quad (7.43)$$

Положительно определенную матрицу \mathbf{A} можно записать в виде разложения Холецки $\mathbf{A} = \mathbf{LL}^T$, где \mathbf{L} – нижнетреугольная матрица. В таком случае

$$\det(\mathbf{A}) = \det(\mathbf{L})\det(\mathbf{L}^T) = \det(\mathbf{L})^2, \quad (7.44)$$

так что

$$\log \det(\mathbf{A}) = 2 \log \det(\mathbf{L}) = 2 \log \prod_i L_{ii} = 2 \text{trace}(\log(\text{diag}(\mathbf{L}))). \quad (7.45)$$

7.1.4.3. Ранг матрицы

Столбцовым рангом матрицы \mathbf{A} называется размерность пространства, натянутого на ее столбцы (иначе говоря, линейной оболочки столбцов), а **строковым рангом** – размерность пространства, натянутого на ее строки. Одним из важных результатов линейной алгебры является тот факт (его можно доказать с помощью сингулярного разложения, рассматриваемого в разделе 7.5), что для любой матрицы \mathbf{A} $\text{columnrank}(\mathbf{A}) = \text{rowrank}(\mathbf{A})$, поэтому обе величины обозначаются просто как $\text{rank}(\mathbf{A})$ и называются **рангом** \mathbf{A} . Ниже перечислены основные свойства ранга:

- для любой $\mathbf{A} \in \mathbb{R}^{m \times n}$ $\text{rank}(\mathbf{A}) \leq \min(m, n)$. Если $\text{rank}(\mathbf{A}) = \min(m, n)$, то говорят, что матрица \mathbf{A} **полного ранга**, в противном случае, что она **неполного ранга**;
- для любой $\mathbf{A} \in \mathbb{R}^{m \times n}$ $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{A}^T) = \text{rank}(\mathbf{A}^T \mathbf{A}) = \text{rank}(\mathbf{A} \mathbf{A}^T)$;
- для любых $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times p}$ $\text{rank}(\mathbf{AB}) \leq \min(\text{rank}(\mathbf{A}), \text{rank}(\mathbf{B}))$;
- для любых $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$ $\text{rank}(\mathbf{A} + \mathbf{B}) \leq \text{rank}(\mathbf{A}) + \text{rank}(\mathbf{B})$.

Можно показать, что квадратная матрица обратима тогда и только тогда, когда она имеет полный ранг.

7.1.4.4. Числа обусловленности

Число обусловленности матрицы \mathbf{A} является мерой ее численной устойчивости при вычислениях. Оно определяется следующим образом:

$$\kappa(\mathbf{A}) \triangleq \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|, \quad (7.46)$$

где $\|\mathbf{A}\|$ – норма матрицы. Можно показать, что $\kappa(\mathbf{A}) \geq 1$. (Число обусловленности зависит от используемой нормы; мы будем предполагать ℓ_2 -норму, если не оговорено противное.)

Говорят, что \mathbf{A} **хорошо обусловлена**, если $\kappa(\mathbf{A})$ мало (близко к 1), и **плохо обусловлена**, если $\kappa(\mathbf{A})$ велико. Большое число обусловленности означает, что \mathbf{A} почти сингулярна. Эта величина является лучшим показателем близости к сингулярности, чем величина определителя. Например, предположим, что $\mathbf{A} = 0.1\mathbf{I}_{100 \times 100}$. Тогда $\det(\mathbf{A}) = 10^{-100}$, и это позволяет предположить, что \mathbf{A} почти сингулярна, но $\kappa(\mathbf{A}) = 1$, т. е. \mathbf{A} хорошо обусловлена, что и неудивительно, поскольку оператор $\mathbf{A}\mathbf{x}$ просто умножает все элементы \mathbf{x} на 0.1.

Чтобы лучше понять смысл числа обусловленности, рассмотрим линейную систему уравнений $\mathbf{A}\mathbf{x} = \mathbf{b}$. Если \mathbf{A} не сингулярная, то у этой системы есть единственное решение $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. Заменим \mathbf{b} на $\mathbf{b} + \Delta\mathbf{b}$; как это повлияет на \mathbf{x} ? Новое решение должно удовлетворять уравнению:

$$\mathbf{A}(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b} + \Delta\mathbf{b}, \quad (7.47)$$

где

$$\Delta\mathbf{x} = \mathbf{A}^{-1}\Delta\mathbf{b}. \quad (7.48)$$

Мы говорим, что \mathbf{A} хорошо обусловлена, если небольшое изменение $\Delta\mathbf{b}$ приводит к небольшому изменению $\Delta\mathbf{x}$; в противном случае мы говорим, что \mathbf{A} плохо обусловлена.

Например, рассмотрим матрицу:

$$\mathbf{A} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 + 10^{-10} & 1 - 10^{-10} \end{pmatrix}, \quad \mathbf{A}^{-1} = \begin{pmatrix} 1 - 10^{-10} & 10^{-10} \\ 1 + 10^{-10} & -10^{-10} \end{pmatrix}. \quad (7.49)$$

Для $\mathbf{b} = (1, 1)$ решением является $\mathbf{x} = (1, 1)$. Если изменить \mathbf{b} на $\Delta\mathbf{b}$, то решение изменится на

$$\Delta\mathbf{x} = \mathbf{A}^{-1}\Delta\mathbf{b} = \begin{pmatrix} \Delta b_1 - 10^{10}(\Delta b_1 - \Delta b_2) \\ \Delta b_1 + 10^{10}(\Delta b_1 - \Delta b_2) \end{pmatrix}. \quad (7.50)$$

Таким образом, небольшое изменение \mathbf{b} может привести к сколь угодно большому изменению \mathbf{x} , потому что \mathbf{A} плохо обусловлена ($\kappa(\mathbf{A}) = 2 \times 10^{10}$).

В случае ℓ_2 -нормы число обусловленности равно отношению наибольшего сингулярного числа к наименьшему (определены в разделе 7.5), а сингулярные числа равны квадратным корням из собственных значений:

$$\kappa(\mathbf{A}) = \sigma_{\max}/\sigma_{\min} = \sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}}. \quad (7.51)$$

Еще глубже проникнуть в смысл чисел обусловленности можно, рассмотрев квадратичную целевую функцию $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x}$. Линии уровня этой функции эллиптические, как показано в разделе 7.4.4. При увеличении числа об-

условленности **A** эллипсы становятся все более вытянутыми вдоль некоторых осей, соответствующих очень узкой долине в пространстве функций. Если $k = 1$ (минимально возможное значение), то линии уровня будут круговыми.

7.1.5. Специальные типы матриц

В этом разделе мы рассмотрим некоторые специальные виды матриц с различной структурой.

7.1.5.1. Диагональная матрица

Диагональной называется матрица, все элементы которой, расположенные все главной диагонали, равны 0. Обычно она обозначается:

$$\mathbf{D} = \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix}. \quad (7.52)$$

Единичной матрицей, обозначаемой $\mathbf{I} \in \mathbb{R}^{n \times n}$, называется квадратная матрица, в которой на главной диагонали находятся единицы, а все остальные элементы равны 0, $\mathbf{I} = \text{diag}(1, 1, \dots, 1)$. Она обладает тем свойством, что

$$\mathbf{A}\mathbf{I} = \mathbf{A} = \mathbf{I}\mathbf{A}, \quad (7.53)$$

где размер **I** определяется размерами **A**, так чтобы умножение матриц было возможно.

Операция $\mathbf{d} = \text{diag}(\mathbf{D})$ извлекает из матрицы вектор диагональных элементов. Операция $\mathbf{D} = \text{diag}(\mathbf{d})$ преобразует вектор в диагональную матрицу.

Блочно-диагональной называется матрица, которая содержит матрицы на главной диагонали и нули во всех остальных местах, например:

$$\begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{pmatrix}. \quad (7.54)$$

Ленточной называется матрица, в которой ненулевые элементы находятся на главной диагонали и не далее, чем в k позициях по обе стороны от нее, где k – ширина ленты. Например, **тридиагональная** матрица размера 6×6 имеет вид:

$$\begin{bmatrix} A_{11} & A_{12} & 0 & \dots & \dots & 0 \\ A_{21} & A_{22} & A_{23} & \ddots & \ddots & \vdots \\ 0 & A_{32} & A_{33} & A_{34} & \ddots & \vdots \\ \vdots & \ddots & A_{43} & A_{44} & A_{45} & 0 \\ \vdots & \ddots & \ddots & A_{54} & A_{55} & A_{56} \\ 0 & \dots & \dots & 0 & A_{65} & A_{66} \end{bmatrix}. \quad (7.55)$$

7.1.5.2. Треугольные матрицы

В **верхнетреугольной матрице** ненулевые элементы находятся только на диагонали и над ней. В **нижнетреугольной матрице** ненулевые элементы находятся только на диагонали и под ней.

У треугольных матриц есть полезное свойство: диагональные элементы A являются ее собственными значениями, поэтому определитель равен произведению диагональных элементов: $\det(A) = \prod_i A_{ii}$.

7.1.5.3. Положительно определенные матрицы

Для квадратной матрицы $A \in \mathbb{R}^{n \times n}$ скалярное значение $\mathbf{x}^T A \mathbf{x}$ называется **квадратичной формой**. В явном виде:

$$\mathbf{x}^T A \mathbf{x} = \sum_{i=1}^n \sum_{j=1}^n A_{ij} x_i x_j. \quad (7.56)$$

Заметим, что

$$\mathbf{x}^T A \mathbf{x} = (\mathbf{x}^T A \mathbf{x})^T = \mathbf{x}^T A^T \mathbf{x} = \mathbf{x}^T \left(\frac{1}{2} A + \frac{1}{2} A^T \right) \mathbf{x}. \quad (7.57)$$

Поэтому часто неявно предполагается, что матрицы, входящие в квадратичную форму, симметричны.

Дадим следующие определения.

- Симметричная матрица $A \in \mathbb{S}^n$ называется **положительно определенной**, если для любого ненулевого вектора $\mathbf{x} \in \mathbb{R}^n$ $\mathbf{x}^T A \mathbf{x} > 0$. Обычно это записывают в виде $A > 0$ (или просто $A > 0$). Если может случиться, что $\mathbf{x}^T A \mathbf{x} = 0$, то говорят, что матрица **положительно полуопределенная**. Множество всех положительно определенных матриц обозначает \mathbb{S}_{++}^n .
- Симметричная матрица $A \in \mathbb{S}^n$ называется **отрицательно определенной** ($A < 0$ или просто $A < 0$), если для любого ненулевого вектора $\mathbf{x} \in \mathbb{R}^n$ $\mathbf{x}^T A \mathbf{x} < 0$. Если может случиться, что $\mathbf{x}^T A \mathbf{x} = 0$, то говорят, что матрица **отрицательно полуопределенная**.
- Симметричная матрица $A \in \mathbb{S}^n$ называется **неопределенной**, если она не является ни положительно полуопределенной, ни отрицательно полуопределенной, т. е. если существуют такие $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$, что $\mathbf{x}_1^T A \mathbf{x}_1 > 0$ и $\mathbf{x}_2^T A \mathbf{x}_2 < 0$.

Очевидно, что если A положительно определенная, то $-A$ отрицательно определенная, и наоборот. Аналогично если A положительно полуопределенная, то $-A$ отрицательно полуопределенная, и наоборот. Можно показать, что положительно и отрицательно определенные матрицы обратимы.

В разделе 7.4.3.1 будет показано, что симметричная матрица является положительно определенной тогда и только тогда, когда ее собственные значения положительны. Отметим, что, если все элементы A положительны, это еще не значит, что A положительно определенная. Например, $A = \begin{pmatrix} 4 & 3 \\ 3 & 2 \end{pmatrix}$

не является положительно определенной. С другой стороны, положительно определенная матрица может иметь отрицательные элементы, например,

$$\mathbf{A} = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}.$$

Достаточным условием положительной определенности (вещественной симметричной) матрицы является **диагональное преобладание**, т. е. в каждой строке абсолютная величина диагонального элемента должна быть больше суммы абсолютных величин все остальных элементов. Точнее,

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \text{ для всех } i. \quad (7.58)$$

В двумерном случае вещественная симметричная матрица $2 \times 2 \begin{pmatrix} a & b \\ b & d \end{pmatrix}$ является положительно определенной тогда и только тогда, когда $a > 0$, $d > 0$ и $ad > b^2$.

Наконец, один частный случай положительно определенных матриц встречается настолько часто, что заслуживает особого упоминания. Для любой матрицы $\mathbf{A} \in \mathbb{R}^{m \times n}$ (необязательно симметричной или даже квадратной) матрица Грама $\mathbf{G} = \mathbf{A}^T \mathbf{A}$ всегда положительно полуопределена. Более того, если $m \geq n$ (и для удобства предполагается, что матрица \mathbf{A} полного ранга), то $\mathbf{G} = \mathbf{A}^T \mathbf{A}$ является положительно определенной.

7.1.5.4. Ортогональные матрицы

Два вектора $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ называются **ортогональными**, если $\mathbf{x}^T \mathbf{y} = 0$. Вектор $\mathbf{x} \in \mathbb{R}^n$ называется **нормированным**, если $\|\mathbf{x}\|^2 = 1$. Множество попарно ортогональных и нормированных векторов называется **ортонормированным**. Квадратная матрица $\mathbf{U} \in \mathbb{R}^{n \times n}$ называется **ортогональной**, если множество ее столбцов ортонормировано. (Обратите внимание на различный смысл термина «ортогональный» применительно к векторам и матрицам.) Если элементы \mathbf{U} – комплексные числа, то матрицу называют не ортогональной, а **унитарной**.

Из определения ортогональности и нормированности сразу следует, что \mathbf{U} ортогональна тогда и только тогда, когда

$$\mathbf{U}^T \mathbf{U} = \mathbf{I} = \mathbf{U} \mathbf{U}^T. \quad (7.59)$$

Иными словами, матрицей, обратной к ортогональной, является транспонированная. Заметим, что если \mathbf{U} не квадратная, т. е. $\mathbf{U} \in \mathbb{R}^{m \times n}$, где $n < m$, но ее столбцы все равно ортонормированы, то $\mathbf{U}^T \mathbf{U} = \mathbf{I}$, но $\mathbf{U} \mathbf{U}^T \neq \mathbf{I}$. Мы будем использовать термин «ортогональная» только в случае, когда \mathbf{U} квадратная.

Примером ортогональной матрицы является **матрица поворота** (см. упражнение 7.1). Например, поворот в трехмерном пространстве на угол α вокруг оси z описывается матрицей:

$$\mathbf{R}(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (7.60)$$

Если $\alpha = 45^\circ$, то эта матрица принимает вид

$$\mathbf{R}(45) = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad (7.61)$$

где $1/\sqrt{2} = 0.7071$. Как видим, $\mathbf{R}(-\alpha) = \mathbf{R}(\alpha)^{-1} = \mathbf{R}(\alpha)^T$, т. е. это ортогональная матрица.

Одним из замечательных свойств ортогональных матриц является то, что умножение вектора на такую матрицу не изменяет его евклидовой нормы, т. е.

$$\|\mathbf{U}\mathbf{x}\|_2 = \|\mathbf{x}\|_2 \quad (7.62)$$

для любого ненулевого $\mathbf{x} \in \mathbb{R}^n$ и ортогональной матрицы $\mathbf{U} \in \mathbb{R}^{n \times n}$.

Аналогично можно показать, что угол между двумя векторами сохраняется после преобразования их ортогональной матрицей. Косинус угла между \mathbf{x} и \mathbf{y} равен:

$$\cos(\alpha(\mathbf{x}, \mathbf{y})) = \frac{\mathbf{x}^T \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}, \quad (7.63)$$

так что

$$\cos(\alpha(\mathbf{U}\mathbf{x}, \mathbf{U}\mathbf{y})) = \frac{(\mathbf{U}\mathbf{x})^T (\mathbf{U}\mathbf{y})}{\|\mathbf{U}\mathbf{x}\| \|\mathbf{U}\mathbf{y}\|} = \frac{\mathbf{x}^T \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} = \cos(\alpha(\mathbf{x}, \mathbf{y})). \quad (7.64)$$

Короче говоря, преобразования с помощью ортогональных матриц являются обобщениями вращения (если $\det(\mathbf{U}) = 1$) и отражения (если $\det(\mathbf{U}) = -1$), поскольку они сохраняют длины и углы.

Заметим, что существует метод ортогонализации Грама–Шмидта, позволяющий сделать любую квадратную матрицу ортогональной, но здесь мы его не рассматриваем.

7.2. УМНОЖЕНИЕ МАТРИЦ

Произведением двух матриц $\mathbf{A} \in \mathbb{R}^{m \times n}$ и $\mathbf{B} \in \mathbb{R}^{n \times p}$ является матрица:

$$\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}, \quad (7.65)$$

где

$$C_{ij} = \sum_{k=1}^n A_{ik} B_{kj}. \quad (7.66)$$

Заметим, что для существования произведения число столбцов **A** должно быть равно числу строк **B**.

Умножение матриц занимает время $O(mnp)$, хотя существуют более быстрые способы. Кроме того, для значительного ускорения умножения можно воспользоваться специализированным оборудованием, например графическими (GPU) и тензорными (TPU) процессорами, которые позволяют выполнять операции над строками (или столбцами) параллельно.

Полезно знать некоторые основные свойства умножения матриц:

- умножение матриц **ассоциативно**: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$;
- умножение матриц **дистрибутивно** относительно сложения: $\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$;
- в общем случае умножение матриц **не коммутативно**, т. е. $\mathbf{AB} \neq \mathbf{BA}$.
(Во всех случаях предполагается, что размеры матриц совместимы.)

Ниже мы обсудим важные частные случаи умножения матриц.

7.2.1. Умножение векторов

Скалярным, или **внутренним**, **произведением** двух векторов $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ называется вещественное число:

$$\langle \mathbf{x}, \mathbf{y} \rangle \triangleq \mathbf{x}^T \mathbf{y} = \sum_{i=1}^n x_i y_i. \quad (7.67)$$

Заметим, что всегда $\mathbf{x}^T \mathbf{y} = \mathbf{y}^T \mathbf{x}$.

Внешним произведением векторов $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n$ (необязательно одинакового размера) называется следующая матрица:

$$\mathbf{xy}^T \in \mathbb{R}^{m \times n} = \begin{bmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_m y_1 & x_m y_2 & \cdots & x_m y_n \end{bmatrix}. \quad (7.68)$$

7.2.2. Произведение матрицы на вектор

Произведением матрицы $\mathbf{A} \in \mathbb{R}^{m \times n}$ на вектор $\mathbf{x} \in \mathbb{R}^n$ называется вектор $\mathbf{y} = \mathbf{Ax} \in \mathbb{R}^m$. Есть два способа интерпретировать умножение матрицы на вектор, и мы рассмотрим оба.

Если записать **A** по строкам, то $\mathbf{y} = \mathbf{Ax}$ можно выразить следующим образом:

$$\mathbf{y} = \mathbf{A}\mathbf{x} = \begin{bmatrix} - & \mathbf{a}_1^\top & - \\ - & \mathbf{a}_2^\top & - \\ & \vdots & \\ - & \mathbf{a}_m^\top & - \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{a}_1^\top \mathbf{x} \\ \mathbf{a}_2^\top \mathbf{x} \\ \vdots \\ \mathbf{a}_m^\top \mathbf{x} \end{bmatrix}. \quad (7.69)$$

Иными словами, i -й элемент \mathbf{y} равен скалярному произведению i -й строки \mathbf{A} и \mathbf{x} , $y_i = \mathbf{a}_i^\top \mathbf{x}$.

С другой стороны, давайте запишем \mathbf{A} по столбцам. В таком случае

$$\mathbf{y} = \mathbf{A}\mathbf{x} = \begin{bmatrix} | & | & & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} | \\ \mathbf{a}_1 \\ | \end{bmatrix} x_1 + \begin{bmatrix} | \\ \mathbf{a}_2 \\ | \end{bmatrix} x_2 + \cdots + \begin{bmatrix} | \\ \mathbf{a}_n \\ | \end{bmatrix} x_n. \quad (7.70)$$

Иными словами, \mathbf{y} – линейная комбинация столбцов \mathbf{A} , где коэффициентами являются элементы \mathbf{x} . Столбцы \mathbf{A} можно рассматривать как векторы базиса, определяющего линейное подпространство. Мы можем строить в этом подпространстве векторы в виде линейных комбинаций базисных векторов. Детали см. в разделе 7.1.2.

7.2.3. Произведение матриц

Далее мы рассмотрим четыре разных (но, конечно, эквивалентных) способа интерпретации произведения матриц $\mathbf{C} = \mathbf{AB}$.

Сначала будем рассматривать произведение матриц как множество произведений векторов. Самая очевидная точка зрения, вытекающая непосредственно из определения, заключается в том, что элемент (i, j) матрицы \mathbf{C} равен скалярному произведению i -й строки \mathbf{A} и j -го столбца \mathbf{B} . Символически это выглядит так:

$$\mathbf{C} = \mathbf{AB} = \begin{bmatrix} - & \mathbf{a}_1^\top & - \\ - & \mathbf{a}_2^\top & - \\ & \vdots & \\ - & \mathbf{a}_m^\top & - \end{bmatrix} \begin{bmatrix} | & | & \cdots & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_p \\ | & | & & | \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^\top \mathbf{b}_1 & \mathbf{a}_1^\top \mathbf{b}_2 & \cdots & \mathbf{a}_1^\top \mathbf{b}_p \\ \mathbf{a}_2^\top \mathbf{b}_1 & \mathbf{a}_2^\top \mathbf{b}_2 & \cdots & \mathbf{a}_2^\top \mathbf{b}_p \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_m^\top \mathbf{b}_1 & \mathbf{a}_m^\top \mathbf{b}_2 & \cdots & \mathbf{a}_m^\top \mathbf{b}_p \end{bmatrix}. \quad (7.71)$$

Напомним, что поскольку $\mathbf{A} \in \mathbb{R}^{m \times n}$ и $\mathbf{B} \in \mathbb{R}^{n \times p}$, то $\mathbf{a}_i \in \mathbb{R}^n$ и $\mathbf{b}_j \in \mathbb{R}^m$, поэтому эти скалярные произведения имеют смысл. Это самое «естественное» представление – когда \mathbf{A} записана по строкам, а \mathbf{B} по столбцам (см. иллюстрацию на рис. 7.5).

Альтернативно можно записать \mathbf{A} по столбцам, а \mathbf{B} по строкам, тогда \mathbf{AB} будет интерпретироваться как сумма внешних произведений. В символическом виде:

$$C = AB = \begin{bmatrix} | & | & \dots & | \\ a_1 & a_2 & \dots & a_n \\ | & | & \dots & | \end{bmatrix} \begin{bmatrix} - & b_1^T & - \\ - & b_2^T & - \\ \vdots & \vdots & \vdots \\ - & b_n^T & - \end{bmatrix} = \sum_{i=1}^n a_i b_i^T. \quad (7.72)$$

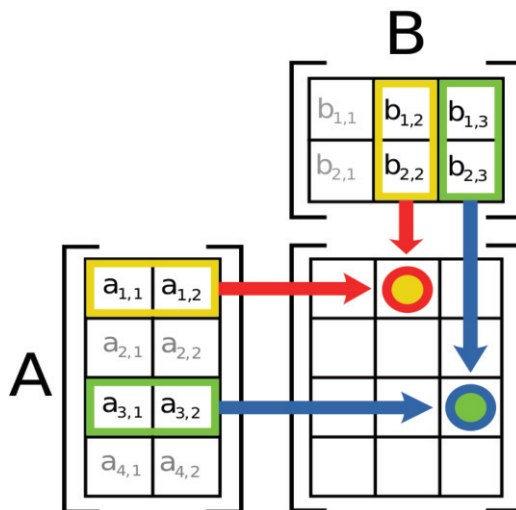


Рис. 7.5 ❖ Умножение матриц.

Взято со страницы https://en.wikipedia.org/wiki/Matrix_multiplication.

Печатается с разрешения автора «Википедии» Bilou

Иначе говоря, AB равно сумме по всем i внешних произведений i -го столбца A на i -ю строку B . Так как в этом случае $a_i \in \mathbb{R}^m$ и $b_i \in \mathbb{R}^p$, размерность внешнего произведения $a_i b_i^T$ равна $m \times p$, что совпадает с размерностью C .

Можно также рассматривать произведение матриц как множество произведение матрицы на вектор. Именно, если записать B по столбцам, то можно рассматривать столбцы C произведения матрицы A на столбцы B . В символическом виде:

$$C = AB = A \begin{bmatrix} | & | & \dots & | \\ b_1 & b_2 & \dots & b_p \\ | & | & \dots & | \end{bmatrix} = \begin{bmatrix} | & | & \dots & | \\ Ab_1 & Ab_2 & \dots & Ab_p \\ | & | & \dots & | \end{bmatrix}. \quad (7.73)$$

Здесь i -й столбец C равен произведению матрицы A на вектор справа, $c_i = Ab_i$. Эти произведения матрицы на вектор в свою очередь могут быть интерпретированы обоими способами, описанными в предыдущем разделе.

Наконец, аналогичную интерпретацию мы получаем, записывая A по строкам и рассматривая строки C как произведения строк A на матрицу B . В символическом виде:

$$\mathbf{C} = \mathbf{AB} = \begin{bmatrix} - & \mathbf{a}_1^T & - \\ - & \mathbf{a}_2^T & - \\ & \vdots & \\ - & \mathbf{a}_m^T & - \end{bmatrix} \mathbf{B} = \begin{bmatrix} - & \mathbf{a}_1^T \mathbf{B} & - \\ - & \mathbf{a}_2^T \mathbf{B} & - \\ & \vdots & \\ - & \mathbf{a}_m^T \mathbf{B} & - \end{bmatrix}. \quad (7.74)$$

Здесь i -я строка \mathbf{C} равна произведению матрицы на вектор слева, $\mathbf{c}_i^T = \mathbf{a}_i^T \mathbf{B}$. Может показаться перебором уделять так много внимания умножению матриц, тем более что все интерпретации непосредственно вытекают из первоначального определения, данного в начале раздела. Однако практически вся линейная алгебра так или иначе связана с перемножением матриц, поэтому стоило потратить некоторое время, чтобы выработать интуитивное понимание различных точек зрения.

И напоследок еще пара слов о нотации. Мы используем \mathbf{A}^2 как сокращенную запись \mathbf{AA} , т. е. произведения матриц. Для обозначения поэлементного возведения матрицы в квадрат мы пишем $\mathbf{A}^{\odot 2} = [A_{ij}^2]$. (Если \mathbf{A} – диагональная матрица, то $\mathbf{A}^2 = \mathbf{A}^{\odot 2}$.)

Можно также определить операцию, обратную к \mathbf{A}^2 – **матричный квадратный корень**: мы говорим, что $\mathbf{A} = \sqrt{\mathbf{M}}$, если $\mathbf{A}^2 = \mathbf{M}$. Для обозначения поэлементного квадратного корня из матрицы применяется нотация $[\sqrt{M_{ij}}]$.

7.2.4. Приложение: манипулирование матрицами данных

Рассмотрим случай, когда \mathbf{X} – матрица плана $N \times D$, строки которой представляют примеры данных. К этой матрице можно применить разнообразные операции предобработки, которые мы опишем ниже. (Запись этих операций в матричной форме полезна, потому что нотация получается компактной и позволяет быстро реализовывать методы с помощью кода для работы с матрицами.)

7.2.4.1. Суммирование срезов матрицы

Пусть \mathbf{X} – матрица $N \times D$. Мы можем просуммировать ее элементы по строкам, умножив слева на матрицу $1 \times N$, составленную из единиц, в результате чего получится матрица $1 \times D$:

$$\mathbf{1}_N^T \mathbf{X} = \left(\sum_n x_{n1} \quad \cdots \quad \sum_n x_{nD} \right). \quad (7.75)$$

Отсюда можно найти среднее векторов данных:

$$\bar{\mathbf{x}}^T = \frac{1}{N} \mathbf{1}_N^T \mathbf{X}. \quad (7.76)$$

Чтобы просуммировать элементы матрицы по столбцам, умножим ее справа на матрицу $D \times 1$, составленную из единиц, и получим матрицу $N \times 1$:

$$\mathbf{X}\mathbf{1}_D = \begin{pmatrix} \sum_d x_{1d} \\ \vdots \\ \sum_d x_{Nd} \end{pmatrix}. \quad (7.77)$$

Чтобы просуммировать все элементы матрицы, умножим ее слева и справа на вектор единиц:

$$\mathbf{1}_N^\top \mathbf{X}\mathbf{1}_D = \sum_{ij} X_{ij}. \quad (7.78)$$

Таким образом, среднее по всем элементам равно:

$$\bar{x} = \frac{1}{ND} \mathbf{1}_N^\top \mathbf{X}\mathbf{1}_D. \quad (7.79)$$

7.2.4.2. Масштабирование строк и столбцов матрицы

Часто требуется масштабировать строки или столбцы матрицы данных (например, чтобы стандартизировать их). Покажем, как записать это в матричной нотации.

Умножив \mathbf{X} слева на диагональную матрицу $\mathbf{S} = \text{diag}(\mathbf{s})$, где \mathbf{s} – вектор длины N , мы умножим каждую строку \mathbf{X} на соответствующий масштабный коэффициент:

$$\text{diag}(\mathbf{s})\mathbf{X} = \begin{pmatrix} s_1 & \cdots & 0 \\ & \ddots & \\ 0 & \cdots & s_N \end{pmatrix} \begin{pmatrix} x_{1,1} & \cdots & x_{1,D} \\ & \ddots & \\ x_{N,1} & \cdots & x_{N,D} \end{pmatrix} = \begin{pmatrix} s_1 x_{1,1} & \cdots & s_1 x_{1,D} \\ & \ddots & \\ s_N x_{N,1} & \cdots & s_N x_{N,D} \end{pmatrix}. \quad (7.80)$$

Умножая \mathbf{X} справа на диагональную матрицу $\mathbf{S} = \text{diag}(\mathbf{s})$, где \mathbf{s} – вектор длины D , мы умножаем каждый столбец \mathbf{X} на соответствующий масштабный коэффициент:

$$\mathbf{X}\text{diag}(\mathbf{s}) = \begin{pmatrix} x_{1,1} & \cdots & x_{1,D} \\ & \ddots & \\ x_{N,1} & \cdots & x_{N,D} \end{pmatrix} \begin{pmatrix} s_1 & \cdots & 0 \\ & \ddots & \\ 0 & \cdots & s_D \end{pmatrix} = \begin{pmatrix} s_1 x_{1,1} & \cdots & s_D x_{1,D} \\ & \ddots & \\ s_1 x_{N,1} & \cdots & s_D x_{N,D} \end{pmatrix}. \quad (7.81)$$

Таким образом, операцию стандартизации из раздела 10.2.8 можно записать в матричной форме следующим образом:

$$\text{standardize}(\mathbf{X}) = (\mathbf{X} - \mathbf{1}_N \boldsymbol{\mu}^\top) \text{diag}(\boldsymbol{\sigma})^{-1}, \quad (7.82)$$

где $\boldsymbol{\mu} = \bar{\mathbf{x}}$ – эмпирическое среднее, а $\boldsymbol{\sigma}$ – вектор эмпирических стандартных отклонений.

7.2.4.3. Матрица сумм квадратов и матрица рассеяния

Матрицей сумм квадратов называется матрица размера $D \times D$, определенная следующим образом:

$$\mathbf{S}_0 \triangleq \mathbf{X}^T \mathbf{X} = \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T = \sum_{n=1}^N \begin{pmatrix} x_{n,1}^2 & \cdots & x_{n,1}x_{n,D} \\ & \ddots & \\ x_{n,D}x_{n,1} & \cdots & x_{n,D}^2 \end{pmatrix}. \quad (7.83)$$

Матрицей рассеяния называется матрица размера $D \times D$, определенная следующим образом:

$$\mathbf{S}_{\bar{\mathbf{x}}} \triangleq \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T = \left(\sum_n \mathbf{x}_n \mathbf{x}_n^T \right) - N \bar{\mathbf{x}} \bar{\mathbf{x}}^T. \quad (7.84)$$

Как видим, это матрица сумм квадратов, примененная к центрированным данным. Точнее, обозначим $\tilde{\mathbf{X}}$ матрицу, получающуюся из \mathbf{X} вычитанием среднего $\bar{\mathbf{x}} = (1/N)\mathbf{X}^T \mathbf{1}_N$ из каждой строки. Тогда центрированная матрица данных вычисляется по формуле:

$$\tilde{\mathbf{X}} = \mathbf{X} - \mathbf{1}_N \bar{\mathbf{x}}^T = \mathbf{X} - \frac{1}{N} \mathbf{1}_N \mathbf{1}_N^T \mathbf{X} = \mathbf{C}_N \mathbf{X}, \quad (7.85)$$

где

$$\mathbf{C}_N \triangleq \mathbf{I}_N - \frac{1}{N} \mathbf{1}_N \mathbf{1}_N^T \quad (7.86)$$

называется **центрирующей матрицей**. Теперь матрицу рассеяния можно вычислить следующим образом:

$$\mathbf{S}_{\bar{\mathbf{x}}} = \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} = \mathbf{X}^T \mathbf{C}_N^T \mathbf{C}_N \mathbf{X} = \mathbf{X}^T \mathbf{C}_N \mathbf{X}, \quad (7.87)$$

где мы воспользовались тем фактом, что \mathbf{C}_N симметрична и идемпотентна, т. е. $\mathbf{C}_N^k = \mathbf{C}_N$ для $k = 1, 2, \dots$ (так как после вычитания среднего повторное вычитание не приводит ни к каким изменениям).

7.2.4.4. Матрица Грама

Матрица $\mathbf{X}\mathbf{X}^T$ размера $N \times N$, состоящая из скалярных произведений, называется **матрицей Грама**:

$$\mathbf{K} \triangleq \mathbf{X}\mathbf{X}^T = \begin{pmatrix} \mathbf{x}_1^T \mathbf{x}_1 & \cdots & \mathbf{x}_1^T \mathbf{x}_N \\ & \ddots & \\ \mathbf{x}_N^T \mathbf{x}_1 & \cdots & \mathbf{x}_N^T \mathbf{x}_N \end{pmatrix}. \quad (7.88)$$

Иногда мы хотим вычислить скалярные произведения центрированных векторов данных, $\tilde{\mathbf{K}} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}^T$. Однако, работая с матрицей сходства признаков, а не с исходными признаками, мы имеем только доступ к \mathbf{K} , но не к \mathbf{X} . (Примеры будут приведены в разделах 20.4.4 и 20.4.6.) К счастью, можно вычислить $\tilde{\mathbf{K}}$ по \mathbf{K} , применив **двойное центрирование**:

$$\tilde{\mathbf{K}} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top = \mathbf{C}_N\mathbf{K}\mathbf{C}_N = \mathbf{K} - \frac{1_N}{N}\mathbf{K} - \mathbf{K}\frac{1_N}{N} + \frac{1_N}{N}\mathbf{K}\frac{1_N}{N}. \quad (7.89)$$

Здесь мы вычитаем из \mathbf{K} средние по строкам и по столбцам и прибавляем глобальное среднее, которое было вычтено дважды, чтобы средние $\tilde{\mathbf{K}}$ по строкам и по столбцам были равны нулю.

Чтобы убедиться в справедливости (7.89), перепишем эту формулу в скалярном виде:

$$\tilde{K}_{ij} = \tilde{\mathbf{x}}_i^\top \tilde{\mathbf{x}}_j = \left(\mathbf{x}_i - \frac{1}{N} \sum_{k=1}^N \mathbf{x}_k \right)^\top \left(\mathbf{x}_j - \frac{1}{N} \sum_{l=1}^N \mathbf{x}_l \right) \quad (7.90)$$

$$= \mathbf{x}_i^\top \mathbf{x}_j - \frac{1}{N} \sum_{k=1}^N \mathbf{x}_i^\top \mathbf{x}_k - \frac{1}{N} \sum_{k=1}^N \mathbf{x}_j^\top \mathbf{x}_k + \frac{1}{N^2} \sum_{k=1}^N \sum_{l=1}^N \mathbf{x}_k^\top \mathbf{x}_l. \quad (7.91)$$

7.2.4.5. Матрица расстояний

Пусть \mathbf{X} – матрица данных размера $N_x \times D$, а \mathbf{Y} – другая матрица данных размера $N_y \times D$. Мы можем вычислить квадраты попарных расстояний между элементами этих матриц:

$$\mathbf{D}_{ij} = (\mathbf{x}_i - \mathbf{y}_j)^\top (\mathbf{x}_i - \mathbf{y}_j) = \|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^\top \mathbf{y}_j + \|\mathbf{y}_j\|^2. \quad (7.92)$$

Теперь запишем это в матричной форме. Обозначим $\hat{\mathbf{x}} = [\|\mathbf{x}_1\|^2, \dots, \|\mathbf{x}_{N_x}\|^2] = \text{diag}(\mathbf{X}\mathbf{X}^\top)$ вектор, каждый элемент которого является квадратом нормы примера из \mathbf{X} . Аналогично определим $\hat{\mathbf{y}}$. Тогда имеем:

$$\mathbf{D} = \hat{\mathbf{x}}\mathbf{1}_{N_y}^\top - 2\mathbf{X}\mathbf{Y}^\top + \mathbf{1}_{N_x}\hat{\mathbf{y}}^\top. \quad (7.93)$$

В случае, когда $\mathbf{X} = \mathbf{Y}$, имеем:

$$\mathbf{D} = \hat{\mathbf{x}}\mathbf{1}_N^\top - 2\mathbf{X}\mathbf{X}^\top + \mathbf{1}_N\hat{\mathbf{x}}^\top. \quad (7.94)$$

Это векторное вычисление зачастую оказывается гораздо быстрее, чем использование циклов for.

7.2.5. Произведения Кронекера*

Если \mathbf{A} – матрица $m \times n$, а \mathbf{B} – матрица $p \times q$, то **произведением Кронекера** $\mathbf{A} \otimes \mathbf{B}$ называется блочная матрица размера $mp \times nq$:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}. \quad (7.95)$$

Например,

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} & a_{12}b_{11} & a_{12}b_{12} & a_{12}b_{13} \\ a_{11}b_{21} & a_{11}b_{22} & a_{11}b_{23} & a_{12}b_{21} & a_{12}b_{22} & a_{12}b_{23} \\ a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} & a_{22}b_{11} & a_{22}b_{12} & a_{22}b_{13} \\ a_{21}b_{21} & a_{21}b_{22} & a_{21}b_{23} & a_{22}b_{21} & a_{22}b_{22} & a_{22}b_{23} \\ a_{31}b_{11} & a_{31}b_{12} & a_{31}b_{13} & a_{32}b_{11} & a_{32}b_{12} & a_{32}b_{13} \\ a_{31}b_{21} & a_{31}b_{22} & a_{31}b_{23} & a_{32}b_{21} & a_{32}b_{22} & a_{32}b_{23} \end{bmatrix}. \quad (7.96)$$

Приведем два полезных тождества:

$$(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}; \quad (7.97)$$

$$(\mathbf{AB})\text{vec}(\mathbf{C}) = \text{vec}(\mathbf{BCA}^T). \quad (7.98)$$

Дополнительные свойства см. в работе [Loa00].

7.2.6. Суммирование Эйнштейна*

Суммирование Эйнштейна, или **einsum**, – краткая нотация для работы с тензорами. Это соглашение было введено Эйнштейном [Ein16, раздел 5], который позднее шутил: «Я сделал выдающееся открытие в математике – опустил знак суммы всюду, где суммирование должно производиться по индексу, встречающемуся дважды...» [Pai05, стр. 216]. Например, вместо того чтобы записывать произведение матриц в виде $C_{ij} = \sum_k A_{ik}B_{kj}$, мы можем просто написать $C_{ij} = A_{ik}B_{kj}$, опустив \sum_k .

В качестве более сложного примера рассмотрим трехмерный тензор S_{ntk} , где n индексирует примеры в пакете, t – позиции внутри последовательности, а k – слова в унитарном представлении. Обозначим W_{kd} матрицу погружений, которая отображает разреженные унитарные векторы, принадлежащие \mathbb{R}^k в плотные векторы, принадлежащие \mathbb{R}^d . Мы можем преобразовать пакет последовательностей унитарных векторов в пакет последовательностей погружений следующим образом:

$$E_{ntd} = \sum_k S_{ntk} W_{kd}. \quad (7.99)$$

Вычислить сумму векторов погружений для каждой последовательности (и получить глобальное представление каждого мешка слов) можно так:

$$E_{nd} = \sum_k \sum_t S_{ntk} W_{kd}. \quad (7.100)$$

Наконец, мы можем подвергнуть представление каждого вектора последовательности еще одному линейному преобразованию V_{dc} с целью отобразить его в логиты над классификатором с метками:

$$L_{nc} = \sum_d E_{nd} V_{dc} = \sum_d \sum_k \sum_t S_{ntk} W_{kd} V_{dc}. \quad (7.101)$$

В нотации суммирования Эйнштейна можно написать $L_{nc} = S_{ntk}W_{kd}V_{dc}$. Здесь производится суммирование по k и d , потому что эти индексы встречаются дважды в правой части. А суммирование по t производится, потому что этот индекс не встречается в левой части.

Суммирование Эйнштейна реализовано в NumPy, Tensorflow, PyTorch и т. д. Особенно полезно то, что оно позволяет выполнять тензорное умножение в сложных выражениях в оптимальном порядке, минимизирующем время и расход промежуточной памяти¹. В программе code.problml.ai/book1/einsum_demo имеются хорошие примеры работы с библиотекой.

Заметим, что быстроедействие einsum зависит от порядка, в котором выполняются операции, а он зависит от форм релевантных аргументов. При оптимальном порядке минимизируется ширина результирующего графа вычислений, как объяснено в работе [GASG18]. В общем случае время вычисления оптимального порядка экспоненциально возрастает с увеличением числа аргументов, поэтому обычно применяют жадную аппроксимацию. Однако если мы ожидаем, что одно и то же вычисление будет повторяться много раз с тензорами одинаковой формы, но, быть может, разного содержания, то можно один раз вычислить оптимальный порядок и использовать его многократно.

7.3. ОБРАЩЕНИЕ МАТРИЦ

В этом разделе мы обсудим, как обращаться матрицы разных видов.

7.3.1. Обращение квадратной матрицы

Обратной матрицей для квадратной матрицы $A \in \mathbb{R}^{n \times n}$ называется однозначно определенная матрица A^{-1} такая, что

$$A^{-1}A = I = AA^{-1}. \quad (7.102)$$

Заметим, что A^{-1} существует тогда и только тогда, когда $\det(A) \neq 0$. Если $\det(A) = 0$, то матрица называется **сингулярной**.

Ниже перечислены свойства обратной матрицей, везде предполагается, что $A, B \in \mathbb{R}^{n \times n}$ – несингулярные матрицы:

$$(A^{-1})^{-1} = A; \quad (7.103)$$

$$(AB)^{-1} = B^{-1}A^{-1}; \quad (7.104)$$

$$(A^{-1})^T = (A^T)^{-1} \triangleq A^{-T}. \quad (7.105)$$

¹ Эти оптимизации реализованы в библиотеке opt-einsum [GASG18]. Ее основная функциональность включается в функции einsum пакетов NumPy и JAX, если задан параметр optimize=True.

В случае матриц 2×2 выражение для \mathbf{A}^{-1} настолько простое, что обратную матрицу можно выписать в явном виде. Имеем

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad \mathbf{A}^{-1} = \frac{1}{|\mathbf{A}|} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}. \quad (7.106)$$

Для блочно-диагональной матрицы обратная получается обращением каждого блока по отдельности, например:

$$\begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}^{-1} \end{pmatrix}. \quad (7.107)$$

7.3.2. Дополнения Шура*

В этом разделе мы дадим обзор некоторых полезных результатов, касающихся матриц с блочной структурой.

Теорема 7.3.1 (обращение клеточной матрицы). *Рассмотрим клеточную матрицу общего вида*

$$\mathbf{M} = \begin{pmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{pmatrix}, \quad (7.108)$$

где \mathbf{E} и \mathbf{H} предполагаются обратимыми. Имеем

$$\mathbf{M}^{-1} = \begin{pmatrix} (\mathbf{M}/\mathbf{H})^{-1} & -(\mathbf{M}/\mathbf{H})^{-1}\mathbf{F}\mathbf{H}^{-1} \\ -\mathbf{H}^{-1}\mathbf{G}(\mathbf{M}/\mathbf{H})^{-1} & \mathbf{H}^{-1} + \mathbf{H}^{-1}\mathbf{G}(\mathbf{M}/\mathbf{H})^{-1}\mathbf{F}\mathbf{H}^{-1} \end{pmatrix} \quad (7.109)$$

$$= \begin{pmatrix} \mathbf{E}^{-1} + \mathbf{E}^{-1}\mathbf{F}(\mathbf{M}/\mathbf{E})^{-1}\mathbf{G}\mathbf{E}^{-1} & -\mathbf{E}^{-1}\mathbf{F}(\mathbf{M}/\mathbf{E})^{-1} \\ -(\mathbf{M}/\mathbf{E})^{-1}\mathbf{G}\mathbf{E}^{-1} & (\mathbf{M}/\mathbf{E})^{-1} \end{pmatrix}, \quad (7.110)$$

где

$$\mathbf{M} = \mathbf{H} \triangleq \mathbf{E} - \mathbf{F}\mathbf{H}^{-1}\mathbf{G}; \quad (7.111)$$

$$\mathbf{M} = \mathbf{E} \triangleq \mathbf{H} - \mathbf{G}\mathbf{E}^{-1}\mathbf{F}. \quad (7.112)$$

Мы говорим, что \mathbf{M}/\mathbf{H} является **дополнением Шура** матрицы \mathbf{M} относительно \mathbf{H} , а \mathbf{M}/\mathbf{E} – **дополнением Шура** матрицы \mathbf{M} относительно \mathbf{E} . (Причины такой нотации будут объяснены в формуле (7.133).)

Формулы (7.109) и (7.110) называются **формулами обращения клеточных матриц**.

Доказательство. Если бы мы могли преобразовать \mathbf{M} к блочно-диагональному виду, то обратить ее было бы проще. Чтобы обнулить правый верхний блок \mathbf{M} , можно умножить его слева на матрицу, как показано ниже:

$$\begin{pmatrix} \mathbf{I} & -\mathbf{F}\mathbf{H}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{pmatrix} = \begin{pmatrix} \mathbf{E} - \mathbf{F}\mathbf{H}^{-1}\mathbf{G} & \mathbf{0} \\ \mathbf{G} & \mathbf{H} \end{pmatrix}. \quad (7.113)$$

Аналогично, чтобы обнулить левый нижний блок, можно умножить его справа на матрицу, как показано ниже:

$$\begin{pmatrix} \mathbf{E} - \mathbf{F}\mathbf{H}^{-1}\mathbf{G} & \mathbf{0} \\ \mathbf{G} & \mathbf{H} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{H}^{-1}\mathbf{G} & \mathbf{I} \end{pmatrix} = \begin{pmatrix} \mathbf{E} - \mathbf{F}\mathbf{H}^{-1}\mathbf{G} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} \end{pmatrix}. \quad (7.114)$$

Собирая все вместе, получаем:

$$\underbrace{\begin{pmatrix} \mathbf{I} & -\mathbf{F}\mathbf{H}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{pmatrix}}_{\mathbf{M}} \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{H}^{-1}\mathbf{G} & \mathbf{I} \end{pmatrix}}_{\mathbf{Z}} = \underbrace{\begin{pmatrix} \mathbf{E} - \mathbf{F}\mathbf{H}^{-1}\mathbf{G} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} \end{pmatrix}}_{\mathbf{W}}. \quad (7.115)$$

Теперь обратим обе части:

$$\mathbf{Z}^{-1}\mathbf{M}^{-1}\mathbf{X}^{-1} = \mathbf{W}^{-1}; \quad (7.116)$$

$$\mathbf{M}^{-1} = \mathbf{Z}\mathbf{W}^{-1}\mathbf{X}. \quad (7.117)$$

Подстановка определений дает

$$\begin{pmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{H}^{-1}\mathbf{G} & \mathbf{I} \end{pmatrix} \begin{pmatrix} (\mathbf{M}/\mathbf{H})^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{H}^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{I} & -\mathbf{F}\mathbf{H}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \quad (7.118)$$

$$= \begin{pmatrix} (\mathbf{M}/\mathbf{H})^{-1} & \mathbf{0} \\ -\mathbf{H}^{-1}\mathbf{G}(\mathbf{M}/\mathbf{H})^{-1} & \mathbf{H}^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{I} & -\mathbf{F}\mathbf{H}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \quad (7.119)$$

$$= \begin{pmatrix} (\mathbf{M}/\mathbf{H})^{-1} & -(\mathbf{M}/\mathbf{H})^{-1}\mathbf{F}\mathbf{H}^{-1} \\ -\mathbf{H}^{-1}\mathbf{G}(\mathbf{M}/\mathbf{H})^{-1} & \mathbf{H}^{-1} + \mathbf{H}^{-1}\mathbf{G}(\mathbf{M}/\mathbf{H})^{-1}\mathbf{F}\mathbf{H}^{-1} \end{pmatrix}. \quad (7.120)$$

Альтернативно можно было бы разложить матрицу \mathbf{M} в терминах \mathbf{E} и $\mathbf{M}/\mathbf{E} = (\mathbf{H} - \mathbf{G}\mathbf{E}^{-1}\mathbf{F})$, что дает

$$\begin{pmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{pmatrix} = \begin{pmatrix} \mathbf{E}^{-1} + \mathbf{E}^{-1}\mathbf{F}(\mathbf{M}/\mathbf{E})^{-1}\mathbf{G}\mathbf{E}^{-1} & -\mathbf{E}^{-1}\mathbf{F}(\mathbf{M}/\mathbf{E})^{-1} \\ -(\mathbf{M}/\mathbf{E})^{-1}\mathbf{G}\mathbf{E}^{-1} & (\mathbf{M}/\mathbf{E})^{-1} \end{pmatrix}. \quad (7.121)$$

7.3.3. Лемма об обращении матрицы*

Приравняв левый верхний блок первой матрицы в формуле (7.119) и левый верхний блок матрицы в формуле (7.121), получаем:

$$(\mathbf{M}/\mathbf{H})^{-1} = (\mathbf{E} - \mathbf{F}\mathbf{H}^{-1}\mathbf{G})^{-1} = \mathbf{E}^{-1} + \mathbf{E}^{-1}\mathbf{F}(\mathbf{H} - \mathbf{G}\mathbf{E}^{-1}\mathbf{F})^{-1}\mathbf{G}\mathbf{E}^{-1}. \quad (7.122)$$

Это утверждение называется **леммой об обращении матрицы** или **формулой Шермана–Моррисона–Вудбери**. Ее типичное приложение к ма-

шинному обучению выглядит следующим образом. Пусть \mathbf{X} – матрица $N \times D$, а Σ – диагональная матрица $N \times N$. Тогда имеем (подставляя $\mathbf{E} = \Sigma$, $\mathbf{F} = \mathbf{G}^T = \mathbf{X}$ и $\mathbf{H}^{-1} = -\mathbf{I}$) следующий результат:

$$(\Sigma + \mathbf{X}\mathbf{X}^T)^{-1} = \Sigma^{-1} - \Sigma^{-1}\mathbf{X}(\mathbf{I} + \mathbf{X}^T\Sigma^{-1}\mathbf{X})^{-1}\mathbf{X}^T\Sigma^{-1}. \quad (7.123)$$

Для вычисления левой части нужно время $O(N^3)$, а для вычисления правой части – время $O(D^3)$.

Еще одно приложение касается вычисления **модификации первого ранга** обратной матрицы. Пусть $\mathbf{E} = \mathbf{A}$, $\mathbf{F} = \mathbf{u}$, $\mathbf{G} = \mathbf{v}^T$, $H = -1$. Тогда имеем:

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{u}(-1 - \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u})^{-1}\mathbf{v}^T\mathbf{A}^{-1}, \quad (7.124)$$

$$= \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}. \quad (7.125)$$

Это тождество называется **формулой Шермана–Моррисона**.

7.3.4. Лемма об определителе матрицы*

Теперь мы воспользуемся приведенными выше результатами для эффективного вычисления определителя матрицы с блочной структурой.

Из формулы (7.115) имеем:

$$|\mathbf{X}||\mathbf{M}||\mathbf{Z}| = |\mathbf{W}| = |\mathbf{E} - \mathbf{F}\mathbf{H}^{-1}\mathbf{G}||\mathbf{H}|; \quad (7.126)$$

$$\left| \begin{pmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{pmatrix} \right| = |\mathbf{E} - \mathbf{F}\mathbf{H}^{-1}\mathbf{G}||\mathbf{H}|; \quad (7.127)$$

$$|\mathbf{M}| = |\mathbf{M}/\mathbf{H}||\mathbf{H}|; \quad (7.128)$$

$$|\mathbf{M}/\mathbf{H}| = \frac{|\mathbf{M}|}{|\mathbf{H}|}. \quad (7.129)$$

Как видим, \mathbf{M}/\mathbf{H} действует как оператор деления.

Далее имеем:

$$|\mathbf{M}| = |\mathbf{M}/\mathbf{H}||\mathbf{H}| = |\mathbf{M}/\mathbf{E}||\mathbf{E}|; \quad (7.130)$$

$$|\mathbf{M}/\mathbf{H}| = \frac{|\mathbf{M}/\mathbf{E}||\mathbf{E}|}{|\mathbf{H}|}; \quad (7.131)$$

$$|\mathbf{E} - \mathbf{F}\mathbf{H}^{-1}\mathbf{G}| = |\mathbf{H} - \mathbf{G}\mathbf{E}^{-1}\mathbf{F}||\mathbf{H}^{-1}||\mathbf{E}|. \quad (7.132)$$

Отсюда (полагая $\mathbf{E} = \mathbf{A}$, $\mathbf{F} = -\mathbf{u}$, $\mathbf{G} = \mathbf{v}^T$, $\mathbf{H} = 1$) получаем:

$$|\mathbf{A} + \mathbf{u}\mathbf{v}^T| = (1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u})|\mathbf{A}|. \quad (7.133)$$

Это тождество называется **леммой об определителе матрицы**.

7.3.5. Приложение: вывод условных распределений для многомерного гауссова распределения

Рассмотрим совместное гауссово распределение вида $p(\mathbf{x}_1, \mathbf{x}_2) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, где

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}. \quad (7.134)$$

В разделе 3.2.3 мы говорили, что

$$p(\mathbf{x}_1|\mathbf{x}_2) = \mathcal{N}(\mathbf{x}_1|\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2), \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}). \quad (7.135)$$

Сейчас мы выведем этот результат с помощью дополнений Шура.

Разложим совместное распределение $p(\mathbf{x}_1, \mathbf{x}_2)$ в произведение $p(\mathbf{x}_2)p(\mathbf{x}_1|\mathbf{x}_2)$ следующим образом:

$$p(\mathbf{x}_1|\mathbf{x}_2) \propto \exp \left\{ -\frac{1}{2} \begin{pmatrix} \mathbf{x}_1 - \boldsymbol{\mu}_1 \\ \mathbf{x}_2 - \boldsymbol{\mu}_2 \end{pmatrix}^T \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{x}_1 - \boldsymbol{\mu}_1 \\ \mathbf{x}_2 - \boldsymbol{\mu}_2 \end{pmatrix} \right\}. \quad (7.136)$$

После применения формулы (7.118) это выражение принимает вид:

$$p(\mathbf{x}_1|\mathbf{x}_2) \propto \exp \left\{ -\frac{1}{2} \begin{pmatrix} \mathbf{x}_1 - \boldsymbol{\mu}_1 \\ \mathbf{x}_2 - \boldsymbol{\mu}_2 \end{pmatrix}^T \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} & \mathbf{I} \end{pmatrix} \begin{pmatrix} (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{22}^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 - \boldsymbol{\mu}_1 \\ \mathbf{x}_2 - \boldsymbol{\mu}_2 \end{pmatrix} \right\} \quad (7.137)$$

$$\times \begin{pmatrix} \mathbf{I} & -\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 - \boldsymbol{\mu}_1 \\ \mathbf{x}_2 - \boldsymbol{\mu}_2 \end{pmatrix} \quad (7.138)$$

$$= \exp \left\{ -\frac{1}{2} (\mathbf{x}_1 - \boldsymbol{\mu}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2))^T (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} \right. \quad (7.139)$$

$$\left. (\mathbf{x}_1 - \boldsymbol{\mu}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2)) \right\} \times \exp \left\{ -\frac{1}{2} (\mathbf{x}_2 - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}_{22}^{-1} (\mathbf{x}_2 - \boldsymbol{\mu}_2) \right\}. \quad (7.140)$$

Это выражение имеет вид:

$$\exp(\text{квадратичная форма от } \mathbf{x}_1, \mathbf{x}_2) \times \exp(\text{квадратичная форма от } \mathbf{x}_2). \quad (7.141)$$

Таким образом, мы успешно разложили совместное распределение в произведение:

$$p(\mathbf{x}_1, \mathbf{x}_2) = p(\mathbf{x}_1|\mathbf{x}_2)p(\mathbf{x}_2) \quad (7.142)$$

$$= \mathcal{N}(\mathbf{x}_1|\boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2})\mathcal{N}(\mathbf{x}_2|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}), \quad (7.143)$$

где параметры условного распределения можно получить из приведенных выше формул:

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(\mathbf{x}_2 - \mu_2); \quad (7.144)$$

$$\Sigma_{1|2} = \Sigma/\Sigma_{22} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}. \quad (7.145)$$

Мы также можем воспользоваться тем фактом, что $|\mathbf{M}| = |\mathbf{M}/\mathbf{H}| |\mathbf{H}|$, и проверить, что нормировочные постоянные правильны:

$$(2\pi)^{(d_1+d_2)/2} |\Sigma|^{\frac{1}{2}} = (2\pi)^{(d_1+d_2)/2} (|\Sigma/\Sigma_{22}| |\Sigma_{22}|)^{\frac{1}{2}} \quad (7.146)$$

$$= (2\pi)^{d_1/2} |\Sigma/\Sigma_{22}|^{\frac{1}{2}} (2\pi)^{d_2/2} |\Sigma_{22}|^{\frac{1}{2}}, \quad (7.147)$$

где $d_1 = \dim(\mathbf{x}_1)$ и $d_2 = \dim(\mathbf{x}_2)$.

7.4. СПЕКТРАЛЬНОЕ РАЗЛОЖЕНИЕ

В этом разделе мы вкратце изложим стандартные факты, относящиеся к **спектральному разложению** квадратных (вещественных) матриц.

7.4.1. Основные сведения

Говорят, что $\lambda \in \mathbb{R}$ является **собственным значением** квадратной матрицы $\mathbf{A} \in \mathbb{R}^{n \times n}$, а $\mathbf{u} \in \mathbb{R}^n$ – соответствующим ему **собственным вектором**, если

$$\mathbf{A}\mathbf{u} = \lambda\mathbf{u}; \mathbf{u} \neq 0. \quad (7.148)$$

Интуитивно понятно, что означает это определение: умножение \mathbf{A} на вектор \mathbf{u} порождает вектор того же направления, но умноженный на коэффициент λ . Например, если \mathbf{A} – матрица поворота, то \mathbf{u} направлен вдоль оси вращения и $\lambda = 1$.

Отметим, что для любого собственного вектора $\mathbf{u} \in \mathbb{R}^n$ и скаляра $c \in \mathbb{R}$

$$\mathbf{A}(c\mathbf{u}) = c\mathbf{A}\mathbf{u} = c\lambda\mathbf{u} = \lambda(c\mathbf{u}). \quad (7.149)$$

Следовательно, $c\mathbf{u}$ также является собственным вектором. Поэтому из всех собственных векторов, ассоциированных с собственным значением λ , мы обычно выделяем нормированный, т. е. имеющий длину 1 (это тоже не до конца устраняет неопределенность, потому что нормированными являются оба вектора \mathbf{u} и $-\mathbf{u}$, но с этим мы готовы смириться).

Мы можем переписать уравнение выше и сказать, что (λ, \mathbf{x}) образуют пару (собственное значение, собственный вектор) \mathbf{A} , если

$$(\lambda\mathbf{I} - \mathbf{A})\mathbf{u} = 0, \mathbf{u} \neq 0. \quad (7.150)$$

Уравнение $(\lambda\mathbf{I} - \mathbf{A})\mathbf{u} = 0$ имеет ненулевое решение тогда и только тогда, когда ядро матрицы $(\lambda\mathbf{I} - \mathbf{A})$ непусто, т. е. эта матрица сингулярна или

$$\det(\lambda \mathbf{I} - \mathbf{A}) = 0. \quad (7.151)$$

Это так называемое **характеристическое уравнение** \mathbf{A} (см. упражнение 7.2) n решений этого уравнения (возможно, комплексных) дают n собственных значений λ_i и соответствующих им собственных векторов \mathbf{u}_i . Принято сортировать собственные векторы в порядке убывания абсолютной величины их собственных значений.

Собственные значения и собственные векторы обладают следующими свойствами:

- след матрицы равен сумме ее собственных значений:

$$\operatorname{tr}(\mathbf{A}) = \sum_{i=1}^n \lambda_i; \quad (7.152)$$

- определитель \mathbf{A} равен произведению ее собственных значений:

$$\det(\mathbf{A}) = \prod_{i=1}^n \lambda_i; \quad (7.153)$$

- ранг \mathbf{A} равен числу ненулевых собственных значений \mathbf{A} ;
- если \mathbf{A} не сингулярна, то $1/\lambda_i$ – собственное значение \mathbf{A}^{-1} , которому соответствует собственный вектор \mathbf{u}_i , т. е. $\mathbf{A}^{-1}\mathbf{u}_i = (1/\lambda_i)\mathbf{u}_i$;
- собственные значения диагональной или треугольной матрицы совпадают с ее диагональными элементами.

7.4.2. Диагонализация

Мы можем записать все уравнения собственных векторов одновременно в виде

$$\mathbf{A}\mathbf{U} = \mathbf{U}\mathbf{\Lambda}, \quad (7.154)$$

где столбцы $\mathbf{U} \in \mathbb{R}^{n \times n}$ – собственные векторы \mathbf{A} , а $\mathbf{\Lambda}$ – диагональная матрица, элементами которой являются собственные значения \mathbf{A} , т. е.

$$\mathbf{U} \in \mathbb{R}^{n \times n} = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & & | \end{bmatrix}, \quad \mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_n). \quad (7.155)$$

Если собственные векторы \mathbf{A} линейно независимы, то матрица \mathbf{U} обратима, так что

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1}. \quad (7.156)$$

Матрица, которую можно записать в таком виде, называется **диагонализируемой**.

7.4.3. Собственные значения и собственные векторы симметричных матриц

Можно показать, что если матрица \mathbf{A} вещественная и симметричная, то все ее собственные значения вещественны, а собственные векторы ортонормированы, т. е. $\mathbf{u}_i^T \mathbf{u}_j = 0$, если $i \neq j$, и $\mathbf{u}_i^T \mathbf{u}_i = 1$, где \mathbf{u}_i – собственные векторы. В матричной форме это записывается в виде $\mathbf{U}^T \mathbf{U} = \mathbf{U} \mathbf{U}^T = \mathbf{I}$, т. е. что \mathbf{U} – ортогональная матрица.

Поэтому мы можем представить \mathbf{A} в виде:

$$\mathbf{A} = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^T = \begin{pmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & & | \end{pmatrix} \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix} \begin{pmatrix} - & \mathbf{u}_1^T & - \\ - & \mathbf{u}_2^T & - \\ & \vdots & \\ - & \mathbf{u}_n^T & - \end{pmatrix} \quad (7.157)$$

$$= \lambda_1 \begin{pmatrix} | \\ \mathbf{u}_1 \\ | \end{pmatrix} \begin{pmatrix} - & \mathbf{u}_1^T & - \end{pmatrix} + \cdots + \lambda_n \begin{pmatrix} | \\ \mathbf{u}_n \\ | \end{pmatrix} \begin{pmatrix} - & \mathbf{u}_n^T & - \end{pmatrix} = \sum_{i=1}^n \lambda_i \mathbf{u}_i \mathbf{u}_i^T. \quad (7.158)$$

Следовательно, умножение на симметричную матрицу \mathbf{A} можно интерпретировать как умножение на матрицу поворота \mathbf{U}^T , матрицу масштабирования $\mathbf{\Lambda}$ и матрицу поворота в обратном направлении \mathbf{U} .

Диагонализированную матрицу легко обратить. Поскольку $\mathbf{A} = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^T$, где $\mathbf{U}^T = \mathbf{U}^{-1}$, имеем:

$$\mathbf{A}^{-1} = \mathbf{U} \mathbf{\Lambda}^{-1} \mathbf{U}^T = \sum_{i=1}^d \frac{1}{\lambda_i} \mathbf{u}_i \mathbf{u}_i^T. \quad (7.159)$$

Это соответствует повороту, отмене масштабирования и обратному повороту.

7.4.3.1. Проверка на положительную определенность

Свойство диагонализации можно использовать также для того, чтобы показать, что симметричная матрица положительно определена тогда и только тогда, когда все ее собственные значения положительны. Чтобы убедиться в этом, заметим, что

$$\mathbf{x}^T \mathbf{A} \mathbf{x} = \mathbf{x}^T \mathbf{U} \mathbf{\Lambda} \mathbf{U}^T \mathbf{x} = \mathbf{y}^T \mathbf{\Lambda} \mathbf{y} = \sum_{i=1}^n \lambda_i y_i^2, \quad (7.160)$$

где $\mathbf{y} = \mathbf{U}^T \mathbf{x}$. Поскольку y_i^2 всегда неотрицательно, знак этого выражения зависит только от коэффициентов λ_i . Если все $\lambda_i > 0$, то матрица является положительно определенной; если все $\lambda_i \geq 0$, то положительно полуопределенной. Аналогично, если все $\lambda_i < 0$ или $\lambda_i \leq 0$, то \mathbf{A} является соответственно отрицательно определенной или полуопределенной. Наконец, если \mathbf{A} имеет

как положительные, так и отрицательные собственные значения, то она неопределенная.

7.4.4. Геометрия квадратичных форм

Квадратичной формой называется функция, которую можно записать в виде

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x}, \quad (7.161)$$

где $\mathbf{x} \in \mathbb{R}^n$, а \mathbf{A} – положительно определенная симметричная матрица размера $n \times n$. Пусть $\mathbf{A} = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^T$ – диагонализация \mathbf{A} (см. раздел 7.4.3). Тогда

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} = \mathbf{x}^T \mathbf{U} \mathbf{\Lambda} \mathbf{U}^T \mathbf{x} = \mathbf{y}^T \mathbf{\Lambda} \mathbf{y} = \sum_{i=1}^n \lambda_i y_i^2, \quad (7.162)$$

где $\mathbf{y}_i = \mathbf{x}^T \mathbf{u}_i$ и $\lambda_i > 0$ (так как \mathbf{A} положительно определенная). Линии уровня $f(\mathbf{x})$ являются гиперэллипсоидами. Например, на двумерной плоскости имеем

$$\lambda_1 y_1^2 + \lambda_2 y_2^2 = r \quad (7.163)$$

– уравнение эллипса (см. иллюстрацию на рис. 7.6). Собственные векторы определяют ориентацию эллипса, а собственные значения – его вытянутость.

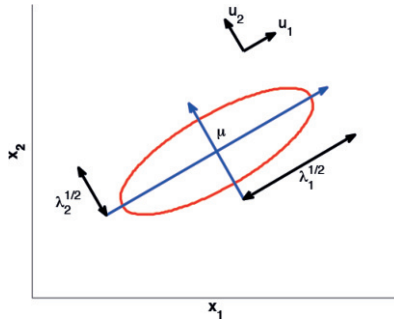


Рис. 7.6 ❖ Линии уровня квадратичной формы $(\mathbf{x} - \boldsymbol{\mu})^T \mathbf{A} (\mathbf{x} - \boldsymbol{\mu})$ на плоскости. Большая и малая оси эллипса определены первыми двумя собственными векторами \mathbf{A} , \mathbf{u}_1 и \mathbf{u}_2 . На основе рис. 2.7 из работы [Bis06]. Построено программой по адресу figures.problml.ai/book1/7.6

7.4.5. Стандартизация и отбеливание данных

Пусть имеется набор данных $\mathbf{X} \in \mathbb{R}^{N \times D}$. Часто данные подвергают предварительной обработке, так чтобы в каждом столбце было нулевое среднее и единичная дисперсия. Эта процедура называется **стандартизацией** данных и будет обсуждаться в разделе 10.2.8. Хотя стандартизация приводит дисперсию к 1, она не устраняет корреляцию между столбцами. Для этого

данные необходимо **отбелить**. Обозначим ковариационную матрицу $\Sigma = (1/N)\mathbf{X}^T\mathbf{X}$ и пусть $\Sigma = \mathbf{E}\mathbf{D}\mathbf{E}^T$ – ее диагонализация. Иначе говоря, пусть $[\mathbf{U}, \mathbf{S}, \mathbf{V}]$ – сингулярное разложение \mathbf{X} (так что $\mathbf{E} = \mathbf{V}$, а $\mathbf{D} = \mathbf{S}^2$, см. раздел 20.1.3.3). Теперь определим

$$\mathbf{W}_{pca} = \mathbf{D}^{-\frac{1}{2}}\mathbf{E}^T. \quad (7.164)$$

Это называется матрицей **PCA-отбеливания**. (Метод PCA мы обсудим в разделе 20.1.) Пусть $\mathbf{y} = \mathbf{W}_{pca}\mathbf{x}$ – преобразованный вектор. Мы можем проверить, что его ковариация отбелена, следующим образом:

$$\text{Cov}[\mathbf{y}] = \mathbf{W}\mathbf{E}[\mathbf{x}\mathbf{x}^T]\mathbf{W}^T = \mathbf{W}\Sigma\mathbf{W} = (\mathbf{D}^{-\frac{1}{2}}\mathbf{E}^T)(\mathbf{E}\mathbf{D}\mathbf{E}^T)(\mathbf{E}\mathbf{D}^{-\frac{1}{2}}) = \mathbf{I}. \quad (7.165)$$

Матрица отбеливания определена не однозначно, поскольку любой поворот $\mathbf{W} = \mathbf{R}\mathbf{W}_{pca}$ сохраняет свойство отбеливания, т. е. $\mathbf{W}^T\mathbf{W} = \Sigma^{-1}$. Например, взяв $\mathbf{R} = \mathbf{E}$, получим:

$$\mathbf{W}_{zca} = \mathbf{E}\mathbf{D}^{-\frac{1}{2}}\mathbf{E}^T = \Sigma^{-\frac{1}{2}} = \mathbf{V}\mathbf{S}^{-1}\mathbf{V}^T. \quad (7.166)$$

Это так называемое **отбеливание Махаланобиса**, или **ZCA** (ZCA означает «zero-phase component analysis» – метод нуль-фазовых компонент, он был впервые описан в работе [BS97]). Преимущество ZCA-отбеливания перед PCA-отбеливанием заключается в том, что получающиеся в результате преобразования данные максимально близки к оригинальным (в смысле наименьших квадратов) [Aho17]. Это показано на рис. 7.7. Применительно к изображениям векторы данных после преобразования ZCA продолжают выглядеть как изображения. Это полезно, когда метод применяется как часть системы глубокого обучения.

7.4.6. Степенной метод

Теперь мы опишем простой итеративный метод вычисления собственного вектора, соответствующего наибольшему собственному значению вещественной симметричной матрицы, – **степенной метод**. Он может быть полезен, когда матрица очень большая, но разреженная. Например, он используется в алгоритме Google **PageRank** для вычисления стационарного распределения матрицы переходов во Всемирной паутине (ее размер приблизительно 3×3 млрд!). В разделе 7.4.7 мы увидим, как использовать этот метод для вычисления последующих собственных векторов и собственных значений.

Пусть \mathbf{A} – матрица с ортонормированными собственными векторами \mathbf{u}_i и собственными значениями $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_m| \geq 0$, так что $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$. Обозначим $\mathbf{v}_{(0)}$ произвольный вектор в образе \mathbf{A} , т. е. $\mathbf{A}\mathbf{x} = \mathbf{v}_{(0)}$ для некоторого \mathbf{x} . Тогда можно записать $\mathbf{v}_{(0)}$ в виде

$$\mathbf{v}_0 = \mathbf{U}(\mathbf{\Lambda}\mathbf{U}^T\mathbf{x}) = a_1\mathbf{u}_1 + \dots + a_m\mathbf{u}_m \quad (7.167)$$

для некоторых постоянных a_i . Теперь можно повторно умножать \mathbf{v} на \mathbf{A} , перенормируя на каждом шаге:

$$\mathbf{v}_t \propto \mathbf{A}\mathbf{v}_{t-1}. \quad (7.168)$$

(Перенормирование нужно для обеспечения численной устойчивости.)

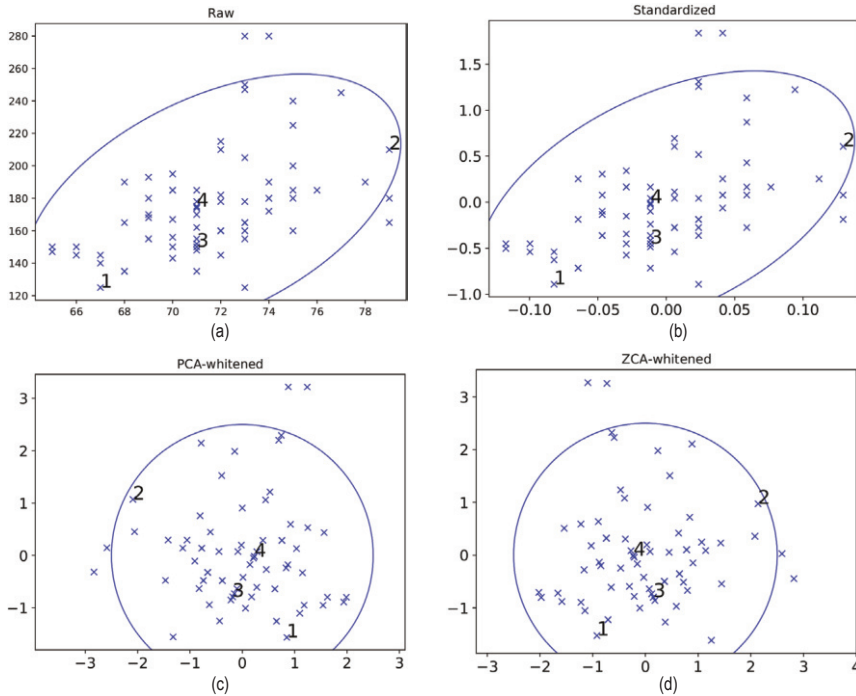


Рис. 7.7 ❖ (а) Данные о росте и весе. (б) Стандартизованные. (с) Отбеленные методом PCA. (д) Отбеленные методом ZCA. Числа относятся к первым 4 точкам, но всего точек 73. Построено программой по адресу figures.problm.ai/book1/7.7

Поскольку \mathbf{v}_t – кратное $\mathbf{A}^t \mathbf{v}_0$, имеем:

$$\mathbf{v}_t \propto a_1 \lambda_1^t \mathbf{u}_1 + a_2 \lambda_2^t \mathbf{u}_2 + \dots + a_m \lambda_m^t \mathbf{u}_m \quad (7.169)$$

$$= \lambda_1^t (a_1 \mathbf{u}_1 + a_1 (\lambda_2/\lambda_1)^t \mathbf{u}_2 + \dots + a_m (\lambda_m/\lambda_1)^t \mathbf{u}_m) \quad (7.170)$$

$$\rightarrow \lambda_1^t a_1 \mathbf{u}_1, \quad (7.171)$$

так как $|\lambda_k|/|\lambda_1| < 1$ для $k > 1$ (в предположении, что собственные значения отсортированы в порядке убывания). Таким образом, мы видим, что последовательность сходится к \mathbf{u}_1 , хотя и не очень быстро (ошибка уменьшается приблизительно в $|\lambda_2|/|\lambda_1|$ раз на каждой итерации). Единственное требование состоит в том, чтобы начальное значение удовлетворяло условию $\mathbf{v}_0^\top \mathbf{u}_1 \neq 0$, что для случайно выбранного \mathbf{v}_0 с большой вероятностью верно.

Теперь обсудим, как вычислить соответствующее собственное значение λ_1 . Определим **отношение Рэля**:

$$R(\mathbf{A}, \mathbf{x}) \triangleq \frac{\mathbf{x}^\top \mathbf{A} \mathbf{x}}{\mathbf{x}^\top \mathbf{x}}. \quad (7.172)$$

Отсюда

$$R(\mathbf{A}, \mathbf{u}_i) \triangleq \frac{\mathbf{u}_i^\top \mathbf{A} \mathbf{u}_i}{\mathbf{u}_i^\top \mathbf{u}_i} = \frac{\lambda_i \mathbf{u}_i^\top \mathbf{u}_i}{\mathbf{u}_i^\top \mathbf{u}_i} = \lambda_i. \quad (7.173)$$

Таким образом, мы легко можем вычислить λ_1 , зная \mathbf{u}_1 и \mathbf{A} (см. демонстрационную программу `code.problm.ai/book1/power_method_demo`).

7.4.7. Понижение порядка

Предположим, что мы вычислили первый собственный вектор \mathbf{u}_1 и его собственное значение λ_1 степенным методом. Теперь опишем, как вычислить последующие собственные векторы и значения. Поскольку собственные векторы ортонормированы, а собственные значения вещественны, мы можем исключить из матрицы компоненту \mathbf{u}_1 следующим образом:

$$\mathbf{A}^{(2)} = (\mathbf{I} - \mathbf{u}_1 \mathbf{u}_1^\top) \mathbf{A}^{(1)} = \mathbf{A}^{(1)} - \mathbf{u}_1 \mathbf{u}_1^\top \mathbf{A}^{(1)} = \mathbf{A}^{(1)} - \lambda_1 \mathbf{u}_1 \mathbf{u}_1^\top. \quad (7.174)$$

Это называется **понижением порядка** матрицы. Затем можно применить степенной метод к $\mathbf{A}^{(2)}$ и найти второе по величине собственное значение и соответствующий ему собственный вектор в подпространстве, ортогональном \mathbf{u}_1 .

В разделе 20.1.2 мы покажем, что оптимальная оценка $\hat{\mathbf{W}}$ для модели PCA (описана в разделе 20.1) дается первыми K собственными векторами эмпирической ковариационной матрицы. Поэтому для реализации PCA можно использовать метод понижения порядка. Его также можно модифицировать для реализации разреженного PCA [Mac09].

7.4.8. Собственные векторы оптимизируют квадратичные формы

Мы можем воспользоваться исчислением матриц для решения задачи оптимизации способом, который непосредственно приводит к анализу собственных значений и векторов. Рассмотрим следующую задачу условной оптимизации с ограничением в виде равенства:

$$\max_{\mathbf{x} \in \mathbb{R}^n} \mathbf{x}^\top \mathbf{A} \mathbf{x} \quad \text{при условии } \|\mathbf{x}\|_2^2 \quad (7.175)$$

для симметричной матрицы $\mathbf{A} \in \mathbb{S}^n$. Стандартный способ решения задач оптимизации с ограничениями в виде равенств – построить лагранжиан, т. е. целевую функцию, включающую ограничения (см. раздел 8.5.1). В данном случае лагранжиан имеет вид

$$\mathcal{L}(\mathbf{x}, \lambda) = \mathbf{x}^T \mathbf{A} \mathbf{x} + \lambda(1 - \mathbf{x}^T \mathbf{x}), \quad (7.176)$$

где λ называется множителем Лагранжа, ассоциированным с ограничением в виде равенства. Можно показать, что, для того чтобы \mathbf{x}^* была точкой оптимума, градиент лагранжиана в точке \mathbf{x}^* должен быть равен нулю (это не единственное, но обязательное условие). Таким образом,

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda) = 2\mathbf{A}^T \mathbf{x} - 2\lambda \mathbf{x} = 0. \quad (7.177)$$

Заметим, что это простое линейное уравнение вида $\mathbf{A} \mathbf{x} = \lambda \mathbf{x}$. Это показывает, что единственные точки, которые могут доставлять максимум (или минимум) форме $\mathbf{x}^T \mathbf{A} \mathbf{x}$, в предположении, что $\mathbf{x}^T \mathbf{x} = 1$, – собственные векторы матрицы \mathbf{A} .

7.5. СИНГУЛЯРНОЕ РАЗЛОЖЕНИЕ (SVD)

Теперь обсудим сингулярное разложение (SVD), которое обобщает спектральное разложение на прямоугольные матрицы.

7.5.1. Основные сведения

Любую вещественную матрицу \mathbf{A} размера $m \times n$ можно разложить в произведение

$$\mathbf{A} = \mathbf{U} \mathbf{S} \mathbf{V}^T = \sigma_1 \begin{pmatrix} | \\ \mathbf{u}_1 \\ | \end{pmatrix} \begin{pmatrix} - & \mathbf{v}_1^T & - \end{pmatrix} + \dots + \sigma_r \begin{pmatrix} | \\ \mathbf{u}_r \\ | \end{pmatrix} \begin{pmatrix} - & \mathbf{v}_r^T & - \end{pmatrix}, \quad (7.178)$$

где \mathbf{U} – матрица $m \times m$ с ортонормированными столбцами (т. е. $\mathbf{U}^T \mathbf{U} = \mathbf{I}_m$), \mathbf{V} – матрица $n \times n$ с ортонормированными строками и столбцами (т. е. $\mathbf{V}^T \mathbf{V} = \mathbf{V} \mathbf{V}^T = \mathbf{I}_n$), а \mathbf{S} – матрица $m \times n$, содержащая $r = \min(m, n)$ **сингулярных чисел** $\sigma_i \geq 0$ на главной диагонали и нули во всех остальных позициях. Столбцы \mathbf{U} называются левыми **сингулярными векторами**, а столбцы \mathbf{V} – правыми сингулярными векторами. Пример показан на рис. 7.8.

Из рис. 7.8а ясно, что если $m > n$, то существует не более n сингулярных чисел, поэтому последние $m - n$ столбцов \mathbf{U} несущественны (т. к. они будут умножены на 0). В **экономном SVD**, называемой также **тощим SVD**, эти ненужные элементы не вычисляются. Иными словами, если записать матрицу \mathbf{U} в виде $\mathbf{U} = [\mathbf{U}_1, \mathbf{U}_2]$, то вычислять нужно только \mathbf{U}_1 . На рис. 7.8b показан противоположный случай, когда $m < n$; мы представили \mathbf{V} в виде $[\mathbf{V}_1, \mathbf{V}_2]$ и вычисляем только \mathbf{V}_1 .

Стоимость вычисления SVD имеет порядок $O(\min(mn^2, m^2n))$. Подробнее об этом методе можно прочитать в стандартных учебниках линейной алгебры.

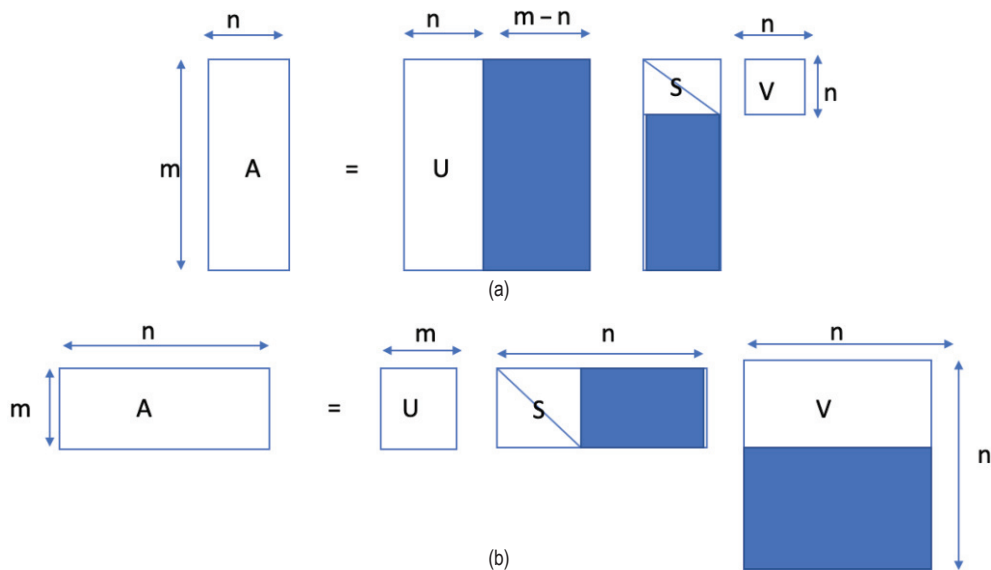


Рис. 7.8 ❖ Сингулярное разложение матрицы, $A = USV^T$.

Закрашенные части каждой матрицы в экономной версии не вычисляются.

(a) Высокая узкая матрица. (b) Низкая широкая матрица

7.5.2. Связь между сингулярным и спектральным разложением

Если A – вещественная, симметричная, положительно определенная матрица, то сингулярные числа равны собственным значениям, а левые и правые сингулярные векторы совпадают с собственными векторами (с точностью до знака):

$$A = USV^T = USU^T = USU^{-1}. \quad (7.179)$$

Заметим, однако, что NumPy всегда возвращает сингулярные числа в порядке убывания, тогда как собственные значения могут быть не отсортированы.

В общем случае для любой вещественной матрицы A если $A = USV^T$, то имеем:

$$A^T A = VS^T U^T USV^T = V(S^T S)V^T. \quad (7.180)$$

Отсюда

$$(A^T A)V = VD_n, \quad (7.181)$$

поэтому собственные векторы $A^T A$ совпадают с V , правыми сингулярными векторами A , а собственные значения $A^T A$ образуют диагональную матрицу $D_n = S^T S$ размера $n \times n$, содержащую квадраты сингулярных чисел. Аналогично

$$\mathbf{A}\mathbf{A}^T = \mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{V}\mathbf{S}^T\mathbf{U}^T = \mathbf{U}(\mathbf{S}\mathbf{S}^T)\mathbf{U}^T, \quad (7.182)$$

$$(\mathbf{A}\mathbf{A}^T)\mathbf{U} = \mathbf{U}\mathbf{D}_m, \quad (7.183)$$

так что собственные векторы $\mathbf{A}\mathbf{A}^T$ совпадают с \mathbf{U} , левыми сингулярными векторами \mathbf{A} , а собственные значения $\mathbf{A}\mathbf{A}^T$ образуют диагональную матрицу $\mathbf{D}_m = \mathbf{S}\mathbf{S}^T$ размера $m \times m$, содержащую квадраты сингулярных чисел. Резюмируем:

$$\mathbf{U} = \text{evec}(\mathbf{A}\mathbf{A}^T), \mathbf{V} = \text{evec}(\mathbf{A}^T\mathbf{A}), \mathbf{D}_m = \text{eval}(\mathbf{A}\mathbf{A}^T), \mathbf{D}_n = \text{eval}(\mathbf{A}^T\mathbf{A}). \quad (7.184)$$

Если мы просто используем вычисленные (ненулевые) части в экономном SVD, то можем определить:

$$\mathbf{D} = \mathbf{S}^2 = \mathbf{S}^T\mathbf{S} = \mathbf{S}\mathbf{S}^T. \quad (7.185)$$

Заметим также, что спектральное разложение существует не всегда, даже для квадратных матриц \mathbf{A} , тогда как сингулярное разложение имеется всегда.

7.5.3. Псевдообратная матрица

Псевдообратная матрица Мура–Пенроуза, обозначаемая \mathbf{A}^\dagger , – это однозначно определенная матрица, обладающая следующими четырьмя свойствами:

$$\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}, \mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger, (\mathbf{A}\mathbf{A}^\dagger)^T = \mathbf{A}\mathbf{A}^\dagger, (\mathbf{A}^\dagger\mathbf{A})^T = \mathbf{A}^\dagger\mathbf{A}. \quad (7.186)$$

Если \mathbf{A} квадратная и несингулярная, то $\mathbf{A}^\dagger = \mathbf{A}^{-1}$.

Если $m > n$ (высокая узкая матрица) и столбцы \mathbf{A} линейно независимы (т. е. \mathbf{A} имеет полный ранг), то

$$\mathbf{A}^\dagger = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \quad (7.187)$$

– то же выражение, которое возникает в нормальных уравнениях (см. раздел 11.2.2.1). В этом случае матрица \mathbf{A}^\dagger – левая обратная для \mathbf{A} , потому что

$$\mathbf{A}^\dagger\mathbf{A} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{A} = \mathbf{I}, \quad (7.188)$$

но правой обратной она не является, потому что матрица

$$\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \quad (7.189)$$

имеет только ранг n и, значит, не может быть единичной матрицей размера $m \times m$.

Если $m < n$ (низкая широкая матрица) и строки \mathbf{A} линейно независимы (т. е. \mathbf{A} имеет полный ранг), то псевдообратная матрица имеет вид:

$$\mathbf{A}^\dagger = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}. \quad (7.190)$$

В этом случае \mathbf{A}^\dagger является правой обратной для \mathbf{A} .

Мы можем вычислить псевдообратную матрицу, воспользовавшись SVD-разложением $\mathbf{A} = \mathbf{USV}^T$. В частности, можно показать, что

$$\mathbf{A}^\dagger = \mathbf{V}[\text{diag}(1/\sigma_1, \dots, 1/\sigma_r, 0, \dots, 0)]\mathbf{U}^T, \quad (7.191)$$

где r – ранг матрицы. Таким образом, \mathbf{A}^\dagger играет роль обратной матрицы для неквадратных матриц:

$$\mathbf{A}^\dagger = \mathbf{A}^{-1} = (\mathbf{USV}^T)^{-1} = \mathbf{VS}^{-1}\mathbf{U}^T, \quad (7.192)$$

где по определению $\mathbf{S}^{-1} = \text{diag}(\sigma_1^{-1}, \dots, \sigma_r^{-1}, 0, \dots, 0)$.

7.5.4. SVD и образ и ядро матрицы*

В этом разделе мы покажем, что левые и правые сингулярные векторы образуют ортонормированный базис образа и ядра соответственно.

Из формулы (7.178) имеем

$$\mathbf{Ax} = \sum_{j:\sigma_j>0} \sigma_j(\mathbf{v}_j^T \mathbf{x}) \mathbf{u}_j = \sum_{j=1}^r \sigma_j(\mathbf{v}_j^T \mathbf{x}) \mathbf{u}_j, \quad (7.193)$$

где r – ранг \mathbf{A} . Следовательно, любой вектор \mathbf{Ax} можно записать в виде линейной комбинации левых сингулярных векторов $\mathbf{u}_1, \dots, \mathbf{u}_r$, так что образ \mathbf{A} имеет вид

$$\text{range}(\mathbf{A}) = \text{span}(\{\mathbf{u}_j : \sigma_j > 0\}) \quad (7.194)$$

и размерность r .

Чтобы найти базис ядра, определим второй вектор $\mathbf{y} \in \mathbb{R}^n$, являющийся линейной комбинацией только правых сингулярных векторов с ненулевыми сингулярными числами:

$$\mathbf{y} = \sum_{j:\sigma_j>0} c_j \mathbf{v}_j = \sum_{j=r+1}^n c_j \mathbf{v}_j. \quad (7.195)$$

Поскольку \mathbf{v}_j ортонормированы, имеем:

$$\mathbf{Ay} = \mathbf{U} \begin{pmatrix} \sigma_1 \mathbf{v}_1^T \mathbf{y} \\ \vdots \\ \sigma_r \mathbf{v}_r^T \mathbf{y} \\ \sigma_{r+1} \mathbf{v}_{r+1}^T \mathbf{y} \\ \vdots \\ \sigma_n \mathbf{v}_n^T \mathbf{y} \end{pmatrix} = \mathbf{U} \begin{pmatrix} \sigma_1 0 \\ \vdots \\ \sigma_r 0 \\ 0 \mathbf{v}_{r+1}^T \mathbf{y} \\ \vdots \\ 0 \mathbf{v}_n^T \mathbf{y} \end{pmatrix} = \mathbf{U} \mathbf{0} = \mathbf{0}. \quad (7.196)$$

Поэтому правые сингулярные векторы образуют ортонормированный базис ядра:

$$\text{nullspace}(\mathbf{A}) = \text{span}(\{\mathbf{v}_j : \sigma_j = 0\}) \quad (7.197)$$

размерности $n - r$. Мы видим, что

$$\dim(\text{range}(\mathbf{A})) + \dim(\text{nullspace}(\mathbf{A})) = r + (n - r) = n. \quad (7.198)$$

Словами это часто выражают так:

$$\text{ранг} + \text{дефект} = n. \quad (7.199)$$

Это утверждение называется **теоремой о ранге и дефекте**. Из нее следует, что ранг матрицы равен количеству ненулевых сингулярных чисел.

7.5.5. Усеченное сингулярное разложение

Пусть $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ – сингулярное разложение \mathbf{A} и $\hat{\mathbf{A}}_K = \mathbf{U}_K\mathbf{S}_K\mathbf{V}_K^T$, где используются первые K столбцов \mathbf{U} и \mathbf{V} . Можно показать, что это оптимальная аппроксимация ранга K в том смысле, что она доставляет минимум величине $\|\mathbf{A} - \hat{\mathbf{A}}_K\|_F^2$.

Если $K = r = \text{rank}(\mathbf{A})$, то это разложение не вносит никакой ошибки. Но при $K < r$, когда сингулярное разложение **усеченное**, какая-то ошибка возникает. Если сингулярные значения быстро убывают, что характерно для естественных данных (см., например, рис. 7.10), то ошибка будет небольшой. Общее число параметров, необходимых для представления матрицы $N \times D$ с помощью аппроксимации ранга K , равно:

$$NK + KD + K = K(N + D + 1). \quad (7.200)$$

В качестве примера рассмотрим изображение с разрешением 200×320 пикселей на рис. 7.9 (слева вверху). Для его описания нужно 64 000 чисел. Мы видим, что аппроксимация ранга 20, описываемая всего $(200 + 320 + 1) \times 20 = 10\,420$ числами, дает хорошее качество.



Рис. 7.9 ❖ Аппроксимации изображения с низким рангом. Слева вверху: оригинальное изображение размера 200×320 , имеющее ранг 200. Последующие изображения имеют ранг 2, 5 и 20. Построено программой по адресу figures.probl.ai/book1/7.9

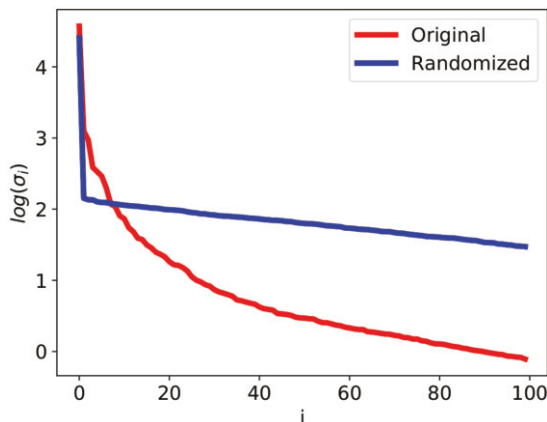


Рис. 7.10 ❖ Первые 100 логарифмов сингулярных чисел для изображения клоуна (красная линия) и для матрицы данных, полученной случайной перетасовкой пикселей (синяя линия). Построено программой по адресу figures.probl.ai/book1/7.10. На основе рис. 14.24 из работы [HTF09]

Можно показать, что ошибка этой аппроксимации равна

$$\|\mathbf{A} - \hat{\mathbf{A}}\|_F = \sum_{k=K+1}^r \sigma_k, \quad (7.201)$$

где σ_k – k -е сингулярное значение \mathbf{A} . Кроме того, можно показать, что SVD – лучшая аппроксимация матрицы, имеющая ранг K (в смысле минимизации нормы Фробениуса).

7.6. ДРУГИЕ МАТРИЧНЫЕ РАЗЛОЖЕНИЯ*

В этом разделе мы кратко опишем еще некоторые полезные матричные разложения.

7.6.1. LU-разложение

Мы можем разложить любую квадратную матрицу \mathbf{A} в произведение нижнетреугольной матрицы \mathbf{L} и верхнетреугольной матрицы \mathbf{U} . Например,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}. \quad (7.202)$$

В общем случае перед созданием такого разложения может потребоваться переставить элементы матрицы. Для демонстрации предположим, что $a_{11} = 0$. Так как $a_{11} = l_{11}u_{11}$, то либо l_{11} , либо u_{11} (либо оба вместе) должны быть

равны 0, но тогда \mathbf{L} или \mathbf{U} была бы сингулярной. Чтобы избежать этого, на первом шаге алгоритма можно просто переставить строки, так чтобы первый элемент стал не равен нулю. Это повторяется и на последующих шагах. Обозначим этот процесс

$$\mathbf{PA} = \mathbf{LU}, \quad (7.203)$$

где \mathbf{P} – матрица перестановки, т. е. квадратная бинарная матрица, в которой $P_{ij} = 1$, если строка j переставляется со строкой i . Это называется **выбором ведущего элемента**.

7.6.2. QR-разложение

Пусть имеется матрица $\mathbf{A} \in \mathbb{R}^{m \times n}$, представляющая множество линейно независимых базисных векторов (так что $m \geq n$), и мы хотим найти последовательность ортонормированных векторов $\mathbf{q}_1, \mathbf{q}_2, \dots$, на которую натянуты подпространства $\text{span}(\mathbf{a}_1)$, $\text{span}(\mathbf{a}_1, \mathbf{a}_2)$ и т. д. Иначе говоря, мы хотим найти векторы \mathbf{q}_j и коэффициенты r_{ij} такие, что

$$\begin{pmatrix} | & | & & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_n \\ | & | & & | \end{pmatrix} = \begin{pmatrix} | & | & & | \\ \mathbf{q}_1 & \mathbf{q}_2 & \dots & \mathbf{q}_n \\ | & | & & | \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ & r_{22} & \dots & r_{2n} \\ & & \ddots & \\ & & & r_{nn} \end{pmatrix}. \quad (7.204)$$

Это можно записать в виде:

$$\mathbf{a}_1 = r_{11}\mathbf{q}_1, \quad (7.205)$$

$$\mathbf{a}_2 = r_{12}\mathbf{q}_1 + r_{22}\mathbf{q}_2, \quad (7.206)$$

\vdots

$$\mathbf{a}_n = r_{1n}\mathbf{q}_1 + \dots + r_{nn}\mathbf{q}_n, \quad (7.207)$$

т. е. на \mathbf{q}_1 натянуто пространство \mathbf{a}_1 , на \mathbf{q}_1 и \mathbf{q}_2 – пространство $\{\mathbf{a}_1, \mathbf{a}_2\}$ и т. д.

В матричной нотации имеем

$$\mathbf{A} = \hat{\mathbf{Q}}\hat{\mathbf{R}}, \quad (7.208)$$

где $\hat{\mathbf{Q}}$ – матрица размера $m \times n$ с ортонормированными столбцами, а $\hat{\mathbf{R}}$ – верхнетреугольная матрица размера $n \times n$. Это называется **редуцированным**, или **экономным, QR-разложением** \mathbf{A} ; см. рис. 7.11.

При полном QR-разложении в конец $\hat{\mathbf{Q}}$ дописывается еще $m - n$ ортонормированных столбцов, так что получается квадратная ортогональная матрица \mathbf{Q} , удовлетворяющая равенствам $\mathbf{Q}\mathbf{Q}^T = \mathbf{Q}^T\mathbf{Q} = \mathbf{I}$. Кроме того, мы добавляем составленные из нулей строки в конец $\hat{\mathbf{R}}$, так что получается матрица размера $m \times n$, по-прежнему верхнетреугольная, которая обозначается \mathbf{R} : см. рис. 7.11. Нулевые элементы \mathbf{R} сводят на нет новые столбцы \mathbf{Q} , поэтому результат получается такой же, как при вычислении $\hat{\mathbf{Q}}\hat{\mathbf{R}}$.

QR-разложение часто используется при решении систем линейных уравнений (см. раздел 11.2.2.3).

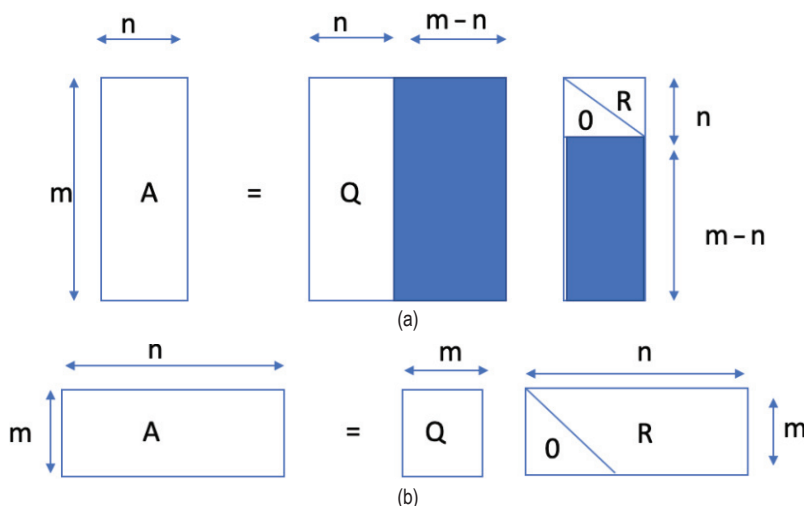


Рис. 7.11 ❖ QR-разложение, $A = QR$, где $Q^T Q = I$ и R – верхнетреугольная матрица. (a) Высокая узкая матрица. Закрашенные части в экономной версии не вычисляются, потому что не нужны. (b) Низкая широкая матрица

7.6.3. Разложение Холецки

Любую симметричную положительно определенную матрицу можно разложить в произведение $A = R^T R$, где R – верхнетреугольная матрица с вещественными положительными элементами на диагонали. (Это разложение можно записать также в виде $A = LL^T$, где $L = R^T$ – нижнетреугольная матрица.) Это **разложение Холецки**, или **квадратный корень из матрицы**. А NumPy оно реализовано в модуле `np.linalg.cholesky`. Вычислительная сложность этой операции составляет $O(V^3)$, где V – число переменных, но для разреженных матриц может быть и меньше. Ниже описано несколько применений этого разложения.

7.6.3.1. Приложение: выборка из многомерного гауссова распределения

Разложение Холецки ковариационной матрицы можно использовать для выборки из многомерного гауссова распределения. Пусть $y \sim \mathcal{N}(\mu, \Sigma)$ и $\Sigma = LL^T$. Сначала выбираем $x \sim \mathcal{N}(0, I)$, это легко, потому что нужно лишь произвести выборку из d раздельных одномерных гауссовых распределений. Затем положим $y = Lx + \mu$. Это допустимо, потому что

$$\text{Cov}[y] = L\text{Cov}[x]L^T = LIL^T = \Sigma. \quad (7.209)$$

Код приведен по адресу code.problml.ai/book1/cholesky_demo.

7.7. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ*

Важное применение линейной алгебры – изучение систем линейных уравнений. Например, рассмотрим систему из трех уравнений:

$$3x_1 + 2x_2 - x_3 = 1, \quad (7.210)$$

$$2x_1 - 2x_2 + 4x_3 = -2, \quad (7.211)$$

$$-x_1 + \frac{1}{2}x_2 - x_3 = 0. \quad (7.212)$$

Ее можно представить в матричной форме следующим образом:

$$\mathbf{Ax} = \mathbf{b}, \quad (7.213)$$

где

$$\mathbf{A} = \begin{pmatrix} 3 & 2 & -1 \\ 2 & -2 & 4 \\ -1 & \frac{1}{2} & -1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 1 \\ -2 \\ 0 \end{pmatrix}. \quad (7.214)$$

Решением является вектор $\mathbf{x} = [1, -2, -2]$.

В общем случае, когда имеется m уравнений и n неизвестных, \mathbf{A} будет матрицей размера $m \times n$, а \mathbf{b} – вектором $m \times 1$. Если $m = n$ (и матрица \mathbf{A} полного ранга), существует единственное решение. Если $m < n$, то система **недоопределенная**, поэтому решение не единственно. Если $m > n$, то система **переопределенная**, потому что ограничений больше, чем неизвестных, и не все прямые пересекаются в одной точке (см. иллюстрацию на рис. 7.12). Ниже мы обсудим, как вычислять решения в каждом из этих случаев.

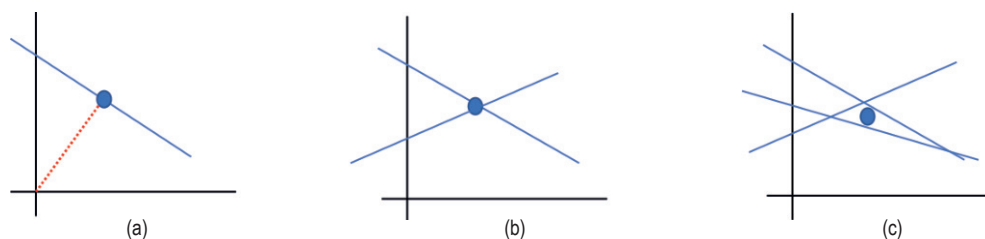


Рис. 7.12 ❖ Решение системы m линейных уравнений с $n = 2$ переменными. (а) $m = 1 < n$, т. е. система недоопределенная. Решение с минимальной нормой показано синей точкой. (Красный пунктирный отрезок ортогонален синей прямой, а его длина равна расстоянию до начала координат.) (б) $m = n = 2$, поэтому решение существует и единственно. (с) $m = 3 > n$, поэтому точного решения не существует. Показано решение по методу наименьших квадратов

7.7.1. Решение квадратных систем

В случае $m = n$ решение \mathbf{x} можно найти, вычислив LU-разложение, $\mathbf{A} = \mathbf{LU}$, а затем поступив следующим образом:

$$\mathbf{Ax} = \mathbf{b}, \quad (7.215)$$

$$\mathbf{LUx} = \mathbf{b}, \quad (7.216)$$

$$\mathbf{Ux} = \mathbf{L}^{-1}\mathbf{b} \triangleq \mathbf{y}, \quad (7.217)$$

$$\mathbf{x} = \mathbf{U}^{-1}\mathbf{y}. \quad (7.218)$$

Здесь важно, что обе матрицы \mathbf{L} и \mathbf{U} треугольные, поэтому можно обойтись без вычисления обратных матриц, а использовать метод **обратного хода**.

Именно, найти решение $\mathbf{y} = \mathbf{L}^{-1}\mathbf{b}$ без обращения матриц можно следующим образом. Сначала запишем:

$$\begin{pmatrix} L_{11} & & & \\ L_{21} & L_{22} & & \\ & & \ddots & \\ L_{n1} & L_{n2} & \cdots & L_{nn} \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}. \quad (7.219)$$

Начинаем с решения уравнения $L_{11}y_1 = b_1$, из которого находим y_1 и подставляем в уравнение

$$L_{21}y_1 + L_{22}y_2 = b_2, \quad (7.220)$$

из которого находим y_2 . Эта процедура повторяется рекурсивно и зачастую обозначается **оператором обратной косой черты**, $\mathbf{y} = \mathbf{L} \backslash \mathbf{b}$. Найдя \mathbf{y} , мы можем точно так же, обратным ходом, найти $\mathbf{x} = \mathbf{U}^{-1}\mathbf{y}$.

7.7.2. Решение недоопределенных систем (оценка по наименьшей норме)

В этом разделе мы рассмотрим недоопределенные системы, когда $m < n^1$. Предполагается, что строки линейно независимы, т. е. матрица \mathbf{A} полного ранга.

Если $m < n$, то существует несколько возможных решений, и все они имеют вид

$$\{\mathbf{x} : \mathbf{Ax} = \mathbf{b}\} = \{\mathbf{x}_p + \mathbf{z} : \mathbf{z} \in \text{nullspace}(\mathbf{A})\}, \quad (7.221)$$

где \mathbf{x}_p – какое-то решение. Принято выбирать решение с минимальной ℓ_2 -нормой:

¹ Наше изложение частично основано на заметках к лекциям Стивена Бойда по адресу <http://ee263.stanford.edu/lectures/min-norm>.

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{x}\|_2^2 \quad \text{такое, что } \mathbf{Ax} = \mathbf{b}. \quad (7.222)$$

Вычислить решение с минимальной нормой можно, воспользовавшись правой псевдообратной матрицей:

$$\mathbf{x}_{\text{pinv}} = \mathbf{A}^T(\mathbf{AA}^T)^{-1}\mathbf{b}. \quad (7.223)$$

Чтобы убедиться в этом, предположим, что \mathbf{x} – какое-то другое решение, т. е. $\mathbf{Ax} = \mathbf{b}$ и $\mathbf{A}(\mathbf{x} - \mathbf{x}_{\text{pinv}}) = \mathbf{0}$. Тогда

$$\begin{aligned} (\mathbf{x} - \mathbf{x}_{\text{pinv}})^T \mathbf{x}_{\text{pinv}} &= (\mathbf{x} - \mathbf{x}_{\text{pinv}})^T \mathbf{A}^T (\mathbf{AA}^T)^{-1} \mathbf{b} \\ &= (\mathbf{A}(\mathbf{x} - \mathbf{x}_{\text{pinv}}))^T (\mathbf{AA}^T)^{-1} \mathbf{b} = 0, \end{aligned} \quad (7.224)$$

а потому $(\mathbf{x} - \mathbf{x}_{\text{pinv}}) \perp \mathbf{x}_{\text{pinv}}$. По теореме Пифагора, норма \mathbf{x} равна:

$$\|\mathbf{x}\|^2 = \|\mathbf{x}_{\text{pinv}} + \mathbf{x} - \mathbf{x}_{\text{pinv}}\|^2 = \|\mathbf{x}_{\text{pinv}}\|^2 + \|\mathbf{x} - \mathbf{x}_{\text{pinv}}\|^2 \geq \|\mathbf{x}_{\text{pinv}}\|^2. \quad (7.225)$$

Таким образом, любое решение, отличное от \mathbf{x}_{pinv} , имеет большую норму.

Решить задачу оптимизации с ограничениями (7.222) можно и по-другому – минимизировав следующую целевую функцию без ограничений:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{x}^T \mathbf{x} + \boldsymbol{\lambda}^T (\mathbf{Ax} - \mathbf{b}). \quad (7.226)$$

В разделе 8.5.1 показано, что условия оптимальности имеют вид:

$$\nabla_{\mathbf{x}} \mathcal{L} = 2\mathbf{x} + \mathbf{A}^T \boldsymbol{\lambda} = \mathbf{0}; \quad \nabla_{\boldsymbol{\lambda}} \mathcal{L} = \mathbf{Ax} - \mathbf{b} = \mathbf{0}. \quad (7.227)$$

Из первого условия находим $\mathbf{x} = -\mathbf{A}^T \boldsymbol{\lambda} / 2$. Подставляя во второе уравнение, получаем

$$\mathbf{Ax} = -\frac{1}{2} \mathbf{AA}^T \boldsymbol{\lambda} = \mathbf{b}, \quad (7.228)$$

откуда следует, что $\boldsymbol{\lambda} = -2(\mathbf{AA}^T)^{-1}\mathbf{b}$. Отсюда $\mathbf{x} = \mathbf{A}^T(\mathbf{AA}^T)^{-1}\mathbf{b}$, а это и есть решение в виде правой псевдообратной матрицы.

Существует интересная связь с регуляризированным методом наименьших квадратов. Положим, $J_1 = \|\mathbf{Ax} - \mathbf{b}\|_2^2$ и $J_2 = \|\mathbf{x}\|_2^2$. Решение с минимальной нормой минимизирует J_2 при ограничении $J_1 = 0$. Теперь рассмотрим следующую целевую функцию в виде взвешенной суммы:

$$\mathcal{L}(\mathbf{x}) = J_2 + \lambda J_1 = \|\mathbf{x}\|_2^2 + \lambda \|\mathbf{b} - \mathbf{Ax}\|_2^2. \quad (7.229)$$

Поскольку это выпуклая задача, лагранжиан достигает минимума, когда градиент равен нулю, так что

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}) = 2\mathbf{x} - 2\lambda \mathbf{A}^T (\mathbf{b} - \mathbf{Ax}) = \mathbf{0} \Rightarrow \mathbf{x}(\mathbf{I} + \lambda \mathbf{A}^T \mathbf{A}) = \lambda \mathbf{A}^T \mathbf{b}. \quad (7.230)$$

При $\lambda \rightarrow \infty$ получаем

$$\hat{\mathbf{x}} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}, \quad (7.231)$$

где $(A^T A)^{-1} A^T$ – левая псевдообратная матрица для A . Альтернативно (но эквивалентно) можно назначить вес μ ℓ_2 -регуляризатору $\|x\|_2^2$ и устремить его к нулю. Из приведенной выше формулы получаем:

$$(\mu I + A^T A)^{-1} A^T \rightarrow A^T (A A^T)^{-1}. \quad (7.232)$$

Итак, решение по методу регуляризованных наименьших квадратов в пределе, когда регуляризация отсутствует, сходится к решению по минимальной норме.

7.7.3. Решение переопределенных систем (оценка по методу наименьших квадратов)

Если $m > n$, то мы имеем переопределенную систему, которая, как правило, не имеет точного решения, но можно попытаться найти решение, которое как можно лучше удовлетворяет всем ограничениям, представленным в форме $Ax = b$. Это можно сделать путем минимизации следующей **целевой функции метода наименьших квадратов**:

$$f(x) = \frac{1}{2} \|Ax - b\|_2^2. \quad (7.233)$$

Используя результаты матричного исчисления из раздела 7.8, находим градиент:

$$g(x) = \frac{\partial}{\partial x} f(x) = A^T Ax - A^T b. \quad (7.234)$$

Для нахождения оптимума следует решить уравнение $g(x) = 0$, т. е.

$$A^T Ax = A^T b. \quad (7.235)$$

Эти уравнения называются **нормальными**, потому что в точке оптимума вектор $b - Ax$ нормален (ортогонален) образу A , как объясняется в разделе 11.2.2.2. Соответствующее решение \hat{x} , полученное методом **обыкновенных наименьших квадратов** (ordinary least squares – **OLS**), имеет вид:

$$\hat{x} = (A^T A)^{-1} A^T b. \quad (7.236)$$

Величина $A^\dagger = (A^T A)^{-1} A^T$ – левая псевдообратная матрица для (неквадратной) матрицы A (детали см. в разделе 7.5.3).

Можно убедиться, что это решение единственное, показав, что гессиан положительно определен. В данном случае гессиан имеет вид:

$$H(x) = \frac{\partial^2}{\partial x^2} f(x) = A^T A. \quad (7.237)$$

Если матрица A полного ранга (т. е. столбцы A линейно независимы), то H положительно определенная, т. к. для любого $v > 0$ имеем:

$$\mathbf{v}^T(\mathbf{A}^T\mathbf{A})\mathbf{v} = (\mathbf{A}\mathbf{v})^T(\mathbf{A}\mathbf{v}) = \|\mathbf{A}\mathbf{v}\|^2 > 0. \quad (7.238)$$

Следовательно, для матриц полного ранга целевая функция метода наименьших квадратов имеет единственный глобальный минимум.

7.8. МАТРИЧНОЕ ИСЧИСЛЕНИЕ

Предмет анализа, или **исчисления** (бесконечно малых) – вычисление «скорости изменения» функций при изменении их аргументов. Это чрезвычайно важно для машинного обучения, как и почти для всех численных методов. В этом разделе мы приведем краткую сводку некоторых стандартных результатов. Иногда мы будем использовать понятия и обозначения матричной алгебры, введенные в главе 7. Дополнительные сведения с точки зрения глубокого обучения см. в работе [РН18].

7.8.1. Производные

Рассмотрим функцию скалярного аргумента $f: \mathbb{R} \rightarrow \mathbb{R}$. Определим ее **производную** в точке a как величину

$$f'(x) \triangleq \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (7.239)$$

в предположении, что предел существует. Эта величина измеряет скорость изменения выхода при перемещении на малое расстояние от x в пространстве входов. $f'(x)$ – можно интерпретировать как угол наклона касательной к графику $f(x)$ и потому

$$f(x+h) \approx f(x) + f'(x)h \quad (7.240)$$

для малых h .

Мы можем вычислить **конечноразностную** аппроксимацию производной, взяв конечный размер шага h :

$$\begin{aligned} f'(x) &\equiv \underbrace{\lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}}_{\text{прямая разность}} = \underbrace{\lim_{h \rightarrow 0} \frac{f(x+h/2) - f(x-h/2)}{h}}_{\text{центральная разность}} \\ &= \underbrace{\lim_{h \rightarrow 0} \frac{f(x) - f(x-h)}{h}}_{\text{обратная разность}}. \end{aligned} \quad (7.241)$$

Чем меньше размер шага h , тем лучше оценка. Но если h слишком мало, то возможны ошибки из-за сокращения чисел.

Дифференцирование можно рассматривать как оператор, отображающий функции в функции, $D(f) = f'$, где $f'(x)$ вычисляет производную в точке x (в предположение, что производная в этой точке существует). Использование

записи со штрихом f' для обозначения производной называется **нотацией Лагранжа**. Вторая производная, измеряющая скорость изменения градиента, обозначается f'' . n -я производная обозначается $f^{(n)}$.

Имеется альтернативная **нотация Лейбница**, когда функция обозначается $y = f(x)$, а ее производная — $\frac{dy}{dx}$ или $\frac{d}{dx}f(x)$. Чтобы обозначить вычисление производной в точке a , пишут $\left.\frac{df}{dx}\right|_{x=a}$.

7.8.2. Градиенты

Понятие производной можно обобщить на функции векторного аргумента, $f: \mathbb{R}^n \rightarrow \mathbb{R}$, определив **частную производную** f по x_i :

$$\frac{\partial f}{\partial x_i} = \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}, \quad (7.242)$$

где \mathbf{e}_i — i -й единичный вектор.

Градиентом функции в точке \mathbf{x} называется вектор, составленный из ее частных производных:

$$\mathbf{g} = \frac{\partial f}{\partial \mathbf{x}} = \nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}. \quad (7.243)$$

Указать, в какой точке вычисляется градиент, можно следующим образом:

$$\mathbf{g}(\mathbf{x}^*) \triangleq \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}^*}. \quad (7.244)$$

Как видим, оператор ∇ (произносится «набла») отображает функцию $f: \mathbb{R}^n \rightarrow \mathbb{R}$ в другую функцию $\mathbf{g}: \mathbb{R}^n \rightarrow \mathbb{R}^n$. Поскольку $\mathbf{g}()$ — просто векторная функция, ее называют **векторным полем**. А производную f' иногда называют **скалярным полем**.

7.8.3. Производная по направлению

Производная по направлению измеряет скорость изменения функции $f: \mathbb{R}^n \rightarrow \mathbb{R}$ вдоль направления \mathbf{v} в пространстве. Она определяется следующим образом:

$$D_{\mathbf{v}}f(\mathbf{x}) = \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{v}) - f(\mathbf{x})}{h}. \quad (7.245)$$

Ее можно аппроксимировать численно с помощью двух обращений к функции f независимо от n . С другой стороны, для численной аппроксимации стандартного градиента требуется $n + 1$ обращений (или $2n$ при использовании центральных разностей).

Заметим, что производная по направлению \mathbf{v} является скалярным произведением градиента \mathbf{g} и вектора \mathbf{v} :

$$D_{\mathbf{v}}f(\mathbf{x}) = \nabla f(\mathbf{x}) \cdot \mathbf{v}. \quad (7.246)$$

7.8.4. Полная производная*

Предположим, что одни аргументы функции зависят от других. Конкретно, предположим, что функция имеет вид $f(t, x(t), y(t))$. Определим **полную производную** f по t следующим образом:

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}. \quad (7.247)$$

Если умножить обе части на дифференциал dt , то получим **полный дифференциал**:

$$df = \frac{\partial f}{\partial t} dt + \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy. \quad (7.248)$$

Он измеряет, насколько изменяется f при изменении t – как вследствие прямого влияния t на f , так и влияния t на x и y .

7.8.5. Якобиан

Рассмотрим функцию, которая отображает один вектор в другой, $\mathbf{f}: \mathbb{R}^n \rightarrow \mathbb{R}^m$. **Якобиан** (или **матрица Якоби**) этой функции – это матрица размера $m \times n$, составленная из частных производных:

$$\mathbf{J}_f(\mathbf{x}) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}^\top} \triangleq \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \nabla f_1(\mathbf{x})^\top \\ \vdots \\ \nabla f_m(\mathbf{x})^\top \end{pmatrix}. \quad (7.249)$$

Заметим, что мы расположили все частные производные одной компоненты функции по разным переменным в строках матрицы; такую запись иногда называют **по строкам** (numerator) или **нотацией Якоби**¹.

¹ Гораздо более полное рассмотрение нотации см. в статье https://en.wikipedia.org/wiki/Matrix_calculus.

7.8.5.1. Умножение якобиана на вектор

Произведение якобиана на вектор (JVP) определяется как результат умножения справа матрицы Якоби $\mathbf{J} \in \mathbb{R}^{m \times n}$ на вектор $\mathbf{v} \in \mathbb{R}^n$:

$$\mathbf{J}_f(\mathbf{x})\mathbf{v} = \begin{pmatrix} \nabla f_1(\mathbf{x})^\top \\ \vdots \\ \nabla f_m(\mathbf{x})^\top \end{pmatrix} \mathbf{v} = \begin{pmatrix} \nabla f_1(\mathbf{x})^\top \mathbf{v} \\ \vdots \\ \nabla f_m(\mathbf{x})^\top \mathbf{v} \end{pmatrix}. \quad (7.250)$$

Как видим, эту величину можно численно аппроксимировать с помощью всего двух обращений к f .

Произведение вектора на якобиан (VJP) определяется как результат умножения слева матрицы Якоби $\mathbf{J} \in \mathbb{R}^{m \times n}$ на вектор $\mathbf{v} \in \mathbb{R}^m$:

$$\mathbf{u}^\top \mathbf{J}_f(\mathbf{x}) = \mathbf{u}^\top \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right) = \left(\mathbf{u} \cdot \frac{\partial f}{\partial x_1}, \dots, \mathbf{u} \cdot \frac{\partial f}{\partial x_n} \right). \quad (7.251)$$

Вычисление JVP более эффективно, когда $m \geq n$, а вычисление VJP – когда $m \leq n$. О том, как этим можно воспользоваться для автоматического дифференцирования в графе вычислений в глубоких нейронных сетях, см. раздел 13.3.

7.8.5.2. Якобиан композиции

Иногда приходится вычислять якобиан композиции двух функций. Пусть $h(\mathbf{x}) = g(f(\mathbf{x}))$. В силу правила дифференцирования сложной функции имеем:

$$\mathbf{J}_h(\mathbf{x}) = \mathbf{J}_g(f(\mathbf{x}))\mathbf{J}_f(\mathbf{x}). \quad (7.252)$$

Например, предположим, что $f: \mathbb{R} \rightarrow \mathbb{R}^2$ и $g: \mathbb{R}^2 \rightarrow \mathbb{R}^2$. Тогда

$$\frac{\partial \mathbf{g}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial}{\partial \mathbf{x}} g_1(f_1(\mathbf{x}), f_2(\mathbf{x})) \\ \frac{\partial}{\partial \mathbf{x}} g_2(f_1(\mathbf{x}), f_2(\mathbf{x})) \end{pmatrix} = \begin{pmatrix} \frac{\partial g_1}{\partial f_1} \frac{\partial f_1}{\partial \mathbf{x}} + \frac{\partial g_1}{\partial f_2} \frac{\partial f_2}{\partial \mathbf{x}} \\ \frac{\partial g_2}{\partial f_1} \frac{\partial f_1}{\partial \mathbf{x}} + \frac{\partial g_2}{\partial f_2} \frac{\partial f_2}{\partial \mathbf{x}} \end{pmatrix} \quad (7.253)$$

$$= \frac{\partial \mathbf{g}}{\partial \mathbf{f}^\top} \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial g_1}{\partial f_1} & \frac{\partial g_1}{\partial f_2} \\ \frac{\partial g_2}{\partial f_1} & \frac{\partial g_2}{\partial f_2} \end{pmatrix} \begin{pmatrix} \frac{\partial f_1}{\partial \mathbf{x}} \\ \frac{\partial f_2}{\partial \mathbf{x}} \end{pmatrix}. \quad (7.254)$$

7.8.6. Гессиан

Если функция $f: \mathbb{R}^n \rightarrow \mathbb{R}$ дважды дифференцируема, то **гессиан** (или **матрица Гесса**) определяется как (симметричная) матрица, составленная из вторых частных производных:

$$\mathbf{H}_f = \frac{\partial^2 \mathbf{g}}{\partial \mathbf{x}^2} = \nabla^2 f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ & \ddots & \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}. \quad (7.255)$$

Как видим, гессиан является якобианом градиента.

7.8.7. Градиенты часто встречающихся функций

В этом разделе мы приведем без доказательства градиенты некоторых широко распространенных функций.

7.8.7.1. Функции, отображающие скаляры в скаляры

Рассмотрим дифференцируемую функцию $f: \mathbb{R} \rightarrow \mathbb{R}$. Вот несколько полезных тождеств из одномерного вещественного анализа, с которыми вы, должно быть, хорошо знакомы:

$$\frac{d}{dx} cx^n = cnx^{n-1}; \quad (7.256)$$

$$\frac{d}{dx} \log(x) = 1/x; \quad (7.257)$$

$$\frac{d}{dx} \exp(x) = \exp(x); \quad (7.258)$$

$$\frac{d}{dx} [f(x) + g(x)] = \frac{df(x)}{dx} + \frac{dg(x)}{dx}; \quad (7.259)$$

$$\frac{d}{dx} [f(x)g(x)] = f(x)\frac{dg(x)}{dx} + g(x)\frac{df(x)}{dx}; \quad (7.260)$$

$$\frac{d}{dx} f(u(x)) = \frac{du}{dx} \frac{df(u)}{du}. \quad (7.261)$$

Формула (7.261) называется **правилом дифференцирования сложной функции**.

7.8.7.2. Функции, отображающие векторы в скаляры

Рассмотрим дифференцируемую функцию $f: \mathbb{R}^n \rightarrow \mathbb{R}$. Вот несколько полезных тождеств¹:

¹ Некоторые из этих тождеств взяты со страницы <http://www.cs.nyu.edu/~roweis/notes/matrixid>.

$$\frac{\partial(\mathbf{a}^\top \mathbf{x})}{\partial \mathbf{x}} = \mathbf{a}; \quad (7.262)$$

$$\frac{\partial(\mathbf{b}^\top \mathbf{A} \mathbf{x})}{\partial \mathbf{x}} = \mathbf{A}^\top \mathbf{b}; \quad (7.263)$$

$$\frac{\partial(\mathbf{x}^\top \mathbf{A} \mathbf{x})}{\partial \mathbf{x}} = (\mathbf{A} + \mathbf{A}^\top) \mathbf{x}. \quad (7.264)$$

Эти тождества легко доказываются, если раскрыть соответствующую квадратичную форму и применить формулы скалярного анализа.

7.8.7.3. Функции, отображающие матрицы в скаляры

Рассмотрим функцию $f: \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$, отображающую матрицу в скаляр. Мы пользуемся следующей естественной организацией матрицы производных:

$$\frac{\partial f}{\partial \mathbf{X}} = \begin{pmatrix} \frac{\partial f}{\partial x_{11}} & \cdots & \frac{\partial f}{\partial x_{1n}} \\ \vdots & & \\ \frac{\partial f}{\partial x_{m1}} & \cdots & \frac{\partial f}{\partial x_{mn}} \end{pmatrix}. \quad (7.265)$$

Ниже приведены некоторые полезные тождества.

Тождества, включающие квадратичные формы

Можно доказать следующие результаты:

$$\frac{\partial f}{\partial \mathbf{X}}(\mathbf{a}^\top \mathbf{X} \mathbf{b}) = \mathbf{a} \mathbf{b}^\top; \quad (7.266)$$

$$\frac{\partial f}{\partial \mathbf{X}}(\mathbf{a}^\top \mathbf{X}^\top \mathbf{b}) = \mathbf{b} \mathbf{a}^\top. \quad (7.267)$$

Тождества, включающие след матрицы

Можно доказать следующие результаты:

$$\frac{\partial f}{\partial \mathbf{X}} \text{tr}(\mathbf{A} \mathbf{X} \mathbf{B}) = \mathbf{A}^\top \mathbf{B}^\top; \quad (7.268)$$

$$\frac{\partial f}{\partial \mathbf{X}} \text{tr}(\mathbf{X}^\top \mathbf{A}) = \mathbf{A}; \quad (7.269)$$

$$\frac{\partial f}{\partial \mathbf{X}} \text{tr}(\mathbf{X}^{-1} \mathbf{A}) = -\mathbf{X}^{-\top} \mathbf{A}^\top \mathbf{X}^{-\top}; \quad (7.270)$$

$$\frac{\partial f}{\partial \mathbf{X}} \text{tr}(\mathbf{X}^\top \mathbf{A} \mathbf{X}) = (\mathbf{A} + \mathbf{A}^\top) \mathbf{X}. \quad (7.271)$$

Тождества, включающие определитель матрицы

Можно доказать следующие результаты:

$$\frac{\partial f}{\partial \mathbf{X}} \det(\mathbf{AXB}) = \det(\mathbf{AXB}) \mathbf{X}^{-\top}; \quad (7.272)$$

$$\frac{\partial f}{\partial \mathbf{X}} \ln(\det(\mathbf{X})) = \mathbf{X}^{-\top}. \quad (7.273)$$

7.9. УПРАЖНЕНИЯ

Упражнение 7.1 [ортогональные матрицы].

- a. Поворот в трехмерном пространстве на угол α вокруг оси z описывается следующей матрицей:

$$\mathbf{R}(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (7.274)$$

Докажите, что матрица \mathbf{R} ортогональная, т. е. $\mathbf{R}^{\top} \mathbf{R} = \mathbf{I}$ для любого α .

- b. Найдите единственный собственный вектор \mathbf{v} матрицы \mathbf{R} с собственным значением 1.0 и единичной нормой (т. е. $\|\mathbf{v}\|^2 = 1$)? (Ответ должен быть одинаковым для любого α .) *Указание:* подумайте о геометрической интерпретации собственных векторов.

Упражнение 7.2 [нахождение собственных векторов вручную*].

Найдите собственные значения и собственные векторы следующей матрицы:

$$\mathbf{A} = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}. \quad (7.275)$$

Вычислите результаты вручную и проверьте их с помощью программы на Python.

Глава 8

Оптимизация

В написании этой главы участвовали Фредерик Кунстнер, Си И Мень, Аарон Мишкин, Шаран Васвани и Марк Шмидт.

8.1. ВВЕДЕНИЕ

В главе 4 мы видели, что главной задачей машинного обучения является оценивание параметров (или обучение модели). Для этого требуется решить **задачу оптимизации**, т. е. найти значения множества переменных $\theta \in \Theta$, доставляющие минимум скалярной **функции потерь**, или **функции стоимости** $\mathcal{L} : \Theta \rightarrow \mathbb{R}$:

$$\theta^* \in \underset{\theta \in \Theta}{\operatorname{argmin}} \mathcal{L}(\theta). \quad (8.1)$$

Будем предполагать, что **пространство параметров** $\Theta \subseteq \mathbb{R}^D$, где D – число переменных, по которым производится оптимизация. Таким образом, нас будет интересовать **непрерывная**, а не **дискретная оптимизация**.

Если мы хотим **максимизировать функцию оценки**, или **функцию вознаграждения** $R(\theta)$, то можем вместо этого минимизировать функцию $\mathcal{L}(\theta) = -R(\theta)$. Мы будем употреблять общий термин **целевая функция** для обозначения функции, которую требуется максимизировать или минимизировать. Алгоритм нахождения оптимума целевой функции часто называют **решателем**.

Далее в этой главе мы обсудим различные виды решателей для разных целевых функций, обращая особое внимание на методы, применяемые в сообществе машинного обучения. Дополнительные сведения об оптимизации вообще можно найти в таких прекрасных учебниках, как [KW19b, BV04, NW06, Ber15, Ber16], а также в обзорных статьях, например [BCN18, Sun+19b, PPS18, Pey20].

8.1.1. Локальная и глобальная оптимизация

Точка, удовлетворяющая условию (8.1), называется **глобальным оптимумом**. Нахождение такой точки называется **глобальной оптимизацией**.

В общем случае задача нахождения глобального оптимума вычислительно неразрешима [Neu04]. В таких случаях мы пытаемся найти **локальный оптимум**. Для непрерывных задач он определяется как точка θ^* , в которой стоимость меньше (или равна) стоимости в «близких» точках. Формально говорят, что θ^* является **локальным минимумом**, если

$$\exists \delta > 0, \forall \theta \in \Theta \text{ такое, что } \|\theta - \theta^*\| < \delta, \mathcal{L}(\theta^*) \leq \mathcal{L}(\theta). \quad (8.2)$$

Локальный минимум может быть окружен другими локальными минимумами той же целевой функции, в таком случае он называется **плоским**. Говорят, что точка является точкой **строгого локального минимума**, если целевая функция в ней строго меньше, чем в соседних точках:

$$\exists \delta > 0, \forall \theta \in \Theta, \theta \neq \theta^* : \|\theta - \theta^*\| < \delta, \mathcal{L}(\theta^*) < \mathcal{L}(\theta). \quad (8.3)$$

Локальный максимум (в том числе строгий) определяется аналогично (см. иллюстрацию на рис. 8.1а).

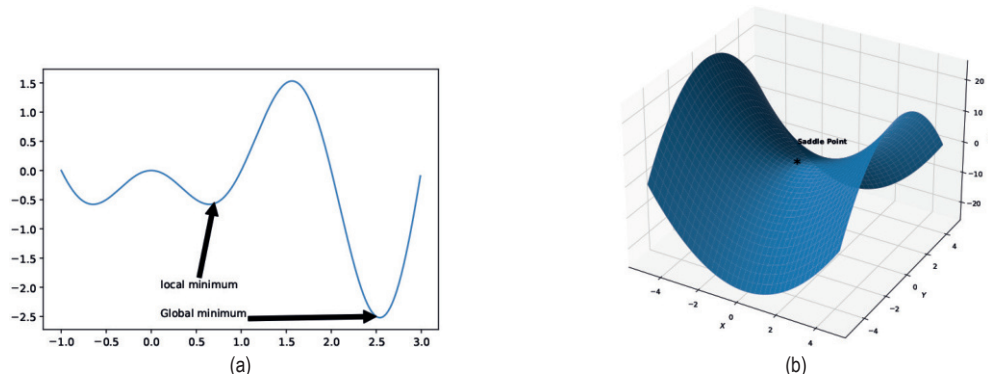


Рис. 8.1 ❖ (а) Локальный и глобальный минимум в одномерном случае. Построено программой по адресу figures.probl.ai/book1/8.1. (б) Седловая точка в трехмерном пространстве. Построено программой по адресу figures.probl.ai/book1/8.1

И последнее замечание о терминологии: если алгоритм гарантированно сходится к стационарной точке, начав работу из любой точки, то говорят, что он **глобально сходящийся**. Но это не означает, что он сходится к глобальному оптимуму (что вносит некоторую путаницу), а означает лишь сходимость к некоторой стационарной точке.

8.1.1.1. Условия оптимальности для локальных и глобальных оптимумов

Для непрерывных, дважды дифференцируемых функций можно точно охарактеризовать точки, соответствующие локальным минимумам. Обозначим $g(\theta) = \nabla \mathcal{L}(\theta)$ вектор градиента, а $H(\theta) = \nabla^2 \mathcal{L}(\theta)$ – гессиан. (Если вы забыли, что

означают эти понятия, обратитесь к разделу 7.8.) Рассмотрим точку $\theta^* \in \mathbb{R}^D$, и пусть $\mathbf{g}^* = \mathbf{g}(\theta)|_{\theta^*}$ – градиент в этой точке, а $\mathbf{H}^* = \mathbf{H}(\theta)|_{\theta^*}$ – соответствующий гессиан. Можно показать, что следующие условия характеризуют любой локальный минимум:

- необходимое условие: если θ^* – локальный минимум, то должно быть $\mathbf{g}^* = \mathbf{0}$ (т. е. θ^* должна быть **стационарной точкой**) и \mathbf{H}^* должна быть положительно определенной;
- достаточное условие: если $\mathbf{g}^* = \mathbf{0}$ и \mathbf{H}^* положительно определенная, то θ^* является локальным оптимумом.

Чтобы понять, почему первое условие необходимо, предположим, что мы находимся в точке θ^* , в которой градиент отличен от нуля: в такой точке можно было бы уменьшить функцию, сдвинувшись на небольшое расстояние в направлении отрицательного градиента, поэтому она не была бы оптимальной. Поэтому градиент должен быть равен нулю. (Для негладких функций необходимое условие заключается в том, что в точке минимума в нуль должен обращаться локальный субградиент.) Чтобы понять, почему условие нулевого градиента недостаточно, заметим, что стационарная точка может быть локальным минимумом, локальным максимумом или **седловой точкой**, в которой одни направления ведут вверх, а другие вниз (см. рис. 8.1b). Точнее, в седловой точке часть собственных значений гессиана будет положительна, а часть отрицательна. Однако если в некоторой точке гессиан положительно полуопределен, то могут быть только направления, ведущие вверх, и плоские направления. Это означает, что целевую функцию нельзя уменьшить в окрестности данной точки, т. е. мы обязательно имеем локальный минимум. Если же гессиан строго положительно определен, то мы находимся в нижней точке «миски», где все направления ведут вверх; этого достаточно, чтобы точка была минимумом.

8.1.2. Условная и безусловная оптимизация

В задаче **безусловной оптимизации** мы ищем в пространстве параметров θ любое значение, минимизирующее потерю. Однако часто имеется набор **ограничений** на допустимые значения. Обычно весь набор ограничений \mathcal{C} разбивается на **ограничения в виде неравенств**, $g_j(\theta) \leq 0$ для $j \in \mathcal{J}$, и **ограничения в виде равенств**, $h_k(\theta) = 0$ для $k \in \mathcal{E}$. Например, можно записать ограничение «сумма должна быть равна единице» в виде равенства $h(\theta) = (1 - \sum_{i=1}^D \theta_i) = 0$, а ограничение неотрицательности параметров с помощью D неравенств $g_i(\theta) = -\theta_i \leq 0$.

Определим **допустимое множество** как подмножество пространства параметров, удовлетворяющее ограничениям:

$$\mathcal{C} = \{\theta : g_j(\theta) \leq 0 : j \in \mathcal{J}, h_k(\theta) = 0 : k \in \mathcal{E}\} \subseteq \mathbb{R}^D. \quad (8.4)$$

Теперь задача **условной оптимизации** принимает вид:

$$\theta^* \in \operatorname{argmin}_{\theta \in \mathcal{C}} \mathcal{L}(\theta). \quad (8.5)$$

Если $\mathcal{C} = \mathbb{R}^D$, то мы имеем задачу **безусловной оптимизации**.

При добавлении ограничений может измениться количество оптимумов функции. Например, функция, которая раньше была неограниченной (и поэтому не имела корректно определенных глобального максимума или минимума) может «обрести» несколько максимумов или минимумов после добавления ограничений, как показано на рис. 8.2. Но если добавить слишком много ограничений, то может оказаться, что допустимое множество пусто. Задача нахождения хотя бы одной точки, принадлежащей допустимому множеству, называется **задачей существования**, она сама по себе может быть трудной проблемой.

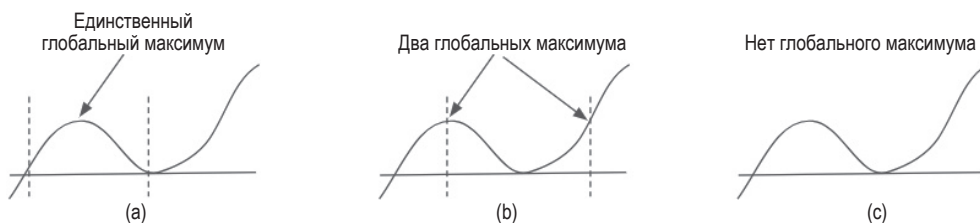


Рис. 8.2 ♦ Условная максимизация невыпуклой функции одной переменной. Область между пунктирными вертикальными прямыми – допустимое множество. (a) Существует единственный глобальный максимум, потому что функция выпукла на допустимом множестве. (b) Существует два глобальных максимума, оба на границе допустимого множества. (c) В случае безусловной оптимизации эта функция не имеет глобального максимума, потому что неограниченна

Стандартная стратегия решения задач условной оптимизации – завести штрафующие члены, которые измеряют степень нарушения ограничения. Эти члены прибавляются к целевой функции, после чего решается задача безусловной оптимизации. Частным случаем такой комбинированной целевой функции является **лагранжиан** (детали см. в разделе 8.5).

8.1.3. Выпуклая и невыпуклая оптимизация

В случае **выпуклой оптимизации** мы требуем, чтобы целевая функция была определена на выпуклом множестве и выпукла на нем (определение этих терминов см. в разделе 8.1.3). В таких задачах локальный минимум одновременно является глобальным. Поэтому многие модели проектируются так, чтобы обучаемая целевая функция была выпуклой.

8.1.3.1. Выпуклые множества

Множество \mathcal{S} называется **выпуклым**, если для любых $\mathbf{x}, \mathbf{x}' \in \mathcal{S}$ имеет место

$$\lambda \mathbf{x} + (1 - \lambda) \mathbf{x}' \in \mathcal{S}, \forall \lambda \in [0, 1]. \quad (8.6)$$

То есть все точки отрезка, соединяющего \mathbf{x} и \mathbf{x}' , лежат внутри множества. Примеры выпуклых и невыпуклых множеств приведены на рис. 8.3.

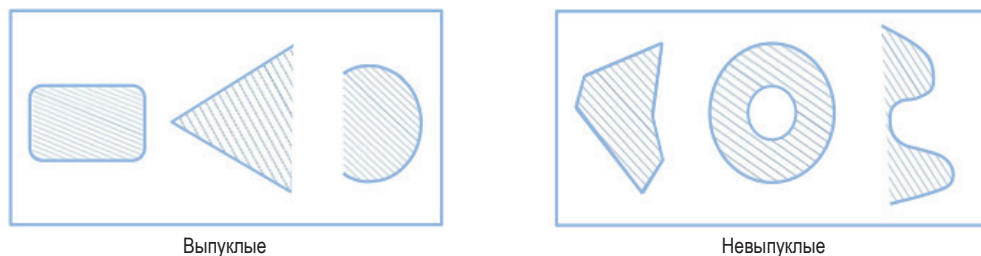


Рис. 8.3 ❖ Выпуклые и невыпуклые множества

8.1.3.2. Выпуклые функции

Функция f называется **выпуклой**, если ее **надграфик** (множество точек, расположенных выше графика функций, см. рис. 8.4а) является выпуклым множеством. Эквивалентное определение: функция $f(x)$ называется выпуклой, если она определена на выпуклом множестве S и для любых $x, y \in S$ и для любого $0 \leq \lambda \leq 1$ имеет место неравенство:

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y). \quad (8.7)$$

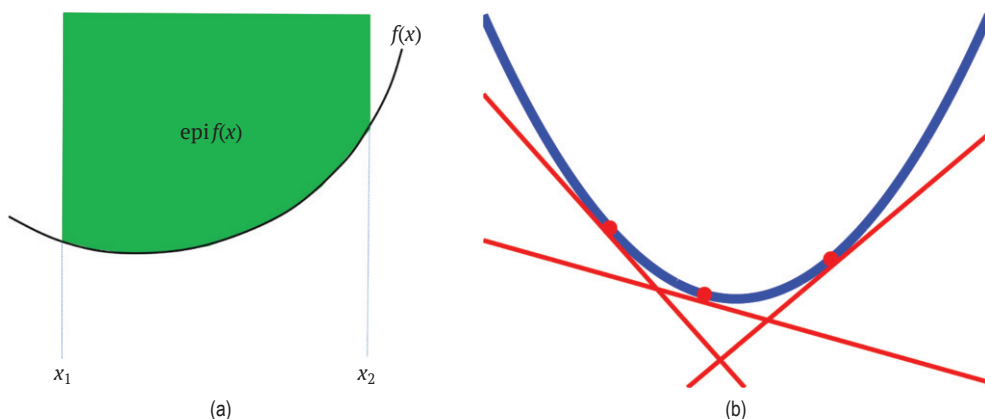


Рис. 8.4 ❖ (а) Надграфик функции. (б) Для выпуклой функции $f(x)$ надграфик можно представить в виде пересечения полупространств, нижние линейные границы которых определены **сопряженной функцией** $f^*(\lambda) = \max_x \lambda x - f(x)$

Пример выпуклой функции на вещественной прямой приведен на рис. 8.5а. Функция называется **строго выпуклой**, если это неравенство строгое. Функция $f(x)$ называется **вогнутой**, если $-f(x)$ выпукла, и **строго вогнутой**, если $-f(x)$ строго выпукла. Пример функции, не являющейся ни выпуклой, ни вогнутой, приведен на рис. 8.5б.

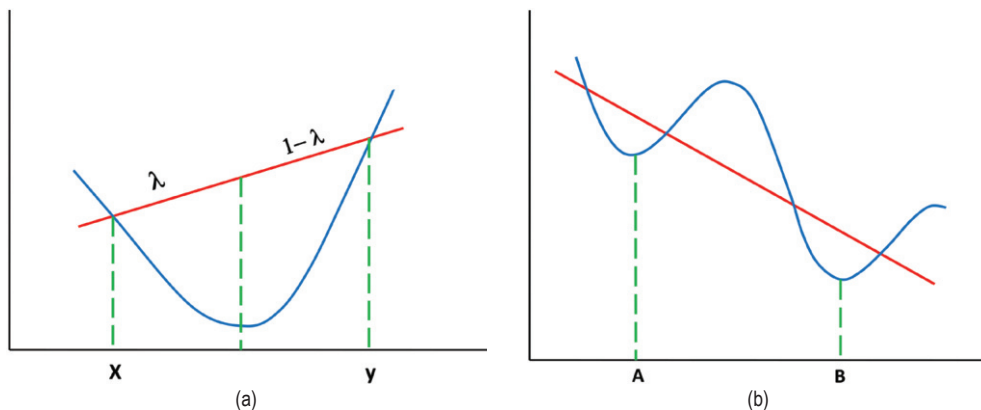


Рис. 8.5 ❖ (а) Выпуклая функция. Хорда, соединяющая точки $(x, f(x))$ и $(y, f(y))$, расположена выше функции. (б) Функция, не являющаяся ни выпуклой, ни вогнутой. А – локальный минимум, В – глобальный минимум

Приведем несколько примеров выпуклых функций одной переменной:

x^2 ;
 e^{ax} ;
 $-\log x$;
 $x^a, a > 1, x > 0$;
 $|x|^a, a \geq 1$;
 $x \log x, x > 0$.

8.1.3.3. Характеристика выпуклых функций

Интуитивно понятно, что выпуклая функция выглядит как миска. Формально можно доказать следующий важный результат.

Теорема 8.1.1. *Предположим, что функция $f: \mathbb{R}^n \rightarrow \mathbb{R}$ дважды дифференцируема в своей области определения. Тогда f выпукла тогда и только тогда, когда матрица $\mathbf{H} = \nabla^2 f(\mathbf{x})$ положительно полуопределенная (раздел 7.1.5.3) для всех $\mathbf{x} \in \text{dom}(f)$. При этом f строго выпукла, если \mathbf{H} положительно определенная.*

Например, рассмотрим квадратичную форму:

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x}. \quad (8.8)$$

Эта функция выпукла, если \mathbf{A} положительно полуопределенная, и строго выпукла, если \mathbf{A} положительно определенная. Она не является ни выпуклой, ни вогнутой, если собственные значения \mathbf{A} имеют разные знаки (см. рис. 8.6).

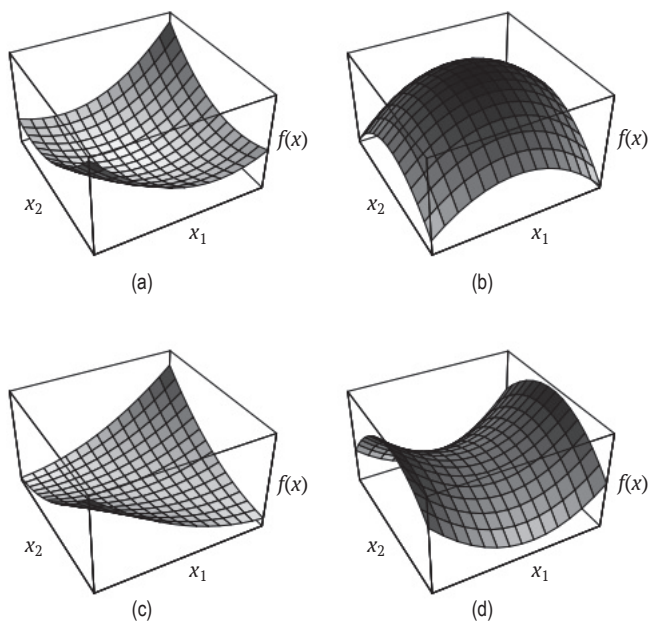


Рис. 8.6 ❖ Квадратичная форма $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x}$ с двумя переменными. (a) \mathbf{A} положительно определенная, поэтому f выпуклая. (b) \mathbf{A} отрицательно определенная, поэтому f вогнутая. (c) \mathbf{A} положительно полуопределенная, но сингулярная, поэтому f выпуклая, но не строго выпуклая. Обратите внимание на долину постоянной ширины в середине. (d) \mathbf{A} неопределенная, поэтому f не является ни выпуклой, ни вогнутой. Стационарная точка в середине поверхности является седловой точкой. На основе рис. 5 из [She94]

8.1.3.4. Сильно выпуклые функции

Функция f называется **сильно выпуклой** с параметром $m > 0$, если для любых \mathbf{x}, \mathbf{y} в области определения f имеет место неравенство:

$$(\nabla f(\mathbf{x}) - \nabla f(\mathbf{y}))^T (\mathbf{x} - \mathbf{y}) \geq m \|\mathbf{x} - \mathbf{y}\|_2^2. \quad (8.9)$$

Сильно выпуклая функция является также строго выпуклой, но обратное неверно.

Если функция f дважды непрерывно дифференцируема, то она сильно выпукла с параметром m тогда и только тогда, когда $\nabla^2 f(\mathbf{x}) \geq m \mathbf{I}$ для любого \mathbf{x} в области определения, где \mathbf{I} – единичная матрица, $\nabla^2 f$ – гессиан, а символ \geq означает, что матрица $\nabla^2 f(\mathbf{x}) - m \mathbf{I}$ положительно полуопределенная. Это эквивалентно условию, что минимальное собственное значение $\nabla^2 f(\mathbf{x})$ не меньше m для всех \mathbf{x} . Если областью определения является вещественная прямая, то $\nabla^2 f(\mathbf{x})$ – это просто вторая производная $f''(x)$, поэтому условие принимает вид $f''(x) \geq m$. Если $m = 0$, значит, гессиан положительно полуопределен (а если область определения – вещественная прямая, значит, $f''(x) \geq 0$), откуда следует, что функция выпуклая и, быть может, строго выпуклая, но не сильно выпуклая.

Различие между выпуклостью, строгой выпуклостью и сильной выпуклостью довольно тонкое. Чтобы лучше разобраться в этом вопросе, рассмотрим случай, когда f определена на вещественной прямой и дважды непрерывно дифференцируема. Тогда эти различия можно охарактеризовать следующим образом:

- f является выпуклой тогда и только тогда, когда $f''(x) \geq 0$ для всех x ;
- f является строго выпуклой, если $f''(x) > 0$ для всех x (примечание: это условие достаточно, но не необходимо);
- f является сильно выпуклой тогда и только тогда, когда $f''(x) \geq m > 0$ для всех x .

Можно показать, что функция f сильно выпукла с параметром m тогда и только тогда, когда функция

$$J(x) = f(x) - \frac{m}{2} \|x\|^2 \quad (8.10)$$

является выпуклой.

8.1.4. Гладкая и негладкая оптимизация

В случае **гладкой оптимизации** целевая функция и ограничения являются непрерывно дифференцируемыми функциями. Степень гладкости гладких функций можно количественно охарактеризовать с помощью **постоянной Липшица**. В одномерном случае так называется любая постоянная $L \geq 0$ такая, что для любых вещественных x_1 и x_2 имеет место неравенство:

$$|f(x_1) - f(x_2)| \leq L|x_1 - x_2|. \quad (8.11)$$

Это показано на рис. 8.8: для заданной постоянной L значение функции не может измениться более чем на L при изменении аргумента на одну единицу. Это понятие можно обобщить на функции с векторным аргументом при использовании подходящей нормы.

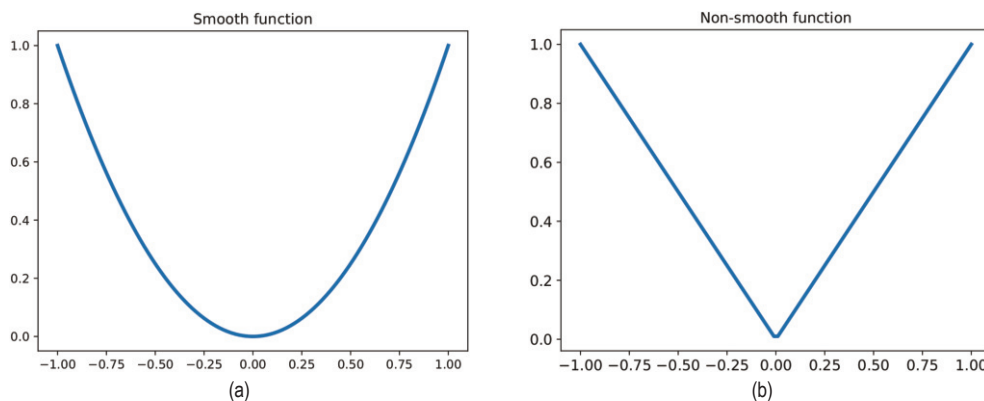


Рис. 8.7 ❖ (а) Гладкая функция одной переменной. (б) Негладкая функция одной переменной (имеется точка негладкости в начале координат). Построено программой по адресу figures.problai.com/book1/8.7

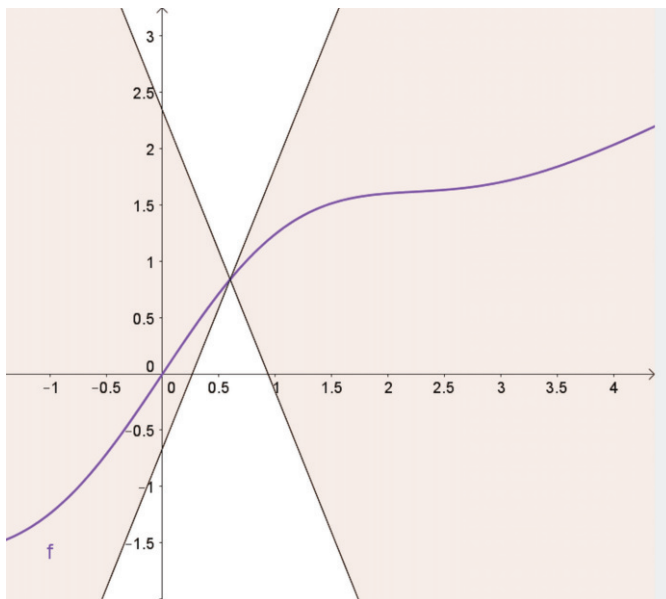


Рис. 8.8 ❖ Для липшицевой функции f существует двойной конус (белого цвета), вершину которого можно перемещать вдоль графика функции f , так что весь график будет оставаться вне конуса. Из статьи https://en.wikipedia.org/wiki/Lipschitz_continuity. Печатается с разрешения автора «Википедии» Taschee

В случае **негладкой оптимизации** существуют точки, в которых градиент целевой функции или ограничений не определен (см. пример на рис. 8.7). В некоторых задачах оптимизации целевую функцию можно разбить на две части: содержащую только гладкие члены и содержащую негладкие члены:

$$\mathcal{L}(\theta) = \mathcal{L}_s(\theta) + \mathcal{L}_r(\theta), \quad (8.12)$$

где \mathcal{L}_s – гладкая (дифференцируемая), а \mathcal{L}_r – негладкая функция. Часто такую целевую функцию называют **составной**. В машинном обучении \mathcal{L}_s – обычно потеря на обучающем наборе, а \mathcal{L}_r – регуляризатор, например ℓ_1 -норма θ . Такую составную структуру можно задействовать в различных алгоритмах.

8.1.4.1. Субградиенты

В этом разделе мы обобщим понятие производной на функции, имеющие локальные негладкости. В частности, для выпуклой функции нескольких переменных, $f: \mathbb{R}^n \rightarrow \mathbb{R}$, говорят, что $\mathbf{g} \in \mathbb{R}^n$ является **субградиентом** f в точке $\mathbf{x} \in \text{dom}(f)$, если для любой точки $\mathbf{z} \in \text{dom}(f)$:

$$f(\mathbf{z}) \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{z} - \mathbf{x}). \quad (8.13)$$

Заметим, что субградиент может существовать, даже если f не дифференцируема в точке, как показано на рис. 8.9.

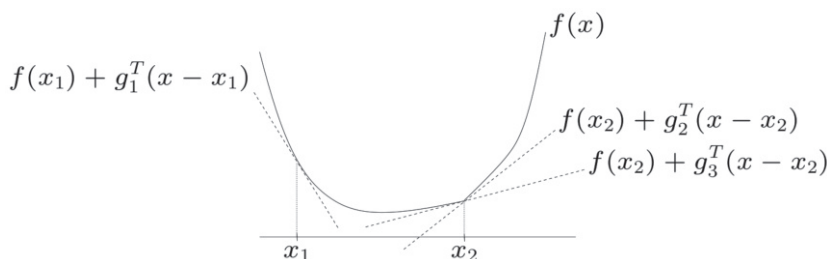


Рис. 8.9 ❖ Субградиенты. В точке x_1 выпуклая функция f дифференцируема, а g_1 (производная f в x_1) — однозначно определенный субградиент в x_1 . В точке x_2 f не дифференцируема, потому что существует «угол». Однако в ней имеется много субградиентов, из которых на рисунке показано только два. Взято из https://web.stanford.edu/class/ee364b/lectures/subgradients_slides. Печатается с разрешения Стивена Бойда

Функция f называется **субдифференцируемой** в точке x , если в x существует хотя бы один субградиент. Множество таких субградиентов называется **субдифференциалом** f в x и обозначается $\partial f(x)$.

Например, рассмотрим функцию модуля $f(x) = |x|$. Ее субдифференциал имеет вид

$$\partial f(x) = \begin{cases} \{-1\}, & \text{если } x < 0 \\ [-1, 1], & \text{если } x = 0, \\ \{+1\}, & \text{если } x > 0 \end{cases} \quad (8.14)$$

где $[-1, 1]$ обозначает отрезок между -1 и 1 включительно (см. иллюстрацию на рис. 8.10).

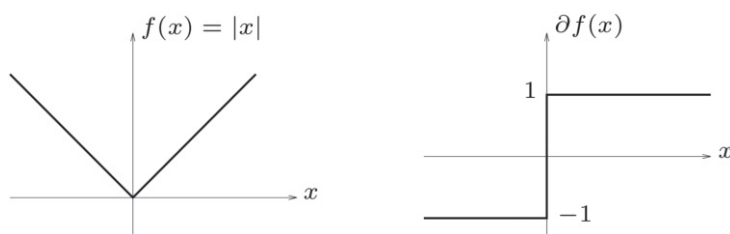


Рис. 8.10 ❖ Функция модуля (слева) и ее субдифференциал (справа). Взято из https://web.stanford.edu/class/ee364b/lectures/subgradients_slides. Печатается с разрешения Стивена Бойда

8.2. МЕТОДЫ ПЕРВОГО ПОРЯДКА

В этом разделе мы рассмотрим итеративные методы оптимизации, в которых используются **первые** производные целевой функции, т. е. они могут вычислять направления, указывающие «вниз», но игнорируют информацию

о кривизне. Во всех этих алгоритмах пользователь должен задать начальную точку θ_0 . Затем на каждой итерации t выполняется обновление вида:

$$\theta_{t+1} = \theta_t + \rho_t d_t, \quad (8.15)$$

где ρ_t называется **размером шага** или **скоростью обучения**, а d_t – направление спуска, например противоположное градиенту, который равен $g_t = \nabla_{\theta} \mathcal{L}(\theta)|_{\theta_t}$. Такие шаги обновления продолжаются, пока метод не достигнет стационарной точки, в которой градиент равен нулю.

8.2.1. Направление спуска

Говорят, что направление d является **направлением спуска**, если существует достаточно малое (но ненулевое) ρ такое, что при перемещении на величину ρ в направлении d значение функции гарантированно уменьшается. Формально мы требуем, чтобы существовало $\rho_{\max} > 0$ такое, что

$$\mathcal{L}(\theta + \rho d) < \mathcal{L}(\theta) \quad (8.16)$$

для всех $0 < \rho < \rho_{\max}$. Градиент на текущей итерации

$$g_t \triangleq \nabla \mathcal{L}(\theta)|_{\theta_t} = \nabla \mathcal{L}(\theta_t) = g(\theta_t) \quad (8.17)$$

направлен в сторону максимального увеличения f , поэтому противоположное направление (отрицательного градиента) является направлением спуска. Можно показать, что направлением спуска является также любое направление d , для которого угол θ между d и $-g_t$ меньше 90° и удовлетворяется условие:

$$d^T g_t = \|d\| \|g_t\| \cos(\theta) < 0. \quad (8.18)$$

На первый взгляд, лучше всего было бы выбрать направление $d_t = -g_t$. Оно называется **направлением скорейшего спуска**. Однако при таком выборе алгоритм может работать очень медленно. Ниже мы рассмотрим более быстрые варианты.

8.2.2. Размер шага (скорость обучения)

В машинном обучении последовательность размеров шагов $\{\rho_t\}$ называется **планом изменения скорости обучения** (learning rate schedule). Есть несколько широко используемых методов выбора такого плана, некоторые из них мы обсудим ниже (см. также раздел 8.4.3, где обсуждаются планы для стохастической оптимизации).

8.2.2.1. Постоянный размер шага

Проще всего использовать постоянный размер шага, $\rho_t = \rho$. Однако если шаг слишком велик, то метод может вообще не сойтись, а если слишком мал,

то сходимость будет очень медленной. Например, рассмотрим выпуклую функцию:

$$\mathcal{L}(\boldsymbol{\theta}) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2. \quad (8.19)$$

Выберем в качестве направления спуска $\mathbf{d}_t = -\mathbf{g}_t$. На рис. 8.11 показано, что происходит, когда мы движемся в этом направлении с шагом постоянного размера, начав из точки $(0, 0)$. На рис. 8.11a выбран малый размер шага $\rho = 0.1$; как видим, осуществляется медленное перемещение вдоль долины. На рис. 8.11b взят больший размер шага $\rho = 0.6$; теперь наблюдается осцилляция – переход с одного склона долины на другой, – и алгоритм вообще не сходится к оптимуму, хотя это выпуклая задача.

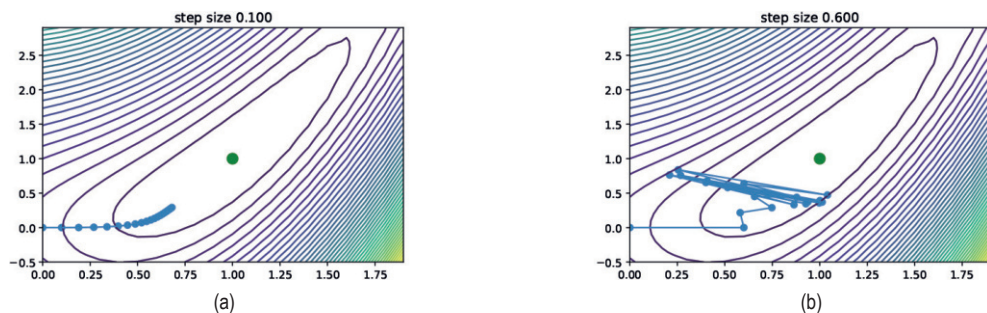


Рис. 8.11 ❖ Скорейший спуск для простой выпуклой функции, начинающийся из точки $(0, 0)$; показано 20 шагов постоянного размера. Глобальный минимум находится в точке $(1, 1)$. (a) $\rho = 0.1$. (b) $\rho = 0.6$. Построено программой по адресу figures.problm.ai/book1/8.11

В некоторых случаях можно вывести теоретическую верхнюю границу максимально допустимого размера шага. Например, рассмотрим квадратичную целевую функцию $\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^T \mathbf{A} \boldsymbol{\theta} + \mathbf{b}^T \boldsymbol{\theta} + \mathbf{c}$, где $\mathbf{A} \succcurlyeq \mathbf{0}$. Можно показать, что метод скорейшего спуска сходится к глобальному оптимуму тогда и только тогда, когда размер шага удовлетворяет условию

$$\rho < \frac{2}{\lambda_{\max}(\mathbf{A})}, \quad (8.20)$$

где $\lambda_{\max}(\mathbf{A})$ – наибольшее собственное значение \mathbf{A} . Интуитивно понять причину этого результата можно, представив себя мяч, скатывающийся в долину. Мы хотим, чтобы шаг не превышал угловой коэффициент в направлении самого крутого спуска, а именно это и измеряет наибольшее собственное значение (см. раздел 3.2.2).

Вообще, соблюдение условия $\rho < 2/L$, где L – постоянная Липшица градиента (раздел 8.1.4), гарантирует сходимость. Поскольку эта постоянная в общем случае неизвестна, обычно приходится адаптировать размер шага, и ниже мы обсудим такую методику.

8.2.2.2. Линейный поиск

Для нахождения оптимального размера шага можно поискать значение, которое максимально уменьшает целевую функцию в выбранном направлении, для чего решить одномерную задачу оптимизации:

$$\rho_t = \underset{\rho > 0}{\operatorname{argmin}} \phi_t(\rho) = \underset{\rho > 0}{\operatorname{argmin}} \mathcal{L}(\boldsymbol{\theta}_t + \rho \mathbf{d}_t). \quad (8.21)$$

Это называется **линейным поиском**, потому мы ищем вдоль прямой, определенной вектором \mathbf{d}_t .

Если функция потерь выпуклая, то эта подзадача также выпуклая, потому что $\phi_t(\rho) = \mathcal{L}(\boldsymbol{\theta}_t + \rho \mathbf{d}_t)$ – выпуклая функция от аффинной функции ρ при фиксированных $\boldsymbol{\theta}_t$ и \mathbf{d}_t . Например, рассмотрим квадратичную потерю:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2} \boldsymbol{\theta}^T \mathbf{A} \boldsymbol{\theta} + \mathbf{b}^T \boldsymbol{\theta} + c. \quad (8.22)$$

Вычисление производной ϕ дает:

$$\frac{d\phi(\rho)}{d\rho} = \frac{d}{d\rho} \left[\frac{1}{2} (\boldsymbol{\theta} + \rho \mathbf{d})^T \mathbf{A} (\boldsymbol{\theta} + \rho \mathbf{d}) + \mathbf{b}^T (\boldsymbol{\theta} + \rho \mathbf{d}) + c \right] \quad (8.23)$$

$$= \mathbf{d}^T \mathbf{A} (\boldsymbol{\theta} + \rho \mathbf{d}) + \mathbf{d}^T \mathbf{b} \quad (8.24)$$

$$= \mathbf{d}^T (\mathbf{A} \boldsymbol{\theta} + \mathbf{b}) + \rho \mathbf{d}^T \mathbf{A} \mathbf{d}. \quad (8.25)$$

Решая уравнение $d\phi(\rho)/d\rho = 0$, получаем:

$$\rho = - \frac{\mathbf{d}^T (\mathbf{A} \boldsymbol{\theta} + \mathbf{b})}{\mathbf{d}^T \mathbf{A} \mathbf{d}}. \quad (8.26)$$

Использование оптимального размера шага называется **точным линейным поиском**. Но обычно такой точности не требуется. Существует несколько методов, например **метод поиска с возвратом Армихо**, которые пытаются обеспечить достаточное уменьшение целевой функции, не тратя слишком много времени на решение уравнения (8.21). В частности, мы можем начать с текущего размера шага (или какого-нибудь максимального значения), а затем уменьшать его, умножая на $0 < \beta < 1$ на каждой итерации, пока не будет выполнено условие Армихо–Голдстейна:

$$\mathcal{L}(\boldsymbol{\theta}_t + \rho \mathbf{d}_t) \leq \mathcal{L}(\boldsymbol{\theta}_t) + c \rho \mathbf{d}_t^T \nabla \mathcal{L}(\boldsymbol{\theta}_t), \quad (8.27)$$

где $c \in [0, 1]$ – постоянная, обычно $c = 10^{-4}$. На практике начальное значение линейного поиска и конкретный алгоритм возврата могут заметно влиять на качество алгоритма. Детали см. в [NW06, раздел 3.1].

8.2.3. Скорость сходимости

Нас интересуют алгоритмы оптимизации, которые быстро сходятся к (локальному) оптимуму. Для некоторых выпуклых задач, в которых градиент ограничен постоянной Липшица, можно показать, что градиентный спуск сходится с **линейной скоростью**. Это означает, что существует такое число $0 < \mu < 1$, что

$$|\mathcal{L}(\boldsymbol{\theta}_{t+1}) - \mathcal{L}(\boldsymbol{\theta}_*)| \leq \mu |\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}_*)|. \quad (8.28)$$

Здесь μ называется **скоростью сходимости**.

Для некоторых простых задач скорость сходимости можно вычислить в явном виде. Например, рассмотрим квадратичную целевую функцию $\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^T \mathbf{A} \boldsymbol{\theta} + \mathbf{b}^T \boldsymbol{\theta} + c$, где $\mathbf{A} \succ 0$. Предположим, что используется метод скорейшего спуска с точным линейным поиском. Можно показать (см., например, [Ber15]), что скорость сходимости равна:

$$\mu = \left(\frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} \right)^2, \quad (8.29)$$

где λ_{\max} – наибольшее, а λ_{\min} – наименьшее собственное значение \mathbf{A} . Это можно переписать в виде $\mu = \left(\frac{\kappa - 1}{\kappa + 1} \right)^2$, где $\kappa = \lambda_{\max}/\lambda_{\min}$ – число обусловленности \mathbf{A} . Интуитивно понятно, что число обусловленности измеряет степень «асимметрии» поверхности, т. е. ее отличия от симметричной «миски» (дополнительные сведения о числах обусловленности см. в разделе 7.1.4.4).

На рис. 8.12 показано влияние числа обусловленности на скорость сходимости. Слева приведен пример, в котором $\mathbf{A} = [20, 5; 5, 2]$, $\mathbf{b} = [-14; -6]$ и $c = 10$, так что $\kappa(\mathbf{A}) = 30.234$. В примере справа $\mathbf{A} = [20, 5; 5, 16]$, $\mathbf{b} = [-14; -6]$ и $c = 10$, так что $\kappa(\mathbf{A}) = 1.8541$. Мы видим, что метод скорейшего спуска сходится гораздо быстрее для задачи с меньшим числом обусловленности.

В более общем случае неквадратичных функций целевая функция часто бывает локально квадратичной в окрестности локального оптимума. Поэтому скорость сходимости зависит от числа обусловленности гессиана, $\kappa(\mathbf{H})$, в этой точке. Нередко скорость сходимости можно улучшить, оптимизируя на каждом шаге суррогатную целевую функцию (или модель), гессиан которой близок к гессиану истинной целевой функции (см. обсуждение в разделе 8.3).

Хотя линейный поиск работает хорошо, на рис. 8.12 видно, что траектория скорейшего спуска с точным линейным поиском **зигзагообразная**, что неэффективно. Эту проблему можно решить с помощью метода **сопряженных градиентов** (см., например, [She94]).

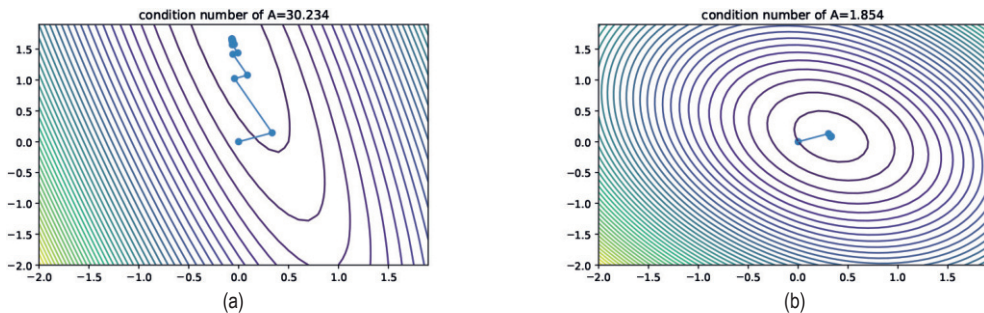


Рис. 8.12 ❖ Влияние числа обусловленности κ на скорость сходимости скорейшего спуска с точным линейным поиском. (а) Большое κ . (б) Малое κ . Построено программой по адресу figures.problai.com/book1/8.12

8.2.4. Метод импульса

Градиентный спуск может продвигаться очень медленно на плоских участках поверхности функции потерь, как показано на рис. 8.11. Ниже мы обсудим некоторые решения этой проблемы.

8.2.4.1. Импульс

Простая эвристика – метод **тяжелого шарика**, или метод **импульса** [Ber99], – заключается в том, чтобы продвигаться быстрее в направлениях, которые раньше оказались хорошими, и замедлять движение в направлениях, где градиент внезапно изменился, – как ведет себя шарик, скатывающийся по склону. Это можно реализовать следующим образом:

$$\mathbf{m}_t = \beta \mathbf{m}_{t-1} + \mathbf{g}_{t-1}; \quad (8.30)$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \rho_t \mathbf{m}_t, \quad (8.31)$$

где \mathbf{m}_t – **импульс** (масса, умноженная на скорость) и $0 < \beta < 1$. Типичное значение β равно 0.9. При $\beta = 0$ метод сводится к градиентному спуску.

Мы видим, что \mathbf{m}_t напоминает экспоненциально взвешенное скользящее среднее прошлых градиентов (см. раздел 4.4.2.2):

$$\mathbf{m}_t = \beta \mathbf{m}_{t-1} + \mathbf{g}_{t-1} = \beta^2 \mathbf{m}_{t-2} + \beta \mathbf{g}_{t-2} + \mathbf{g}_{t-1} = \cdots = \sum_{\tau=0}^{t-1} \beta^\tau \mathbf{g}_{t-\tau-1}. \quad (8.32)$$

Если все прошлые градиенты постоянны, скажем равны \mathbf{g} , то выражение упрощается:

$$\mathbf{m}_t = \mathbf{g} \sum_{\tau=0}^{t-1} \beta^\tau. \quad (8.33)$$

Масштабный коэффициент представляет собой геометрическую прогрессию, сумма которой в пределе равна:

Это объясняет, почему ускоренный метод градиентов Нестерова иногда называют импульсом Нестерова, а также показывает, из-за чего он может оказаться быстрее стандартного метода импульса: вектор импульса уже указывает приблизительно в правильном направлении, поэтому измерение градиента в новой точке, $\theta_t + \beta m_t$, может оказаться точнее, чем в текущей точке θ_t .

Можно доказать, что ускоренный метод градиентов Нестерова быстрее метода скорейшего спуска для выпуклых функций, когда β и ρ_t выбираются правильно. Он называется «ускоренным» именно из-за этой улучшенной скорости сходимости, оптимальной для градиентных методов, в которых используется только информация первого порядка, когда целевая функция выпукла и ее градиенты липшицевы. На практике, однако, метод Нестерова может оказаться медленнее скорейшего спуска и даже бывает неустойчивым, если β или ρ_t заданы неудачно.

8.3. Методы второго порядка

Алгоритмы оптимизации, в которых используется только градиент, называются методами **первого порядка**. Их достоинство в том, что вычисление и хранение градиента обходится дешево, но они не учитывают кривизну поверхности, а потому могут сходиться медленно, как было показано на рис. 8.12. Методы оптимизации **второго порядка** включают в рассмотрение кривизну разными способами (например, с помощью гессиана), что может ускорить сходимость. Некоторые из этих методов мы обсудим ниже.

8.3.1. Метод Ньютона

Классическим методом второго порядка является **метод Ньютона**. Обновление в нем имеет вид

$$\theta_{t+1} = \theta_t - \rho_t \mathbf{H}_t^{-1} \mathbf{g}_t, \quad (8.39)$$

где предполагается, что матрица

$$\mathbf{H}_t \triangleq \nabla^2 \mathcal{L}(\theta)|_{\theta_t} = \nabla^2 \mathcal{L}(\theta_t) = \mathbf{H}(\theta_t) \quad (8.40)$$

положительно определенная и, следовательно, обновление определено корректно. Псевдокод метода Ньютона приведен в алгоритме 1. Интуитивно понятно, почему он работает быстрее градиентного спуска: обратная матрица \mathbf{H}^{-1} «исправляет» асимметрию в локальной кривизне, преобразуя топологию, показанную на рис. 8.12a, в топологию на рис. 8.12b.

Этот алгоритм можно вывести следующим образом. Разложим $\mathcal{L}(\theta)$ в ряд Тейлора с точностью до двух членов в окрестности θ_t :

$$\mathcal{L}_{\text{quad}}(\theta) = \mathcal{L}(\theta_t) + \mathbf{g}_t^\top (\theta - \theta_t) + \frac{1}{2} (\theta - \theta_t)^\top \mathbf{H}_t (\theta - \theta_t). \quad (8.41)$$

Алгоритм 1. Метод Ньютона минимизации функции

```

1  Инициализировать  $\theta_0$ ;
2  for  $t = 1, 2, \dots$  до достижения сходимости do
3      Вычислить  $\mathbf{g}_t = \nabla \mathcal{L}(\theta_t)$ ;
4      Вычислить  $\mathbf{H}_t = \nabla^2 \mathcal{L}(\theta_t)$ ;
5      Решить уравнение  $\mathbf{H}_t \mathbf{d}_t = -\mathbf{g}_t$  относительно  $\mathbf{d}_t$ ;
6      Воспользоваться линейным поиском для нахождения размера шага  $\rho_t$  вдоль  $\mathbf{d}_t$ ;
7       $\theta_{t+1} = \theta_t + \rho_t \mathbf{d}_t$ ;

```

Минимум $\mathcal{L}_{\text{quad}}$ достигается в точке:

$$\theta = \theta_t - \mathbf{H}_t^{-1} \mathbf{g}_t. \quad (8.42)$$

Таким образом, если квадратичная аппроксимация приемлема, то следует выбирать в качестве направления спуска $\mathbf{d}_t = -\mathbf{H}_t^{-1} \mathbf{g}_t$ (см. иллюстрацию на рис. 8.14а). Заметим, что в «чистом» методе Ньютона размер шага $\rho_t = 1$. Однако ничто не мешает использовать линейный поиск для нахождения наилучшего размера шага; это дает более устойчивый алгоритм, потому что при $\rho_t = 1$ он не всегда сходится глобально.

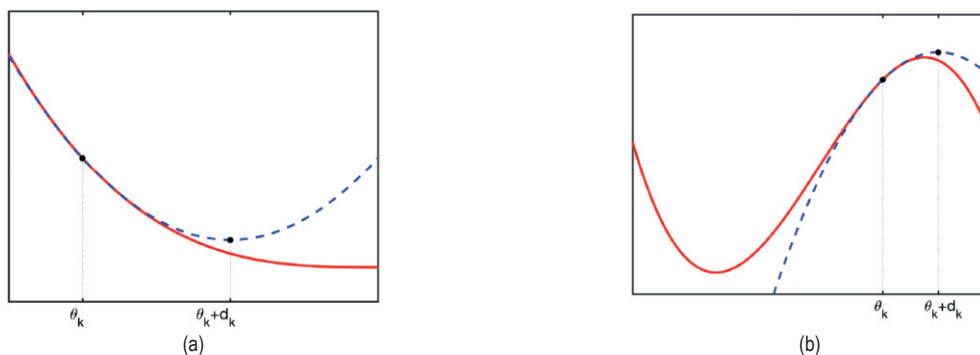


Рис. 8.14 ❖ Применение метода Ньютона для минимизации функции одной переменной. (а) Сплошная кривая – график функции $\mathcal{L}(x)$. Пунктирная кривая $\mathcal{L}_{\text{quad}}(\theta)$ – аппроксимация ее второй производной в точке θ_t . Приращение d_t необходимо прибавить к θ_t для приближения к минимуму $\mathcal{L}_{\text{quad}}(\theta)$. На основе рис. 13.4 из работы [Van06]. Построено программой по адресу figures.problm.ai/book1/8.14. (б) Применение метода Ньютона к невыпуклой функции. Мы аппроксимируем исходную функцию квадратичной в окрестности текущей точки θ_t и приближаемся к ее стационарной точке $\theta_{t+1} = \theta_t + d_t$. К сожалению, это приводит нас к локальному максимуму, а не минимуму f . Это означает, что нужно аккуратнее выбирать область квадратичной аппроксимации. На основе рис. 13.11 из работы [Van06]. Построено программой по адресу figures.problm.ai/book1/8.14

Если применить этот метод к линейной регрессии, то мы придем к оптимуму за один шаг, потому что (как показано в разделе 11.2.2.1) $\mathbf{H} = \mathbf{X}^T \mathbf{X}$ и $\mathbf{g} = \mathbf{X}^T \mathbf{X} \mathbf{w} - \mathbf{X}^T \mathbf{y}$, так что обновление Ньютона принимает вид

$$\begin{aligned} \mathbf{w}_1 &= \mathbf{w}_0 - \mathbf{H}^{-1} \mathbf{g} = \mathbf{w}_0 - (\mathbf{X}^T \mathbf{X})^{-1} (\mathbf{X}^T \mathbf{X} \mathbf{w}_0 - \mathbf{X}^T \mathbf{y}) \\ &= \mathbf{w}_0 - \mathbf{w}_0 + (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}, \end{aligned} \quad (8.43)$$

что является оценкой по методу обыкновенных наименьших квадратов. Однако в случае применения этого метода к логистической регрессии может потребоваться несколько итераций для сходимости к глобальному оптимуму (мы обсудим это в разделе 10.2.6).

8.3.2. BFGS и другие квазиньютоновские методы

Квазиньютоновские методы, иногда называемые методами с **переменной метрикой**, итеративно строят приближение к гессиану, используя информацию, почерпнутую из вектора градиента на каждом шаге. Самый известный из таких методов – **BFGS** (названный по фамилиям авторов, Бройден, Флетчер, Гольдфарб и Шанно), в нем аппроксимация гессиана $\mathbf{B}_t \approx \mathbf{H}_t$ имеет вид:

$$\mathbf{B}_{t+1} = \mathbf{B}_t + \frac{\mathbf{y}_t \mathbf{y}_t^T}{\mathbf{y}_t^T \mathbf{s}_t} - \frac{(\mathbf{B}_t \mathbf{s}_t)(\mathbf{B}_t \mathbf{s}_t)^T}{\mathbf{s}_t^T \mathbf{B}_t \mathbf{s}_t}; \quad (8.45)$$

$$\mathbf{s}_t = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}; \quad (8.45)$$

$$\mathbf{y}_t = \mathbf{g}_t - \mathbf{g}_{t-1}. \quad (8.46)$$

Это обновление ранга 2 матрицы. Если \mathbf{B}_0 положительно определена и размер шага ρ выбран с помощью линейного поиска и удовлетворяет как условию Армихо (8.27), так и следующему условию на кривизну:

$$\nabla \mathcal{L}(\boldsymbol{\theta}_t + \rho \mathbf{d}_t) \geq c_2 \rho \mathbf{d}_t^T \nabla \mathcal{L}(\boldsymbol{\theta}_t), \quad (8.47)$$

– то \mathbf{B}_{t+1} останется положительно определенной. Постоянная c_2 выбирается из интервала $(c, 1)$, где c – настраиваемый параметр в формуле (8.27). Оба условия на величину шага называются **условиями Вульфа**. Обычно начинают с диагональной аппроксимации, $\mathbf{B}_0 = \mathbf{I}$. Таким образом, BFGS можно рассматривать как аппроксимацию гессиана в виде «диагональная матрица плюс матрица низкого ранга».

В BFGS можно вместо этого итеративно обновлять аппроксимацию обратного гессиана, $\mathbf{C}_t \approx \mathbf{H}_t^{-1}$, следующим образом:

$$\mathbf{C}_{t+1} = \left(\mathbf{I} - \frac{\mathbf{s}_t \mathbf{y}_t^T}{\mathbf{y}_t^T \mathbf{s}_t} \right) \mathbf{C}_t \left(\mathbf{I} - \frac{\mathbf{y}_t \mathbf{s}_t^T}{\mathbf{y}_t^T \mathbf{s}_t} \right) + \frac{\mathbf{s}_t \mathbf{s}_t^T}{\mathbf{y}_t^T \mathbf{s}_t}. \quad (8.48)$$

Поскольку для хранения аппроксимации гессиана по-прежнему нужна память объемом $O(D^2)$, при решении очень больших задач можно применять алгоритм **BFGS с ограниченной памятью**, или **L-BFGS**, в котором ранг аппроксимации контролируется посредством использования только M последних пар $(\mathbf{s}_t, \mathbf{y}_t)$, а более старая информация игнорируется. Вместо того чтобы хранить \mathbf{B}_t явно, мы просто храним эти векторы в памяти, а затем аппроксимируем $\mathbf{H}_t^{-1} \mathbf{g}_t$, вычисляя последовательность скалярных про-

изведений с хранимыми векторами \mathbf{s}_t и \mathbf{y}_t . Тогда размер потребной памяти уменьшается до $O(MD)$. Как правило, выбор M в диапазоне 5–20 достаточен для хорошей производительности [NW06, стр. 177].

Отметим, что в библиотеке `sklearn` LBFGS является решателем по умолчанию для задач логистической регрессии¹.

8.3.3. Методы на основе доверительных областей

Если целевая функция невыпуклая, то гессиан \mathbf{H}_t может не быть положительно определенным, так что $\mathbf{d}_t = -\mathbf{H}_t^{-1}\mathbf{g}_t$ необязательно является направлением спуска. Для функции одной переменной это показано на рис. 8.14b – как видим, метод Ньютона может находить локальный максимум вместо локального минимума.

Вообще говоря, когда квадратичная аппроксимация в методе Ньютона недопустима, мы сталкиваемся с проблемой. Однако обычно существует локальная область вокруг текущей итерации, где целевую функцию можно безопасно аппроксимировать квадратичной. Обозначим эту область \mathcal{R}_t , а модель (или аппроксимацию) целевой функции – $M(\delta)$, где $\delta = \theta - \theta_t$. Тогда на каждом шаге мы можем решить задачу:

$$\delta^* = \operatorname{argmin}_{\delta \in \mathcal{R}_t} M_t(\delta). \quad (8.49)$$

Это называется **оптимизацией в доверительной области**. (Это можно считать «противоположностью» линейному поиску в том смысле, что мы сначала выбираем расстояние, на которое хотим сместиться, определяемое \mathcal{R}_t , и затем находим оптимальное направление, а не наоборот – сначала выбрать направление, а затем найти оптимальное расстояние.)

Обычно предполагается, что $M_t(\delta)$ является квадратичной аппроксимацией:

$$M_t(\delta) = \mathcal{L}(\theta_t) + \mathbf{g}_t^\top \delta + \frac{1}{2} \delta^\top \mathbf{H}_t \delta, \quad (8.50)$$

где $\mathbf{g}_t = \nabla_{\theta} \mathcal{L}(\theta)|_{\theta_t}$ – градиент, а $\mathbf{H}_t = \nabla_{\theta}^2 \mathcal{L}(\theta)|_{\theta_t}$ – гессиан. Кроме того, обычно предполагается, что \mathcal{R}_t – шар радиуса r , т. е. $\mathcal{R}_t = \{\delta : \|\delta\|_2 \leq r\}$. При таких условиях мы можем преобразовать условную оптимизацию в безусловную следующим образом:

$$\delta^* = \operatorname{argmin}_{\delta} M(\delta) + \lambda \|\delta\|_2^2 = \operatorname{argmin}_{\delta} \mathbf{g}^\top \delta + \frac{1}{2} \delta^\top (\mathbf{H} + \lambda \mathbf{I}) \delta \quad (8.51)$$

для некоторого множителя Лагранжа $\lambda > 0$, зависящего от радиуса r (см. обсуждение множителей Лагранжа в разделе 8.5.1). Эту задачу можно решить, взяв

$$\delta = -(\mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{g}. \quad (8.52)$$

¹ См. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

Это называется **регуляризацией Тихонова**, см. иллюстрацию на рис. 8.15.

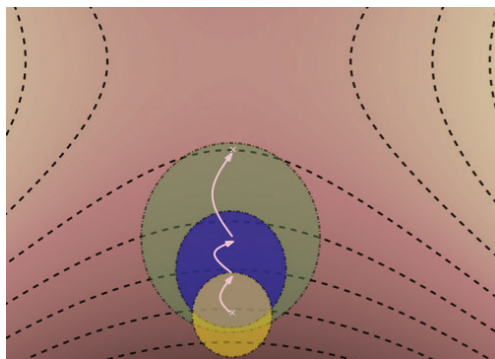


Рис. 8.15 ❖ Подход на основе доверительной области. Пунктирные кривые – это линии уровня исходной невыпуклой целевой функции. Кругами представлены последовательные квадратичные аппроксимации. На основе рис. 4.2 из работы [Pas14]. Печатается с разрешения Развана Паскану

Заметим, что если прибавить к \mathbf{H} матрицу $\lambda \mathbf{I}$ с достаточно большим λ , то результирующая матрица гарантированно будет положительно определенной. При $\lambda \rightarrow 0$ этот метод сводится к методу Ньютона, но при достаточно больших λ все отрицательные собственные значения станут положительными (а все нулевые – равными λ).

8.4. СТОХАСТИЧЕСКИЙ ГРАДИЕНТНЫЙ СПУСК

В этом разделе мы рассмотрим **стохастическую оптимизацию**, цель которой – минимизировать среднее значение функции

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{z})}[\mathcal{L}(\boldsymbol{\theta}, \mathbf{z})], \quad (8.53)$$

где \mathbf{z} – случайная величина, являющаяся аргументом целевой функции. Это может быть «шум», вносимый окружающей средой, или обучающий пример, случайно выбранный из обучающего набора, как будет объяснено ниже.

Предполагается, что на каждой итерации наблюдается $\mathcal{L}_t(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}, \mathbf{z}_t)$, где $\mathbf{z}_t \sim q$. Также предполагается, что имеется способ вычисления несмещенной оценки градиента \mathcal{L} . Если распределение $q(\mathbf{z})$ не зависит от оптимизируемых параметров, то можно взять $\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} \mathcal{L}_t(\boldsymbol{\theta}_t)$. В таком случае алгоритм можно записать следующим образом:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \rho_t \nabla \mathcal{L}(\boldsymbol{\theta}_t, \mathbf{z}_t) = \boldsymbol{\theta}_t - \rho_t \mathbf{g}_t. \quad (8.54)$$

Этот метод называется **стохастическим градиентным спуском**, или **СГС** (англ. SGD). При условии, что оценка градиента несмещенная, он сходится к стационарной точке, коль скоро размер шага ρ_t уменьшается с некоторой скоростью (см. обсуждение в разделе 8.4.3).

8.4.1. Приложение к задачам с конечной суммой

СГС применяется в машинном обучении очень широко. Чтобы понять, почему это так, вспомним (см. раздел 4.3), что многие процедуры обучения моделей основаны на минимизации эмпирического риска, что подразумевает минимизацию следующей функции потерь:

$$\mathcal{L}(\boldsymbol{\theta}_t) = \frac{1}{N} \sum_{n=1}^N \ell(\mathbf{y}_n, f(\mathbf{x}_n; \boldsymbol{\theta}_t)) = \frac{1}{N} \sum_{n=1}^N \mathcal{L}_n(\boldsymbol{\theta}_t). \quad (8.55)$$

Это называется **задачей с конечной суммой**. Градиент такой целевой функции имеет вид:

$$\mathbf{g}_t = \frac{1}{N} \sum_{n=1}^N \nabla_{\boldsymbol{\theta}} \mathcal{L}_n(\boldsymbol{\theta}_t) = \frac{1}{N} \sum_{n=1}^N \nabla_{\boldsymbol{\theta}} \ell(\mathbf{y}_n, f(\mathbf{x}_n; \boldsymbol{\theta}_t)). \quad (8.56)$$

Для его вычисления необходимо просуммировать по всем N обучающим примерам, а при больших N это может занять много времени. По счастью, можно аппроксимировать эту процедуру, выбрав **мини-пакет**, содержащий $B \ll N$ примеров, и вычислив

$$\mathbf{g}_t \approx \frac{1}{|B_t|} \sum_{n \in B_t} \nabla_{\boldsymbol{\theta}} \mathcal{L}_n(\boldsymbol{\theta}_t) = \frac{1}{|B_t|} \sum_{n \in B_t} \nabla_{\boldsymbol{\theta}} \ell(\mathbf{y}_n, f(\mathbf{x}_n; \boldsymbol{\theta}_t)), \quad (8.57)$$

где B_t – множество случайно выбранных примеров, используемых на итерации t^1 . Это несмещенная аппроксимация эмпирического среднего (8.56). Следовательно, мы можем безопасно использовать ее вместе с СГС.

Хотя теоретическая скорость сходимости СГС меньше, чем для пакетного ГС (в частности, скорость сходимости СГС сублинейна), на практике СГС обычно работает быстрее, т. е. время выполнения одного шага значительно меньше [BV08; BV11]. Чтобы понять, почему СГС может продвигаться к цели быстрее, чем градиентный спуск на полном пакете, предположим, что имеется набор данных, состоящий всего из одного примера, повторенного K раз. Время обучения на таком пакете будет по крайней мере в K раз больше, чем для СГС, потому что будет расходоваться на вычисление градиента для повторяющихся примеров. Даже при отсутствии дубликатов, пакетное обучение может оказаться расточительным, так как на ранних этапах обучения у параметров еще нет хороших оценок, так что тратить время на точное вычисление градиента не имеет смысла.

¹ На практике обычно применяется выборка B_t без возвращения. Однако по исчерпании набора данных (т. е. после одной эпохи обучения) можно случайно перетасовать примеры, чтобы все мини-пакеты в следующей эпохе отличались от предыдущих. Этот вариант СГС анализируется в работе [HS19].

8.4.2. Пример: СГС для обучения модели линейной регрессии

В этом разделе мы покажем, как использовать СГС для обучения модели линейной регрессии. Напомним (см. раздел 4.2.7), что целевая функция имеет вид:

$$\mathcal{L}(\theta) = \frac{1}{2N} \sum_{n=1}^N (\mathbf{x}_n^T \theta - y_n)^2 = \frac{1}{2N} \|\mathbf{X}\theta - \mathbf{y}\|_2^2. \quad (8.58)$$

Градиент равен:

$$\mathbf{g}_t = \frac{1}{N} \sum_{n=1}^N (\theta_n^T \mathbf{x}_n - y_n) \mathbf{x}_n. \quad (8.59)$$

Теперь рассмотрим применение СГС с размером мини-пакета $B = 1$. Обновление принимает вид

$$\theta_{t+1} = \theta_t - \rho_t (\theta_t^T \mathbf{x}_n - y_n) \mathbf{x}_n, \quad (8.60)$$

где $n = n(t)$ – индекс примера, выбранного на итерации t . Алгоритм в целом называется алгоритмом **наименьших средних квадратов** (least mean squares – **LMS**), а также **дельта-правилом**, или **правилом Видроу–Хоффа**.

На рис. 8.16 показаны результаты применения этого алгоритма к данным на рис. 11.2. Алгоритм начинает работу с $\theta = (-0.5, 2)$ и сходится (в том смысле, что величина $\|\theta_t - \theta_{t-1}\|_2^2$ становится ниже порога 10^{-2}) примерно за 26 итераций. Заметим, что СГС (а значит, и LMS) может потребовать нескольких проходов по данным для нахождения минимума.

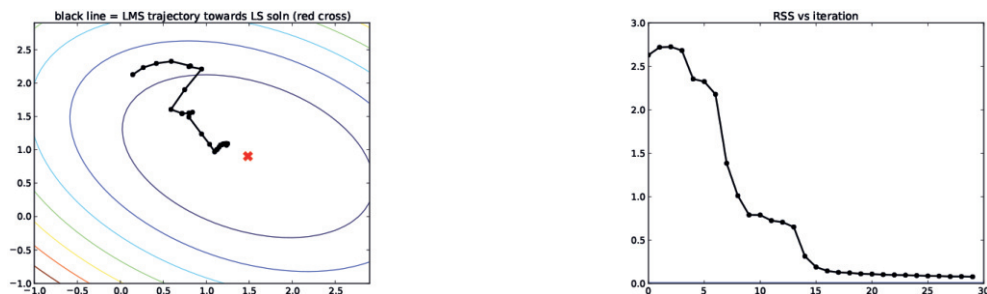


Рис. 8.16 ❖ Алгоритм LMS. Слева: мы начинаем с $\theta = (-0.5, 2)$ и медленно сходимся к решению по методу наименьших квадратов $\hat{\theta} = (1.45, 0.93)$ (красный крестик). Справа: график зависимости целевой функции от времени. Отметим, что она не убывает монотонно. Построено программой по адресу figures.problm.ai/book1/8.16

8.4.3. Выбор размера шага (скорости обучения)

При использовании СГС нужно осторожно выбирать скорость обучения, необходимую для сходимости. Так, на рис. 8.17 приведен график зависимости потери от скорости обучения в случае применения СГС к обучению классификатора на основе нейронной сети (детали см. в главе 13). Мы видим, что кривая имеет U-образную форму: при слишком малой скорости обучения модель оказывается недообученной, а при слишком большой – неустойчивой (ср. с рис. 8.11b); в обоих случаях она не сходится к локальному оптимуму.

Для выбора хорошей скорости обучения можно применить эвристику, предложенную в работе [Smi18], – начать с малой скорости и постепенно увеличивать ее, оценивая качество на небольшом числе мини-пакетов. Затем строится график типа показанного на рис. 8.17 и выбирается скорость обучения, которой соответствует наименьшая потеря. (На практике лучше выбирать чуть меньшую скорость, чтобы гарантировать устойчивость.)

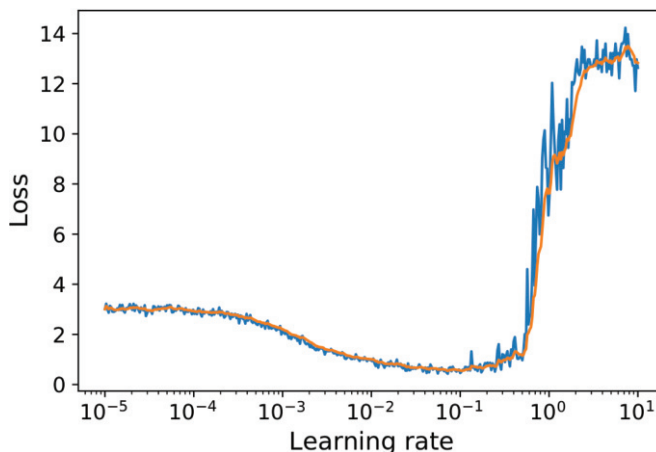


Рис. 8.17 ❖ Зависимость потери от скорости обучения (по горизонтальной оси). График зависимости потери на обучающем наборе от скорости обучения небольшого многослойного перцептрона на наборе данных FashionMNIST с применением стандартного СГС. (Исходная потеря показана синим цветом, сглаженная по методу EWMA – оранжевым.) Построено программой по адресу figures.probml.ai/book1/8.17

Вместо того чтобы выбирать одну постоянную скорость обучения, можно использовать **план изменения скорости обучения**, предусматривающий корректировку размера шага со временем. Теоретически СГС будет сходиться, если план скоростей обучения удовлетворяет **условиям Роббинса–Монро**:

$$\rho_t \rightarrow 0, \quad \frac{\sum_{t=1}^{\infty} \rho_t^2}{\sum_{t=1}^{\infty} \rho_t} \rightarrow 0. \quad (8.61)$$

Ниже перечислено несколько типичных планов скоростей обучения:

$$\rho_t = \rho_i, \text{ если } t_i \leq t \leq t_{i+1} \text{ кусочно-постоянный;} \quad (8.62)$$

$$\rho_t = \rho_0 e^{-\lambda t} \text{ с экспоненциальным затуханием;} \quad (8.63)$$

$$\rho_t = \rho_0 (\beta t + 1)^{-\alpha} \text{ с полиномиальным затуханием.} \quad (8.64)$$

Для кусочно-постоянного плана t_i – множество моментов времени, в которые изменяется скорость обучения. Например, можно положить $\rho_i = \rho_0 \gamma^i$, так что начальная скорость обучения будет уменьшаться в γ раз при прохождении каждого порога. На рис. 8.18а это показано для $\rho_0 = 1$ и $\gamma = 0.9$. Это называется **ступенчатым затуханием**. Иногда моменты смены скорости вычисляются адаптивно, для чего оценивается момент выхода на плато потери на обучающем или контрольном наборе; это называется **уменьшением при выходе на плато**. Экспоненциальное затухание обычно уменьшает скорость обучения очень быстро, как показано на рис. 8.18б. Чаще всего выбирается полиномиальное затухание с $\alpha = 0.5$ и $\beta = 1$, как показано на рис. 8.18с; это соответствует **плану по закону квадратного корня**, $\rho_t = \rho_0 \frac{1}{\sqrt{t+1}}$.

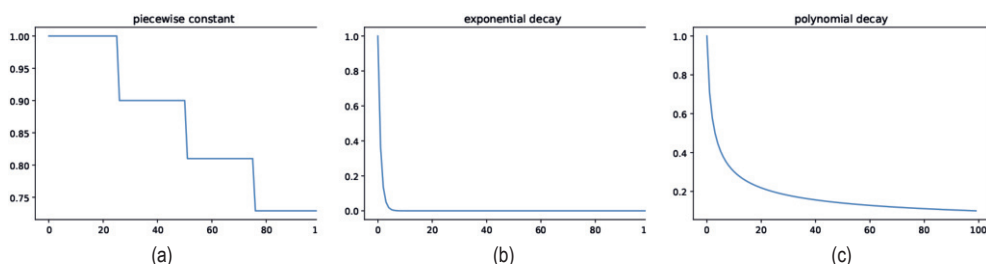


Рис. 8.18 ❖ Некоторые широко распространенные планы скоростей обучения. (а) Кусочно-постоянный. (б) Экспоненциальное затухание. (с) Полиномиальное затухание. Построено программой по адресу figures.problml.ai/book1/8.18

В глубоком обучении встречается еще один план – быстро увеличить скорость обучения, а затем постепенно снижать ее, как показано на рис. 8.19а. Это называется **прогревом скорости обучения** или **однофазным планом скоростей обучения** [Smi18]. Обоснование такое: в начале параметры могут находиться в плохо обусловленной части поверхности функции потерь, поэтому большой шаг может привести к чрезмерной «болтанке» (см. рис. 8.11б) и невозможности спуститься вниз по склону. Однако при медленной скорости обучения алгоритм может обнаружить более плоские области поверхности, где уместно использовать больший шаг. Оказавшись там, можно будет продвигаться к цели быстрее. Но, чтобы гарантировать сходимость к точке, мы должны уменьшить скорость обучения до 0. Более подробное обсуждение см. в [Got+19].

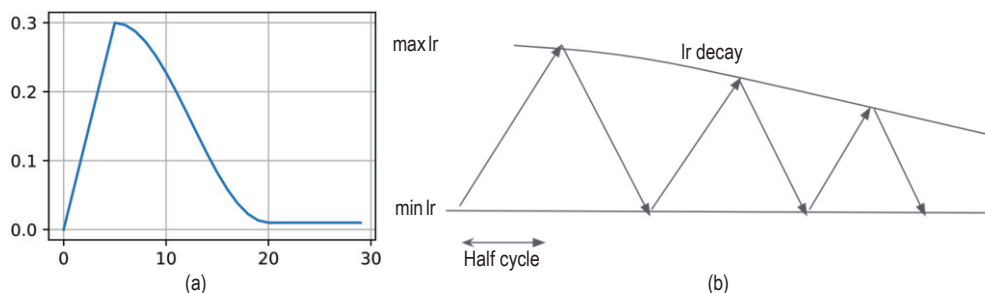


Рис. 8.19 ❖ (a) Линейный разогрев с косинусоидальным охлаждением.
(b) Циклический план скоростей обучения

Можно также увеличивать и уменьшать скорость обучения несколько раз, циклически. Этот **циклический план скоростей обучения** [Smi18] был популяризирован в курсе fast.ai. На рис. 8.19b показан его вариант с треугольной формой циклов. Обоснование заключается в том, что этот подход позволяет уйти от локальных минимумов. Минимальную и максимальную скорость обучения можно найти в ходе описанного выше начального «холодного прогона», а величину полупериода выбрать исходя из того, сколько перезапусков допускает бюджет времени, отведенного на обучение. Сходный подход, получивший название **стохастический градиентный спуск с теплым перезапуском**, был описан в работе [LH17], авторы которой предложили сохранять все контрольные точки, посещенные после каждого охлаждения, и использовать их в качестве членов ансамбля моделей (об ансамблевом обучении см. раздел 18.2).

Альтернативой использованию эвристик для оценки скорости обучения является линейный поиск (раздел 8.2.2.2). В случае СГС это непросто, потому что из-за зашумленных градиентов вычислить условие Армихо трудно [CS20]. Однако в работе [Vas+19] показано, что это можно сделать, если дисперсия шума градиента стремится к нулю со временем. Такое возможно, если модель настолько гибкая, что может идеально интерполировать обучающий набор.

8.4.4. Итеративное усреднение

Оценки параметров, порождаемые СГС, могут быть очень неустойчивыми во времени. Чтобы уменьшить дисперсию оценки, мы можем вычислить среднее:

$$\bar{\theta}_t = \frac{1}{t} \sum_{i=1}^t \theta_i = \frac{1}{t} \theta_t + \frac{t-1}{t} \bar{\theta}_{t-1}, \quad (8.65)$$

где θ_t – обычные итерации СГС. Это называется **итеративным усреднением**, или **усреднением Поляка–Рупперта** [Rup88].

В работе [PJ92] доказано, что оценка $\bar{\theta}_t$ дает наилучшую возможную асимптотическую скорость сходимости среди всех алгоритмов СГС, сопоставимую по качеству с методами, использующими информацию второго порядка, например гессианы. У такого усреднения могут быть и статистические преимущества. Например, в работе [NR18] доказано, что в случае линейной регрессии этот метод эквивалентен ℓ_2 -регуляризации (т. е. гребневой регрессии).

Вместо экспоненциального скользящего среднего в итерациях СГС в методе **стохастического усреднения весов** (Stochastic Weight Averaging – **SWA**) [Izm+18] используется обычное среднее в сочетании с модифицированным планом скоростей обучения. В отличие от стандартного усреднения Поляка–Рупперта, целью которого являлось повышение скорости сходимости, в SWA пологость в целевых функциях используется для обучения глубоких нейронных сетей с целью найти решения, допускающие лучшее обобщение.

8.4.5. Уменьшение дисперсии*

В этом разделе мы обсудим различные способы уменьшения дисперсии в СГС. Иногда это позволяет повысить теоретическую скорость сходимости от сублинейной до линейной (т. е. такой же, как в градиентном спуске с полным пакетом) [SLRB17; JZ13; DBLJ14]. Эти методы уменьшают дисперсию градиентов, а не самих параметров и предназначены для работы на задачах с конечной суммой.

8.4.5.1. SVRG

Основная идея **стохастического градиента с уменьшенной регрессией** (stochastic variance reduced gradient – **SVRG**) [JZ13] – использовать управляющую переменную для хранения оценки опорного значения градиента на основе полного пакета, с которой затем сравниваются стохастические градиенты.

Точнее, мы периодически (например, один раз за эпоху) вычисляем полный градиент в точке «моментального снимка» параметров модели, $\tilde{\theta}$; соответствующий «точный» градиент равен, следовательно, $\nabla \mathcal{L}(\tilde{\theta})$. На шаге t мы вычисляем обычный стохастический градиент при текущих параметрах, $\nabla \mathcal{L}_t(\theta_t)$, а также при параметрах снимка, $\nabla \mathcal{L}_t(\tilde{\theta})$, которые используются как эталон для сравнения. Затем можно использовать следующую улучшенную оценку градиента:

$$\mathbf{g}_t = \nabla \mathcal{L}_t(\theta_t) - \nabla \mathcal{L}_t(\tilde{\theta}) + \nabla \mathcal{L}(\tilde{\theta}) \quad (8.66)$$

для вычисления θ_{t+1} . Эта оценка несмещенная, так как $\mathbb{E}[\nabla \mathcal{L}_t(\tilde{\theta})] = \nabla \mathcal{L}(\tilde{\theta})$. Кроме того, для обновления нужно вычислить только два градиента, поскольку $\nabla \mathcal{L}(\tilde{\theta})$ достаточно вычислить один раз за эпоху. В конце эпохи мы обновляем параметры снимка $\tilde{\theta}$, опираясь на последнее значение θ_t или на скользящее среднее по всем итерациям, и обновляем ожидаемое опорное значение. (Снимки можно вычислять реже, но тогда опорное значение не

будет коррелировать с целевой функцией, что может негативно отразиться на качестве модели, как показано в работе [DB18].)

Итерации SVRG вычислительно быстрее, чем в случае градиентного спуска с полным пакетом, но тем не менее SVRG может достигать той же теоретической скорости сходимости, что ГС.

8.4.5.2. SAGA

В этом разделе мы опишем алгоритм **стохастического усредненного ускоренного градиента** (stochastic averaged gradient accelerated – **SAGA**) [DBL14]. В отличие от SVRG, он требует только одного вычисления градиента на полном пакете, в самом начале работы. Однако за экономию времени приходится расплачиваться дополнительным потреблением памяти. Точнее, необходимо хранить N векторов градиента. Это позволяет динамически пересчитывать аппроксимацию глобального градиента, удаляя старый локальный градиент из общей суммы и заменяя его новым локальным градиентом. Это называется методом **агрегированного градиента**.

Точнее, сначала мы инициализируем алгоритм, вычисляя $\mathbf{g}_n^{\text{local}} = \nabla \mathcal{L}_n(\theta_0)$ для всех n и среднее $\mathbf{g}^{\text{avg}} = (1/N) \sum_{n=1}^N \mathbf{g}_n^{\text{local}}$. Затем на t -й итерации мы используем оценку градиента

$$\mathbf{g}_t = \nabla \mathcal{L}_n(\theta_t) - \mathbf{g}_n^{\text{local}} + \mathbf{g}^{\text{avg}}, \quad (8.67)$$

где $n \sim \text{Unif}\{1, \dots, N\}$ – индекс примера. Далее мы выполняем обновление $\mathbf{g}_n^{\text{local}} = \nabla \mathcal{L}_n(\theta_t)$ и \mathbf{g}^{avg} , заменяя старое $\mathbf{g}_n^{\text{local}}$ новым значением.

Преимущество перед SVRG достигается за счет того, что нужно выполнить лишь один проход по всему пакету в самом начале. (На самом деле и начальный проход необязателен, потому что мы можем вычислять \mathbf{g}^{avg} «лениво», включая только те градиенты, которые уже видели ранее.) Недостатком является заметный дополнительный расход памяти. Однако если признаки (а значит, и градиенты) разрежены, то затраты на память приемлемы. В действительности алгоритм SAGA рекомендуется в `sklearn` для логистической регрессии, когда N велико, а \mathbf{x} разрежен¹.

8.4.5.3. Применение в глубоком обучении

Методы уменьшения дисперсии широко применяются для обучения моделей МО с выпуклыми целевыми функциями, например линейных моделей. Однако использование SVRG в практике традиционного глубокого обучения наталкивается на различные трудности. Например, нормировка пакета (раздел 14.2.4.1), приращение данных (раздел 19.1) и прореживание (раздел 13.5.4) нарушают предположения метода, так как различия в потере будут случайными и зависящими не только от параметров и индекса примера n . Дополнительные сведения см., например, в работах [DB18; Arn+19].

¹ См. https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression.

8.4.6. Предобусловленный СГС

В этом разделе мы рассмотрим **предобусловленный СГС** с таким обновлением:

$$\theta_{t+1} = \theta_t - \theta_t \mathbf{M}_t^{-1} \mathbf{g}_t, \quad (8.68)$$

где \mathbf{M}_t – **матрица предобусловливания**, или просто **предобусловливатель**, обычно выбирается положительно определенной. К сожалению, зашумленность оценок градиента затрудняет надежную оценку гесссиана, из-за чего становится сложно использовать методы, описанные в разделе 8.3. Кроме того, искать направление обновления для полной матрицы предобусловливания вычислительно дорого. Поэтому на практике чаще всего используют диагональный предобусловливатель \mathbf{M}_t . Такие предобусловливатели необязательно используют информацию второго порядка, но часто дают ускорение, сравнимое с настоящим СГС. (Вероятностную интерпретацию этих эвристик см. в [Roo+21].)

8.4.6.1. AdaGrad

Метод AdaGrad (сокращение от «adaptive gradient» – «адаптивный градиент»), описанный в работе [DHS11], первоначально был разработан для оптимизации выпуклых целевых функций, когда многие элементы вектора градиента равны нулю; они могут соответствовать признакам, редко присутствующим во входных данных, например редким словам. Обновление производится по следующей формуле:

$$\theta_{t+1,d} = \theta_{t,d} - \rho_t \frac{1}{\sqrt{s_{t,d} + \varepsilon}} g_{t,d}, \quad (8.69)$$

где

$$s_{t,d} = \sum_{i=1}^t g_{i,d}^2 \quad (8.70)$$

– сумма квадратов градиента, а $\varepsilon > 0$ – малый член, предотвращающий деление на ноль. Эквивалентно обновление можно записать в векторной форме:

$$\Delta \theta_t = -\rho_t \frac{1}{\sqrt{\mathbf{s}_t + \varepsilon}} \mathbf{g}_t, \quad (8.71)$$

где извлечение квадратного корня и деление производятся поэлементно. Если рассматривать этот метод как предобусловленный СГС, то это эквивалентно матрице $\mathbf{M}_t = \text{diag}(\mathbf{s}_t + \varepsilon)^{1/2}$. Это пример **адаптивной скорости обучения**; общий размер шага ρ_t все же нужно выбрать, но результат не так сильно зависит от него, как в стандартном СГС. Конкретно, обычно полагают $\rho_t = \rho_0$.

8.4.6.2. RMSProp и AdaDelta

Определяющим свойством AdaGrad является тот факт, что знаменатель растет со временем, поэтому эффективная скорость обучения падает. Это необходимо, чтобы гарантировать сходимость, но может негативно отразиться на производительности, если знаменатель убывает слишком быстро.

Альтернатива – использовать экспоненциально взвешенное скользящее среднее (EWMA, раздел 4.4.2.2) прошлых квадратов градиента:

$$\mathbf{s}_{t+1,d} = \beta \mathbf{s}_{t,d} + (1 - \beta) g_{t,d}^2. \quad (8.72)$$

На практике обычно берут $\beta \sim 0.9$, т. е. назначают больший вес недавним примерам. В этом случае

$$\sqrt{s_{t,d}} \approx \text{RMS}(\mathbf{g}_{1:t,d}) = \sqrt{\frac{1}{t} \sum_{\tau=1}^t g_{\tau,d}^2}, \quad (8.73)$$

где RMS означает «root mean squared» (корень из среднеквадратического значения). Поэтому и метод (основанный на более раннем методе **RPROP** из работы [RB93]) называется **RMSProp** [Hin14]. Обновление в RMSProp описывается формулой:

$$\Delta \boldsymbol{\theta}_t = -\rho_t \frac{1}{\sqrt{\mathbf{s}_t + \varepsilon}} \mathbf{g}_t. \quad (8.74)$$

Метод **AdaDelta** был независимо предложен в работе [Zei12] и похож на RMSProp. Однако в дополнение к накоплению EWMA градиентов в $\hat{\mathbf{s}}$ он еще и хранит EWMA обновлений $\boldsymbol{\delta}_t$, чтобы можно было использовать обновление вида

$$\Delta \boldsymbol{\theta}_t = -\rho_t \frac{\sqrt{\boldsymbol{\delta}_{t-1} + \varepsilon}}{\sqrt{\mathbf{s}_t + \varepsilon}} \mathbf{g}_t, \quad (8.75)$$

где

$$\boldsymbol{\delta}_t = \beta \boldsymbol{\delta}_{t-1} + (1 - \beta) (\Delta \boldsymbol{\theta}_t)^2, \quad (8.76)$$

а \mathbf{s}_t такое же, как в RMSProp. Преимуществом является тот факт, что «единицы измерения» числителя и знаменателя сокращаются, так что мы просто поэлементно умножаем градиент на скаляр. Это устраняет необходимость настраивать скорость обучения ρ_t , а значит, можно просто положить $\rho_t = 1$, хотя в популярных реализациях AdaDelta ρ_t все равно является настраиваемым гиперпараметром. Однако, поскольку эти адаптивные скорости обучения необязательно уменьшаются со временем (если только явно не выбирать ρ_t соответствующим образом), нельзя дать гарантию, что метод сойдется к решению.

8.4.6.3. Adam

Можно объединить RMSProp с методом моментов. Именно, вычислим EWMA градиентов (как в методе моментов) и квадраты градиентов (как в RMSProp):

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t; \quad (8.77)$$

$$\mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2. \quad (8.78)$$

Затем произведем такое обновление:

$$\Delta \boldsymbol{\theta}_t = -\rho_t \frac{1}{\sqrt{\mathbf{s}_t} + \varepsilon} \mathbf{m}_t. \quad (8.79)$$

Получающийся в итоге метод называется **Adam** (adaptive moment estimation – оценка адаптивного момента) [KB15]. Стандартные значения постоянных таковы: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-6}$. (Если положить $\beta_1 = 0$ и отказаться от корректировки смещения, то мы вернемся к RMSProp без использования момента). В качестве общей скорости обучения часто используют фиксированное значение, например $\rho_t = 0.001$. Как и в предыдущем случае, адаптивная скорость обучения необязательно уменьшается со временем, поэтому сходимость не гарантируется (см. раздел 8.4.6.4).

Если инициализировать $\mathbf{m}_0 = \mathbf{s}_0 = \mathbf{0}$, то начальные оценки будут смещены в сторону малых значений. Поэтому авторы рекомендуют использовать моменты со скорректированным смещением, которые увеличивают значения на ранних стадиях процесса оптимизации. Эти оценки имеют вид:

$$\hat{\mathbf{m}}_t = \mathbf{m}_t / (1 - \beta_1^t); \quad (8.80)$$

$$\hat{\mathbf{s}}_t = \mathbf{s}_t / (1 - \beta_2^t). \quad (8.81)$$

Преимущество корректировки смещения показано на рис. 4.3.

8.4.6.4. Проблемы, связанные с адаптивной скоростью обучения

При использовании методов диагонального масштабирования общая скорость обучения равна $\rho_0 \mathbf{M}_t^{-1}$ и изменяется со временем. Поэтому такие методы часто называют методами с **адаптивной скоростью обучения**. Однако они все равно требуют задания базовой скорости обучения ρ_0 .

Поскольку методы на основе EWMA обычно применяются в стохастической постановке, когда оценки градиентов зашумленные, адаптация скорости обучения в них может стать причиной расходимости даже в выпуклых задачах [RKK18]. Были предложены различные решения этой проблемы, в том числе AMSGrad [RKK18], Padam [CG18; Zho+18] и Yogi [Zah+18]. Например, в Yogi обновление отличается от выполняемого в Adam заменой

$$\mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2 = \mathbf{s}_{t-1} + (1 - \beta_2) (\mathbf{g}_t^2 - \mathbf{s}_{t-1}) \quad (8.82)$$

на

$$\mathbf{s}_t = \mathbf{s}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2 \odot \text{sgn}(\mathbf{g}_t^2 - \mathbf{s}_{t-1}). \quad (8.83)$$

8.4.6.5. Недиагональные матрицы предобуславливания

Хотя рассмотренные выше методы способны адаптировать скорость обучения каждого параметра, они не решают более фундаментальную проблему плохой обусловленности из-за корреляции параметров, а потому не всегда дают такое ускорение по сравнению со стандартным СГС, на которое мы рассчитывали.

Один из способов ускорить сходимость – воспользоваться следующей матрицей предобуславливания (соответствующий метод называется **полноматричным Adagrad** [DHS11]):

$$\mathbf{M}_t = [(\mathbf{G}_t \mathbf{G}_t^\top)^{\frac{1}{2}} + \varepsilon \mathbf{I}_D]^{-1}, \quad (8.84)$$

где

$$\mathbf{G}_t = [\mathbf{g}_t, \dots, \mathbf{g}_1]. \quad (8.85)$$

Здесь $\mathbf{g}_i = \nabla_{\mathbf{v}} c(\mathbf{v}_i)$ – D -мерный вектор градиента, вычисленный на шаге i . К сожалению, \mathbf{M}_t – матрица размера $D \times D$, поэтому ее хранение и обращение обходятся дорого.

В алгоритме **Shampoo** [GKS18] производится аппроксимация \mathbf{M} блочно-диагональной матрицей, по одному разу в каждом слое модели, а затем структура произведения Кронекера используется для ее эффективного обращения. (Метод называется «шампунь», потому что в нем используется «кондиционер» – обуславливатель). В недавней работе [Ani+20] этот метод был масштабирован и успешно применен для обучения очень больших глубоких моделей за рекордное время.

8.5. УСЛОВНАЯ ОПТИМИЗАЦИЯ

В этом разделе мы рассмотрим следующую **задачу условной оптимизации**:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \mathcal{C}}{\operatorname{argmin}} \mathcal{L}(\boldsymbol{\theta}), \quad (8.86)$$

где допустимое множество, или множество ограничений имеет вид:

$$\mathcal{C} = \{\boldsymbol{\theta} \in \mathbb{R}^D : h_i(\boldsymbol{\theta}) = 0, i \in \mathcal{E}, g_j(\boldsymbol{\theta}) \leq 0, j \in \mathcal{J}\}, \quad (8.87)$$

где \mathcal{E} – множество **ограничений в виде равенств**, а \mathcal{J} – множество **ограничений в виде неравенств**.

Например, пусть имеется квадратичная целевая функция $\mathcal{L}(\boldsymbol{\theta}) = \theta_1^2 + \theta_2^2$ с ограничением (связью) в виде линейного равенства $h(\boldsymbol{\theta}) = 1 - \theta_1 - \theta_2 = 0$.

На рис. 8.20(a) показаны линии уровня h , а также отрезок, представляющий поверхность связи. Мы пытаемся найти точку θ^* на этом отрезке, расположенную как можно ближе к началу координат. Из геометрии задачи ясно, что оптимальным решением является точка $\theta = (0.5, 0.5)$, обозначенная черным кружочком.

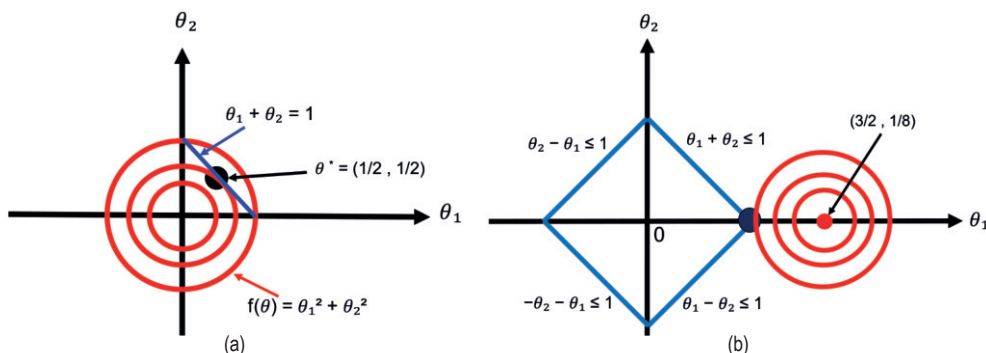


Рис. 8.20 ❖ Некоторые задачи условной оптимизации. Красные окружности – линии уровня целевой функции $\mathcal{L}(\theta)$. Оптимальное решение, удовлетворяющее ограничениям, показано черным кружочком. (a) Синий отрезок – ограничение в виде равенства $h(\theta) = 0$. (b) Синие отрезки описывают ограничения в виде неравенства $|\theta_1| + |\theta_2| \leq 1$ (сравните с рис.11.8 (слева))

В следующих разделах мы кратко опишем теорию и алгоритмы, лежащие в основе условной оптимизации. Дополнительные сведения можно найти в других книгах, например [BV04; NW06; Ber15; Ber16].

8.5.1. Множители Лагранжа

В этом разделе мы обсудим, как решать задачи оптимизации с ограничениями в виде равенств. Сначала предположим, что есть всего одно ограничение, $h(\theta) = 0$.

Прежде всего отметим, что для любой точки, лежащей на поверхности связи, вектор $\nabla h(\theta)$ будет ортогонален поверхности связи. Действительно, рассмотрим другую, близко расположенную точку $\theta + \varepsilon$, тоже лежащую на поверхности. Воспользовавшись разложением в ряд Тейлора с точностью до членов первого порядка в окрестности θ , получим

$$h(\theta + \varepsilon) \approx h(\theta) + \varepsilon^T \nabla h(\theta). \quad (8.88)$$

Поскольку и θ , и $\theta + \varepsilon$ лежат на поверхности связи, то должно быть $h(\theta) = h(\theta + \varepsilon)$, а значит, $\varepsilon^T \nabla h(\theta) \approx 0$. Так как ε параллелен поверхности связи, $\nabla h(\theta)$ должен быть перпендикулярен ей.

Мы ищем такую точку θ^* на поверхности связи, которая доставляет минимум $\mathcal{L}(\theta)$. Мы только что показали, что она должна удовлетворять условию ортогональности $\nabla h(\theta^*)$ к поверхности связи. Кроме того, для такой точки

$\nabla \mathcal{L}(\boldsymbol{\theta})$ тоже должен быть ортогонален поверхности связи, поскольку иначе мы могли бы уменьшить $\mathcal{L}(\boldsymbol{\theta})$, сдвинувшись на короткое расстояние вдоль поверхности связи. Так как $\nabla h(\boldsymbol{\theta})$ и $\nabla \mathcal{L}(\boldsymbol{\theta})$ ортогональны поверхности связи в точке $\boldsymbol{\theta}^*$, они должны быть параллельны (или антипараллельны) друг другу. Следовательно, должна существовать постоянная $\lambda^* \in \mathbb{R}$ такая, что

$$\nabla \mathcal{L}(\boldsymbol{\theta}^*) = \lambda^* \nabla h(\boldsymbol{\theta}^*). \quad (8.89)$$

(Мы не можем просто приравнять векторы градиентов, потому что у них могут быть разные модули.) Постоянная λ^* называется **множителем Лагранжа**, она может быть положительна, отрицательна или равна нулю. Последний случай имеет место, когда $\nabla f(\boldsymbol{\theta}^*) = 0$.

Мы можем следующим образом преобразовать (8.89) в целевую функцию, называемую **лагранжианом**, для которой требуется найти стационарную точку:

$$\mathcal{L}(\boldsymbol{\theta}, \lambda) \triangleq \mathcal{L}(\boldsymbol{\theta}) + \lambda h(\boldsymbol{\theta}). \quad (8.90)$$

В стационарной точке лагранжиана имеем

$$\nabla_{\boldsymbol{\theta}, \lambda} \mathcal{L}(\boldsymbol{\theta}, \lambda) = 0 \Leftrightarrow \lambda \nabla_{\boldsymbol{\theta}} h(\boldsymbol{\theta}) = \nabla \mathcal{L}(\boldsymbol{\theta}), h(\boldsymbol{\theta}) = 0. \quad (8.91)$$

Эта точка называется **критической** и удовлетворяет исходному ограничению $h(\boldsymbol{\theta}) = 0$ и условию (8.89).

Если имеется $m > 1$ ограничений, то можно построить из них новую функцию ограничения следующим образом:

$$L(\boldsymbol{\theta}, \lambda) = \mathcal{L}(\boldsymbol{\theta}) + \sum_{j=1}^m \lambda_j h_j(\boldsymbol{\theta}). \quad (8.92)$$

Теперь у нас имеется $D + m$ уравнений от $D + m$ неизвестных, и мы можем использовать стандартные методы безусловной оптимизации для нахождения стационарной точки. Примеры будут приведены ниже.

8.5.1.1. Пример: двумерная квадратичная целевая функция с одним линейным ограничением в виде равенства

Рассмотрим минимизацию функции $\mathcal{L}(\boldsymbol{\theta}) = \theta_1^2 + \theta_2^2$ при ограничении $\theta_1 + \theta_2 = 1$ (эта задача показана на рис. 8.20(а)). Лагранжиан имеет вид:

$$L(\theta_1, \theta_2, \lambda) = \theta_1^2 + \theta_2^2 + \lambda(\theta_1 + \theta_2 - 1). \quad (8.93)$$

Имеем следующие условия в стационарной точке:

$$\frac{\partial}{\partial \theta_1} L(\theta_1, \theta_2, \lambda) = 2\theta_1 + \lambda = 0; \quad (8.94)$$

$$\frac{\partial}{\partial \theta_2} L(\theta_1, \theta_2, \lambda) = 2\theta_2 + \lambda = 0; \quad (8.95)$$

$$\frac{\partial}{\partial \lambda} L(\theta_1, \theta_2, \lambda) = \theta_2 + \theta_2 - 1 = 0. \quad (8.96)$$

Из уравнений (8.94) и (8.95) находим $2\theta_1 = -\lambda = 2\theta_2$, откуда $\theta_1 = \theta_2$. Из уравнения (8.96) находим $2\theta_1 = 1$. Таким образом, $\theta^* = (0.5, 0.5)$, как мы и утверждали ранее. Более того, этот минимум глобальный, потому что целевая функция выпуклая, а ограничение аффинное.

8.5.2. Условия Каруша–Куна–Таккера

В этом разделе мы обобщим множители Лагранжа на ограничения в виде неравенств.

Сначала рассмотрим случай одного неравенства $g(\theta) \leq 0$. Чтобы найти оптимум, мы будем рассматривать задачу без ограничений, в которую добавим штраф в виде функции с бесконечной ступенькой:

$$\hat{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \infty \mathbb{I}(g(\theta) > 0). \quad (8.97)$$

Однако это разрывная функция, которую трудно оптимизировать.

Вместо этого мы создадим нижнюю границу вида $\mu g(\theta)$, где $\mu \geq 0$. Это даст нам следующий лагранжиан:

$$\mathcal{L}(\theta, \mu) = \mathcal{L}(\theta) + \mu g(\theta). \quad (8.98)$$

Заметим, что ступенчатую функцию можно восстановить:

$$\hat{\mathcal{L}}(\theta) = \max_{\mu \geq 0} \mathcal{L}(\theta, \mu) = \begin{cases} \infty, & \text{если } g(\theta) > 0 \\ \mathcal{L}(\theta) & \text{в противном случае} \end{cases}. \quad (8.99)$$

Таким образом, наша задача оптимизации принимает вид:

$$\min_{\theta} \max_{\mu \geq 0} \mathcal{L}(\theta, \mu). \quad (8.100)$$

Теперь рассмотрим общий случай, когда имеется несколько ограничений в виде неравенств, $g(\theta) \leq 0$, и несколько ограничений в виде равенств, $h(\theta) = 0$. **Обобщенный лагранжиан** имеет вид:

$$\mathcal{L}(\theta, \mu, \lambda) = \mathcal{L}(\theta) + \sum_i \mu_i g_i(\theta) + \sum_j \lambda_j h_j(\theta). \quad (8.101)$$

(Мы вправе заменить $-\lambda_j h_j$ на $+\lambda_j h_j$, потому что знак может быть любым.) Задача оптимизации принимает вид:

$$\min_{\theta} \max_{\varepsilon \geq 0, \lambda} \mathcal{L}(\theta, \mu, \lambda). \quad (8.102)$$

Если \mathcal{L} и g выпуклы, то все критические точки этой задачи должны удовлетворять следующему критерию (при некоторых условиях, см. [BV04, раздел 5.2.3]):

- все ограничения выполнены (это называется **допустимостью**):

$$\mathbf{g}(\boldsymbol{\theta}) \leq \mathbf{0}, \mathbf{h}(\boldsymbol{\theta}) = \mathbf{0}; \quad (8.103)$$

- решение является стационарной точкой:

$$\nabla \mathcal{L}(\boldsymbol{\theta}^*) + \sum_i \mu_i \nabla g_i(\boldsymbol{\theta}^*) + \sum_j \lambda_j \nabla h_j(\boldsymbol{\theta}^*) = \mathbf{0}; \quad (8.104)$$

- штраф за ограничения в виде неравенств указывает в правильном направлении (это называется **допустимостью для двойственной задачи**):

$$\boldsymbol{\mu} \geq \mathbf{0}; \quad (8.105)$$

- множители Лагранжа выбирают всю слабину в неактивных ограничениях, т. е. либо $\mu_i = 0$, либо

$$\boldsymbol{\mu} \odot \mathbf{g} = \mathbf{0}. \quad (8.106)$$

Это называется **условием дополняющей нежесткости**.

Чтобы понять, почему выполняется последнее условие, рассмотрим (для простоты) случай одного ограничения в виде неравенства, $g(\boldsymbol{\theta}) \leq 0$. Оно либо **активно**, т. е. $g(\boldsymbol{\theta}) = 0$, либо неактивно, т. е. $g(\boldsymbol{\theta}) < 0$. В первом случае решение лежит на границе связи и $g(\boldsymbol{\theta}) = 0$ становится ограничением типа равенства; тогда мы имеем $\nabla \mathcal{L} = \mu \nabla g$ для некоторой постоянной $\mu \neq 0$ в силу (8.89). Во втором случае решение не лежит на границе связи; мы по-прежнему имеем $\nabla \mathcal{L} = \mu \nabla g$, но теперь $\mu = 0$.

Это так называемые условия **Каруша–Куна–Такера (ККТ)**. Если \mathcal{L} – выпуклая функция и ограничения определяют выпуклое множество, то условия RRN необходимы и достаточны для (глобальной) оптимальности.

8.5.3. Линейное программирование

Рассмотрим оптимизацию линейной функции при линейных ограничениях. В **стандартной форме** эта задача представляется следующим образом:

$$\min_{\boldsymbol{\theta}} \mathbf{c}^\top \boldsymbol{\theta} \quad \text{при условиях} \quad \mathbf{A}\boldsymbol{\theta} \leq \mathbf{b}, \boldsymbol{\theta} \geq \mathbf{0}. \quad (8.107)$$

Допустимое множество является выпуклым **политопом**, т. е. выпуклым множеством, образованным пересечением полупространств. Пример на плоскости показан на рис. 8.21a. На рис. 8.21b показана линейная функция стоимости, которая убывает в направлении правого нижнего угла. Мы видим, что самой нижней точкой допустимого множества является вершина политопы. На самом деле можно доказать, что точка оптимума всегда совпадает с вершиной политопы в предположении, что решение единственно. Если решений несколько, то прямая будет параллельна какой-то грани. Может случиться и так, что в допустимом множестве нет точек оптимума, тогда говорят, что задача неразрешима.

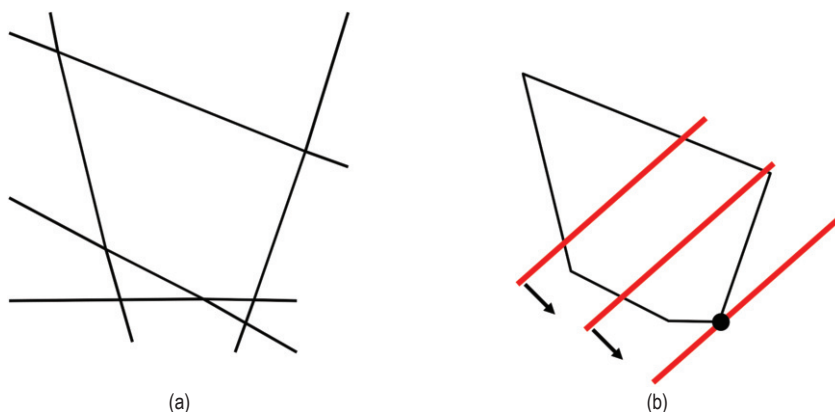


Рис. 8.21 ❖ (а) Выпуклый политоп на плоскости, определенный пересечением линейных ограничений. (б) Изображение допустимого множества, а также линейной целевой функции. Красные прямые – линии уровня целевой функции, а стрелки указывают направление ее улучшения. Мы видим, что оптимальное решение совпадает с одной из вершин политопа

8.5.3.1. Симплекс-метод

Можно показать, что точки оптимума в задаче линейного программирования (ЛП) находятся в вершинах политопа, определяющего допустимое множество (см. пример на рис. 8.21b). **Симплекс-метод** решает задачу ЛП, для чего переходит от одной вершины к другой и на каждом шаге ищет ребро, которое лучше остальных улучшает целевую функцию.

В худшем случае время работы симплекс-метода экспоненциально зависит от D , хотя на практике он обычно очень эффективен. Существуют также различные алгоритмы с полиномиальным временем работы, например метод внутренней точки, но на практике они часто оказываются более медленными.

8.5.3.2. Приложения

У линейного программирования много приложений в науке, технике и бизнесе. Оно также полезно в некоторых задачах машинного обучения. Например, в разделе 11.6.1. показано, как с его помощью решить задачу робастной линейной регрессии. Полезно оно и для оценки состояний в графовых моделях (см., например, [SGJ11]).

8.5.4. Квадратичное программирование

Рассмотрим задачу минимизации квадратичной целевой функции при линейных ограничениях в виде равенств и неравенств. Такие задачи называются **квадратичным программированием** (quadratic programming – **QP**) и могут быть записаны в виде:

$$\min_{\theta} \frac{1}{2} \theta^T \mathbf{H} \theta + \mathbf{c}^T \theta \quad \text{при условиях } \mathbf{A} \theta \leq \mathbf{b}, \mathbf{A}_{eq} \theta = \mathbf{b}_{eq}. \quad (8.108)$$

Если матрица \mathbf{H} положительно полуопределенная, то это задача выпуклой оптимизации.

8.5.4.1. Пример: квадратичная целевая функция в двумерном случае с линейными ограничениями в виде равенств

В качестве конкретного примера минимизируем функцию

$$\mathcal{L}(\theta) = \left(\theta_1 - \frac{3}{2} \right)^2 + \left(\theta_2 - \frac{1}{8} \right)^2 = \frac{1}{2} \theta^T \mathbf{H} \theta + \mathbf{c}^T \theta + \text{const}, \quad (8.109)$$

где $\mathbf{H} = 2\mathbf{I}$ и $\mathbf{c} = -(3, 1/4)$, при ограничении

$$|\theta_1| + |\theta_2| \leq 1. \quad (8.110)$$

Смотрите иллюстрацию на рис. 8.20b.

Ограничения можно переписать в виде

$$\theta_1 + \theta_2 \leq 1, \quad \theta_1 - \theta_2 \leq 1, \quad -\theta_1 + \theta_2 \leq 1, \quad -\theta_1 - \theta_2 \leq 1 \quad (8.111)$$

или более компактно в виде

$$\mathbf{A} \theta - \mathbf{b} \leq \mathbf{0}, \quad (8.112)$$

где $\mathbf{b} = \mathbf{1}$ и

$$\mathbf{A} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \\ -1 & 1 \\ -1 & -1 \end{pmatrix}. \quad (8.113)$$

Это и есть стандартная форма QP.

Из геометрии задачи, показанной на рис. 8.20b, видно, что ограничения, соответствующие обеим левым граням ромба, неактивны (поскольку мы пытаемся подобраться как можно ближе к центру окружности, которая находится вне допустимой области и справа от нее). Если обозначить $g_i(\theta)$ ограничение в виде неравенства, соответствующее строке i матрицы \mathbf{A} , то это означает, что $g_3(\theta^*) > 0$ и $g_4(\theta^*) > 0$ и, следовательно, в силу условий дополняющей нежесткости $\mu_3^* = \mu_4^* = 0$. Поэтому эти неактивные ограничения можно исключить.

Из условий ККТ мы знаем, что

$$\mathbf{H} \theta + \mathbf{c} + \mathbf{A}^T \mu = \mathbf{0}. \quad (8.114)$$

Применяя их к подзадаче с активными ограничениями, получаем

$$\begin{pmatrix} 2 & 0 & 1 & 1 \\ 0 & 2 & 1 & -1 \\ 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \\ \mu_1 \\ \mu_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 1/4 \\ 1 \\ 1 \end{pmatrix}. \quad (8.115)$$

Следовательно, решение имеет вид:

$$\theta^* = (1, 0)^T, \quad \mu^* = (0.625, 0.375, 0, 0)^T. \quad (8.116)$$

Заметим, что оптимальное значение θ совпадает с одной из вершин «шара» по норме ℓ_1 (имеющего форму ромба).

8.5.4.2. Приложения

У квадратичного программирования есть несколько приложений в МО. В разделе 11.4 мы покажем, как его использовать для разреженной линейной регрессии, а в разделе 17.3 – применим в методе опорных векторов.

8.5.5. Смешанно-целочисленное программирование*

В задаче **целочисленного линейного программирования** (ЦЛП) требуется минимизировать линейную целевую функцию при линейных ограничениях, но переменные являются целыми, а не вещественными числами. В стандартной форме задача имеет вид:

$$\min_{\theta} c^T \theta \text{ при условиях } A\theta \leq b, \theta \geq 0, \theta \in \mathbb{Z}^D, \quad (8.117)$$

где \mathbb{Z} – множество целых чисел. Если некоторые переменных могут быть вещественными, то говорят о задаче **смешанно-целочисленного программирования** (СЦЛП). (Если все переменные вещественные, то мы возвращаемся к стандартной задаче ЛП.)

СЦЛП имеет много применений, в том числе в маршрутизации транспортных потоков, планировании и упаковке в контейнеры. Оно также полезно в некоторых задачах МО, например для формальной верификации поведения некоторых видов глубоких нейронных сетей [And+18] и для доказательства свойств устойчивости ГНС к враждебному вмешательству (в худшем случае) [ТХТ19].

8.6. ПРОКСИМАЛЬНЫЙ ГРАДИЕНТНЫЙ МЕТОД*

Часто нас интересует оптимизация целевой функции вида

$$\mathcal{L}(\theta) = \mathcal{L}_s(\theta) + \mathcal{L}_r(\theta), \quad (8.118)$$

где \mathcal{L}_s – дифференцируемая (гладкая) функция, а \mathcal{L}_r – выпуклая, но необязательно дифференцируемая функция (т. е. она может быть негладкой). Например, \mathcal{L}_s может быть отрицательным логарифмическим правдоподобием (NLL), а \mathcal{L}_r – индикаторной функцией, бесконечной в случае нарушения ограничения (см. раздел 8.6.1), или же ℓ_1 -нормой некоторых параметров (см. раздел 8.6.2) или измерять, насколько далеко параметры отстоят от допустимых дискретизированных значений (см. раздел 8.6.3).

Один из подходов к решению таких задач – **проксимальный градиентный метод** (см., например, [PB+14; PSW15]). Грубо говоря, его идея заключается в том, чтобы сделать шаг размера ρ в направлении градиента, а затем спроецировать получившееся обновление параметров в пространство с учетом \mathcal{L}_r . Точнее, обновление имеет вид

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\rho_t \mathcal{L}_r}(\boldsymbol{\theta}_t - \rho_t \nabla \mathcal{L}_s(\boldsymbol{\theta}_t)), \quad (8.119)$$

где $\text{prox}_{\rho_f}(\boldsymbol{\theta})$ – **проксимальный оператор** \mathcal{L}_r (масштабированный на η), вычисленный в точке $\boldsymbol{\theta}$:

$$\text{prox}_{\eta \mathcal{L}_r}(\boldsymbol{\theta}) \triangleq \arg\min_{\mathbf{z}} \left(\mathcal{L}_r(\mathbf{z}) + \frac{1}{2\eta} \|\mathbf{z} - \boldsymbol{\theta}\|_2^2 \right). \quad (8.120)$$

(Множитель $1/2$ – произвольное соглашение.) Мы можем переписать проксимальный оператор как решение задачи условной оптимизации:

$$\text{prox}_{\eta \mathcal{L}_r}(\boldsymbol{\theta}) = \arg\min_{\mathbf{z}} \mathcal{L}_r(\mathbf{z}) \text{ при условии } \|\mathbf{z} - \boldsymbol{\theta}\|_2 \leq \rho, \quad (8.121)$$

где граница ρ зависит от масштабного коэффициента η . Таким образом, мы видим, что проксимальная проекция минимизирует функцию, оставаясь при этом близкой (т. е. проксимальной) к текущей итерации. Примеры будут приведены ниже.

8.6.1. Спроецированный градиентный спуск

Пусть требуется решить задачу

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}_s(\boldsymbol{\theta}) \text{ при условии } \boldsymbol{\theta} \leq \mathcal{C}, \quad (8.122)$$

где \mathcal{C} – выпуклое множество. Например, могут быть ограничения вида $\mathcal{C} = \{\boldsymbol{\theta} : \mathbf{l} \leq \boldsymbol{\theta} \leq \mathbf{u}\}$, когда задаются нижняя и верхняя граница каждого элемента. Для некоторых элементов границы могут быть бесконечными, если мы не хотим налагать никаких ограничений в данном направлении. Так, если мы просто хотим, чтобы параметры были неотрицательными, то полагаем $l_d = 0$ и $u_d = \infty$ для всех направлений d .

Мы можем преобразовать эту задачу условной оптимизации в безусловную, прибавив **штрафующий член** к исходной целевой функции:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_s(\boldsymbol{\theta}) + \mathcal{L}_r(\boldsymbol{\theta}), \quad (8.123)$$

где $\mathcal{L}_r(\boldsymbol{\theta})$ – индикаторная функция для выпуклого множества \mathcal{C} :

$$\mathcal{L}_r(\boldsymbol{\theta}) = I_{\mathcal{C}}(\boldsymbol{\theta}) = \begin{cases} 0, & \text{если } \boldsymbol{\theta} \in \mathcal{C} \\ \infty, & \text{если } \boldsymbol{\theta} \notin \mathcal{C} \end{cases} \quad (8.124)$$

Для решения задачи (8.123) можно воспользоваться проксимальным градиентным спуском. Проксимальный оператор для индикаторной функции эквивалентен проекции на множество \mathcal{C} :

$$\text{proj}_{\mathcal{C}}(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}' \in \mathcal{C}}{\text{argmin}} \|\boldsymbol{\theta}' - \boldsymbol{\theta}\|_2. \quad (8.125)$$

Этот метод называется **спроецированным градиентным спуском** (см. иллюстрацию на рис. 8.22). Например, рассмотрим ограничения $\mathcal{C} = \{\boldsymbol{\theta} : \mathbf{l} \leq \boldsymbol{\theta} \leq \mathbf{u}\}$. Оператор проецирования в этом случае можно вычислить поэлементно путем обрезания на границах:

$$\text{proj}_{\mathcal{C}}(\boldsymbol{\theta})_d = \begin{cases} l_d, & \text{если } \theta_d \leq l_d \\ \theta_d, & \text{если } l_d \leq \theta_d \leq u_d \\ u_d, & \text{если } \theta_d \geq u_d \end{cases} \quad (8.126)$$

Например, если мы хотим гарантировать, что все элементы неотрицательны, то можно взять такой оператор:

$$\text{proj}_{\mathcal{C}}(\boldsymbol{\theta}) = \boldsymbol{\theta}_+ = [\max(\theta_1, 0), \dots, \max(\theta_D, 0)]. \quad (8.127)$$

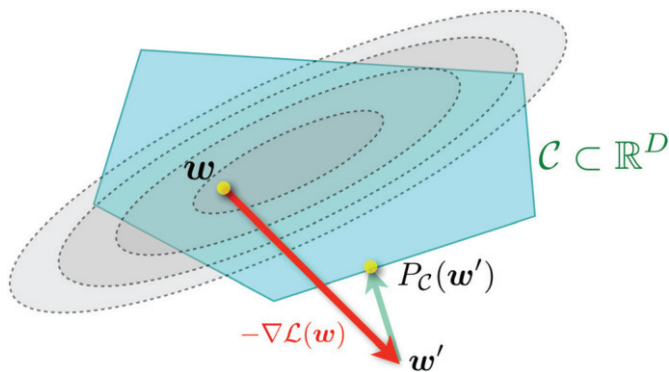


Рис. 8.22 ❖ Спроецированный градиентный спуск. \mathbf{w} – текущая оценка параметра, \mathbf{w}' – обновление после шага в направлении градиента, а $P_{\mathcal{C}}(\mathbf{w}')$ проецирует его на множество ограничений \mathcal{C} . Взято из статьи <https://bit.ly/3eJ3BhZ>. Печатается с разрешения
Мартина Джагги

О применении этого метода к разреженной линейной регрессии см. раздел 11.4.9.2.

8.6.2. Проксимальный оператор для регуляризатора по норме ℓ_1

Рассмотрим линейный предиктор вида $f(\mathbf{x}, \boldsymbol{\theta}) = \sum_{d=1}^D \theta_d x_d$. Если $\theta_d = 0$ для какого-то измерения d , то соответствующий признак x_d можно игнорировать. Такая форма **отбора признаков** может быть полезна и для уменьшения переобучения, и как способ улучшить интерпретируемость модели. Мы можем поощрить выбор нулевых (а не просто малых) весов, штрафую по норме ℓ_1 :

$$\|\boldsymbol{\theta}\|_1 = \sum_{d=1}^D |\theta_d|. \quad (8.128)$$

Это называется **регуляризатором, стимулирующим разреженность**.

Чтобы понять, почему он стимулирует разреженность, рассмотрим два возможных вектора параметров, один из которых разрежен, $\boldsymbol{\theta} = (1, 0)$, а другой не разрежен, $\boldsymbol{\theta}' = (1/\sqrt{2}, 1/\sqrt{2})$. Норма ℓ_2 в обоих случаях одинакова:

$$\|(1, 0)\|_2^2 = \|(1/\sqrt{2}, 1/\sqrt{2})\|_2^2 = 1. \quad (8.129)$$

Следовательно, ℓ_2 -регуляризация (раздел 4.5.3) не дает предпочтения разреженному решению перед плотным. Но при использовании ℓ_1 -регуляризации разреженное решение оказывается дешевле, потому что

$$\|(1, 0)\|_1 = 1 < \|(1/\sqrt{2}, 1/\sqrt{2})\|_1 = \sqrt{2}. \quad (8.130)$$

Дополнительные сведения о разреженной регрессии см. в разделе 11.4.

Если объединить этот регуляризатор с нашей гладкой потерей, то получим

$$\mathcal{L}(\boldsymbol{\theta}) = \text{NLL}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1. \quad (8.131)$$

Эту целевую функцию можно оптимизировать методом проксимального градиентного спуска. Ключевой вопрос – как вычислить оператор прох для функции $f(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$. Поскольку эта функция разлагается по d измерениям, проксимальную проекцию можно вычислять покомпонентно. Из формулы (8.120) с $\rho = 1$ имеем

$$\text{prox}_{\lambda f}(\theta) = \underset{\theta_0}{\operatorname{argmin}} |\theta_0| + \frac{1}{2\lambda} (z - \theta)^2 = \underset{\theta_0}{\operatorname{argmin}} \lambda |\theta_0| + \frac{1}{2} (z - \theta)^2. \quad (8.132)$$

В разделе 11.4.3 мы покажем, что решение этой задачи имеет вид:

$$\text{prox}_{\lambda f}(\theta) = \begin{cases} \theta - \lambda, & \text{если } \theta \geq \lambda \\ 0, & \text{если } |\theta| \leq \lambda \\ \theta + \lambda, & \text{если } \theta \leq -\lambda \end{cases}. \quad (8.133)$$

Оно называется **оператором мягкого порога**, потому что значения, меньшие λ по абсолютной величине, обнуляются, но без разрывов. Отметим, что этот оператор можно записать более компактно в виде

$$\text{SoftThreshold}(\theta, \lambda) = \text{sign}(\theta)(|\theta| - \lambda)_+, \quad (8.134)$$

где $\theta_+ = \max(\theta, 0)$ – положительная часть θ . В векторном случае его можно применять поэлементно:

$$\text{SoftThreshold}(\boldsymbol{\theta}, \lambda) = \text{sign}(\boldsymbol{\theta}) \odot (|\boldsymbol{\theta}| - \lambda)_+. \quad (8.135)$$

О применении этого метода к разреженной линейной регрессии см. раздел 11.4.9.3.

8.6.3. Применение проксимального оператора в случае квантования

В некоторых приложениях (например, при обучении глубоких нейронных сетей для работы на **оконечных устройствах** с ограниченной памятью, например мобильных телефонах) мы хотим, чтобы параметры были **квантованы**. Так, в крайних случаях, когда каждый параметр может принимать только значения -1 или $+1$, пространство состояний имеет вид $\mathcal{C} = \{-1, +1\}^D$.

Определим регуляризатор, который измеряет расстояние до ближайшего квантованного значения вектора параметров:

$$\mathcal{L}_r(\boldsymbol{\theta}) = \inf_{\boldsymbol{\theta}_0 \in \mathcal{C}} \|\boldsymbol{\theta} - \boldsymbol{\theta}_0\|_1. \quad (8.136)$$

(Можно было использовать также норму ℓ_2 .) В случае, когда $\mathcal{C} = \{-1, +1\}^D$, это выражение принимает вид:

$$\mathcal{L}_r(\boldsymbol{\theta}) = \sum_{d=1}^D \inf_{[\theta_0]_d \in \{\pm 1\}} |\theta_d - [\theta_0]_d| = \sum_{d=1}^D \min\{|\theta_d - 1|, |\theta_d + 1|\} = \|\boldsymbol{\theta} - \text{sign}(\boldsymbol{\theta})\|_1. \quad (8.137)$$

Определим соответствующий оператор квантования:

$$q(\boldsymbol{\theta}) = \text{proj}_{\mathcal{C}}(\boldsymbol{\theta}) = \text{argmin} \mathcal{L}_r(\boldsymbol{\theta}) = \text{sign}(\boldsymbol{\theta}). \quad (8.138)$$

Основная трудность обучения с квантованием состоит в том, что квантование – недифференцируемая операция. Распространенное решение этой проблемы – использовать **сквозной оценщик** (straight-through estimator), в котором применяется аппроксимация $\frac{\partial \mathcal{L}}{\partial q(\boldsymbol{\theta})} \approx \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$ (см., например, [Yin+19]). Соответствующее обновление можно произвести за два шага: сначала вычислить вектор градиента в точке, совпадающей с квантованными значениями текущих параметров, а затем обновить не связанные ограничения параметры, пользуясь приближенным градиентом:

$$\tilde{\theta}_t = \text{proj}_{\mathcal{C}}(\theta_t) = q(\theta_t), \quad (8.139)$$

$$\theta_{t+1} = \theta_t - \rho_t \nabla \mathcal{L}_s(\tilde{\theta}_t). \quad (8.140)$$

Применительно к множеству $\mathcal{C} = \{-1, +1\}^D$ этот метод называется **бинарной связью** (binary connect) [CBD15].

Мы можем улучшить результаты, воспользовавшись проксимальным градиентным спуском, рассматривая квантование как регуляризатор, а не жесткое ограничение; этот вариант называется **ProxQuant** [BWL19]. Обновление имеет вид:

$$\tilde{\theta}_t = \text{prox}_{\lambda \mathcal{L}_r}(\theta_t - \rho_t \nabla \mathcal{L}_s(\theta_t)). \quad (8.141)$$

В случае, когда $\mathcal{C} = \{-1, +1\}^D$, можно показать, что проксимальный оператор является обобщением оператора мягкого порога (8.135):

$$\text{prox}_{\lambda \mathcal{L}_r}(\theta) = \text{SoftThreshold}(\theta, \lambda, \text{sign}(\theta)) \quad (8.142)$$

$$= \text{sign}(\theta) + \text{sign}(\theta - \text{sign}(\theta)) \odot (|\theta - \text{sign}(\theta)| - \lambda)_+. \quad (8.143)$$

Эта идея допускает обобщения и на другие формы квантования, детали см. в работе [Yin+19].

8.6.4. Инкрементные (онлайновые) проксимальные методы

Во многих задачах МО встречается целевая функция в виде суммы потерь, по одному слагаемому на каждый пример. Такие задачи можно решать инкрементно, это частный случай **онлайнового обучения**. На эту постановку обобщаются и проксимальные методы. О вероятностном взгляде на такие методы (в терминах фильтров Калмана) см. работы [АЕМ18; Аку+19].

8.7. Граничная оптимизация*

В этом разделе мы рассмотрим класс алгоритмов, известных под названием **граничная оптимизация** (bound optimization) или **ММ**-алгоритмы. В контексте минимизации ММ означает «majorize-minimize» (мажорирование-минимизация). Мы обсудим частный случай – математическое ожидание-максимизация (expectation-maximization), или **ЕМ**, в разделе 8.7.2.

8.7.1. Общий алгоритм

В этом разделе мы дадим краткий обзор ММ-методов. (Дополнительные сведения можно найти, например, в работах [HL04; Mai15; SBP17; Nad+19].) Чтобы не отходить от принятых в литературе традиций, будем предполагать,

что наша цель – максимизировать некоторую функцию $LL(\theta)$, например логарифмическое правдоподобие, относительно ее параметров θ . Базовый подход в ММ-алгоритмах – построить суррогатную функцию $Q(\theta, \theta^t)$, которая является точной нижней границей $LL(\theta)$, такую, что $Q(\theta, \theta^t) \leq LL(\theta)$ и $Q(\theta^t, \theta^t) = LL(\theta^t)$. Если эти условия выполнены, то говорят, что Q минорирует LL . Затем на каждом шаге выполняется следующее обновление:

$$\theta^{t+1} = \underset{\theta}{\operatorname{argmax}} Q(\theta, \theta^t). \quad (8.144)$$

Это гарантирует монотонное возрастание исходной целевой функции:

$$LL(\theta^{t+1}) \geq Q(\theta^{t+1}, \theta^t) \geq Q(\theta^t, \theta^t) = LL(\theta^t), \quad (8.145)$$

где первое неравенство следует из того, что $Q(\theta^{t+1}, \theta^t)$ – нижняя граница $LL(\theta^{t+1})$ для любого θ^t , второе неравенство – из (8.144), а последнее равенство – из точности границы.

Из этого результата вытекает, что если вы не наблюдаете монотонного возрастания целевой функции, значит, в расчетах и (или) в коде имеется ошибка. Это на удивление эффективная техника отладки.

Этот процесс схематически изображен на рис. 8.23. Штрихпунктирная красная кривая – график исходной функции (например, логарифмического правдоподобия) наблюдаемых данных). Сплошная синяя кривая – нижняя граница, вычисленная в точке θ^t ; она касается целевой функции в θ^t . Затем мы полагаем θ^{t+1} равным точке максимума нижней границы (синей кривой) и строим новую аппроксимацию в этой точке (пунктирная зеленая кривая). Точка максимума новой границы становится равной θ^{t+2} и т. д.

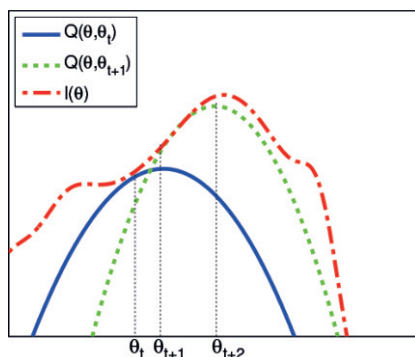


Рис. 8.23 ❖ Алгоритм граничной оптимизации.

На основе рис. 9.14 из работы [Bis06].

Построено программой по адресу figures.problm.ai/book1/8.23.

Если Q – квадратичная нижняя граница, то мы получаем подобие метода Ньютона, который итеративно ищет, а затем оптимизирует квадратичную аппроксимацию, как показано на рис. 8.14а. Разница в том, что оптимизация Q гарантированно ведет к улучшению целевой функции, даже если она

не выпукла, тогда как метод Ньютона может «дать перелет» или привести к уменьшению целевой функции, как показано на рис. 8.24, поскольку это квадратичная аппроксимация, а не граница.

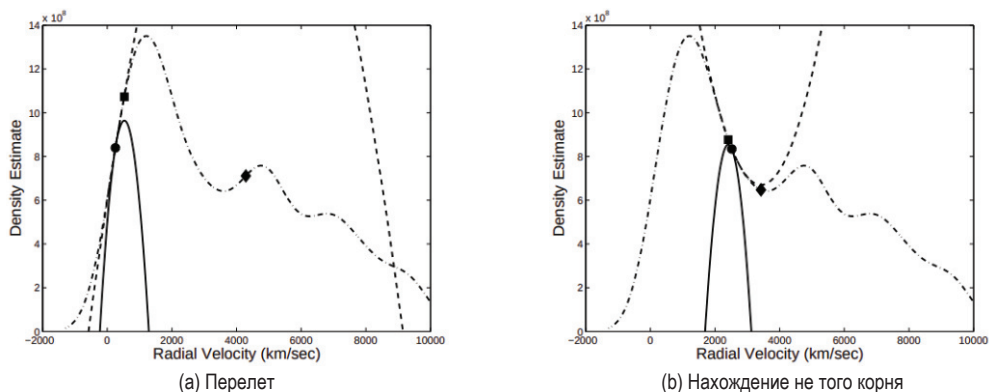


Рис. 8.24 ❖ Квадратичная нижняя граница в ММ-алгоритме (сплошная кривая) и квадратичная аппроксимация в методе Ньютона (пунктирная кривая). Начальные точки обоих алгоритмов отмечены кружочком. Квадратик обозначает результат одного обновления в ММ-алгоритме. Ромбик обозначает результат одного обновления в алгоритме Ньютона. (а) Метод Ньютона дает «перелет», проскакивая глобальный максимум. (б) Метод Ньютона приводит к уменьшению целевой функции. На основе рис. 4 из работы [FT05]. Печатается с разрешения Карло Томаси

8.7.2. ЕМ-алгоритм

В этом разделе мы обсудим алгоритм **математического ожидания-максимизации (ЕМ)** [DLR77; МК97] – алгоритм граничной оптимизации, предназначенный для вычисления оценки параметров MLE или MAP в вероятностных моделях с частично отсутствующими данными и (или) скрытыми переменными. Обозначим y_n видимые данные для примера n , а z_n – скрытые данные.

Основная идея ЕМ-алгоритма – попеременно оценивать скрытые переменные (или отсутствующие значения) на **Е-шаге** (шаге вычисления математического ожидания), а затем использовать полные наблюдаемые данные для вычисления оценки максимального правдоподобия (MLE) на **М-шаге** (шаге максимизации). Разумеется, этот процесс необходимо повторять, потому что ожидаемые значения зависят от параметров, а параметры в свою очередь зависят от ожидаемых значений.

В разделе 8.7.2.1 мы покажем, что ЕМ действительно является ММ-алгоритмом, откуда следует, что эта итеративная процедура сходится к локальному максимуму логарифмического правдоподобия. Скорость сходимости зависит от количества отсутствующих данных, что влияет на точность оценки [XJ96; MD97; SRG03; KKS20].

8.7.2.1. Нижняя граница

Цель ЕМ-алгоритма – максимизировать логарифмическое правдоподобие наблюдаемых данных:

$$LL(\theta) = \sum_{n=1}^N \log p(\mathbf{y}_n | \theta) = \sum_{n=1}^N \log \left[\sum_{\mathbf{z}_n} p(\mathbf{y}_n, \mathbf{z}_n | \theta) \right], \quad (8.146)$$

где \mathbf{y}_n – видимые, а \mathbf{z}_n – скрытые переменные. К сожалению, эту функцию трудно оптимизировать, потому что логарифм нельзя перенести под знак суммы.

ЕМ-алгоритм обходит эту проблему следующим образом. Сначала рассмотрим множество произвольных распределений $q_n(\mathbf{z}_n)$ каждой скрытой переменной \mathbf{z}_n . Логарифмическое правдоподобие наблюдаемых данных можно записать в виде:

$$LL(\theta) = \sum_{n=1}^N \log \left[\sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \frac{p(\mathbf{y}_n, \mathbf{z}_n | \theta)}{q_n(\mathbf{z}_n)} \right]. \quad (8.147)$$

Пользуясь неравенством Йенсена (6.34), мы можем перенести логарифм (являющийся вогнутой функцией) под знак суммы и получить следующую нижнюю границу логарифмического правдоподобия:

$$LL(\theta) \geq \sum_n \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{y}_n, \mathbf{z}_n | \theta)}{q_n(\mathbf{z}_n)} \quad (8.148)$$

$$= \sum_n \underbrace{\mathbb{E}_{q_n} [\log p(\mathbf{y}_n, \mathbf{z}_n | \theta)]}_{\mathcal{E}(\theta, q_n | \mathbf{y}_n)} + \mathbb{H}(q_n) \quad (8.149)$$

$$= \sum_n \mathcal{E}(\theta, q_n | \mathbf{y}_n) \triangleq \mathcal{E}(\theta, \{q_n\} | \mathcal{D}), \quad (8.150)$$

где $\mathbb{H}(q)$ – энтропия распределения вероятностей q , а $\mathcal{E}(\theta, \{q_n\} | \mathcal{D})$ называется **вариационной нижней оценкой**, или **свидетельствующей нижней границей** (evidence lower bound – **ELBO**), поскольку это нижняя граница логарифмического маргинального правдоподобия $\log p(\mathbf{y}_{1:N} | \theta)$, которое называется также **свидетельством**. Оптимизация этой границы лежит в основе вариационного вывода, обсуждавшегося в разделе 4.6.8.3.

8.7.2.2. E-шаг

Мы видим, что нижняя граница представляет собой сумму N членов следующего вида:

$$\mathcal{E}(\theta, q_n | \mathbf{y}_n) = \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{y}_n, \mathbf{z}_n | \theta)}{q_n(\mathbf{z}_n)} \quad (8.151)$$

$$= \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n | \mathbf{y}_n, \theta) p(\mathbf{y}_n | \theta)}{q_n(\mathbf{z}_n)} \quad (8.152)$$

$$= \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} + \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log p(\mathbf{y}_n | \boldsymbol{\theta}) \quad (8.153)$$

$$= -\mathbb{KL}(q_n(\mathbf{z}_n) \| p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta})) + \log p(\mathbf{y}_n | \boldsymbol{\theta}), \quad (8.154)$$

где $\mathbb{KL}(q \| p) \triangleq \sum_{\mathbf{z}} q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z})}$ – расхождение Кульбака–Лейблера (КЛ) между распределениями вероятностей q и p . Мы подробно обсуждали его в разделе 6.2, а сейчас напомним основное нужное нам свойство: $\mathbb{KL}(q \| p) \geq 0$, причем равенство достигается тогда и только тогда, когда $q = p$. Поэтому мы можем максимизировать нижнюю границу $\mathcal{E}(\boldsymbol{\theta}, \{q_n\} | \mathcal{D})$ относительно последовательности $\{q_n\}$, положив каждый ее член равным $q_n^* = p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta})$. Это называется **Е-шагом**. Он гарантирует, что ELBO является точной нижней границей:

$$\mathcal{E}(\boldsymbol{\theta}, \{q_n^*\} | \mathcal{D}) = \sum_n \log p(\mathbf{y}_n | \boldsymbol{\theta}) = LL(\boldsymbol{\theta} | \mathcal{D}). \quad (8.155)$$

Чтобы понять, как это связано с граничной оптимизацией, определим

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) = \mathcal{E}(\boldsymbol{\theta}, \{p(\mathbf{z}_n | \mathbf{y}_n, \boldsymbol{\theta}^t)\}). \quad (8.156)$$

Тогда имеем $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) \leq LL(\boldsymbol{\theta})$ и $Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}^t) = LL(\boldsymbol{\theta}^t)$, как и требуется.

Однако если даже мы не можем вычислить апостериорные распределения $p(\mathbf{z}_n | \mathbf{y}_n; \boldsymbol{\theta}^t)$ точно, то все же можем использовать приближенное распределение $q(\mathbf{z}_n | \mathbf{y}_n; \boldsymbol{\theta}^t)$; это дает неточную нижнюю границу логарифмического правдоподобия. Обобщенный вариант EM-алгоритма называется **вариационным EM** [NH98]. Дополнительные сведения см. во втором томе этой книги, [Mur22].

8.7.2.3. М-шаг

На М-шаге мы должны максимизировать $\mathcal{E}(\boldsymbol{\theta}, \{q_n^t\})$ относительно $\boldsymbol{\theta}$, где q_n^t – распределения, вычисленные на Е-шаге итерации t . Поскольку члены энтропии $\mathbb{H}(q_n)$ постоянны относительно $\boldsymbol{\theta}$, мы можем опустить их на М-шаге. Остается

$$LL^t(\boldsymbol{\theta}) = \sum_n \mathbb{E}_{q_n^t(\mathbf{z}_n)} [\log p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})]. \quad (8.157)$$

Эта функция называется **ожидаемым полным логарифмическим правдоподобием данных**. Если совместная вероятность принадлежит экспоненциальному семейству (раздел 3.4), то мы можем переписать ее в виде

$$LL^t(\boldsymbol{\theta}) = \sum_n \mathbb{E}[\mathcal{T}(\mathbf{y}_n, \mathbf{z}_n)^T \boldsymbol{\theta} - A(\boldsymbol{\theta})] = \sum_n (\mathbb{E}[\mathcal{T}(\mathbf{y}_n, \mathbf{z}_n)]^T \boldsymbol{\theta} - A(\boldsymbol{\theta})), \quad (8.158)$$

где $\mathbb{E}[\mathcal{T}(\mathbf{y}_n, \mathbf{z}_n)]$ – **ожидаемые достаточные статистики**.

На М-шаге мы максимизируем ожидаемое полное логарифмическое правдоподобие данных и получаем

$$\boldsymbol{\theta}^{t+1} = \operatorname{argmax}_{\boldsymbol{\theta}} \sum_n \mathbb{E}_{q_n^t} [\log p(\mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})]. \quad (8.159)$$

В случае экспоненциального семейства задачу максимизации можно решить в замкнутой форме, приравняв моменты ожидаемых достаточных статистик.

Из вышеизложенного видно, что Е-шаг на самом деле не обязан возвращать полный набор апостериорных распределений $\{q(z_n)\}$, а может вернуть только сумму ожидаемых достаточных статистик, $\sum_n \mathbb{E}_{q(z_n)} [T(\mathbf{y}_n, \mathbf{z}_n)]$. Это станет яснее из приведенных ниже примеров.

8.7.3. Пример: ЕМ-алгоритм для смеси гауссовых распределений

В этом разделе мы покажем, как применить ЕМ-алгоритм для вычисления оценок MLE и MAP параметров модели смеси гауссовых распределений (GMM).

8.7.3.1. Е-шаг

На Е-шаге вычисляется **ответственность** кластера k за порождение точки данных n , оцененная с помощью текущих оценок параметров $\boldsymbol{\theta}^{(t)}$:

$$r_{nk}^{(t)} = p^*(z_n = k | \boldsymbol{\theta}^{(t)}) = \frac{\pi_k^{(t)} p(\mathbf{y}_n | \boldsymbol{\theta}_k^{(t)})}{\sum_{k'} \pi_{k'}^{(t)} p(\mathbf{y}_n | \boldsymbol{\theta}_{k'}^{(t)})}. \quad (8.160)$$

8.7.3.2. М-шаг

На М-шаге максимизируется ожидаемое полное логарифмическое правдоподобие данных:

$$LL^t(\boldsymbol{\theta}) = \mathbb{E} \left[\sum_n \log p(z_n | \boldsymbol{\pi}) + \log p(\mathbf{y}_n | z_n, \boldsymbol{\theta}) \right] \quad (8.161)$$

$$= \mathbb{E} \left[\sum_n \log \left(\prod_k \pi_k^{z_{nk}} \right) + \log \left(\prod_k \mathcal{N}(\mathbf{y}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{nk}} \right) \right] \quad (8.162)$$

$$= \sum_n \sum_k \mathbb{E}[z_{nk}] \log \pi_k + \sum_n \sum_k \mathbb{E}[z_{nk}] \log \mathcal{N}(\mathbf{y}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (8.163)$$

$$= \sum_n \sum_k r_{nk}^{(t)} \log(\pi_k) - \frac{1}{2} \sum_n \sum_k r_{nk}^{(t)} [\log |\boldsymbol{\Sigma}_k| + (\mathbf{y}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{y}_n - \boldsymbol{\mu}_k)] + \text{const}, \quad (8.164)$$

где $z_{nk} = \mathbb{I}(z_n = k)$ – унитарное кодирование категориальных значений z_n . Эта целевая функция не что иное, как взвешенная версия стандартной задачи вычисления оценок MLE многомерного гауссова распределения (см. раздел 4.2.6).

Можно показать, что новые оценки параметров имеют вид:

$$\boldsymbol{\mu}_k^{(t+1)} = \frac{\sum_n r_{nk}^{(t)} \mathbf{y}_n}{r_k^{(t)}}; \quad (8.165)$$

$$\begin{aligned} \boldsymbol{\Sigma}_k^{(t+1)} &= \frac{\sum_n r_{nk}^{(t)} (\mathbf{y}_n - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{y}_n - \boldsymbol{\mu}_k^{(t+1)})^\top}{r_k^{(t)}} \\ &= \frac{\sum_n r_{nk}^{(t)} \mathbf{y}_n \mathbf{y}_n^\top}{r_k^{(t)}} - \boldsymbol{\mu}_k^{(t+1)} (\boldsymbol{\mu}_k^{(t+1)})^\top, \end{aligned} \quad (8.166)$$

где $r_k^{(t)} \triangleq \sum_n r_{nk}^{(t)}$ – взвешенное число точек, отнесенных к кластеру k . Среднее кластера k – это просто взвешенное среднее всех точек, отнесенных к кластеру k , а ковариация пропорциональна взвешенной эмпирической матрице рассеяния.

М-шаг для весов смеси – это просто взвешенная форма обычной MLE:

$$\pi_k^{(t+1)} = \frac{1}{N} \sum_n r_{nk}^{(t)} = \frac{r_k^{(t)}}{N}. \quad (8.167)$$

8.7.3.3. Пример

Пример алгоритма в действии показан на рис. 8.25, где 25 точек на плоскости аппроксимируются двухкомпонентной моделью GMM. Набор данных, заимствованный из [Bis06], получен на основе измерений гейзера Старый служака в Йеллоустоунском национальном парке. Конкретно, на графике представлена зависимость момента следующего извержения в минутах от продолжительности извержения в минутах. Перед обработкой данные были стандартизированы путем вычитания среднего и деления на стандартное отклонение; часто это улучшает сходимость. Были выбраны следующие начальные значения: $\boldsymbol{\mu}_1 = (-1, 1)$, $\boldsymbol{\Sigma}_1 = \mathbf{I}$, $\boldsymbol{\mu}_2 = (1, -1)$, $\boldsymbol{\Sigma}_2 = \mathbf{I}$. Затем показаны отнесение к кластерам и соответствующие компоненты смеси на различных итерациях.

Дополнительные сведения о применении GMM для кластеризации см. в разделе 21.4.1.

8.7.3.4. Оценка MAP

Вычисление MLE для GMM часто подвержено численным проблемам и переобучению. Чтобы понять, почему это так, предположим для простоты, что $\boldsymbol{\Sigma}_k = \sigma_k^2 \mathbf{I}$ для всех k . Можно получить бесконечное правдоподобие, отнеся один из центров, скажем $\boldsymbol{\mu}_k$, к одной точке данных, скажем \mathbf{y}_n , потому что тогда правдоподобие этой точки будет равно

$$\mathcal{N}(\mathbf{y}_n | \boldsymbol{\mu}_k = \mathbf{y}_n, \sigma_k^2 \mathbf{I}) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^0. \quad (8.168)$$

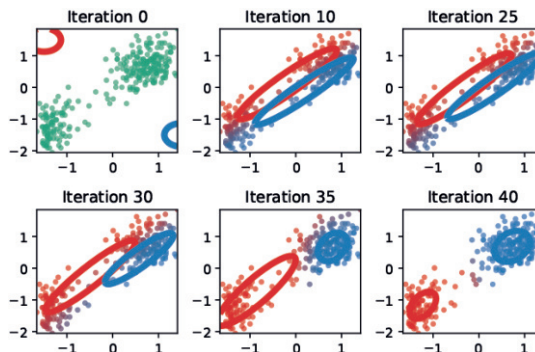


Рис. 8.25 ❖ Иллюстрация EM для модели GMM применительно к данным о гейзере Старый служака. Уровень «красноты» показывает степень уверенности в принадлежности точки к красному кластеру и аналогично для синего; таким образом, фиолетовые точки могут принадлежать обоим кластерам с вероятностью примерно 50/50. На основе рис. 9.8 из [Bis06]. Построено программой по адресу figures.problml.ai/book1/8.25

Следовательно, мы можем устремить этот член к бесконечности, положив $\sigma_k \rightarrow 0$, как показано на рис. 8.26(a). Это называется «проблемой коллапсирующей дисперсии».

Простое решение этой проблемы – вычислять оценку MAP. По счастью, для ее нахождения можно использовать все тот же EM-алгоритм. Наша цель – максимизировать сумму ожидаемого полного логарифмического правдоподобия данных и логарифмического априорного распределения:

$$LL^t(\theta) = \left[\sum_n \sum_k r_{nk}^{(t)} \log \pi_{nk} + \sum_n \sum_k r_{nk}^{(t)} \log p(\mathbf{y}_n | \theta_k) \right] + \log p(\pi) + \sum_k \log p(\theta_k). \quad (8.169)$$

Отметим, что E-шаг вообще не изменяется, но M-шаг необходимо модифицировать, как описано ниже.

В качестве априорного распределения весов смеси естественно использовать распределение Дирихле (раздел 4.6.3.2), $\pi \sim \text{Dir}(\alpha)$, так как оно является сопряженным к категориальному распределению. Оценка MAP имеет вид:

$$\tilde{\pi}_k^{(t+1)} = \frac{r_k^{(t)} + a_k - 1}{N + \sum_k a_k - K}. \quad (8.170)$$

Если использовать равномерное априорное распределение, $\alpha_k = 1$, то она сводится к MLE.

В качестве априорного распределения компонент смеси рассмотрим сопряженное априорное распределение вида

$$p(\mu_k, \Sigma_k) = \text{NIW}(\mu_k, \Sigma_k | \bar{m}, \bar{\kappa}, \bar{\nu}, \bar{S}). \quad (8.171)$$

Оно называется **нормальным обратным распределением Вишарта** (см. второй том этой книги, [Mur22]). Пусть для гиперпараметров μ выбрано $\tilde{\kappa} = 0$, так что μ_k не регуляризованы; таким образом, априорное распределение повлияет только на нашу оценку Σ_k . В этом случае оценки MAP имеют вид:

$$\tilde{\mu}_k^{(t+1)} = \tilde{\mu}_k^{(t)}; \quad (8.172)$$

$$\tilde{\Sigma}_k^{(t+1)} = \frac{\tilde{\mathbf{S}} + \tilde{\Sigma}_k^{(t+1)}}{\tilde{\nu} + r_k^{(t)} + D + 2}, \quad (8.173)$$

где $\hat{\mu}_k$ – MLE для μ_k из формулы (8.165), а $\hat{\Sigma}_k$ – MLE Σ_k из формулы (8.166).

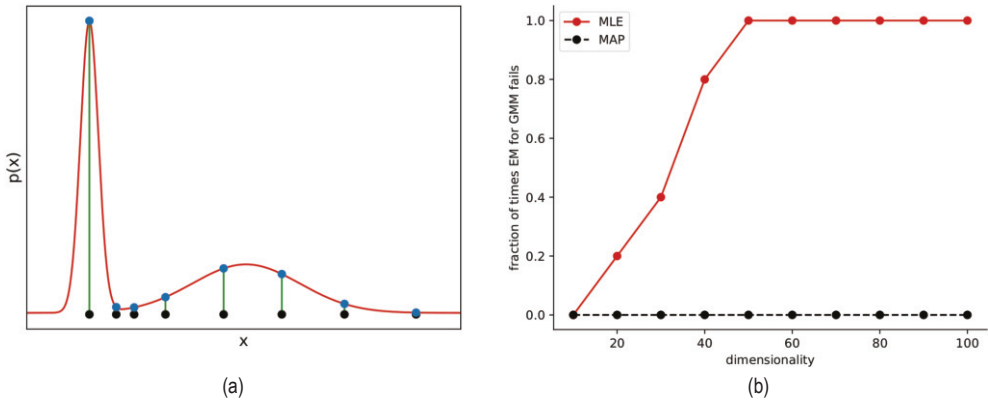


Рис. 8.26 ❖ (a) Как могут возникать сингулярности в функции правдоподобия моделей GMM. Здесь $K = 2$, но первая компонента смеси – узкий пик (с $\sigma_1 \approx 0$) с центром в точке x_1 . На основе рис. 9.7 из [Bis06]. Построено программой по адресу figures.probl.ai/book1/8.26. (b) Преимущество оценки MAP перед оценкой максимального правдоподобия при аппроксимации моделью гауссовой смеси. Показана зависимость доли случаев (в пяти случайных испытаниях), когда каждый метод стал-кивался с численными проблемами, от размерности задачи при $N = 100$ примерах. Сплошная красная линия (верхняя кривая): MLE. Пунктирная черная линия (нижняя кривая): MAP. Построено программой по адресу figures.probl.ai/book1/8.26

Теперь обсудим, как задать априорную ковариацию, $\tilde{\mathbf{S}}$. Одна из возможностей (предложенная в [FR07, стр. 163]) – взять

$$\tilde{\mathbf{S}} = \frac{1}{K^{1/D}} \text{diag}(s_1^2, \dots, s_D^2), \quad (8.174)$$

где $s_d = (1/N) \sum_{n=1}^N (x_{nd} - \bar{x}_d)^2$ – объединенная дисперсия по измерению d . Параметр $\tilde{\nu}$ управляет силой нашей веры в это априорное распределение. Самое слабое априорное распределение, которое все еще можно использовать, получается, если положить $\tilde{\nu} = D + 2$, поэтому такой выбор наиболее распространен.

Теперь продемонстрируем преимущества использования оценки MAP вместо MLE в контексте GMM. Мы применяем EM-алгоритм к некоторым

синтетическим данным с $N = 100$ примерами в D измерениях, используя одну из двух оценок: MLE или MAP. Мы считаем испытание «неудачным», если при вычислениях возникли проблемы из-за сингулярных матриц. Для каждой размерности производится 5 случайных испытаний. Результаты показаны на рис. 8.26b. Мы видим, что даже при умеренно больших D оценка MLE терпит крах, тогда как оценка MAP с подходящим априорным распределением редко сталкивается с численными проблемами.

8.7.3.5. Невыпуклость NLL

Правдоподобие смесовой модели равно

$$LL(\theta) = \sum_{n=1}^N \log \left[\sum_{z_n=1}^K p(y_n, z_n | \theta) \right]. \quad (8.175)$$

В общем случае она будет иметь несколько мод, а значит, единственного глобального оптимума не будет. На рис. 8.27 это показано для смеси двух одномерных гауссовых распределений. Мы видим два одинаково хороших глобальных оптимума, соответствующих двум разным пометкам кластеров; в одном левый пик соответствует $z = 1$, а в другом – $z = 2$. Это называется **проблемой переключения меток**; дополнительные сведения см. в разделе 21.4.1.2.

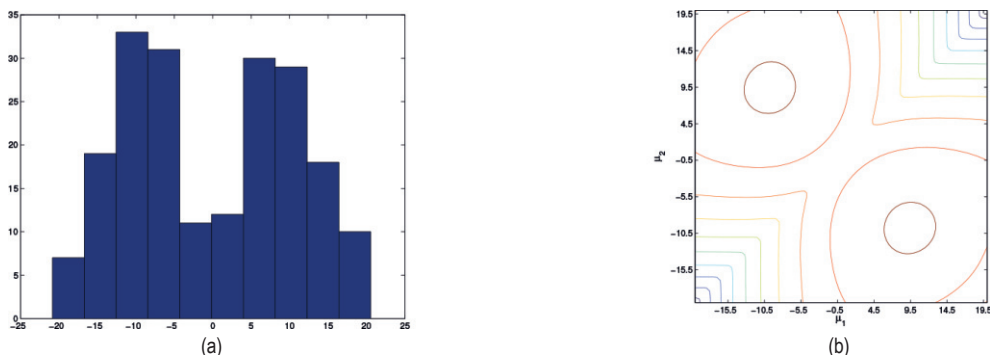


Рис. 8.27 ❖ Слева: $N = 200$ точек, выбранных из смеси двух одномерных гауссовых распределений с $\pi_k = 0.5$, $\sigma_k = 5$, $\mu_1 = -10$ и $\mu_2 = 10$. Справа: поверхность правдоподобия $p(\mathcal{D} | \mu_1, \mu_2)$, когда все остальные параметры имеют истинные значения. Мы видим две симметричных моды, что отражает невозможность идентификации параметров. Построено программой по адресу figures.problml.ai/book1/8.27

На вопрос, сколько мод имеется в функции правдоподобия, ответить трудно. Существует $K!$ возможных способов пометки, но некоторые пики могут сливаться в зависимости от того, насколько далеко отстоят μ_k . Тем не менее число мод может расти экспоненциально. Следовательно, нахождение глобального оптимума – NP-трудная задача [Alo+09; Dri+04]. Поэтому мы вынуждены довольствоваться локальным оптимумом. Чтобы найти хороший локальный оптимум, можно воспользоваться методом Kmeans++ (раздел 21.3.4) для инициализации EM-алгоритма.

8.8. ОПТИМИЗАЦИЯ ЧЕРНОГО ЯЩИКА И ОПТИМИЗАЦИЯ БЕЗ ИСПОЛЬЗОВАНИЯ ПРОИЗВОДНЫХ

В некоторых задачах оптимизации целевая функция представляет собой **черный ящик**, т. е. ее функциональная форма неизвестна. Это значит, что для ее оптимизации невозможно применить градиентные методы. Вместо этого такие задачи требуют методов **оптимизации черного ящика** (black-box optimization – **BBO**), которые также называют **оптимизацией без использования производных** (derivative free optimization – **DFO**).

В МО такие задачи часто возникают при выборе модели. Например, предположим, что имеются некоторые гиперпараметры, $\lambda \in \Lambda$, управляющие типом или сложностью модели. Часто целевая функция $\mathcal{L}(\lambda)$ определяется как потеря на контрольном наборе (см. раздел 4.5.4). Поскольку потеря зависит от оптимальных параметров модели, которые вычисляются с помощью сложного алгоритма, эта целевая функция по существу является черным ящиком¹. Простой подход к решению таких задач – воспользоваться **поиском на сетке**, когда мы оцениваем каждую точку в пространстве параметров и выбираем ту, для которой потеря наименьшая. К сожалению, этот способ не масштабируется на большое число измерений из-за проклятия размерности. Кроме того, даже при низкой размерности он может оказаться дорогостоящим, если вычисление целевой функции черного ящика обходится дорого (например, если прежде чем вычислить потерю на контрольном наборе, необходимо обучить модель). Были предложены различные решения этой проблемы. Дополнительные сведения см. во втором томе книги, [Mur22]

8.9. УПРАЖНЕНИЯ

Упражнение 8.1 [субдифференциал кусочно-линейной функции потерь*].

Пусть $f(x) = (1 - x)_+$ – кусочно-линейная функция потерь, где $(z)_+ = \max(0, z)$. Чему равны $\partial f(0)$, $\partial f(1)$ и $\partial f(2)$?

Упражнение 8.2 [ЕМ-алгоритм для распределения Стюдента].

Выведите уравнения ЕМ-алгоритма для вычисления MLE многомерного распределения Стюдента. Отдельно рассмотрите случаи, когда параметр dof известен и неизвестен. *Указание:* запишите распределение Стюдента как масштабированную смесь гауссовых распределений.

¹ Если оптимальные параметры вычисляются градиентным методом, то мы можем «развернуть» шаги вычисления градиента и создать глубокую цепь, которая отображает обучающие данные в оптимальные параметры и, следовательно, в потерю на контрольном наборе. Затем ее можно оптимизировать (см., например, [Fra+17]). Однако такая техника применима далеко не всегда.

Часть II

ЛИНЕЙНЫЕ МОДЕЛИ

Глава 9

Линейный дискриминантный анализ

9.1. ВВЕДЕНИЕ

В этой главе мы рассмотрим модели классификации вида:

$$p(y = c|\mathbf{x}; \boldsymbol{\theta}) = \frac{p(\mathbf{x}|y = c; \boldsymbol{\theta})p(y = c; \boldsymbol{\theta})}{\sum_{c'} p(\mathbf{x}|y = c'; \boldsymbol{\theta})p(y = c'; \boldsymbol{\theta})}. \quad (9.1)$$

Член $p(y = c; \boldsymbol{\theta})$ – априорное распределение меток класса, а член $p(\mathbf{x}|y = c; \boldsymbol{\theta})$ называется **условной плотностью класса** c .

Модель в целом называется **порождающим классификатором**, потому что она описывает способ *порождения* признаков \mathbf{x} для каждого класса c путем выборки из распределения $p(\mathbf{x}|y = c; \boldsymbol{\theta})$. С другой стороны, **дискриминантный классификатор** непосредственно моделирует апостериорное распределение классов $p(y|\mathbf{x}; \boldsymbol{\theta})$. Мы обсудим плюсы и минусы этих подходов к классификации в разделе 9.4.

Если выбрать условные плотности классов специальным образом, то результирующее апостериорное распределение классов будет линейной функцией от \mathbf{x} , т. е. $\log p(y = c|\mathbf{x}; \boldsymbol{\theta}) = \mathbf{w}^T \mathbf{x} + \text{const}$, где \mathbf{w} выведено из $\boldsymbol{\theta}$. Поэтому метод называется **линейным дискриминантным анализом** (ЛДА, англ. LDA)¹.

9.2. ГАУССОВ ДИСКРИМИНАНТНЫЙ АНАЛИЗ

В этом разделе мы рассмотрим порождающий классификатор, в котором условные плотности классов – многомерные гауссовы распределения:

¹ Этот термин ведет к некоторой путанице по двум причинам. Во-первых, ЛДА относится к порождающим классификаторам. А во-вторых, английская аббревиатура LDA означает также latent Dirichlet allocation – латентное размещение Дирихле – популярную порождающую модель без учителя для мешков слов [BNJ03].

$$p(\mathbf{x}|y = c, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c). \quad (9.2)$$

Поэтому соответствующее апостериорное распределение классов имеет вид

$$p(y = c|\mathbf{x}, \boldsymbol{\theta}) \propto \pi_c \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c), \quad (9.3)$$

где $\pi_c = p(y = c)$ – априорная вероятность метки c . (Заметим, что нормировочную постоянную в знаменателе апостериорного распределения можно игнорировать, потому что она не зависит от c .) Эта модель называется **гауссовым дискриминантным анализом**, или **GDA**.

9.2.1. Квадратичные решающие границы

Из формулы (9.3) видно, что логарифмическое апостериорное распределение меток классов имеет вид:

$$\log p(y = c|\mathbf{x}; \boldsymbol{\theta}) = \log \pi_c - \frac{1}{2} \log |2\pi\boldsymbol{\Sigma}_c| - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}_c^{-1} (\mathbf{x} - \boldsymbol{\mu}_c) + \text{const}. \quad (9.4)$$

Это называется **дискриминантной функцией**. Мы видим, что решающая граница между двумя классами, скажем c и c' , будет квадратичной функцией от \mathbf{x} . Поэтому такой метод называется **квадратичным дискриминантным анализом** (QDA).

Например, рассмотрим двумерные данные, принадлежащие трем разным классам, на рис. 9.1a. Мы аппроксимировали условные плотности классов гауссовыми распределениями с полной ковариационной матрицей (методом, описанным в разделе 9.2.4) и представили результаты на рис. 9.1b. Мы видим, что признаки синего класса в какой-то мере коррелированы, тогда как признаки зеленого класса независимы, а признаки красного класса независимы и изотропны (сферическая ковариация). На рис. 9.2a видно, что получившиеся решающие границы являются квадратичными функциями от \mathbf{x} .

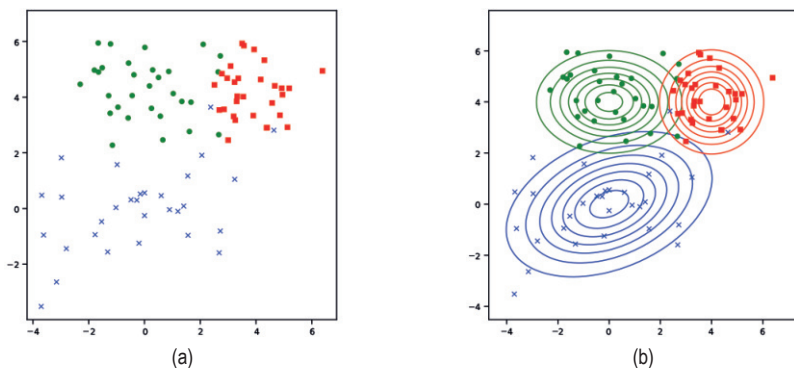


Рис. 9.1 ❖ (a) Двумерные данные, принадлежащие трем разным классам. (b) Аппроксимация каждого класса двумерным гауссовым распределением.

Построено программой по адресу figures.probl.ai/book1/9.1

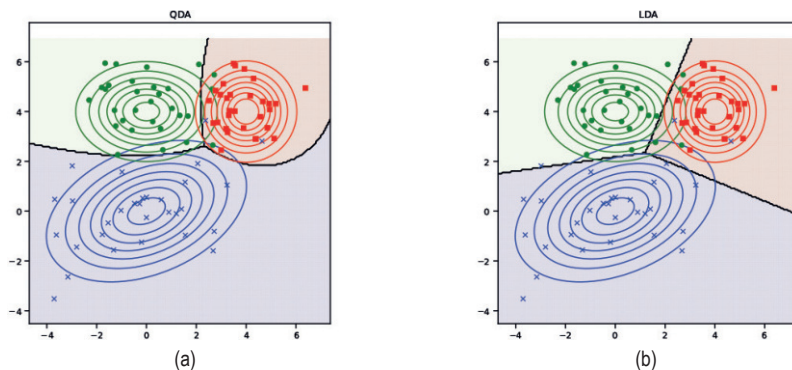


Рис. 9.2 ❖ Аппроксимация данных на рис. 9.1 с помощью гауссова дискриминантного анализа. (а) Ничем не ограниченные ковариации индуцируют квадратичные решающие границы. (б) Связанные ковариации индуцируют линейные решающие границы. Построено программой по адресу figures.problai/book1/9.3

9.2.2. Линейные решающие границы

Теперь рассмотрим частный случай гауссова дискриминантного анализа, когда ковариационные матрицы **связаны** или **разделяются** всеми классами, т. е. $\Sigma_c = \Sigma$. Если Σ не зависит от c , то выражение (9.4) можно упростить следующим образом:

$$\log p(y = c | \mathbf{x}, \boldsymbol{\theta}) = \log \pi_c - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_c)^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_c) + \text{const} \quad (9.5)$$

$$= \underbrace{\log \pi_c - \frac{1}{2} \boldsymbol{\mu}_c^\top \Sigma^{-1} \boldsymbol{\mu}_c}_{\gamma_c} + \underbrace{\mathbf{x}^\top \Sigma^{-1} \boldsymbol{\mu}_c}_{\boldsymbol{\beta}_c} + \underbrace{\text{const} - \frac{1}{2} \mathbf{x}^\top \Sigma^{-1} \mathbf{x}}_{\kappa} \quad (9.6)$$

$$= \gamma_c + \mathbf{x}^\top \boldsymbol{\beta}_c + \kappa. \quad (9.7)$$

Последний член не зависит от c , поэтому является несущественной аддитивной постоянной, которую можно опустить. Таким образом, дискриминантная функция является линейной функцией от \mathbf{x} , так что решающие границы будут линейными. Поэтому метод называется **линейным дискриминантным анализом**, или ЛДА (см. пример на рис. 9.2б).

9.2.3. Связь между ЛДА и логистической регрессией

В этом разделе мы выведем интересную связь между ЛДА и логистической регрессией, введенной в разделе 2.5.3. Благодаря формуле (9.7) мы можем написать:

$$p(y = c|\mathbf{x}, \boldsymbol{\theta}) = \frac{e^{\beta_c^\top \mathbf{x} + \gamma_c}}{\sum_{c'} e^{\beta_{c'}^\top \mathbf{x} + \gamma_{c'}}} = \frac{e^{\mathbf{w}_c^\top [1, \mathbf{x}]}}{\sum_{c'} e^{\mathbf{w}_{c'}^\top [1, \mathbf{x}]}} \quad (9.8)$$

где $\mathbf{w}_c = [\gamma_c, \beta_c]$. Мы видим, что формула (9.8) имеет такой же вид, как модель мультиномиальной логистической регрессии. Ключевое отличие заключается в том, что в ЛДА мы сначала обучаем гауссовы распределения (и априорные распределения классов), чтобы максимизировать совместное правдоподобие $p(\mathbf{x}, y|\boldsymbol{\theta})$, как обсуждалось в разделе 9.2.4, а затем выводим \mathbf{w} из $\boldsymbol{\theta}$. А в логистической регрессии мы оцениваем \mathbf{w} непосредственно с целью максимизировать условное правдоподобие $p(y|\mathbf{x}, \mathbf{w})$. В общем случае результаты получаются разными (см. упражнение 10.3).

Чтобы лучше понять смысл формулы (9.8), рассмотрим бинарный случай, когда апостериорное распределение имеет вид

$$p(y = 1|\mathbf{x}, \boldsymbol{\theta}) = \frac{e^{\beta_1^\top \mathbf{x} + \gamma_1}}{e^{\beta_1^\top \mathbf{x} + \gamma_1} + e^{\beta_0^\top \mathbf{x} + \gamma_0}} = \frac{1}{1 + e^{(\beta_0 - \beta_1)^\top \mathbf{x} + (\gamma_0 - \gamma_1)}} \quad (9.9)$$

$$= \sigma((\boldsymbol{\beta}_1 - \boldsymbol{\beta}_0)^\top \mathbf{x} + (\gamma_1 - \gamma_0)). \quad (9.10)$$

где $\sigma(\eta)$ обозначает сигмоидную функцию.

Тогда

$$\gamma_1 - \gamma_0 = -\frac{1}{2} \boldsymbol{\mu}_1^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1 + \frac{1}{2} \boldsymbol{\mu}_0^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_0 + \log(\pi_1/\pi_0) \quad (9.11)$$

$$= -\frac{1}{2} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu}_1 + \boldsymbol{\mu}_0) + \log(\pi_1/\pi_0). \quad (9.12)$$

Поэтому, если определить

$$\mathbf{w} = \boldsymbol{\beta}_1 - \boldsymbol{\beta}_0 = \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu}_1 + \boldsymbol{\mu}_0); \quad (9.13)$$

$$\mathbf{x}_0 = \frac{1}{2} (\boldsymbol{\mu}_1 + \boldsymbol{\mu}_0) - (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) \frac{\log(\pi_1/\pi_0)}{(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu}_1 + \boldsymbol{\mu}_0)}, \quad (9.14)$$

то будем иметь $\mathbf{w}^\top \mathbf{x}_0 = -(\gamma_1 - \gamma_0)$ и потому

$$p(y = 1|\mathbf{x}, \boldsymbol{\theta}) = \sigma(\mathbf{w}^\top (\mathbf{x} - \mathbf{x}_0)). \quad (9.15)$$

Это уравнение имеет такую же форму, как бинарная логистическая регрессия. Поэтому решающее правило МАР имеет вид

$$\hat{y}(\mathbf{x}) = 1 \text{ тогда и только тогда, когда } \mathbf{w}^\top \mathbf{x} > c, \quad (9.16)$$

где $c = \mathbf{w}^\top \mathbf{x}_0$. Если $\pi_0 = \pi_1 = 0.5$, то пороговую величину можно упростить: $c = \frac{1}{2} \mathbf{w}^\top (\boldsymbol{\mu}_1 + \boldsymbol{\mu}_0)$.

Чтобы интерпретировать это уравнение геометрически, предположим, что $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$. В этом случае вектор $\mathbf{w} = \sigma^{-2} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$ параллелен прямой, соединяющей оба центроида, $\boldsymbol{\mu}_0$ и $\boldsymbol{\mu}_1$. Поэтому, чтобы классифицировать точку, мы

можем спроецировать ее на эту прямую, а затем проверить, к какому центру, μ_0 или μ_1 , она ближе (см. рис. 9.3). Вопрос о том, насколько близко она должна быть, зависит от априорного распределения классов. Если $\pi_1 = \pi_0$, то $x_0 = \frac{1}{2}(\mu_1 + \mu_0)$, т. е. посередине между средними. Если взять $\pi_1 > \pi_0$, то для выбора класса 0 точка должна быть ближе к μ_0 , чем средняя точка. Для $\pi_0 > \pi_1$ дело обстоит ровно наоборот. Таким образом, мы видим, что априорное распределение классов меняет только порог принятия решения, но не общую форму решающей границы. (Аналогичное рассуждение применимо и к случаю нескольких классов.)

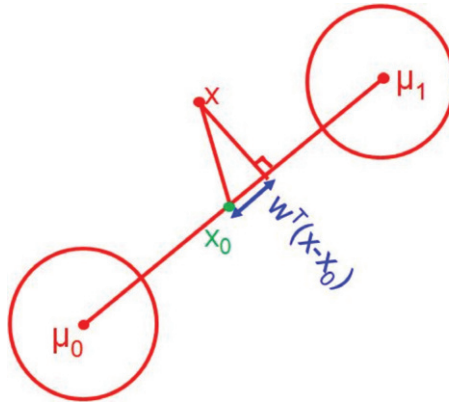


Рис. 9.3 ❖ Геометрия ЛДА в случае 2 классов, когда $\Sigma_1 = \Sigma_2 = I$

9.2.4. Обучение модели

Теперь обсудим, как обучить модель GDA с помощью оценки максимального правдоподобия. Функция правдоподобия выглядит следующим образом:

$$p(\mathcal{D}|\theta) = \prod_{n=1}^N \text{Cat}(y_n|\pi) \prod_{c=1}^C \mathcal{N}(\mathbf{x}_n|\mu_c, \Sigma_c)^{\mathbb{I}(y_n=c)}. \quad (9.17)$$

Поэтому логарифмическое правдоподобие имеет вид:

$$\log p(\mathcal{D}|\theta) = \left[\sum_{n=1}^N \sum_{c=1}^C \mathbb{I}(y_n = c) \log \pi_c \right] + \sum_{c=1}^C \left[\sum_{n: y_n=c} \log \mathcal{N}(\mathbf{x}_n|\mu_c, \Sigma_c) \right]. \quad (9.18)$$

Таким образом, мы видим, что оптимизировать члены π и (μ_c, Σ_c) можно по отдельности.

Из раздела 4.2.4 мы знаем, что MLE априорного распределения классов имеет вид $\hat{\pi}_c = N_c/N$. Пользуясь результатами из раздела 4.2.6, можно вывести следующие MLE для гауссовых распределений:

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{N_c} \sum_{n: y_n=c} \mathbf{x}_n; \quad (9.19)$$

$$\hat{\boldsymbol{\Sigma}}_c = \frac{1}{N_c} \sum_{n: y_n=c} (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_c)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_c)^\top. \quad (9.20)$$

К сожалению, оценка MLE $\hat{\boldsymbol{\Sigma}}_c$ подвержена переобучению (т. е. может оказаться плохо обусловленной), если N_c мало по сравнению с D – размерности пространства входных признаков. Ниже мы обсудим некоторые решения этой проблемы.

9.2.4.1. Связанные ковариационные матрицы

Если $\boldsymbol{\Sigma}_c = \boldsymbol{\Sigma}$, т. е. ковариационные матрицы связаны, то, как уже было показано, мы имеем линейные решающие границы. Обычно это дает более надежную оценку параметров, так как мы можем объединить все примеры из разных классов:

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_{c=1}^C \sum_{n: y_n=c} (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_c)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_c)^\top. \quad (9.21)$$

9.2.4.2. Диагональные ковариационные матрицы

Если постулировать, что матрица $\boldsymbol{\Sigma}_c$ диагональная, то количество параметров уменьшается с $O(CD^2)$ до $O(CD)$, так что проблемы переобучения удастся избежать. Однако при этом мы теряем возможность уловить корреляцию между признаками. (Это называется наивным байесовским предположением, и мы еще обсудим его в разделе 9.3.) Но, несмотря на такую аппроксимацию, этот подход хорошо масштабируется на большое число измерений.

Мы можем еще сильнее ограничить емкость модели, воспользовавшись связанной диагональной ковариационной матрицей. Такая постановка называется «диагональным ЛДА» [BL04].

9.2.4.3. Оценка MAP

Диагональность ковариационной матрицы – довольно сильное предположение. Альтернативный подход – вычислить не MLE, а оценку MAP гауссова распределения с полной (разделяемой) ковариационной матрицей. Опираясь на результаты из раздела 4.5.2, находим, что оценка MAP имеет вид

$$\hat{\boldsymbol{\Sigma}}_{\text{map}} = \lambda \text{diag}(\hat{\boldsymbol{\Sigma}}_{\text{mle}}) + (1 - \lambda) \hat{\boldsymbol{\Sigma}}_{\text{mle}}, \quad (9.22)$$

где λ контролирует степень регуляризации. Эта техника называется **регуляризованным дискриминантным анализом**, или RDA [HTF09, стр. 656].

9.2.5. Классификатор по ближайшему центроиду

Если предположить, что априорное распределение классов равномерное, то можно вычислить метку наиболее вероятного класса следующим образом:

$$\hat{y}(\mathbf{x}) = \operatorname{argmax}_c \log p(y = c | \mathbf{x}, \boldsymbol{\theta}) = \operatorname{argmax}_c (\mathbf{x} - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_c). \quad (9.23)$$

Это называется **классификатором по ближайшему центроиду**, или **классификатором по ближайшему среднему классов** (nearest class mean classifier – NCM), поскольку мы относим \mathbf{x} к классу с ближайшим $\boldsymbol{\mu}_c$, где под расстоянием понимается квадрат расстояния Махаланобиса.

Мы можем заменить его любой другой метрикой и получить решающее правило:

$$\hat{y}(\mathbf{x}) = \operatorname{argmin}_c d^2(\mathbf{x}, \boldsymbol{\mu}_c). \quad (9.24)$$

В разделе 16.2 мы обсудим, как выбирать метрики с помощью обучения, но можно поступить по-простому и положить

$$d^2(\mathbf{x}, \boldsymbol{\mu}_c) = \|\mathbf{x} - \boldsymbol{\mu}_c\|_{\mathbf{W}}^2 = (\mathbf{x} - \boldsymbol{\mu}_c)^\top (\mathbf{W}\mathbf{W}^\top) (\mathbf{x} - \boldsymbol{\mu}_c) = \|\mathbf{W}(\mathbf{x} - \boldsymbol{\mu}_c)\|^2. \quad (9.25)$$

Соответствующее апостериорное распределение классов принимает вид:

$$p(y = c | \mathbf{x}, \boldsymbol{\mu}, \mathbf{W}) = \frac{\exp\left(-\frac{1}{2}\|\mathbf{W}(\mathbf{x} - \boldsymbol{\mu}_c)\|_2^2\right)}{\sum_{c'=1}^C \exp\left(-\frac{1}{2}\|\mathbf{W}(\mathbf{x} - \boldsymbol{\mu}_{c'})\|_2^2\right)}. \quad (9.26)$$

Мы можем оптимизировать \mathbf{W} , применив градиентный спуск к дискриминантной потере. Это называется **обучением метрики ближайшего среднего классов** [Men+12]. Преимущество этой техники в том, что ее можно использовать для обучения новым классам **на одном примере**, так как нужно увидеть всего один помеченный прототип $\boldsymbol{\mu}_c$ для каждого класса (в предположении, что мы уже обучили хорошую матрицу \mathbf{W}).

9.2.6. Линейный дискриминантный анализ Фишера*

Дискриминантный анализ – это порождающий подход к классификации, который требует аппроксимировать признаки многомерным гауссовым распределением (MVN). Как уже было сказано, при большой размерности это может оказаться проблематичным. Альтернативный подход – понизить размерность признаков $\mathbf{x} \in \mathbb{R}^D$, а затем подобрать MVN, аппроксимирующее

признаки меньшей размерности $\mathbf{z} \in \mathbb{R}^K$. Самое простое – воспользоваться матрицей линейного проецирования $\mathbf{z} = \mathbf{W}\mathbf{x}$, где \mathbf{W} – матрица размера $K \times D$. Один из способов нахождения \mathbf{W} дает метод главных компонент (principal components analysis – PCA) (раздел 20.1). Однако PCA – это метод обучения без учителя, который не принимает в расчет метки классов. Поэтому получаемые признаки низкой размерности необязательно оптимальны для классификации, что иллюстрируется на рис. 9.4.

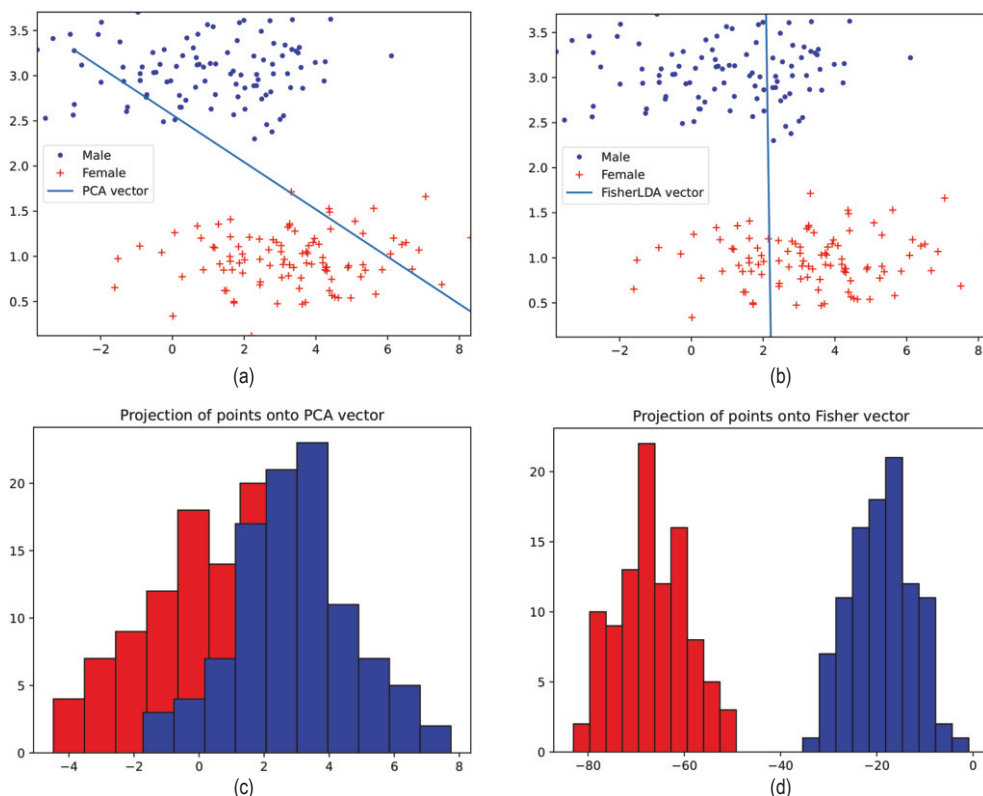


Рис. 9.4 ❖ Линейный дискриминантный анализ применительно к двухклассовому двумерному набору данных, представляющему (стандартизованные) рост и вес взрослых мужчин и женщин. (a) Направление PCA. (b) Направление FLDA. (c) Проекция на направление PCA дает плохое разделение на классы. (d) Проекция на направление FLDA дает хорошее разделение на классы. Построено программой по адресу figures.probl.ai/book1/9.4

Другой подход – воспользоваться градиентными методами для оптимизации логарифмического правдоподобия, которое выводится из апостериорного распределения классов в пространстве низкой размерности, как обсуждалось в разделе 9.2.5.

Третий подход (опирающийся на спектральное разложение, а не на градиентный оптимизатор) – найти матрицу \mathbf{W} такую, что низкоразмерные

данные можно классифицировать настолько хорошо, насколько возможно, с помощью модели гауссовой условной плотности классов. Предположение о гауссовости разумно, потому что мы вычисляем линейные комбинации (потенциально негауссовых) признаков. Этот подход называется **линейным дискриминантным анализом Фишера (FLDA)**.

FLDA – интересный гибрид дискриминантного и порождающего подхода. Недостаток его в том, что он ограничен использованием $K \leq C - 1$ измерений вне зависимости от D по причинам, которые мы объясним позже. В случае двух классов это означает, что мы ищем один вектор \mathbf{w} , на который можно спроецировать данные. Ниже мы найдем оптимальный \mathbf{w} для двух классов. Затем мы обобщим решение на случай нескольких классов и, наконец, дадим вероятностную интерпретацию этой техники.

9.2.6.1. Нахождение оптимального одномерного направления

Теперь найдем оптимальное направление \mathbf{w} для случая двух классов, следуя изложению в книге [Bis06, раздел 4.1.4]. Определим условное среднее классов:

$$\boldsymbol{\mu}_1 = \frac{1}{N_1} \sum_{n:y_n=1} \mathbf{x}_n, \quad \boldsymbol{\mu}_2 = \frac{1}{N_2} \sum_{n:y_n=2} \mathbf{x}_n. \quad (9.27)$$

Обозначим $m_k = \mathbf{w}^T \boldsymbol{\mu}_k$ проекцию каждого среднего на прямую \mathbf{w} , а $z_n = \mathbf{w}^T \mathbf{x}_n$ проекцию данных на эту прямую. Дисперсия спроецированных точек пропорциональна:

$$s_k^2 = \sum_{n:y_n=k} (z_n - m_k)^2. \quad (9.28)$$

Наша цель – найти такое направление \mathbf{w} , которое максимизирует расстояние между средними, $m_2 - m_1$, гарантируя в то же время компактность кластеров, что можно сделать, минимизировав их дисперсию. Тогда естественно взять следующую целевую функцию:

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}. \quad (9.29)$$

Правую часть можно переписать в терминах \mathbf{w} :

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}, \quad (9.30)$$

где \mathbf{S}_B – матрица межклассового рассеяния:

$$\mathbf{S}_B = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T, \quad (9.31)$$

а \mathbf{S}_W – матрица внутриклассового рассеяния:

$$\mathbf{S}_W = \sum_{n:y_n=1} (\mathbf{x}_n - \boldsymbol{\mu}_1)(\mathbf{x}_n - \boldsymbol{\mu}_1)^\top + \sum_{n:y_n=2} (\mathbf{x}_n - \boldsymbol{\mu}_2)(\mathbf{x}_n - \boldsymbol{\mu}_2)^\top. \quad (9.32)$$

Чтобы убедиться в этом, заметим, что

$$\mathbf{w}^\top \mathbf{S}_B \mathbf{w} = \mathbf{w}^\top (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \mathbf{w} = (m_2 - m_1)(m_2 - m_1) \quad (9.33)$$

и

$$\mathbf{w}^\top \mathbf{S}_W \mathbf{w} = \sum_{n:y_n=1} \mathbf{w}^\top (\mathbf{x}_n - \boldsymbol{\mu}_1)(\mathbf{x}_n - \boldsymbol{\mu}_1)^\top \mathbf{w} + \sum_{n:y_n=2} \mathbf{w}^\top (\mathbf{x}_n - \boldsymbol{\mu}_2)(\mathbf{x}_n - \boldsymbol{\mu}_2)^\top \mathbf{w} \quad (9.34)$$

$$= \sum_{n:y_n=1} (z_n - m_1)^2 + \sum_{n:y_n=2} (z_n - m_1)^2. \quad (9.35)$$

Выражение (9.30) – отношение двух скаляров; мы можем взять его производную по \mathbf{w} и приравнять нулю. Можно показать (упражнение 9.1), что $J(\mathbf{w})$ достигает максимума, когда

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w}, \quad (9.36)$$

где

$$\lambda = \frac{\mathbf{w}^\top \mathbf{S}_B \mathbf{w}}{\mathbf{w}^\top \mathbf{S}_W \mathbf{w}}. \quad (9.37)$$

Уравнение (9.36) называется **обобщенной проблемой собственных значений**. Если матрица \mathbf{S}_W обратима, то эту проблему можно свести к обычной проблеме собственных значений:

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w}. \quad (9.38)$$

Однако в случае двух классов существует более простое решение. Именно, поскольку

$$\mathbf{S}_B \mathbf{w} = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \mathbf{w} = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(m_2 - m_1), \quad (9.39)$$

то из (9.38) имеем

$$\lambda \mathbf{w} = \mathbf{S}_W^{-1} (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(m_2 - m_1), \quad (9.40)$$

$$\mathbf{w} \propto \mathbf{S}_W^{-1} (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1). \quad (9.41)$$

Поскольку нас интересует только направление, а не масштабный коэффициент, можно просто положить

$$\mathbf{w} = \mathbf{S}_W^{-1} (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1). \quad (9.42)$$

Это и есть оптимальное решение в случае двух классов. Если $\mathbf{S}_W \propto \mathbf{I}$, т. е. объединенная ковариационная матрица изотропна, то \mathbf{w} пропорционален

вектору, соединяющему средние классов. Интуитивно понятно, что на это направление имеет смысл проецировать, как показано на рис. 9.3.

9.2.6.2. Обобщение на большую размерность и несколько классов

Мы можем обобщить описанную выше идею на несколько классов и подпространства более высокой размерности, найдя *матрицу* проекции \mathbf{W} , которая отображает D -мерное пространство в K -мерное. Обозначим $\mathbf{z}_n = \mathbf{W}\mathbf{x}_n$ низкоразмерную проекцию n -й точки. Пусть $\mathbf{m}_c = (1/N_c)\sum_{n:y_n=c} \mathbf{z}_n$ — соответствующее среднее c -го класса и $\mathbf{m} = (1/N)\sum_{c=1}^C N_c \mathbf{m}_c$ — общее среднее (то и другое вычисляются в пространстве низкой размерности). Определим следующие матрицы рассеяния:

$$\tilde{\mathbf{S}}_W = \sum_{c=1}^C \sum_{n:y_n=c} (\mathbf{z}_n - \mathbf{m}_c)(\mathbf{z}_n - \mathbf{m}_c)^T; \quad (9.43)$$

$$\tilde{\mathbf{S}}_B = \sum_{c=1}^C N_c (\mathbf{m}_c - \mathbf{m})(\mathbf{m}_c - \mathbf{m})^T. \quad (9.44)$$

Наконец, определим целевую функцию, которую требует максимизировать¹:

$$J(\mathbf{W}) = \frac{|\tilde{\mathbf{S}}_B|}{|\tilde{\mathbf{S}}_W|} = \frac{|\mathbf{W}^T \mathbf{S}_B \mathbf{W}|}{|\mathbf{W}^T \mathbf{S}_W \mathbf{W}|}, \quad (9.45)$$

где \mathbf{S}_W и \mathbf{S}_B определены в исходном пространстве высокой размерности очевидным образом (а именно вместо \mathbf{z}_n используется \mathbf{x}_n , вместо $\mathbf{m}_c - \boldsymbol{\mu}_c$, а вместо $\mathbf{m} - \boldsymbol{\mu}$). Можно показать [DHS01], что решением будет $\mathbf{W} = \mathbf{S}_W^{-1/2} \mathbf{U}$, где \mathbf{U} образована K первыми собственными векторами $\mathbf{S}_W^{-1/2} \mathbf{S}_B \mathbf{S}_W^{-1/2}$, в предположении, что \mathbf{S}_W несингулярна. (Если она сингулярна, то сначала можно применить PCA ко всем данным.)

На рис. 9.5 приведен пример применения этого метода к $D = 10$ -мерным речевым данным, представляющим $C = 11$ гласных звуков. Для визуализации данных мы проецируем их на пространство $K = 2$ измерений. Как видим, FLDA дает лучшее разделение на классы, чем PCA.

Отметим, что у FLDA есть ограничение: он может найти линейное подпространство размерности K , не большей $C - 1$ вне зависимости от величины D , поскольку ранг матрицы межклассового рассеяния \mathbf{S}_B равен $C - 1$. (Член -1 появляется из-за члена $\boldsymbol{\mu}$, который является линейной функцией от $\boldsymbol{\mu}_c$.) Это довольно серьезное ограничение, уменьшающее полезность FLDA.

¹ Иногда используется другой критерий ([Fuk90]): $J(\mathbf{W}) = \text{tr}\{\tilde{\mathbf{S}}_W^{-1} \tilde{\mathbf{S}}_B\} = \text{tr}\{(\mathbf{W} \mathbf{S}_W \mathbf{W}^T)^{-1} (\mathbf{W} \mathbf{S}_B \mathbf{W}^T)\}$.

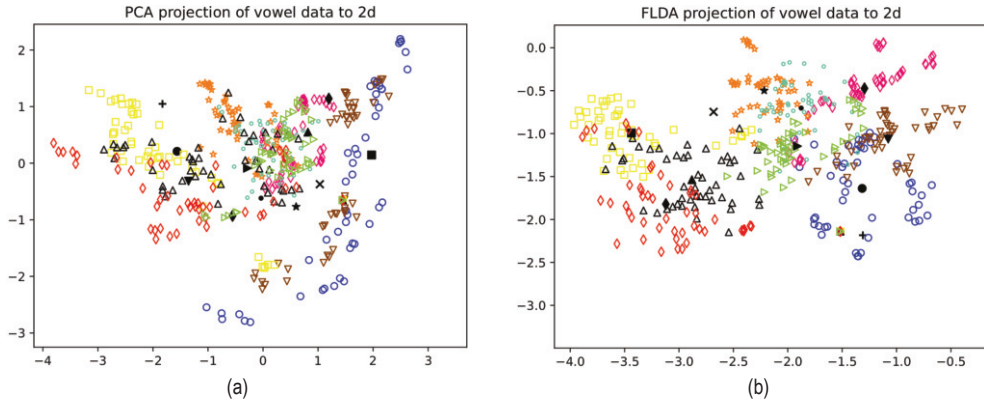


Рис. 9.5 ❖ (a) PCA-проекция данных о гласных на двумерную плоскость. (b) FLDA-проекция данных о гласных на двумерную плоскость. Как видно, наилучшее разделение на классы дает FLDA. На основе рис. 4.11 из [HTF09]. Построено программой по адресу figures.problml.ai/book1/9.5

9.3. НАИВНЫЕ БАЙЕСОВСКИЕ КЛАССИФИКАТОРЫ

В этом разделе мы обсудим простой порождающий подход к классификации, предполагающий, что признаки условно независимы при условии метки класса. Это называется **наивным байесовским предположением**. Модель называется «наивной», потому что на самом деле мы не ожидаем, что признаки будут независимыми, даже при обусловливании меткой класса. Однако хотя наивное байесовское предположение неверно, оно часто приводит к хорошо работающим классификаторам [DP97]. Одна из причин заключается в том, что эта модель проста (в ней всего $O(CD)$ параметров, где C – количество классов, а D – количество признаков), а потому мало подвержена переобучению.

Точнее, наивное байесовское предположение соответствует использованию условной плотности классов следующего вида:

$$p(\mathbf{x}|y = c, \boldsymbol{\theta}) = \prod_{d=1}^D p(x_d|y = c, \theta_{dc}), \quad (9.46)$$

где θ_{dc} – параметры условной плотности классов для класса c и признака d . Поэтому апостериорное распределение меток классов имеет вид

$$p(y = c|\mathbf{x}, \boldsymbol{\theta}) = \frac{p(y = c|\boldsymbol{\pi}) \prod_{d=1}^D p(x_d|y = c, \theta_{dc})}{\sum_{c'} p(y = c'|\boldsymbol{\pi}) \prod_{d=1}^D p(x_d|y = c', \theta_{dc'})}, \quad (9.47)$$

где π_c – априорная вероятность класса c , а $\boldsymbol{\theta} = (\boldsymbol{\pi}, \{\theta_{dc}\})$ – все параметры. Это называется **наивным байесовским классификатором** (naive Bayes classifier – NBC).

9.3.1. Примеры моделей

Нам по-прежнему нужно задать форму распределений вероятностей в формуле (9.46). Она зависит от типа признака x_d . Ниже мы приведем несколько примеров.

- В случае бинарных признаков, $x_d \in \{0, 1\}$, можно взять распределение Бернулли: $p(\mathbf{x}|\mathbf{y} = c; \boldsymbol{\theta}) = \prod_{d=1}^D \text{Ber}(x_d|\theta_{dc})$, где θ_{dc} – вероятность того, что $x_d = 1$ в классе c . Иногда это называют **многомерной бернуллиевой наивной байесовской** моделью. Например, на рис. 9.6 показаны оцененные параметры для каждого класса, когда этой моделью аппроксимируется бинаризованная версия набора данных MNIST. Подход работает на удивление хорошо, его верность на тестовом наборе составляет 84.3 %. (Примеры предсказаний показаны на рис. 9.7.)

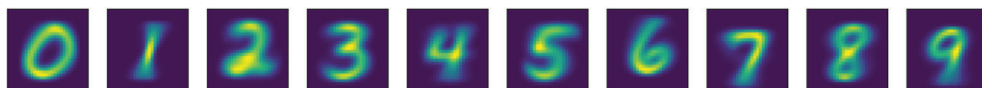


Рис. 9.6 ❖ Визуализация условных плотностей классов с распределением Бернулли для аппроксимации наивной байесовской моделью бинаризованной версии набора данных MNIST. Построено программой по адресу figures.probl.ai/book1/9.6



Рис. 9.7 ❖ Визуализация предсказаний, сделанных моделью на рис. 9.6, применительно к нескольким бинаризованным тестовым изображениям из MNIST. В заголовке показан наиболее вероятный предсказанный класс. Построено программой по адресу figures.probl.ai/book1/9.7

- В случае категориальных признаков, $x_d \in \{1, \dots, K\}$, можно использовать категориальное распределение: $p(\mathbf{x}|\mathbf{y} = c; \boldsymbol{\theta}) = \prod_{d=1}^D \text{Cat}(x_d|\boldsymbol{\theta}_{dc})$, где θ_{dck} – вероятность того, что $x_d = k$ при условии $\mathbf{y} = c$.
- В случае вещественных признаков, $x_d \in \mathbb{R}$, можно использовать одномерное гауссово распределение: $p(\mathbf{x}|\mathbf{y} = c; \boldsymbol{\theta}) = \prod_{d=1}^D \mathcal{N}(x_d|\mu_{dc}, \sigma_{dc}^2)$, где μ_{dc} – среднее признака d , когда метка класса равна c , а σ_{dc}^2 – его дисперсия. (Это эквивалентно гауссову дискриминантному анализу с диагональными ковариационными матрицами.)

9.3.2. Обучение модели

В этом разделе мы обсудим, как обучить наивный байесовский классификатор с использованием оценки максимального правдоподобия. Правдоподобие можно записать следующим образом:

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^N \text{Cat}(y_n|\boldsymbol{\pi}) \prod_{d=1}^D p(x_{nd}|y_n, \boldsymbol{\theta}_d) \quad (9.48)$$

$$= \prod_{n=1}^N \text{Cat}(y_n|\boldsymbol{\pi}) \prod_{d=1}^D \prod_{c=1}^C p(x_{nd}|\boldsymbol{\theta}_{d,c})^{\mathbb{I}(y_n=c)}, \quad (9.49)$$

так что логарифмическое правдоподобие имеет вид:

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \left[\sum_{n=1}^N \sum_{c=1}^C \mathbb{I}(y_n = c) \log \pi_c \right] + \sum_{c=1}^C \sum_{d=1}^D \left[\sum_{n:y_n=c} \log p(x_{nd}|\boldsymbol{\theta}_{dc}) \right]. \quad (9.50)$$

Как видим, оно разлагается на член, относящийся к π , и члены CD для каждого $\boldsymbol{\theta}_{dc}$:

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \log p(\mathcal{D}_y|\boldsymbol{\pi}) + \sum_c \sum_d \log p(\mathcal{D}_{dc}|\boldsymbol{\theta}_{dc}), \quad (9.51)$$

где $\mathcal{D}_y = \{y_n : n = 1 \dots N\}$ – все метки, а $\mathcal{D}_{dc} = \{x_{nd} : y_n = c\}$ – все значения признака d для примеров из класса c . Следовательно, мы можем оценивать эти параметры по отдельности.

В разделе 4.2.4 мы показали, что MLE для π – это вектор эмпирических счетчиков, $\hat{\pi}_c = N_c/N$. Оценки MLE для $\boldsymbol{\theta}_{dc}$ зависят от выбора условной плотности класса для признака d . Ниже мы обсудим несколько вариантов.

- В случае дискретных признаков можно использовать категориальное распределение. Прямолинейное обобщение результатов из раздела 4.2.4 дает следующее выражение для MLE:

$$\hat{\theta}_{dck} = \frac{N_{dck}}{\sum_{k'=1}^K N_{dck'}} = \frac{N_{dck}}{N_c}, \quad (9.52)$$

где $N_{dck} = \sum_{n=1}^N \mathbb{I}(x_{nd} = k; y_n = c)$ – сколько раз признак d принимал значение k на примерах из класса c .

- В случае бинарных признаков категориальным распределением становится распределение Бернулли, а MLE принимает вид

$$\hat{\theta}_{dc} = \frac{N_{dc}}{N_c}, \quad (9.53)$$

что совпадает с эмпирической долей испытаний, в которых признак d появлялся в примерах из класса c .

- В случае вещественных признаков можно использовать гауссово распределение. Прямолинейное обобщение результатов из раздела 4.2.5 дает следующее выражение для MLE:

$$\hat{\mu}_{dc} = \frac{1}{N_c} \sum_{n:y_n=c} x_{nd}; \quad (9.54)$$

$$\hat{\sigma}_{dc}^2 = \frac{1}{N_c} \sum_{n: y_n=c} (x_{nd} - \hat{\mu}_{dc})^2. \quad (9.55)$$

Как видим, обучение наивного байесовского классификатора производится очень просто и эффективно.

9.3.3. Байесовская интерпретация наивной байесовской модели

В этом разделе мы обобщим обсуждение оценки MLE для наивных байесовских классификаторов из раздела 9.3.2 с целью вычисления апостериорного распределения параметров. Для простоты предположим, что признаки категориальные, так что $p(x_d | \theta_{dc}) = \text{Cat}(x_d | \theta_{dc})$, где $\theta_{dck} = p(x_d = k | y = c)$. В разделе 4.6.3.2 мы показали, что сопряженным априорным для категориального правдоподобия является распределение Дирихле, $p(\theta_{dc}) = \text{Dir}(\theta_{dc} | \beta_{dc})$, где β_{dck} можно интерпретировать как набор «псевдосчетчиков», соответствующих счетчикам N_{dck} , известным из априорных данных. Аналогично мы используем априорное распределение Дирихле для частот меток, $p(\pi) = \text{Dir}(\pi | \alpha)$. Используя априорное сопряженное распределение, мы можем вычислить апостериорное распределение в замкнутой форме, как было объяснено в разделе 4.6.3. Именно, имеем

$$p(\theta | \mathcal{D}) = \text{Dir}(\pi | \hat{\alpha}) \prod_{d=1}^D \prod_{c=1}^C \text{Dir}(\theta_{dc} | \hat{\beta}_{dc}), \quad (9.56)$$

где $\hat{\alpha}_c = \tilde{\alpha}_c + N_c$ и $\hat{\beta}_{dck} = \tilde{\beta}_{dck} + N_{dck}$.

Пользуясь результатами из раздела 4.6.3.4, мы можем вывести апостериорное прогнозное распределение следующим образом. Априорное распределение метки имеет вид $p(y | \mathcal{D}) = \text{Cat}(y | \bar{\pi})$, где $\bar{\pi}_c = \hat{\alpha}_c / \sum_{c'} \hat{\alpha}_{c'}$. Для признаков имеем $p(x_d = k | y = c, \mathcal{D}) = \bar{\theta}_{dck}$, где

$$\bar{\theta}_{dck} = \frac{\hat{\beta}_{dck}}{\sum_{k'} \hat{\beta}_{dck'}} = \frac{\hat{\beta}_{dck} + N_{dck}}{\sum_{k'} \hat{\beta}_{dck'} + N_{dck'}} \quad (9.57)$$

– апостериорное среднее параметров.

Если $\hat{\beta}_{dck} = 0$, то это сводится к MLE в формуле (9.52). С другой стороны, полагая $\hat{\beta}_{dck} = 1$, мы прибавляем 1 ко всем эмпирическим счетчикам перед нормировкой. Это называется **сглаживанием прибавлением единицы**, или **сглаживанием Лапласа**. Например, в бинарном случае это дает

$$\bar{\theta}_{dc} = \frac{\hat{\beta}_{dc1} + N_{dc1}}{\hat{\beta}_{dc0} + N_{dc0} + \hat{\beta}_{dc1} + N_{dc1}} = \frac{1 + N_{dc1}}{2 + N_c}. \quad (9.58)$$

Оценив апостериорное распределение параметров, мы можем вычислить прогнозное распределение метки следующим образом:

$$p(y = c|\mathbf{x}, \mathcal{D}) \propto p(y = c|\mathcal{D}) \prod_d p(x_d|y = c, \mathcal{D}) = \bar{\pi}_c \prod_d \prod_k \bar{\theta}_{dck}^{\mathbb{I}(x_d=k)}. \quad (9.59)$$

Это дает полностью байесовскую форму наивной байесовской модели, в которой все параметры исключены путем интегрирования. (В данном случае прогнозное распределение можно получить, просто подставив средние параметры в апостериорное распределение.)

9.3.4. Связь между наивной байесовской моделью и логистической регрессией

В этом разделе мы покажем, что апостериорное распределение классов $p(y|\mathbf{x}, \boldsymbol{\theta})$ для модели NBC имеет такую же форму, как мультиномиальная логистическая регрессия. Для простоты предположим, что все признаки дискретные и у каждого имеется K состояний, хотя результат справедлив для произвольных распределений признаков, принадлежащих экспоненциальному семейству.

Пусть $x_{dk} = \mathbb{I}(x_d = k)$, т. е. \mathbf{x}_d – унитарное кодирование признака d . Тогда условную плотность классов можно записать следующим образом:

$$p(\mathbf{x}|y = c, \boldsymbol{\theta}) = \prod_{d=1}^D \text{Cat}(x_d|y = c, \boldsymbol{\theta}) = \prod_{d=1}^D \prod_{k=1}^K \theta_{dck}^{x_{dk}}. \quad (9.60)$$

Отсюда апостериорное распределение классов имеет вид:

$$\begin{aligned} p(y = c|\mathbf{x}, \boldsymbol{\theta}) &= \frac{\pi_c \prod_d \prod_k \theta_{dck}^{x_{dk}}}{\sum_{c'} \pi_{c'} \prod_d \prod_k \theta_{dc'k}^{x_{dk}}} \\ &= \frac{\exp[\log \pi_c + \sum_d \sum_k x_{dk} \log \theta_{dck}]}{\sum_{c'} \exp[\log \pi_{c'} + \sum_d \sum_k x_{dk} \log \theta_{dc'k}]}. \end{aligned} \quad (9.61)$$

Это можно записать в виде функции softmax

$$p(y = c|\mathbf{x}, \boldsymbol{\theta}) = \frac{e^{\boldsymbol{\beta}_c^\top \mathbf{x} + \gamma_c}}{\sum_{c'=1}^C e^{\boldsymbol{\beta}_{c'}^\top \mathbf{x} + \gamma_{c'}}} \quad (9.62)$$

при подходящем определении $\boldsymbol{\beta}_c$ и γ_c . Это точно такая же форма, как у мультиномиальной логистической регрессии в разделе 2.5.3. Разница в том, что в случае наивной байесовской модели мы оптимизируем совместное правдоподобие $\prod_n p(y_n, \mathbf{x}_n|\boldsymbol{\theta})$, тогда как в случае логистической регрессии – условное правдоподобие $\prod_n p(y_n|\mathbf{x}_n, \boldsymbol{\theta})$. В общем случае результаты различны (см. упражнение 10.3).

9.4. Порождающие и дискриминантные классификаторы

Модель вида $p(\mathbf{x}, y) = p(y)p(\mathbf{x}|y)$ называется **порождающим классификатором**, потому что ее можно использовать для порождения примеров \mathbf{x} из каждого класса y . А модель вида $p(y|\mathbf{x})$ называется **дискриминантным классификатором**, потому что она позволяет только различать классы. Ниже мы опишем плюсы и минусы порождающего и дискриминантного подходов к классификации.

9.4.1. Преимущества дискриминантных классификаторов

Основные преимущества дискриминантных классификаторов таковы.

- **Повышенная верность предсказаний.** Дискриминантные классификаторы зачастую дают гораздо более верные предсказания, чем порождающие [NJ02]. Причина в том, что условное распределение $p(y|\mathbf{x})$ обычно гораздо проще (а значит, легче обучается), чем совместное распределение $p(y, \mathbf{x})$, как показано на рис. 9.8. В частности, дискриминантные модели не «тратят усилий» на моделирование распределения входных признаков.

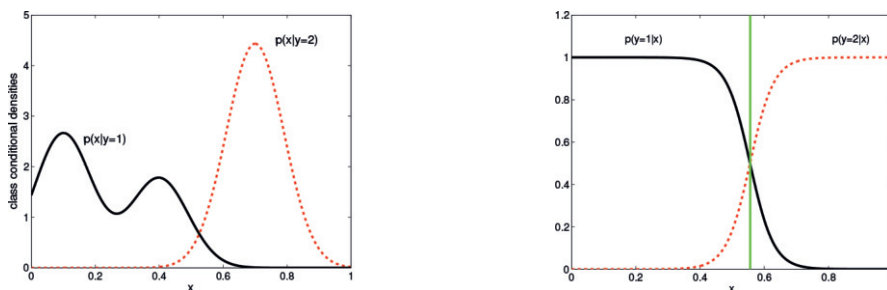


Рис. 9.8 ❖ Условные плотности классов $p(\mathbf{x}|y = c)$ (слева) могут быть более сложными, чем апостериорные распределения классов $p(y = c|\mathbf{x})$ (справа). На основе рис. 1.27 из [Bis06]. Построено программой по адресу figures.probl.ai/book1/9.8

- **Допускают предварительную обработку признаков.** Важное достоинство дискриминантных методов состоит в том, что они позволяют предварительно обрабатывать входные данные любым способом. Например, можно выполнить полиномиальное разложение входных признаков или заменить строку слов векторами погружения (см. раз-

дел 20.5). Определить порождающую модель для таких преобработанных данных часто бывает трудно, потому что новые признаки могут быть сложным образом коррелированы, что затрудняет построение модели.

- **Хорошо откалиброванные вероятности.** Некоторые порождающие классификаторы, например наивный байесовский (см. раздел 9.3), делают сильные предположения о независимости, далеко не всегда верные. Это может приводить к экстремальным апостериорным распределениям вероятностей классов (сосредоточенным вблизи 0 или 1). Дискриминантные модели, например логистическая регрессия, зачастую лучше откалиброваны в части оценок вероятностей, хотя иногда и они нуждаются в корректировке (см., например, [NMC05]).

9.4.2. Преимущества порождающих классификаторов

Основные преимущества порождающих классификаторов таковы.

- **Простота обучения.** Порождающие классификаторы часто очень легко поддаются обучению. Например, в разделе 9.3.2 мы показали, как просто обучить наивный байесовский классификатор с помощью подсчета и усреднения. Напротив, для логистической регрессии необходимо решить сложную задачу оптимизации (детали см. в разделе 10.2.3), а для обучения нейронных сетей – задачу невыпуклой оптимизации. То и другое требует гораздо больше времени.
- **Легко справляются с отсутствием части входных признаков.** Иногда часть входных данных (компонентов \mathbf{x}) недоступна наблюдению. В порождающем классификаторе на такой случай есть простой метод, описанный в разделе 1.5.5. А в дискриминантном классификаторе не существует теоретического решения проблемы, потому что модель предполагает, что \mathbf{x} всегда доступен и по нему можно обусловить.
- **Может обучаться классам по отдельности.** В порождающем классификаторе мы оцениваем параметры условной плотности каждого класса независимо (как показано в разделе 9.3.2), поэтому не нужно заново обучать модель после добавления новых классов. С другой стороны, в дискриминантных моделях все параметры взаимодействуют между собой, поэтому после добавления класса модель должна быть переобучена целиком.
- **Могут справляться с непомеченными обучающими данными.** Порождающие модели просто использовать для обучения с частичным привлечением учителя, когда в модели могут присутствовать как помеченные данные $\mathcal{D}_{xy} = \{(\mathbf{x}_n, y_n)\}$, так и непомеченные, $\mathcal{D}_x = \{\mathbf{x}_n\}$. В дискриминантных моделях это труднее, потому что не существует единственно оптимального способа задействовать \mathcal{D}_x .

9.4.3. Обработка отсутствующих признаков

Иногда во время обучения или тестирования часть входных данных \mathbf{x} отсутствует. В порождающем классификаторе эту ситуацию можно разрешить, исключив отсутствующие значения путем маргинализации. (Мы предполагаем, что отсутствие признака не несет информации о его потенциальном значении.) Напротив, в дискриминантной модели нет однозначно лучшего способа обработать отсутствующие значения, о чем шла речь в разделе 1.5.5.

Например, предположим, что отсутствует значение x_1 . Мы просто должны вычислить

$$p(y = c | \mathbf{x}_{2:D}, \boldsymbol{\theta}) \propto p(y = c | \boldsymbol{\pi}) p(\mathbf{x}_{2:D} | y = c, \boldsymbol{\theta}) \quad (9.63)$$

$$p(y = c | \boldsymbol{\pi}) \sum_{x_1} p(x_1, \mathbf{x}_{2:D} | y = c, \boldsymbol{\theta}). \quad (9.64)$$

В гауссовом дискриминантном анализе можно исключить x_1 , воспользовавшись формулами из раздела 3.2.3.

Если позволить себе наивное байесовское предположение, то ситуация еще упрощается, потому что можно вообще игнорировать член правдоподобия для x_1 . Это следует из того, что

$$\sum_{x_1} p(x_1, \mathbf{x}_{2:D} | y = c, \boldsymbol{\theta}) = \left[\sum_{x_1} p(x_1 | \boldsymbol{\theta}_{1c}) \right] \prod_{d=2}^D p(x_d | \boldsymbol{\theta}_{dc}) = \prod_{d=2}^D p(x_d | \boldsymbol{\theta}_{dc}), \quad (9.65)$$

где мы воспользовались тем фактом, что $\sum_{x_1} p(x_1 | y = c, \boldsymbol{\theta}_{1c}) = 1$.

9.5. УПРАЖНЕНИЯ

Упражнение 9.1 [вывод линейного дискриминанта Фишера].

Покажите, что точка максимума $J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$ удовлетворяет равенству

$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w}$, где $\lambda = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$. Указание: вспомните, что производная частного

двух скалярных функций равна $\frac{d}{dx} \frac{f(x)}{g(x)} = \frac{f'g - fg'}{g^2}$, где $f' = \frac{d}{dx} f(x)$ и $g' = \frac{d}{dx} g(x)$.

Также вспомните, что $\frac{d}{dx} \mathbf{x}^T \mathbf{A} \mathbf{x} = (\mathbf{A} + \mathbf{A}^T) \mathbf{x}$.

Глава 10

Логистическая регрессия

10.1. ВВЕДЕНИЕ

Логистическая регрессия – широко применяемая модель дискриминантной классификации $p(y|\mathbf{x}; \boldsymbol{\theta})$, где $\mathbf{x} \in \mathbb{R}^D$ – входной вектор фиксированной размерности, $y \in \{1, \dots, C\}$ – метка класса, а $\boldsymbol{\theta}$ – параметры. Если $C = 2$, то говорят о **бинарной логистической регрессии**, а если $C > 2$ – то о **мультиномиальной**, или **многоклассовой**.

10.2. БИНАРНАЯ ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ

Бинарной логистической регрессии соответствует следующая модель:

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \text{Ber}(y|\sigma(\mathbf{w}^T \mathbf{x} + b)), \quad (10.1)$$

где σ – сигмоидная функция, определенная в разделе 2.4.2, \mathbf{w} – веса, b – смещение, а $\boldsymbol{\theta} = (\mathbf{w}, b)$ – все параметры. Иными словами,

$$p(y = 1|\mathbf{x}; \boldsymbol{\theta}) = \alpha(a) = \frac{1}{1 + e^{-a}}, \quad (10.2)$$

где $a = \mathbf{w}^T \mathbf{x} + b$ – логарифм отношения шансов, $\log(p/(1 - p))$, где $p = p(y = 1|\mathbf{x}; \boldsymbol{\theta})$, как было объяснено в разделе 2.4.2. (В МО эту величину обычно называют **логитом** или **преактивацией**.)

Иногда мы предпочитаем использовать метки $\tilde{y} \in \{-1, +1\}$ вместо $y \in \{0, 1\}$. Вероятности этих альтернативных меток можно вычислить по формуле

$$p(\tilde{y}|\mathbf{x}, \boldsymbol{\theta}) = \sigma(\tilde{y}a), \quad (10.3)$$

так как $\sigma(-a) = 1 - \sigma(a)$. Такая чуть более компактная нотация широко используется в литературе по МО.

10.2.1. Линейные классификаторы

Сигмоида дает вероятность того, что метка класса $y = 1$. Если потеря при неправильной классификации каждого класса одинакова, то оптимальным решением будет предсказывать $y = 1$ тогда и только тогда, когда класс 1 вероятнее класса 0, как было объяснено в разделе 5.1.2.2. Таким образом,

$$f(\mathbf{x}) = \mathbb{I}(p(y = 1|\mathbf{x}) > p(y = 0|\mathbf{x})) = \mathbb{I}\left(\log \frac{p(y = 1|\mathbf{x})}{p(y = 0|\mathbf{x})} > 0\right) = \mathbb{I}(a > 0), \quad (10.4)$$

где $a = \mathbf{w}^T \mathbf{x} + b$.

Таким образом, мы можем записать функцию предсказания в виде

$$f(\mathbf{x}; \boldsymbol{\theta}) = b + \mathbf{w}^T \mathbf{x} = b + \sum_{d=1}^D w_d x_d, \quad (10.5)$$

где $\mathbf{w}^T \mathbf{x} = \langle \mathbf{w}, \mathbf{x} \rangle$ – скалярное произведение вектора весов \mathbf{w} и вектора признаков \mathbf{x} . Эта функция определяет линейную **гиперплоскость** с нормальным вектором $\mathbf{w} \in \mathbb{R}^D$ и смещением $b \in \mathbb{R}$ от начала координат.

Уравнение (10.5) можно понять, глядя на рис. 10.1a. Здесь мы видим плоскость в трехмерном пространстве признаков, проходящую через точку \mathbf{x}_0 , и нормаль к ней \mathbf{w} . Точки на поверхности удовлетворяют уравнению $\mathbf{w}^T(\mathbf{x} - \mathbf{x}_0) = 0$. Определив $b = -\mathbf{w}^T \mathbf{x}_0$, мы можем переписать это уравнение в виде $\mathbf{w}^T \mathbf{x} + b = 0$. Эта плоскость разделяет трехмерное пространство на два **полупространства** и называется **решающей границей**. Если удастся идеально разделить множество обучающих примеров такой линейной границей (не допустив ни одной ошибки классификации на обучающем наборе), то говорят, что данные **линейно разделимы**. На рис. 10.1b видно, что версия набора данных об ирисах с двумя классами и двумя признаками не допускает линейного разделения.

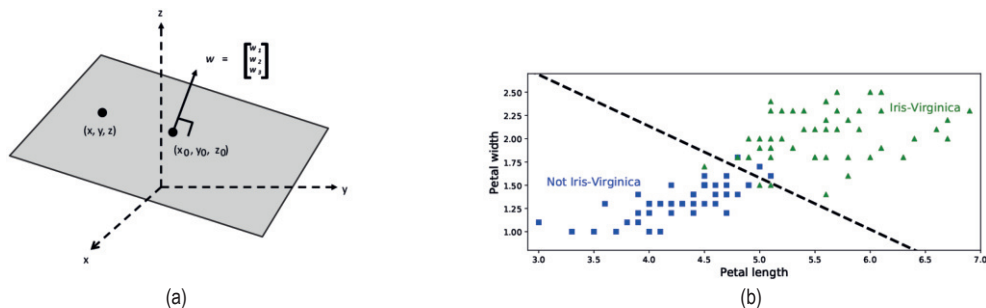


Рис. 10.1 ❖ (a) Визуализация двумерной плоскости в трехмерном пространстве и нормали к поверхности \mathbf{w} , проходящей через точку $\mathbf{x}_0 = (x_0, y_0, z_0)$. Детали см. в тексте. (b) Визуализация оптимальной линейной решающей границы, индуцированной логистической регрессией для варианта набора данных об ирисах с двумя классами и двумя признаками. Построено программой по адресу figures.problm.ai/book1/10.1. На основе рис. 4.24 из [Gér19]

В общем случае имеет место неопределенность относительно правильной метки класса, так что нужно предсказывать распределение вероятностей меток, а не просто решать, по какую сторону решающей границы находится метка. На рис. 10.2 изображены графики $p(y = 1|x_1, x_2; \mathbf{w}) = \sigma(w_1x_1 + w_2x_2)$ для различных векторов весов \mathbf{w} . Вектор \mathbf{w} определяет ориентацию решающей границы, а его модуль, $\|\mathbf{w}\| = \sqrt{\sum_{d=1}^D w_d^2}$, управляет крутизной сигмоиды, а значит, уверенностью в предсказаниях.

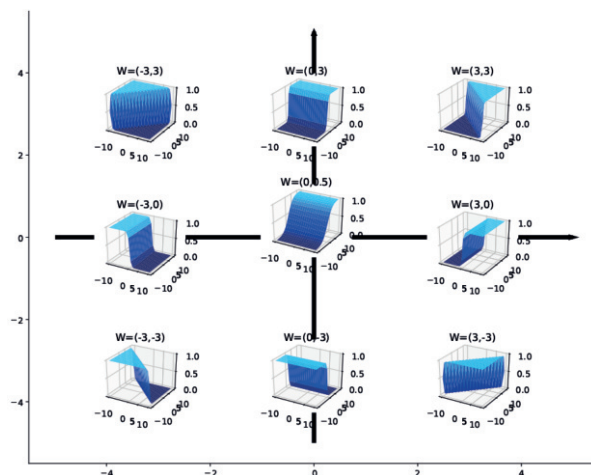


Рис. 10.2 ❖ Графики $\sigma(w_1x_1 + w_2x_2)$. Здесь вектор $\mathbf{w} = (w_1, w_2)$ определяет нормаль к решающей границе. Для точек справа от нее $\sigma(\mathbf{w}^T \mathbf{x}) > 0.5$, а для точек слева $\sigma(\mathbf{w}^T \mathbf{x}) < 0.5$. На основе рис. 39.3 из [Mac03]. Построено программой по адресу figures.problml.ai/book1/10.2

10.2.2. Нелинейные классификаторы

Нередко задачу можно сделать линейно разделимой, предварительно обработав входные данные тем или иным способом. Именно, обозначим $\phi(\mathbf{x})$ – преобразованный вектор входных признаков. Например, предположим, что $\phi(x_1, x_2) = [1, x_1^2, x_2^2]$, и пусть $\mathbf{w} = [-R^2, 1, 1]$. Тогда $\mathbf{w}^T \phi(\mathbf{x}) = x_1^2 + x_2^2 - R^2$, так что решающая граница (на которой $f(\mathbf{x}) = 0$) определяет окружность радиуса R , показанную на рис. 10.3. Результирующая функция f по-прежнему линейно зависит от параметров \mathbf{w} , что важно для упрощения задачи обучения, как мы увидим в разделе 10.2.3. Однако мы можем пойти еще дальше и обучить параметры экстрактора признаков $\phi(\mathbf{x})$ в дополнение к линейным весам \mathbf{w} ; как это делается, мы обсудим в части III.

На рис. 10.3 мы использовали квадратичное преобразование признаков. Можно использовать и полином большей степени, как в разделе 1.2.2.2. На рис. 1.7 был показан результат использования полиномов степени до K в двумерной задаче логистической регрессии. Как и на рис. 1.7, мы видим, что при

увеличении числа параметров модель становится сложнее, что в конечном итоге приводит к переобучению. Как бороться с переобучением, мы обсудим в разделе 10.2.7.

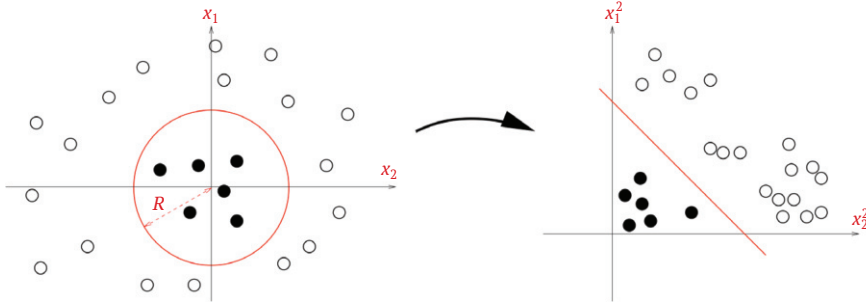


Рис. 10.3 ❖ Как можно преобразовать квадратичную решающую границу в линейную, заменив признаки $\mathbf{x} = (x_1, x_2)$ на $\phi(\mathbf{x}) = (x_1^2, x_2^2)$. Печатается с разрешения Жана-Филиппа Верта

10.2.3. Оценка максимального правдоподобия

В этом разделе мы обсудим, как оценивать параметры модели логистической регрессии с помощью оценки максимального правдоподобия.

10.2.3.1. Целевая функция

Отрицательное логарифмическое правдоподобие (масштабированное на размер набор данных N) вычисляется следующим образом (мы предполагаем, что смещение \mathbf{b} включено в вектор весов \mathbf{w}):

$$\text{NLL}(\mathbf{w}) = -\frac{1}{N} \log p(\mathcal{D}|\mathbf{w}) = -\frac{1}{N} \log \prod_{n=1}^N \text{Ber}(y_n|\mu_n) \quad (10.6)$$

$$= -\frac{1}{N} \sum_{n=1}^N \log[\mu_n^{y_n} \times (1 - \mu_n)^{1-y_n}] \quad (10.7)$$

$$= -\frac{1}{N} \sum_{n=1}^N [y_n \log \mu_n + (1 - y_n) \log(1 - \mu_n)] \quad (10.8)$$

$$= \frac{1}{N} \sum_{n=1}^N \mathbb{H}(y_n, \mu_n), \quad (10.9)$$

где $\mu_n = \sigma(a_n)$ – вероятность класса 1, $a_n = \mathbf{w}^T \mathbf{x}_n$ – логарифм отношения шансов, а $\mathbb{H}(y_n, \mu_n)$ – **бинарная перекрестная энтропия**:

$$\mathbb{H}(p, q) = -[p \log q + (1 - p) \log(1 - q)]. \quad (10.10)$$

Если взять $\tilde{y}_n \in \{-1, +1\}$ вместо $y_n \in \{0, 1\}$, то эту формулу можно переписать в виде:

$$\text{NLL}(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^N [\mathbb{I}(\tilde{y}_n = 1) \log(\sigma(a_n)) + \mathbb{I}(\tilde{y}_n = -1) \log(\sigma(-a_n))] \quad (10.11)$$

$$= -\frac{1}{N} \sum_{n=1}^N \log(\sigma(\tilde{y}_n a_n)) \quad (10.12)$$

$$= -\frac{1}{N} \sum_{n=1}^N \log(1 + \exp(-\tilde{y}_n a_n)). \quad (10.13)$$

Однако в этой книге мы в основном будем использовать нотацию $y_n \in \{0, 1\}$, потому что она проще обобщается на случай нескольких классов (раздел 10.3) и позволяет отчетливее увидеть связь с перекрестной энтропией.

10.2.3.2. Оптимизация целевой функции

Для нахождения оценки максимального правдоподобия (MLE) мы должны решить уравнение:

$$\nabla_{\mathbf{w}} \text{NLL}(\mathbf{w}) = \mathbf{g}(\mathbf{w}) = \mathbf{0}. \quad (10.14)$$

Для его решения можно использовать любой градиентный алгоритм оптимизации, например один из тех, что обсуждались в главе 8. Конкретный пример будет приведен в разделе 10.2.4. Но сначала нужно вывести градиент, и этим мы займемся ниже.

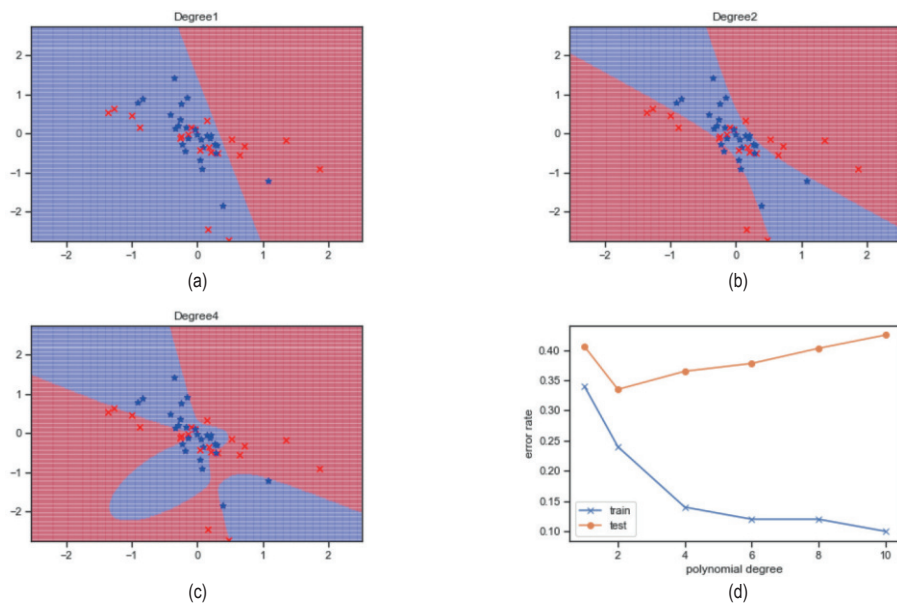


Рис. 10.4 ❖ Полиномиальное преобразование признаков применительно к двумерной задаче логистической регрессии с двумя классами. (а) Степень полинома $K = 1$. (б) Степень полинома $K = 2$. (с) Степень полинома $K = 4$. (д) Зависимость ошибки на обучающем и на тестовом наборе от степени полинома.

Построено программой по адресу figures.problml.ai/book1/10.4

10.2.3.3. Вывод градиента

Хотя для вычисления градиента отрицательного логарифмического правдоподобия (NLL) можно воспользоваться методами автоматического дифференцирования (раздел 13.3), это легко сделать и явно, как показано ниже. По счастью, получающиеся уравнения допускают простую и интуитивно понятную интерпретацию, которую можно использовать и в других методах, — и мы увидим, как именно.

Для начала заметим, что

$$\frac{d\mu_n}{da_n} = \sigma(a_n)(1 - \sigma(a_n)), \quad (10.15)$$

где $a_n = \mathbf{w}^\top \mathbf{x}_n$ и $\mu_n = \sigma(a_n)$. Отсюда по правилу дифференцирования сложной функции (и правилам векторного исчисления, рассмотренным в разделе 7.8) имеем

$$\frac{\partial}{\partial \mathbf{w}_d} \mu_n = \frac{\partial}{\partial \mathbf{w}_d} \sigma(\mathbf{w}^\top \mathbf{x}_n) = \frac{\partial}{\partial a_n} \sigma(a_n) \frac{\partial a_n}{\partial \mathbf{w}_d} = \mu_n(1 - \mu_n) \mathbf{x}_{nd}. \quad (10.16)$$

Градиент члена смещения можно вывести точно так же, подставив в формулу выше вход $x_{n0} = 1$. Однако для простоты мы проигнорируем смещение. Следовательно,

$$\nabla_{\mathbf{w}} \log(\mu_n) = \frac{1}{\mu_n} \nabla_{\mathbf{w}} \mu_n = (1 - \mu_n) \mathbf{x}_n. \quad (10.17)$$

Аналогично

$$\nabla_{\mathbf{w}} \log(1 - \mu_n) = \frac{-\mu_n(1 - \mu_n) \mathbf{x}_n}{1 - \mu_n} \nabla_{\mathbf{w}} \mu_n = -\mu_n \mathbf{x}_n. \quad (10.18)$$

Таким образом, градиент NLL равен:

$$\nabla_{\mathbf{w}} \text{NLL}(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^N [y_n(1 - \mu_n) \mathbf{x}_n - (1 - y_n) \mu_n \mathbf{x}_n] \quad (10.19)$$

$$= -\frac{1}{N} \sum_{n=1}^N [y_n \mathbf{x}_n - y_n \mathbf{x}_n \mu_n - \mathbf{x}_n \mu_n + y_n \mathbf{x}_n \mu_n] \quad (10.20)$$

$$= \frac{1}{N} \sum_{n=1}^N (\mu_n - y_n) \mathbf{x}_n. \quad (10.21)$$

Если интерпретировать $e_n = \mu_n - y_n$ как сигнал ошибки, то легко видеть, что градиент назначает каждому входу x_n вес, равный его ошибке, а затем усредняет результаты. Заметим, что градиент можно записать в матричной форме следующим образом:

$$\nabla_{\mathbf{w}} \text{NLL}(\mathbf{w}) = \frac{1}{N} (\mathbf{1}_N^\top (\text{diag}(\boldsymbol{\mu} - \mathbf{y}(\mathbf{X})))^\top. \quad (10.22)$$

10.2.3.4. Вывод гессиана

Градиентные оптимизаторы ищут стационарную точку, в которой $\mathbf{g}(\mathbf{w}) = \mathbf{0}$. Это может быть глобальный или локальный оптимум. Чтобы с уверенностью утверждать, что стационарная точка является глобальным оптимумом, мы должны показать, что целевая функция **выпуклая**; причины были объяснены в разделе 8.1.1.1. Интуитивно это означает, что NLL имеет **форму миски** с единственной нижней точкой, и это действительно так, как явствует из рис. 10.5b.

Формально мы должны доказать, что гессиан положительно полуопределен, что сейчас и сделаем. (Необходимые сведения из линейной алгебры см. в главе 7.) Можно показать, что гессиан имеет вид

$$\mathbf{H}(\mathbf{w}) = \nabla_{\mathbf{w}} \nabla_{\mathbf{w}}^T \text{NLL}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N (\mu_n(1 - \mu_n) \mathbf{x}_n) \mathbf{x}_n^T = \frac{1}{N} \mathbf{X}^T \mathbf{S} \mathbf{X}, \quad (10.23)$$

где

$$\mathbf{S} \triangleq \text{diag}(\mu_1(1 - \mu_1), \dots, \mu_N(1 - \mu_N)). \quad (10.24)$$

Мы видим, что матрица \mathbf{H} положительно определенная, так как для любого ненулевого вектора \mathbf{v} имеем

$$\mathbf{v}^T \mathbf{X}^T \mathbf{S} \mathbf{X} \mathbf{v} = (\mathbf{v}^T \mathbf{X}^T \mathbf{S}^{1/2})(\mathbf{S}^{1/2} \mathbf{X} \mathbf{v}) = \|\mathbf{v}^T \mathbf{X}^T \mathbf{S}^{1/2}\|_2^2 > 0. \quad (10.25)$$

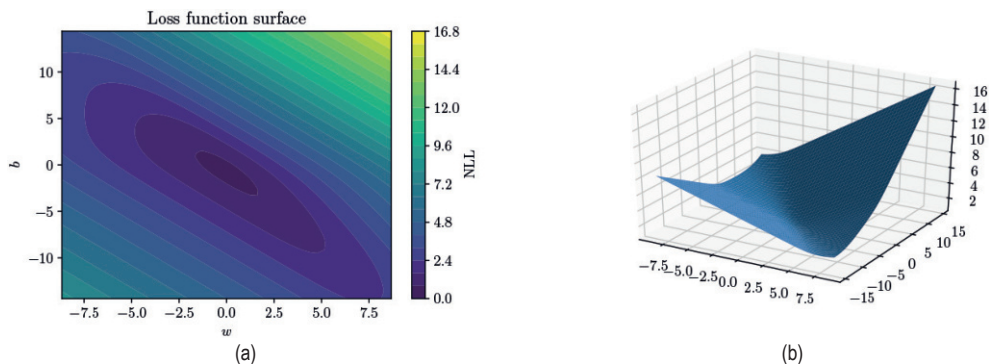


Рис. 10.5 ❖ Поверхность потери NLL для бинарной логистической регрессии применительно к набору данных об ирисах с одним признаком и одним членом смещения. Цель состоит в минимизации функции. Построено программой по адресу figures.probml.ai/book1/10.5

Это следует из того, что $\mu_n > 0$ для всех n , так как мы используем сигмоидную функцию. Следовательно, NLL строго выпукла. Однако на практике значения μ_n , близкие к 0 или 1, могут стать причиной почти сингулярности гессиана. Этого можно избежать с помощью ℓ_2 -регуляризации, как будет описано в разделе 10.2.7.

10.2.4. Стохастический градиентный спуск

Наша цель – решить следующую задачу оптимизации

$$\hat{\mathbf{w}} \triangleq \operatorname{argmin}_{\mathbf{w}} \mathcal{L}(\mathbf{w}), \quad (10.26)$$

где $\mathcal{L}(\mathbf{w})$ – функция потерь, в данном случае отрицательное логарифмическое правдоподобие:

$$\text{NLL}(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^N [y_n \log \mu_n + (1 - y_n) \log(1 - \mu_n)], \quad (10.27)$$

где $\mu_n = \sigma(a_n)$ – вероятность класса 1, а $a_n = \mathbf{w}^T \mathbf{x}_n$ – логарифм отношения шансов.

Существует много алгоритмов решения задачи (10.26), они обсуждались в главе 8. Но, пожалуй, самый простой – стохастический градиентный спуск (раздел 8.4). Используя мини-пакет размера 1, мы получим следующую простую формулу обновления:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \rho_t \nabla_{\mathbf{w}} \text{NLL}(\mathbf{w}_t) = \mathbf{w}_t - \rho_t (\mu_n - y_n) \mathbf{x}_n, \quad (10.28)$$

где мы заменили среднее во всем N примерам в градиенте (10.21) одним случайно выбранным примером n . (Индекс n меняется вместе с t .)

Поскольку мы знаем, что целевая функция выпукла (см. раздел 10.2.3.4), можно показать, что эта процедура сходится к глобальному оптимуму при условии, что скорость обучения убывает с подходящей быстротой (см. раздел 8.4.3). Скорость сходимости можно улучшить, применяя методы уменьшения дисперсии, например SAGA (раздел 8.4.5.2).

10.2.5. Алгоритм перцептрона

Перцептрон, впервые описанный в работе [Ros58], – это детерминированный бинарный классификатор вида

$$f(\mathbf{x}_n; \boldsymbol{\theta}) = \mathbb{I}(\mathbf{w}^T \mathbf{x}_n + b > 0). \quad (10.29)$$

Его можно рассматривать как ограниченный вариант бинарного классификатора на базе логистической регрессии, в которой сигмоидная функция $\sigma(a)$ заменена ступенчатой функцией Хевисайда $H(a) \triangleq \mathbb{I}(a > 0)$. Сравнение этих двух функций приведено на рис. 2.10.

Поскольку функция Хевисайда не дифференцируема, мы не можем использовать для обучения этой модели градиентные методы оптимизации. Вместо этого Розенблатт предложил **алгоритм обучения перцептрона**. Основная его идея – начать со случайных весов, а затем итеративно обновлять их, если модель делает ошибочное предсказание. Точнее, обновление весов производится по формуле

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \rho_t (\hat{y}_n - y_n) \mathbf{x}_n, \quad (10.30)$$

где (x_n, y_n) – помеченный пример, выбранный на итерации t , а ρ_t – скорость обучения, или размер шага. (Мы можем задать размер шага равным 1, поскольку абсолютные величины весов не влияют на решающую границу.) Простую реализацию этого алгоритма см. в коде по адресу code.problm.ai/book1/perceptron_demo_2d.

У правила обновления перцептрона (10.30) есть интуитивно понятная интерпретация: если предсказание правильно, то никаких изменений не вносится, в противном случае веса изменяются в таком направлении, чтобы повысить вероятность правильного ответа. Точнее, если $y_n = 1$ и $\hat{y}_n = 0$, то имеем $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x}_n$, а если $y_n = 0$ и $\hat{y}_n = 1$, то $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{x}_n$.

Сравнивая формулы (10.30) и (10.28), мы замечаем, что правило обновления эквивалентно правилу обновления СГС для бинарной логистической регрессии с аппроксимацией, предполагающей замену мягких вероятностей $\mu_n = p(y_n = 1 | \mathbf{x}_n)$ жесткими метками $\hat{y}_n = f(\mathbf{x}_n)$. Преимущество метода перцептрона заключается в том, что не нужно вычислять вероятности; это может быть полезно, когда пространство меток очень велико. А недостаток в том, что метод сходится, только когда данные линейно разделимы [Nov62], в том время как алгоритм СГС, применяемый для минимизации NLL в логистической регрессии, всегда сходится к глобально оптимальной MLE, даже если данные линейно неразделимы.

В разделе 13.2 мы обобщим перцептроны на нелинейные функции и тем самым сделаем их куда более полезными.

10.2.6. Метод наименьших квадратов с итеративным пересчетом весов

Градиентный спуск – метод оптимизации **первого порядка**, т. е. в нем для навигации по поверхности функции потерь используются только первые производные. Это может оказаться медленно, особенно когда одни направления пространства ведут резко вниз, а у других градиент более пологий, как на рис. 10.5а. В таких задачах гораздо быстрее было бы использовать метод оптимизации **второго порядка**, принимающий во внимание кривизну.

Мы подробно обсуждали такие методы в разделе 8.3. Здесь же мы рассмотрим только простой метод второго порядка, работающий для логистической регрессии. Мы ограничимся постановкой с полным пакетом (т. е. предполагаем, что N мало), потому что заставить методы второго порядка работать в стохастической постановке труднее (описание некоторых методов такого рода см., например, в [Вуг+16; Liu+18b]).

Классическим методом второго порядка является **метод Ньютона**. В нем итеративное обновление имеет вид

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \rho_t \mathbf{H}_t^{-1} \mathbf{g}_t, \quad (10.31)$$

где матрица

$$\mathbf{H}_t \triangleq \nabla^2 \mathcal{L}(\mathbf{w})|_{\mathbf{w}_t} = \nabla^2 \mathcal{L}(\mathbf{w}_t) = \mathbf{H}(\mathbf{w}_t) \quad (10.32)$$

предполагается положительно определенной, чтобы обновление было корректно определено. Если гессиан точный, то размер шага можно положить равным $\rho_t = 1$.

Теперь применим этот метод к логистической регрессии. Напомним (см. раздел 10.2.3.3), что градиент и гессиан соответственно равны

$$\nabla_{\mathbf{w}} \text{NLL}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N (\mu_n - y_n) \mathbf{x}_n; \quad (10.33)$$

$$\mathbf{H} = \frac{1}{N} \mathbf{X}^T \mathbf{S} \mathbf{X}; \quad (10.34)$$

$$\mathbf{S} \triangleq \text{diag}(\mu_1(1 - \mu_1), \dots, \mu_N(1 - \mu_N)). \quad (10.35)$$

Поэтому обновление Ньютона имеет вид:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}^{-1} \mathbf{g}_t \quad (10.36)$$

$$= \mathbf{w}_t + (\mathbf{X}^T \mathbf{S}_t \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{y} - \boldsymbol{\mu}_t) \quad (10.37)$$

$$= (\mathbf{X}^T \mathbf{S}_t \mathbf{X})^{-1} [(\mathbf{X}^T \mathbf{S}_t \mathbf{X}) \mathbf{w}_t + \mathbf{X}^T (\mathbf{y} - \boldsymbol{\mu}_t)] \quad (10.38)$$

$$= (\mathbf{X}^T \mathbf{S}_t \mathbf{X})^{-1} \mathbf{X}^T [\mathbf{S}_t \mathbf{X} \mathbf{w}_t + \mathbf{y} - \boldsymbol{\mu}_t] \quad (10.39)$$

$$= (\mathbf{X}^T \mathbf{S}_t \mathbf{X})^{-1} \mathbf{X}^T \mathbf{S}_t \mathbf{z}_t, \quad (10.40)$$

где мы определили **рабочую характеристику**

$$\mathbf{z}_t \triangleq \mathbf{X} \mathbf{w}_t + \mathbf{S}_t^{-1} (\mathbf{y} - \boldsymbol{\mu}_t) \quad (10.41)$$

и $\mathbf{S}_t = \text{diag}(\mu_{t,n}(1 - \mu_{t,n}))$. Поскольку \mathbf{S}_t – диагональная матрица, мы можем переписать цели в компонентной форме:

$$z_{t,n} = \mathbf{w}_t^T \mathbf{x}_n + \frac{y_n - \mu_{t,n}}{\mu_{t,n}(1 - \mu_{t,n})}. \quad (10.42)$$

Формула (10.40) – пример взвешенной задачи наименьших квадратов (раздел 11.2.2.4), когда минимизируется функция:

$$\sum_{n=1}^N S_{t,n} (z_{t,n} - \mathbf{w}_t^T \mathbf{x}_n)^2. \quad (10.43)$$

Поэтому алгоритм в целом называется **методом наименьших квадратов с итеративным пересчетом весов** (iteratively reweighted least squares – **IRLS**), так как на каждой итерации мы решаем взвешенную задачу наименьших квадратов, в которой матрица весов \mathbf{S}_t изменяется. Псевдокод приведен в алгоритме 2.

Заметим, что **критерий Фишера** – то же самое, что IRLS, с тем отличием, что гессиан фактического логарифмического правдоподобия заменяется его математическим ожиданием, т. е. мы используем информационную матрицу Фишера (раздел 4.7.2) вместо \mathbf{H} . Так как информационная матрица Фишера не зависит от данных, ее можно вычислить заранее, в отличие от гессиана,

который должен пересчитываться на каждой итерации. В задачах с большим числом параметров это может оказаться быстрее.

Алгоритм 2. Метод наименьших квадратов с итеративным пересчетом весов (IRLS)

```

1  w = 0;
2  repeat
3      for  $n = 1 : N$  do
4           $a_n = \mathbf{w}^T \mathbf{x}_n$ ;
5           $\mu_n = \sigma(a_n)$ ;
6           $s_n = \mu_n(1 - \mu_n)$ ;
7           $z_n = a_n + (y_n - \mu_n)/s_n$ ;
8      S =  $\text{diag}(s_{1:N})$ ;
9       $\mathbf{w} = (\mathbf{X}^T \mathbf{S} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{S} \mathbf{z}$ ;
10 until сошелся;
```

10.2.7. Оценка MAP

На рис. 10.4 мы видели, что модель логистической регрессии может оказаться переобученной, если параметров слишком много по сравнению с числом обучающих примеров. Это следствие того, что метод максимального правдоподобия может находить веса, заставляющие решающую границу «извиваться» с целью подогнать ее под примеры. Чтобы получить такое поведение, часто требуется задать большие значения весов. Например, на рис. 10.4 при использовании степени $K = 1$ мы нашли, что MLE двух входных весов (смещение игнорируется) имеет вид:

$$\hat{\mathbf{w}} = [0.51291712, 0.11866937]. \quad (10.44)$$

Если степень $K = 2$, то получаем

$$\hat{\mathbf{w}} = [2.27510513, 0.05970325, 11.84198867, 15.40355969, 2.51242311]. \quad (10.45)$$

А при $K = 4$ имеем

$$\hat{\mathbf{w}} = [-3.07813766, \dots, -59.03196044, 51.77152431, 10.25054164]. \quad (10.46)$$

Один из способов уменьшить переобучение – предотвратить появление таких больших весов. Для этого можно взять априорное гауссово распределение с нулевым средним, $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, C\mathbf{I})$, и использовать оценку MAP, как было описано в разделе 4.5.3. Новая целевая функция принимает вид

$$\mathcal{L}(\mathbf{w}) = \text{NLL}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2, \quad (10.47)$$

где $\|\mathbf{w}\|_2^2 = \sum_{d=1}^D w_d^2$ и $\lambda = 1/C$. Это называется ℓ_2 -регуляризацией, или **уменьшением весов**. Чем больше значение λ , тем больше параметры штрафуются за излишнюю величину (отклонение от нулевого априорного среднего), см. иллюстрацию на рис. 10.6.

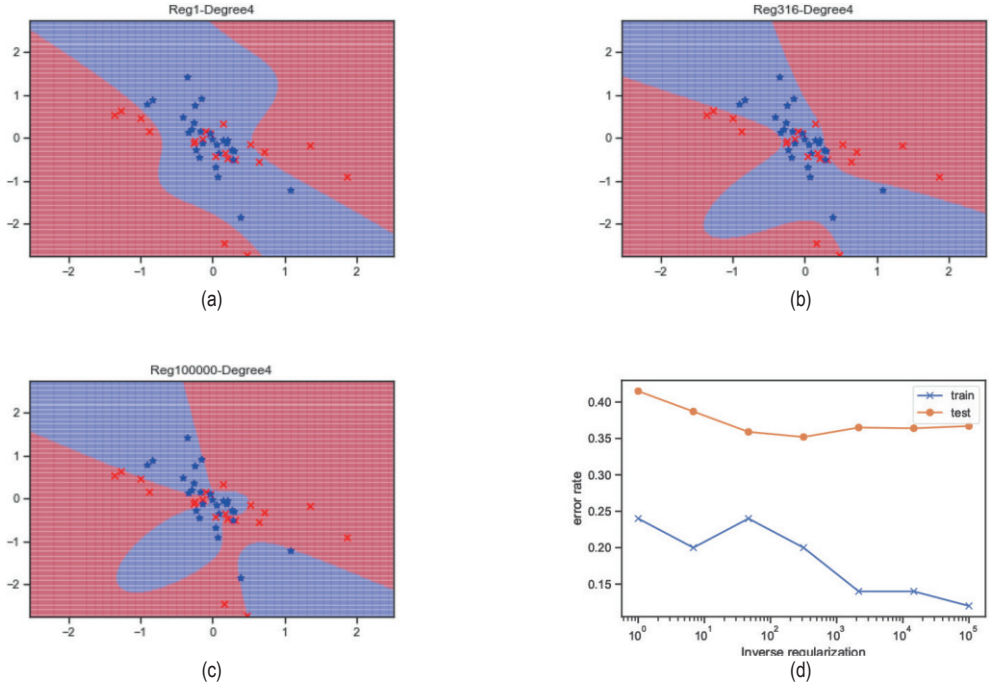


Рис. 10.6 ❖ Уменьшение весов с дисперсией C применительно к двумерной задаче логистической регрессии с двумя классами и полиномом четвертой степени. (a) $C = 1$. (b) $C = 316$. (c) $C = 100\,000$. (d) Зависимость ошибки на обучающем и на тестовом наборе от C . Построено программой по адресу figures.problml.ai/book1/10.6

Мы можем вычислить оценку MAP, немного изменив входные данные для рассмотренных выше градиентных алгоритмов оптимизации. Градиент и гессиан штрафующего отрицательного логарифмического правдоподобия имеют вид:

$$\text{PNLL}(\mathbf{w}) = \text{NLL}(\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}; \quad (10.48)$$

$$\nabla \text{PNLL}(\mathbf{w}) = \mathbf{g}(\mathbf{w}) + 2\lambda \mathbf{w}; \quad (10.49)$$

$$\nabla_{\mathbf{w}}^2 \text{PNLL}(\mathbf{w}) = \mathbf{H}(\mathbf{w}) + 2\lambda \mathbf{I}, \quad (10.50)$$

где $\mathbf{g}(\mathbf{w})$ – градиент, а $\mathbf{H}(\mathbf{w})$ – гессиан нештрафуемого NLL.

Интересный факт, связанный с ℓ_2 -регуляризированной логистической регрессией, см. в упражнении 10.2.

10.2.8. Стандартизация

В разделе 10.2.7 для предотвращения переобучения использовалось изотропное априорное распределение $\mathcal{N}(\mathbf{w}|\mathbf{0}, \lambda^{-1}\mathbf{I})$. Здесь неявно предполагается, что величины всех весов близки, что в свою очередь означает, что мы

ожидаем от входных признаков примерно одинаковой величины. Однако во многих наборах данных шкалы измерения входных признаков различны. В таких случаях принято стандартизировать данные, так чтобы все признаки имели нулевое среднее и единичную дисперсию. Для этого нужно вычесть из каждого признака среднее и разделить результат на стандартное отклонение:

$$\text{standardize}(x_{nd}) = \frac{x_{nd} - \hat{\mu}_d}{\hat{\sigma}_d}; \quad (10.51)$$

$$\hat{\mu}_d = \frac{1}{N} \sum_{n=1}^N x_{nd}; \quad (10.52)$$

$$\hat{\sigma}_d^2 = \frac{1}{N} \sum_{n=1}^N (x_{nd} - \hat{\mu}_d)^2. \quad (10.53)$$

Альтернатива – использовать **минимаксное шкалирование**, когда масштаб входных данных изменяется так, чтобы они попадали в интервал $[0, 1]$. Оба метода гарантируют, что признаки будут сравнимы по величине, что помогает обучать модель и делать статистические выводы, даже если оценка MAP не используется (см. обсуждение этого вопроса в разделе 11.7.5).

10.3. Мультиномиальная логистическая РЕГРЕССИЯ

Мультиномиальной логистической регрессией называется дискриминантная модель классификации вида

$$p(y|\mathbf{x}; \boldsymbol{\theta}) = \text{Cat}(y|\mathcal{S}(\mathbf{W}\mathbf{x} + \mathbf{b})), \quad (10.54)$$

где $\mathbf{x} \in \mathbb{R}^D$ – вектор входов, $y \in \{1, \dots, C\}$ – метка класса, $\mathcal{S}()$ – функция softmax (раздел 2.5.2), \mathbf{W} – матрица весов размера $C \times D$, \mathbf{b} – C -мерный вектор смещения, $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$ – все параметры. Далее мы будем опускать член \mathbf{b} , предполагая, что в начало каждого вектора \mathbf{x} добавлена 1, а к первому столбцу \mathbf{W} прибавлен \mathbf{b} . Таким образом, $\boldsymbol{\theta} = \mathbf{W}$.

Если обозначить $\mathbf{a} = \mathbf{W}\mathbf{x}$ C -мерный вектор **логитов**, то формулу выше можно будет переписать следующим образом:

$$p(y = c|\mathbf{x}; \boldsymbol{\theta}) = \frac{e^{a_c}}{\sum_{c'=1}^C e^{a_{c'}}}. \quad (10.55)$$

В силу условия нормировки $\sum_{c=1}^C p(y_n = c|\mathbf{x}_n; \boldsymbol{\theta}) = 1$ можно положить $\mathbf{w}_C = \mathbf{0}$. (Например, в случае бинарной логистической регрессии, когда $C = 2$, мы обучаем только один вектор весов.) Поэтому параметрам $\boldsymbol{\theta}$ соответствуют матрица весов \mathbf{W} размера $(C - 1) \times D$, где $\mathbf{x}_n \in \mathbb{R}^D$.

Заметим, что в этой модели предполагается, что метки взаимно исключают друг друга, т. е. истинна только одна метка. В некоторых приложениях (например, при **аннотировании изображений**) мы хотим, чтобы можно было предсказывать для входного вектора одну или несколько меток; в таком случае пространство выходов будет множеством подмножеств $\{1, \dots, C\}$. Это называется **многозначной классификацией**, в отличие **многоклассовой классификации**. Ее можно рассматривать как сопоставление входу битового вектора $\mathcal{Y} = \{0, 1\}^C$, c -й элемент которого равен 1, если присутствует c -я метка. Решать такую задачу можно, воспользовавшись модифицированным вариантом бинарной логистической регрессии с несколькими выходами:

$$p(y|\mathbf{x}; \boldsymbol{\theta}) = \prod_{c=1}^C \text{Ber}(y_c | \sigma(\mathbf{w}_c^T \mathbf{x})). \quad (10.56)$$

10.3.1. Линейные и нелинейные классификаторы

Логистическая регрессия вычисляет линейные решающие границы в пространстве входов, как показано на рис. 10.7а для случая, когда $\mathbf{x} \in \mathbb{R}^2$ и имеется $C = 3$ класса. Однако мы всегда можем преобразовать входы так, что границы станут нелинейными. Например, заменим $\mathbf{x} = (x_1, x_2)$ на

$$\boldsymbol{\phi}(\mathbf{x}) = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2]. \quad (10.57)$$

Тогда мы сможем создать квадратичные решающие границы, как показано на рис. 10.7b.

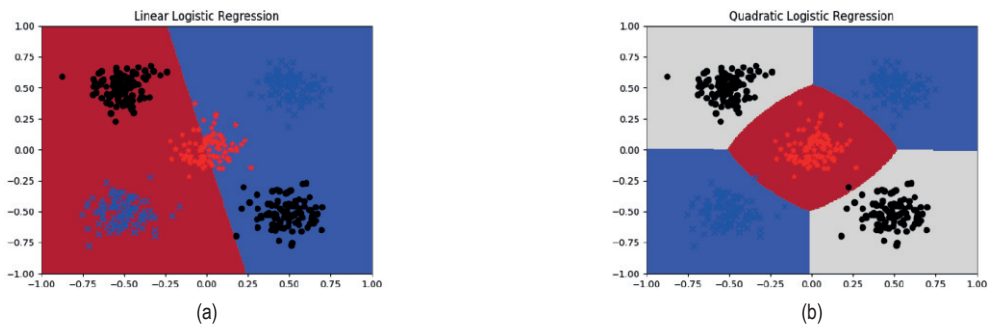


Рис. 10.7 ❖ Пример 3-классовой логистической регрессии с 2d входами. (а) Оригинальные признаки. (b) Квадратичные признаки. Построено программой по адресу figures.problml.ai/book1/10.7

10.3.2. Оценка максимального правдоподобия

В этом разделе мы обсудим, как вычислить оценку максимального правдоподобия (MLE) путем минимизации отрицательного логарифмического правдоподобия (NLL).

10.3.2.1. Целевая функция

NLL имеет вид

$$\text{NLL}(\boldsymbol{\theta}) = -\frac{1}{N} \log \prod_{n=1}^N \prod_{c=1}^C \mu_{nc}^{y_{nc}} = -\frac{1}{N} \sum_{n=1}^N \sum_{c=1}^C y_{nc} \log \mu_{nc} = \frac{1}{N} \sum_{n=1}^N \mathbb{H}(\mathbf{y}_n, \boldsymbol{\mu}_n), \quad (10.58)$$

где $\mu_{nc} = p(y_{nc} = 1 | \mathbf{x}_n, \boldsymbol{\theta}) = \mathcal{S}(f(\mathbf{x}_n; \boldsymbol{\theta}))_c$, \mathbf{y}_n – унитарное кодирование метки (т. е. $y_{nc} = \mathbb{I}(y_n = c)$), а $\mathbb{H}(\mathbf{y}_n, \boldsymbol{\mu}_n)$ – перекрестная энтропия:

$$\mathbb{H}(\mathbf{p}, \mathbf{q}) = -\sum_{c=1}^C p_c \log q_c. \quad (10.59)$$

10.3.2.2. Оптимизация целевой функции

Для нахождения оптимума необходимо решить уравнение $\nabla_{\mathbf{w}} \text{NLL}(\mathbf{w}) = \mathbf{0}$, где \mathbf{w} – векторизованная матрица весов \mathbf{W} , а член смещения для простоты обозначений игнорируется. Найти стационарную точку этой функции мы можем с помощью любого градиентного оптимизатора, несколько примеров будет приведено ниже. Но сначала выведем градиент и гессиан, а затем докажем, что целевая функция выпукла.

10.3.2.3. Вывод градиента

Чтобы вывести градиент NLL, необходимо воспользоваться якобианом функции softmax, который выглядит следующим образом (вы докажете это в упражнении 10.1):

$$\frac{\partial \mu_c}{\partial a_j} = \mu_c (\delta_{cj} - \mu_j), \quad (10.60)$$

где $\delta_{cj} = \mathbb{I}(c = j)$. Например, если имеется 3 класса, то якобиан имеет вид:

$$\left[\frac{\partial \mu_c}{\partial a_j} \right] = \begin{pmatrix} \mu_1(1 - \mu_1) & -\mu_1\mu_2 & -\mu_1\mu_3 \\ -\mu_2\mu_1 & \mu_2(1 - \mu_2) & -\mu_2\mu_3 \\ -\mu_3\mu_1 & -\mu_3\mu_2 & \mu_3(1 - \mu_3) \end{pmatrix}. \quad (10.61)$$

В матричной форме это можно записать так:

$$\frac{\partial \boldsymbol{\mu}}{\partial \mathbf{a}} = (\boldsymbol{\mu} \mathbf{1}^T) \odot (\mathbf{I} - \mathbf{1} \boldsymbol{\mu}^T), \quad (10.62)$$

где \odot – поэлементное произведение, $\boldsymbol{\mu} \mathbf{1}^T$ копирует $\boldsymbol{\mu}$ в столбцы, а $\mathbf{1} \boldsymbol{\mu}^T$ копирует $\boldsymbol{\mu}$ в строки.

Теперь выведем градиент NLL для одного примера с индексом n . Для этого нужно будет развернуть матрицу весов размера $D \times C$ в вектор \mathbf{w} длины CD (или $(C - 1)D$, если одному классу назначен фиксированный нулевой вес), т. е. конкатенировать все ее строки и затем транспонировать в вектор-столбец.

Обозначим \mathbf{w}_j вектор весов, ассоциированный с классом j . Градиент по этому вектору равен (здесь используется дельта-символ Кронекера δ_{jc} , равный 1, если $j = c$, и 0 в противном случае):

$$\nabla_{\mathbf{w}_j} \text{NLL}_n = \sum_c \frac{\partial \text{NLL}_n}{\partial \mu_{nc}} \frac{\partial \mu_{nc}}{\partial a_{nj}} \frac{\partial a_{nj}}{\partial \mathbf{w}_j} \quad (10.63)$$

$$= - \sum_c \frac{y_{nc}}{\mu_{nc}} \mu_{nc} (\delta_{jc} - \mu_{nj}) \mathbf{x}_n \quad (10.64)$$

$$= \sum_c y_{nc} (\mu_{nj} - \delta_{jc}) \mathbf{x}_n \quad (10.65)$$

$$= \left(\sum_c y_{nc} \right) \mu_{nj} \mathbf{x}_n - \sum_c \delta_{jc} y_{nj} \mathbf{x}_n \quad (10.66)$$

$$= (\mu_{nj} - y_{nj}) \mathbf{x}_n. \quad (10.67)$$

Повторив это вычисление для каждого класса, мы получим полный вектор градиента. Градиент всего NLL получается суммированием по примерам, что дает матрицу размера $D \times C$:

$$\mathbf{g}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n (\boldsymbol{\mu}_n - \mathbf{y}_n)^\top. \quad (10.68)$$

Форма такая же, как в случае бинарной логистической регрессии, – член ошибки, умноженный на вход.

10.3.2.4. Вывод гессиана

В упражнении 10.1 вам будет предложено доказать, что гессиан NLL в случае мультиномиальной логистической регрессии равен:

$$\mathbf{H}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N (\text{diag}(\boldsymbol{\mu}_n) - \boldsymbol{\mu}_n \boldsymbol{\mu}_n^\top) \otimes (\mathbf{x}_n \mathbf{x}_n^\top), \quad (10.69)$$

где $\mathbf{A} \otimes \mathbf{B}$ – произведение Кронекера (раздел 7.2.5). Иными словами, блочная подматрица c, c' имеет вид:

$$\mathbf{H}_{c,c'}(\mathbf{w}) = \frac{1}{N} \sum_n \mu_{nc} (\delta_{c,c'} - \mu_{n,c'}) \mathbf{x}_n \mathbf{x}_n^\top. \quad (10.70)$$

Например, при трех признаках и двух классах эта формула принимает вид:

$$\mathbf{H}(\mathbf{w}) = \frac{1}{N} \sum_n \begin{pmatrix} \mu_{n1} - \mu_{n1}^2 & -\mu_{n1}\mu_{n2} \\ -\mu_{n1}\mu_{n2} & \mu_{n2} - \mu_{n2}^2 \end{pmatrix} \otimes \begin{pmatrix} x_{n1}x_{n1} & x_{n1}x_{n2} & x_{n1}x_{n3} \\ x_{n2}x_{n1} & x_{n2}x_{n2} & x_{n2}x_{n3} \\ x_{n3}x_{n1} & x_{n3}x_{n2} & x_{n3}x_{n3} \end{pmatrix} \quad (10.71)$$

$$= \frac{1}{N} \sum_n \begin{pmatrix} (\mu_{n1} - \mu_{n1}^2) \mathbf{X}_n & -\mu_{n1}\mu_{n2} \mathbf{X}_n \\ -\mu_{n1}\mu_{n2} \mathbf{X}_n & (\mu_{n2} - \mu_{n2}^2) \mathbf{X}_n \end{pmatrix}, \quad (10.72)$$

где $\mathbf{X}_n = \mathbf{x}_n \mathbf{x}_n^T$. В упражнении 10.1 также предлагается доказать, что это положительно определенная матрица, т. е. целевая функция выпуклая.

10.3.3. Градиентная оптимизация

Градиент, выведенный в разделе 10.3.2.3, можно использовать для построения алгоритма СГС. Аналогично гессиан из раздела 10.3.2.4 можно использовать для построения алгоритма оптимизации второго порядка. Однако вычисление гессиана обычно обходится дорого, так что его часто аппроксимируют с помощью квазиньютоновских методов, например BFGS с ограниченной памятью (BFGS – инициалы авторов: Бройден, Флетчер, Гольдфарб и Шанно). Детали см. в разделе 8.3.2. Другой подход, похожий на IRLS, описан в разделе 10.3.4.

Все эти методы опираются на вычисление градиента логарифмического правдоподобия, что в свою очередь требует вычисления нормированных вероятностей, которые можно вычислить по вектору логитов $\mathbf{a} = \mathbf{W}\mathbf{x}$:

$$p(y = c|\mathbf{x}) = \exp(a_c - \text{lse}(\mathbf{a})), \quad (10.73)$$

где lse – функция $\log\text{-sum-exp}$, определенная в разделе 2.5.4. По этой причине во многих программных библиотеках определен вариант потери перекрестной энтропии, принимающий на входе ненормированные логиты.

10.3.4. Граничная оптимизация

В этом разделе мы рассмотрим подход к обучению модели логистической регрессии с применением класса алгоритмов граничной оптимизации, описанного в разделе 8.7. Основная идея заключается в итеративном построении нижней границы максимизируемой функции с последующим обновлением границы, так чтобы она «пододвигалась ближе» к истинной функции. Оптимизировать границу часто бывает проще, чем обновлять функцию непосредственно.

Если $LL(\boldsymbol{\theta})$ – вогнутая функция, которую мы хотим максимизировать, то один из способов получить нижнюю границу – воспользоваться границей ее гессиана, т. е. найти такую отрицательно определенную матрицу \mathbf{B} , что $\mathbf{H}(\boldsymbol{\theta}) \succ \mathbf{B}$. В таком случае можно показать, что

$$LL(\boldsymbol{\theta}) \geq LL(\boldsymbol{\theta}^t) + (\boldsymbol{\theta} - \boldsymbol{\theta}^t)^T \mathbf{g}(\boldsymbol{\theta}^t) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^t)^T \mathbf{B}(\boldsymbol{\theta} - \boldsymbol{\theta}^t), \quad (10.74)$$

где $\mathbf{g}(\boldsymbol{\theta}_t) = \nabla LL(\boldsymbol{\theta}_t)$. Если обозначить $Q(\boldsymbol{\theta}, \boldsymbol{\theta}_t)$ правую часть (10.74), то обновление принимает вид:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \mathbf{B}^{-1} \mathbf{g}(\boldsymbol{\theta}^t). \quad (10.75)$$

Это похоже на обновление в методе Ньютона, только используется не фиксированная матрица \mathbf{B} , а матрица $\mathbf{H}(\boldsymbol{\theta}_t)$, которая изменяется на каждой

итерации. Таким образом, мы можем получить некоторые преимущества методов второго порядка, выполнив меньше вычислений.

Применим эту идею к логистической регрессии, следуя работе [Kri+05]. Обозначим $\boldsymbol{\mu}_n(\mathbf{w}) = [p(y_n = 1|\mathbf{x}_n, \mathbf{w}), \dots, p(y_n = C|\mathbf{x}_n, \mathbf{w})]$ и $y_n = [\mathbb{I}(y_n = 1), \dots, \mathbb{I}(y_n = C)]$. Мы хотим максимизировать логарифмическое правдоподобие, имеющее вид:

$$LL(\mathbf{w}) = \sum_{n=1}^N \left[\sum_{c=1}^C y_{nc} \mathbf{w}_c^\top \mathbf{x}_n - \log \sum_{c=1}^C \exp(\mathbf{w}_c^\top \mathbf{x}_n) \right]. \quad (10.76)$$

Его градиент равен (вывод см. в разделе 10.3.2.3)

$$\mathbf{g}(\mathbf{w}) = \sum_{n=1}^N (\mathbf{y}_n - \boldsymbol{\mu}_n(\mathbf{w})) \otimes \mathbf{x}_n, \quad (10.77)$$

где \otimes обозначает произведение Кронекера (в данном случае – просто внешнее произведение двух векторов). Гессиан равен (вывод см. в разделе 10.3.2.4)

$$\mathbf{H}(\mathbf{w}) = - \sum_{n=1}^N (\text{diag}(\boldsymbol{\mu}_n(\mathbf{w}) - \boldsymbol{\mu}_n(\mathbf{w})\boldsymbol{\mu}_n(\mathbf{w})^\top) \otimes (\mathbf{x}_n \mathbf{x}_n^\top). \quad (10.78)$$

Нижнюю границу гессиана можно построить, как описано в работе [Boh92]:

$$\mathbf{H}(\mathbf{w}) \succ -\frac{1}{2}[\mathbf{I} - \mathbf{1}\mathbf{1}^\top/C] \otimes \left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^\top \right) \triangleq \mathbf{B}, \quad (10.79)$$

где \mathbf{I} – C -мерная единичная матрица, а $\mathbf{1}$ – C -мерный вектор, состоящий из одних единиц¹. В бинарном случае эта формула принимает вид:

$$\mathbf{H}(\mathbf{w}) \succ -\frac{1}{2} \left(1 - \frac{1}{2} \right) \left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^\top \right) = -\frac{1}{4} \mathbf{X}^\top \mathbf{X}. \quad (10.80)$$

Это следует из того, что $\mu_n \leq 0.5$, так что $-(\mu_n - \mu_n^2) \geq -0.25$.

Мы можем воспользоваться этой нижней границей для построения ММ-алгоритма нахождения MLE. Обновление принимает вид:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \mathbf{B}^{-1} \mathbf{g}(\mathbf{w}^t). \quad (10.81)$$

Такая итерация может оказаться быстрее IRLS (раздел 10.2.6), потому что мы можем заранее вычислить \mathbf{B}^{-1} за время, не зависящее от N , вместо того чтобы обращать гессиан на каждой итерации. Например, рассмотрим бинарный случай, когда $\mathbf{g}^t = \nabla LL(\mathbf{w}^t) = \mathbf{X}^\top (\mathbf{y} - \boldsymbol{\mu}^t)$, где $\boldsymbol{\mu}^t = [p_n(\mathbf{w}^t), (1 - p_n(\mathbf{w}^t))]_{n=1}^N$. Обновление принимает вид:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - 4(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{g}^t. \quad (10.82)$$

¹ Если зафиксировать $\mathbf{w}_C = \mathbf{0}$, то в этих векторах и матрицах можно использовать $C - 1$ измерений.

Сравните это с формулой (10.37), имеющей вид

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \mathbf{H}^{-1} \mathbf{g}(\mathbf{w}^t) = \mathbf{w}^t - (\mathbf{X}^T \mathbf{S}^t \mathbf{X})^{-1} \mathbf{g}^t, \quad (10.83)$$

где $\mathbf{S}^t = \text{diag}(\boldsymbol{\mu}^t \odot (1 - \boldsymbol{\mu}^t))$. Мы видим, что вычисления по формуле (10.82) быстрее, так как постоянную матрицу $(\mathbf{X}^T \mathbf{X})^{-1}$ можно вычислить предварительно.

10.3.5. Оценка MAP

В разделе 10.2.7 мы обсуждали преимущества ℓ_2 -регуляризации для бинарной логистической регрессии. Эти преимущества имеют место и в многоклассовом случае. Однако имеется и еще одно неожиданное преимущество, касающееся **идентифицируемости** параметров, отмеченное в [HTF09, упражнение 18.3]. (Говорят, что параметры идентифицируемы, если существует единственное значение, доставляющее максимум правдоподобию; эквивалентно мы требуем, чтобы NLL было *строго* выпуклым.)

Чтобы понять, почему идентифицируемость важна, вспомним, что многоклассовая логистическая регрессия имеет вид

$$p(y = c | \mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{k=1}^C \exp(\mathbf{w}_k^T \mathbf{x})}, \quad (10.84)$$

где \mathbf{W} – матрица весов размера $C \times D$. Мы можем произвольно положить $\mathbf{w}_c = \mathbf{0}$ для одного из классов, скажем $c = C$, потому что $p(y = C | \mathbf{x}, \mathbf{W}) = 1 - \sum_{c=1}^{C-1} p(y = c | \mathbf{x}, \mathbf{w})$. В этом случае модель имеет вид:

$$p(y = c | \mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{1 + \sum_{k=1}^{C-1} \exp(\mathbf{w}_k^T \mathbf{x})}. \quad (10.85)$$

Если не зафиксировать какое-нибудь постоянное значение одного из векторов, то параметры будут неидентифицируемыми.

Но предположим, что мы не зафиксировали $\mathbf{w}_c = \mathbf{0}$, т. е. пользуемся формулой (10.84), но добавили ℓ_2 -регуляризацию и оптимизируем

$$\text{PNLL}(\mathbf{W}) = -\sum_{n=1}^N \log p(y_n | \mathbf{x}_n, \mathbf{W}) + \lambda \sum_{c=1}^C \|\mathbf{w}_c\|_2^2, \quad (10.86)$$

где член $1/N$ включен в λ . В точке оптимума имеем $\sum_{c=1}^C \hat{w}_{cj} = 0$ для $j = 1 \dots D$, так что веса автоматически удовлетворяют ограничению нулевой суммы, что делает их однозначно идентифицируемыми. Чтобы понять, почему, заметим, что в точке оптимума

$$\nabla \text{NLL}(\mathbf{w}) + 2\lambda \mathbf{w} = \mathbf{0}; \quad (10.87)$$

$$\sum_n (\mathbf{y}_n - \boldsymbol{\mu}_n) \otimes \mathbf{x}_n = \lambda \mathbf{w}. \quad (10.88)$$

Поэтому для любого измерения признаков j имеем

$$\begin{aligned}\lambda \sum_c w_{cj} &= \sum_n \sum_c (y_{nc} - \mu_{nc}) x_{nj} = \sum_n \left(\sum_c y_{nc} - \sum_c \mu_{nc} \right) x_{nj} \\ &= \sum_n (1 - 1) x_{nj} = 0.\end{aligned}\quad (10.89)$$

Таким образом, если $\lambda > 0$, то $\sum_c \hat{w}_{cj} = 0$, так что сумма весов по всем классам равна нулю для любого измерения признаков.

10.3.6. Классификаторы максимальной энтропии

Напомним, что модель мультиномиальной регрессии можно записать в виде

$$p(y = c | \mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{Z(\mathbf{w}, \mathbf{x})} = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{c'=1}^{C-1} \exp(\mathbf{w}_{c'}^T \mathbf{x})}, \quad (10.90)$$

где $Z(\mathbf{w}, \mathbf{x}) = \sum_c \exp(\mathbf{w}_c^T \mathbf{x})$ – функция разбиения (нормировочная постоянная). Здесь для каждого класса используются одни и те же признаки, но разные векторы весов. Эту модель можно немного обобщить, допустив признаки, зависящие от класса. Такая модель записывается в виде

$$p(y = c | \mathbf{x}, \mathbf{w}) = \frac{1}{Z(\mathbf{w}, \mathbf{x})} \exp(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}, c)), \quad (10.91)$$

где $\boldsymbol{\phi}(\mathbf{x}, c)$ – вектор признаков для класса c . Это называется **классификатором максимальной энтропии**, для краткости **maxent**. (Происхождение этого термина объясняется в разделе 3.4.4.)

Классификаторы максимальной энтропии включают мультиномиальную логистическую регрессию в качестве частного случая. Чтобы убедиться в этом, обозначим $\mathbf{w} = [\mathbf{w}_1, \dots, \mathbf{w}_C]$ и определим вектор признаков следующим образом:

$$\boldsymbol{\phi}(\mathbf{x}, c) = [\mathbf{0}, \dots, \mathbf{x}, \dots, \mathbf{0}], \quad (10.92)$$

где \mathbf{x} включен в c -й блок, а остальные блоки равны нулю. В данном случае $\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}, c) = \mathbf{w}_c^T \mathbf{x}$, так что мы вернулись к мультиномиальной логистической регрессии.

Классификаторы максимальной энтропии очень широко используются в области обработки естественного языка. Например, рассмотрим задачу **разметки семантических ролей**, где классификатор относит слово \mathbf{x} к семантической роли y , например: человек, место или предмет. Мы могли бы определить (бинарные) признаки, например:

$$\phi_1(\mathbf{x}, y) = \mathbb{I}(y = \text{человек} \wedge \mathbf{x} \text{ встречается после «Mr.» или «Mrs.»); \quad (10.93)$$

$$\phi_2(\mathbf{x}, y) = \mathbb{I}(y = \text{человек} \wedge \mathbf{x} \text{ находится в белом списке частых имен); \quad (10.94)$$

$$\phi_3(\mathbf{x}, y) = \mathbb{I}(y = \text{место} \wedge \mathbf{x} \text{ встречается в картах Google}). \quad (10.95)$$

Как видим, используемые нами признаки зависят от метки.

Есть два основных способа создания таких признаков. Первый – вручную задать много потенциально полезных признаков, применяя различные шаблоны, а затем воспользоваться каким-нибудь алгоритмом выделения признаков, например групповым методом LASSO из раздела 11.4.7. Второй – постепенно добавлять в модель признаки, применяя метод эвристического порождения признаков.

10.3.7. Иерархическая классификация

Иногда множество возможных меток можно структурировать, образовав **иерархию**, или **таксономию**. Например, допустим, что нужно предсказать, какое животное изображено на картинке: собака или кошка; если это собака, то она может быть золотистым ретривером или немецкой овчаркой и т. д. Интуитивно кажется, что разумно предсказывать самую точную метку из тех, в которых мы уверены [Den+12], т. е. система должна подстраховывать себя.

В работе [RF17] предложен следующий простой способ это сделать. Сначала создадим модель, включающую бинарную выходную метку для каждого возможного узла в дереве. Перед тем как приступить к обучению модели, произведем **нанизывание меток** (label smearing), применив метки из всех родительских узлов (**гиперонимов**). Например, если изображение помечено как «золотистый ретривер», то мы пометим его также как «собака». Если обучить многозначный классификатор (который порождает вектор $p(y|x)$ бинарных меток) на таких данных, то он будет выполнять иерархическую классификацию, т. е. предсказывать набор меток с разных уровней абстракции.

Однако этот метод мог бы предсказать каждую из меток «золотистый ретривер», «кошка» и «птица» с вероятностью 1.0, потому что модель не улавливает тот факт, что некоторые метки взаимно исключают друг друга. Чтобы предотвратить это, мы можем добавить ограничение взаимного исключения между всеми узлами-братьями, как показано на рис. 10.8. Например, эта модель гарантирует, что $p(\text{mammal}|x) + p(\text{bird}|x) = 1$, так как обе эти метки являются прямыми потомками корневого узла. Далее мы можем распределить вероятность оказаться млекопитающим (mammal) между собаками и кошками, так что $p(\text{dog}|x) + p(\text{cat}|x) = p(\text{mammal}|x)$.

В работах [Den+14; Din+15] этот метод обобщен на условные графовые модели со структурой графа более сложной, чем дерево. Кроме того, допускаются не только жесткие, но и мягкие ограничения между метками.

10.3.8. Работа с большим числом классов

В этом разделе мы обсудим некоторые вопросы, возникающие, когда число потенциальных меток велико, например когда метки соответствуют словам языка.

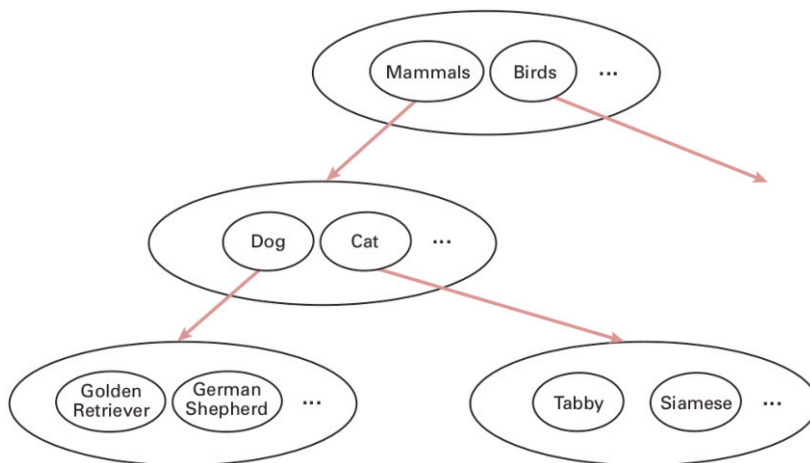


Рис. 10.8 ❖ Простой пример иерархии меток.
Узлы внутри одного эллипса взаимно исключают друг друга

10.3.8.1. Иерархическая softmax-модель

В регулярных softmax-классификаторах нахождение нормировочной постоянной, необходимой для вычисления градиента логарифмического правдоподобия, занимает время $O(C)$, и может стать узким местом, если C велико. Однако если организовать метки в виде дерева, то можно будет вычислить вероятность любой метки за время $O(\log C)$, перемножив вероятности всех ребер на пути от корня к листу. Например, рассмотрим дерево на рис. 10.9. Имеем

$$p(y = \text{I'm} | C) = 0.57 \times 0.68 \times 0.72 = 0.28. \quad (10.96)$$

Таким образом, мы заменили «плоскую» выходную softmax-модель древовидной последовательностью бинарных классификаторов. Это называется **иерархической softmax-моделью** [Goo01; MB05].

Неплохой способ построить такое дерево дает кодирование Хаффмана, когда самые частые метки помещаются в начало дерева, как предложено в работе [Mik+13a]. (Другой подход, основанный на объединении самых распространенных меток в один кластер, описан в работе [Gra+17]. А еще один – основанный на выборке меток – в работе [Tit16].)

10.3.8.2. Несбалансированность классов и длинный хвост

Еще одна проблема, часто возникающая при большом числе классов, – наличие слишком малого числа примеров для большинства классов. Точнее, если N_c – число примеров для класса c , то эмпирическое распределение $p(N_1, \dots, N_C)$ может иметь **длинный хвост**. Результат – крайняя форма **несбалансированности классов** (см., например, [ASR15]). Поскольку редкие классы оказывают меньшее влияние на общую потерю, чем частые, модель может «акцентировать внимание» только на часто встречающихся классах.

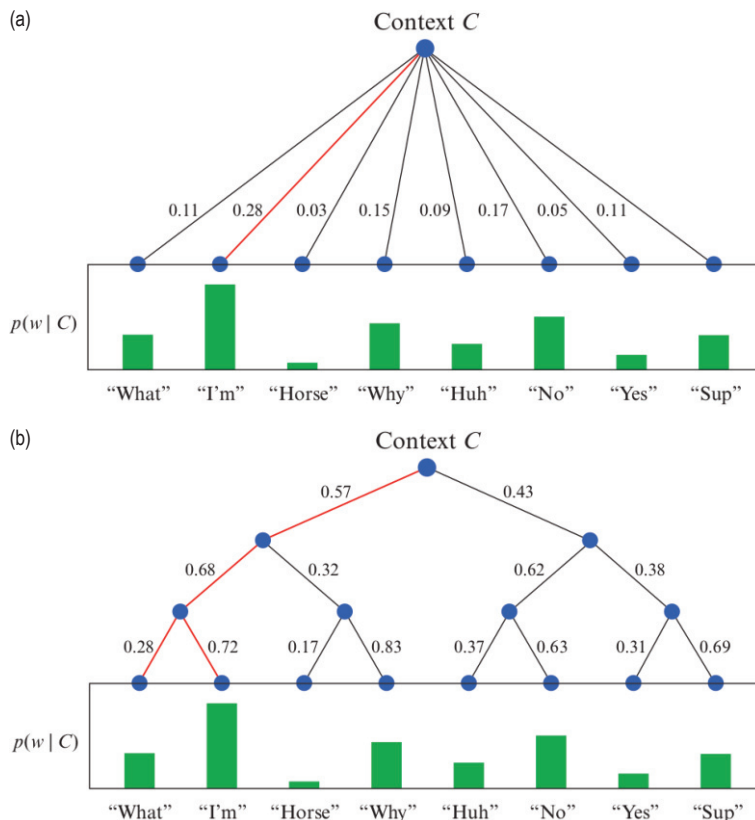


Рис. 10.9 ❖ Плоская и иерархическая softmax-модель $p(w|C)$, где C – входные признаки (контекст), а w – выходная метка (слово). На основе рисунка из статьи <https://www.quora.com/What-is-hierarchical-softmax>

Решить проблему помогает задание членов смещения \mathbf{b} , так что $\mathcal{S}(\mathbf{b})_c = N_c/N$; такая модель соответствует эмпирическому априорному распределению меток даже при использовании весов $\mathbf{w} = 0$. По мере корректировки весов модель может обучиться зависящим от входных данных отклонениям от этого априорного распределения.

Еще один распространенный подход – произвести повторную выборку данных, чтобы сделать их более сбалансированными, до (или во время) обучения. Предположим, что пример выбирается из класса c с вероятностью

$$p_c = \frac{N_c^q}{\sum_i^C N_i^q}. \quad (10.97)$$

Если положить $q = 1$, то мы вернемся к стандартной **выборке со сбалансированными экземплярами**, когда $p_c \propto N_c$; распространенные классы будут выбираться чаще, чем редкие. Если положить $q = 0$, то мы придем к **выборке со сбалансированными классами**, когда $p_c = 1/C$; можно считать, что при этом сначала случайно и равномерно выбирается класс, а затем экземпляр

из этого класса. Наконец, можно рассматривать и другие значения, например $q = 0.5$, этот случай называется **выборкой по правилу квадратного корня** (square-root sampling) [Mah+18].

Есть еще один метод, простой и хорошо справляющийся с длинным хвостом, – использовать **классификатор по ближайшему среднему классов**. Он имеет вид:

$$f(\mathbf{x}) = \operatorname{argmin}_c \|\mathbf{x} - \boldsymbol{\mu}_c\|_2^2, \quad (10.98)$$

где $\boldsymbol{\mu}_c = \frac{1}{N_c} \sum_{n: y_n = c} \mathbf{x}_n$ – среднее признаков, принадлежащих классу c . Это индуцирует апостериорное softmax-распределение, обсуждавшееся в разделе 9.2.5. Мы можем получить гораздо лучшие результаты, если сначала с помощью нейронной сети (см. часть III) найдем хорошие признаки, для чего обучим ГНС-классификатор с потерей перекрестной энтропии на исходных несбалансированных данных. Затем заменим \mathbf{x} на $\boldsymbol{\phi}(\mathbf{x})$ в формуле (10.98). Этот простой подход может дать очень хорошее качество на распределениях с длинным хвостом [Kan+20].

10.4. РОБАСТНАЯ ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ*

Иногда в данных имеются **выбросы**. Зачастую это связано с ошибками при пометке данных – **зашумленными метками**. Чтобы предотвратить негативное воздействие такого загрязнения на модель, используется **робастная логистическая регрессия**. В этом разделе мы обсудим некоторые подходы к проблеме. (Отметим, что эти методы применимы также к ГНС. Более подробный обзор зашумленных меток и их влияния на глубокое обучение см. в работе [Han+20].)

10.4.1. Смесовая модель правдоподобия

Один из самых простых способов определить модель робастной логистической регрессии – модифицировать правдоподобие, так чтобы оно предсказывало, что каждая выходная метка y либо с вероятностью π выбирается из равномерного распределения, либо порождается обычной условной моделью. В бинарном случае это выглядит так:

$$p(y|\mathbf{x}) = \pi \operatorname{Ber}(y|0.5) + (1 - \pi) \operatorname{Ber}(y|\boldsymbol{\sigma}(\mathbf{w}^T \mathbf{x})). \quad (10.99)$$

Такой способ использования смесовой модели с целью сделать модель наблюдений робастной применим к самым разным моделям (например, к ГНС).

Мы можем обучить эту модель стандартными методами, например СГС или методами байесовского вывода, скажем методом Монте-Карло по схеме марковской цепи (МСМС). Для примера давайте создадим «загрязненную версию» одномерного набора данных об ирисах с двумя классами, который

обсуждали в разделе 4.6.7.2. Мы добавим 6 примеров из класса 1 (ирис щети-нистый) с аномальной длиной чашелистика. На рис. 10.10a показаны результаты обучения стандартной (байесовской) модели логистической регрессии на этом наборе данных. На рис. 10.10b показаны результаты обучения описанной выше робастной модели. В последнем случае мы видим, что решающая граница похожа на ту, что выведена на основе незагрязненных данных (рис. 4.20b). Также видно, что апостериорная неопределенность положения решающей границы меньше, чем при использовании неробастной модели.

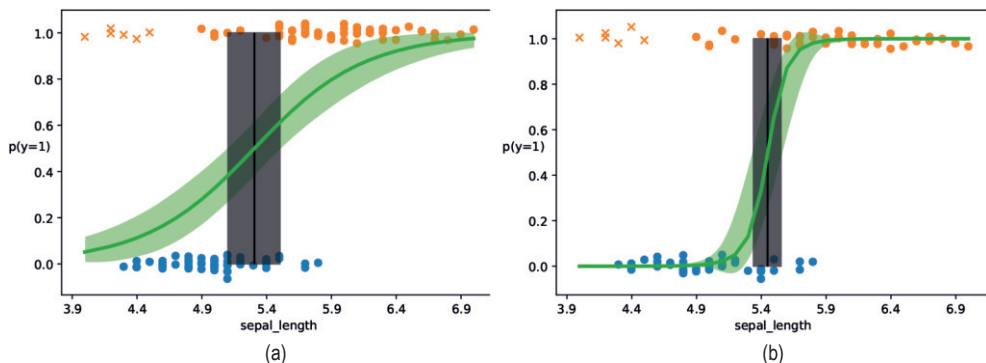


Рис. 10.10 ❖ (а) Логистическая регрессия для данных с выбросами (обозначены крестиками). Обучающие примеры разнесены по вертикали, чтобы избежать чрезмерного наложения. Вертикальная прямая является решающей границей, а широкая полоса – ее апостериорным байесовским доверительным интервалом. (б) То же, что (а), но используется робастная модель со смесовым правдоподобием. На основе рис 4.13 из of [Mar18]. Построено программой по адресу figures.problml.ai/book1/10.10

10.4.2. Дважды смягченная потеря

В этом разделе мы опишем подход к робастной логистической регрессии, предложенный в работе [Ami+19].

Прежде всего заметим, что примеры, расположенные далеко от решающей границы, но помеченные неправильно, могут оказать несоразмерно большое негативное влияние на модель, если функция потерь выпукла [LS10]. С этим можно бороться, заменив обычную потерю перекрестной энтропии «смягченной» версией с параметром температуры $0 \leq t_1 < 1$, который гарантирует, что потеря из-за выбросов будет ограничена. В частности, рассмотрим стандартную функцию потерь, равную относительной энтропии:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \mathbb{H}(\mathbf{y}, \hat{\mathbf{y}}) = \sum_c y_c \log \hat{y}_c, \quad (10.100)$$

где \mathbf{y} – истинное распределение меток (часто унитарное), а $\hat{\mathbf{y}}$ – предсказанное распределение. Определим потерю **смягченной перекрестной энтропии** следующим образом:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \sum_c \left[y_c (\log_{t_1} y_c - \log_{t_1} \hat{y}_c) - \frac{1}{2 - t_1} (y_c^{2-t_1} - \hat{y}_c^{2-t_1}) \right]. \quad (10.101)$$

Это выражение можно упростить, если истинное распределение \mathbf{y} является унитарным, т. е. вся его масса сосредоточена в классе c :

$$\mathcal{L}(c, \hat{\mathbf{y}}) = -\log_{t_1} \hat{y}_c - \frac{1}{2 - t_1} \left(1 - \sum_{c'=1}^C \hat{y}_{c'}^{2-t_1} \right). \quad (10.102)$$

Здесь \log_t – смягченная версия логарифмической функции:

$$\log_t(x) \triangleq \frac{1}{1-t} (x^{1-t} - 1). \quad (10.103)$$

Эта функция монотонно возрастающая и вогнутая, а при $t = 1$ сводится к стандартному натуральному логарифму. (Аналогично смягченная перекрестная энтропия сводится к стандартной перекрестной энтропии при $t = 1$.) Однако смягченная логарифмическая функция ограничена снизу значением $-1/(1-t)$ для $0 \leq t < 1$, а значит, потеря смягченной перекрестной энтропии ограничена сверху (см. рис. 10.1).

Второе наблюдение заключается в том, что для примеров, расположенных близко к решающей границе, но помеченных неправильно, необходимо использовать передаточную функцию (которая отображает активации \mathbb{R}^C в вероятности $[0, 1]^C$), имеющую более тяжелые хвосты, чем функция softmax, основанная на экспоненте, и потому позволяющую «заглядывать дальше» ближайшей окрестности. Напомним, что стандартная softmax определена следующим образом:

$$\hat{y}_c = \frac{a_c}{\sum_{c'=1}^C \exp(a_{c'})} = \exp \left[a_c - \log \sum_{c'=1}^C \exp(a_{c'}) \right], \quad (10.104)$$

где \mathbf{a} – вектор логитов. Получить вариант с тяжелыми хвостами можно, воспользовавшись **смягченной softmax** с параметром температуры $t_2 > 1 > t_1$:

$$\hat{y}_c = \exp_{t_2}(a_c - \lambda_{t_2}(\mathbf{a})), \quad (10.105)$$

где

$$\exp_t(x) \triangleq [1 + (1-t)x]_+^{1/(1-t)} \quad (10.106)$$

– смягченная версия экспоненциальной функции. (Она сводится к обычной экспоненте при $t \rightarrow 1$.) На рис. 10.11 (справа) показано, что смягченная softmax (в случае двух классов) имеет более тяжелые хвосты, что нам и нужно.

Осталось только вычислить $\lambda_{t_2}(\mathbf{a})$. Эта величина должна удовлетворять следующему уравнению неподвижной точки:

$$\sum_{c=1}^C \exp_{t_2}(a_c - \lambda(\mathbf{a})) = 1. \quad (10.107)$$

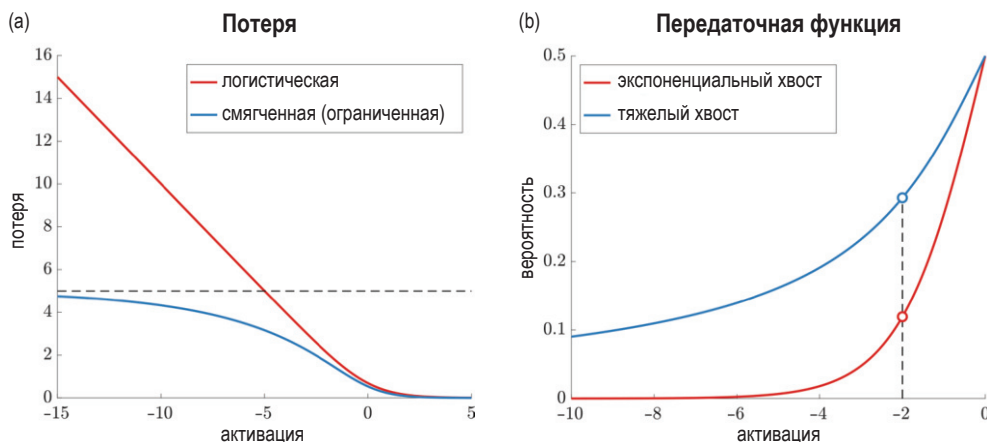


Рис. 10.11 ❖ (а) Логистическая и смягченная логистическая потеря при $t_1 = 0.8$.
 (б) Сигмоидная и смягченная сигмоидная передаточная функция при $t_2 = 2.0$.
 Взято из статьи <https://ai.googleblog.com/2019/08/bi-tempered-logistic-loss-for-training.html>. Печатается с разрешения Эхсана Амида

Решить это уравнение относительно λ можно методом двоичного поиска или с помощью итеративной процедуры в алгоритме 3.

Алгоритм 3. Итеративный алгоритм вычисления $\lambda(\mathbf{a})$ в уравнении (10.107).
 Взят из [AWS19]

```

1  Вход: логиты  $\mathbf{a}$ , температура  $t > 1$ ;
2   $\mu := \max(\mathbf{a})$ ;
3   $\mathbf{a} := \mathbf{a} - \mu$ ;
4  while  $\mathbf{a}$  не сошелся do
5     $Z(\mathbf{a}) := \sum_{c=1}^C \exp_t(a_c)$ ;
6     $\mathbf{a} := Z(\mathbf{a})^{1-t}(\mathbf{a} - \mu \mathbf{1})$ ;
7  Вернуть  $-\log_t(1/Z(\mathbf{a})) + \mu$ 

```

Сочетание смягченной softmax и смягченной перекрестной энтропии приводит к методу **дважды смягченной логистической регрессии** (bi-tempered logistic regression). На рис. 10.12 показан пример ее применения к двумерным данным. В верхнем ряду показана стандартная логистическая регрессия, а в нижнем – дважды смягченная. Первый столбец содержит чистые данные. Во втором столбце данные зашумлены вблизи границы. В робастной версии используются параметры $t_1 = 1$ (стандартная перекрестная энтропия), но $t_2 = 4$ (смягченная softmax с тяжелыми хвостами). В третьем столбце метки зашумлены вдали от границы. В робастной версии используется $t_1 = 0.2$ (смягченная перекрестная энтропия с ограниченной потерей), но $t_2 = 1$ (стандартная softmax). В четвертом столбце присутствуют оба вида шума; в этом случае в робастной версии используются параметры $t_1 = 0.2$ и $t_2 = 4$.

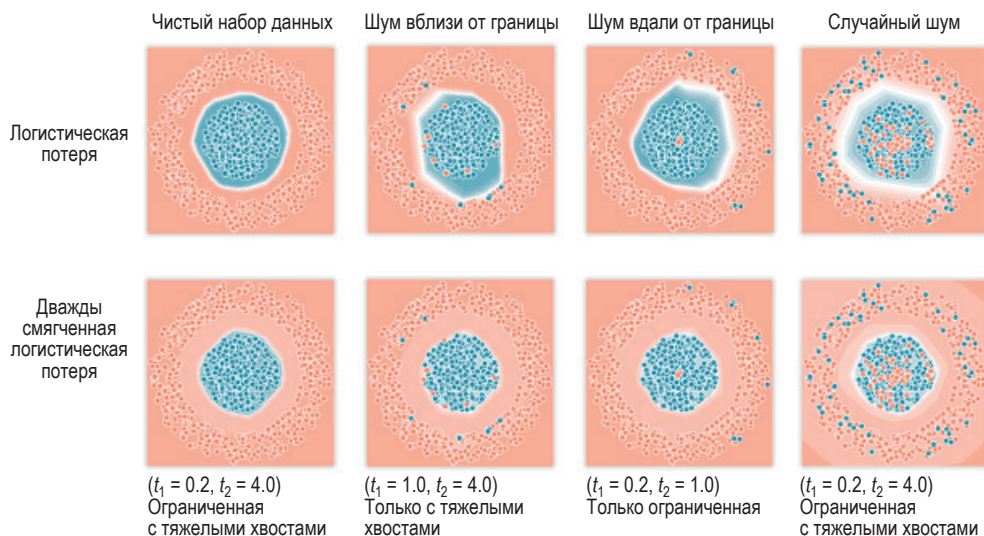


Рис. 10.12 ❖ Стандартная и дважды смягченная логистическая регрессия для данных с зашумленными метками. Взято из статьи <https://ai.googleblog.com/2019/08/bi-tempered-logistic-loss-for-training.html>. Печатается с разрешения Эхсана Амида

10.5. БАЙЕСОВСКАЯ ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ*

До сих пор в центре нашего внимания были точечные оценки параметров, MLE или MAP. Однако иногда мы хотим вычислить апостериорное распределение, $p(\mathbf{w}|\mathcal{D})$, чтобы уловить неопределенность наших предсказаний. Особенно это может быть полезно, когда данных мало, а выбор неверного решения может дорого обойтись.

В отличие от линейной регрессии для модели логистической регрессии точно вычислить апостериорное распределение невозможно. Но в нашем распоряжении широкий спектр приближенных алгоритмов. В этом разделе мы рассмотрим один из самых простых, аппроксимацию Лапласа (раздел 4.6.8.2). Более сложные аппроксимации описаны во второй томе этой книги, [Mur22].

10.5.1. Аппроксимация Лапласа

В разделе 4.6.8.2 мы говорили, что аппроксимация Лапласа аппроксимирует апостериорное распределение гауссовым. Среднее гауссова распределения равно оценке MAP $\hat{\mathbf{w}}$, а ковариационная матрица совпадает с матрицей, обратной гессиану \mathbf{H} и вычисленной в точке оценки MAP, т. е. $p(\mathbf{w}|\mathcal{D}) \approx \mathcal{N}(\mathbf{w}|\hat{\mathbf{w}}, \mathbf{H}^{-1})$. Мы можем найти моду с помощью стандартного метода оптимизации (см. раздел 10.2.7), а затем использовать результаты из раздела 10.2.3.4 для вычисления гессиана в точке моды.

В качестве примера рассмотрим данные на рис. 10.13а. Существует много вариантов задания параметров, при которых прямые идеально разделяют обучающие данные; мы показали четыре примера таких прямых. Поверхность правдоподобия показана на рис. 10.13b. Диагональная прямая соединяет начало координат с точкой сетки с максимальным правдоподобием, $\hat{\mathbf{w}}_{\text{mle}} = (8.0; 3.4)$. (Для неограниченной MLE $\|\mathbf{w}\| = \infty$, как обсуждалось в разделе 10.2.7; эту точку можно получить, продлив диагональную прямую бесконечно далеко вправо.)

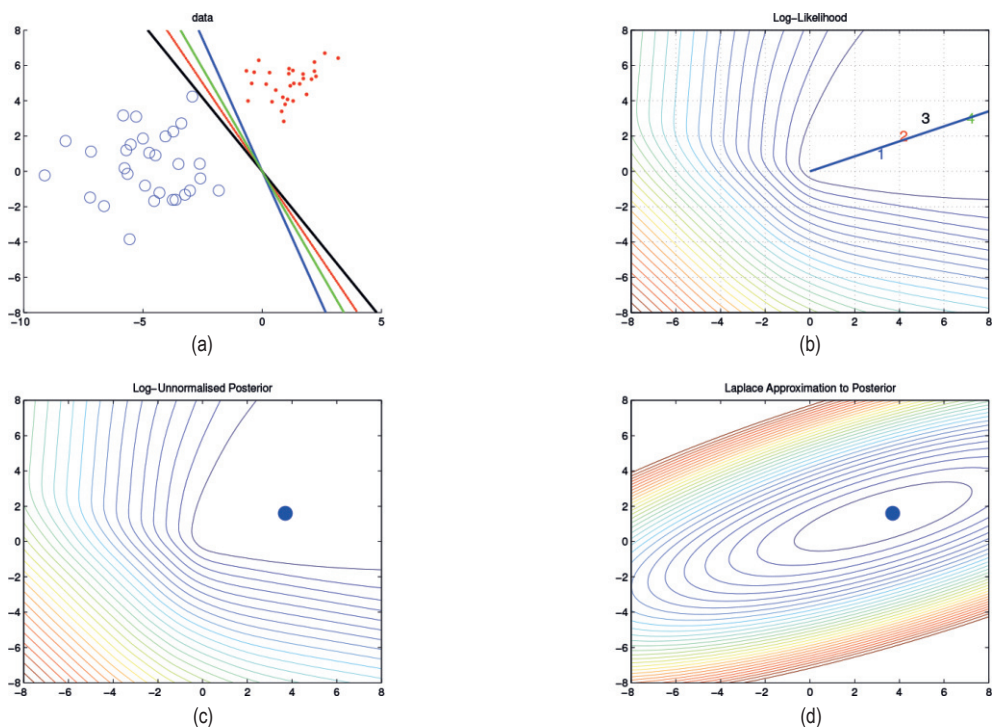


Рис. 10.13 ❖ (а) Данные. (б) Логарифмическое правдоподобие для модели логистической регрессии. Прямая проведена из начала координат в направлении оценки MLE (находящейся в бесконечности). Числа относятся к четырем точкам в пространстве параметров, соответствующим прямым на рисунке (а). (с) Ненормированное логарифмическое апостериорное распределение (в предположении сферического априорного распределения). (d) Аппроксимация Лапласа апостериорного распределения. На основе рисунка Марка Джиролами. Построено программой по адресу figures.probabml.ai/book1/10.13

Для каждой решающей границы на рис. 10.13а мы построили соответствующий вектор параметров на рис. 10.13b. Это точки $\mathbf{w}_1 = (3, 1)$, $\mathbf{w}_2 = (4, 2)$, $\mathbf{w}_3 = (5, 3)$ и $\mathbf{w}_4 = (7, 3)$. Все они удовлетворяют приближенному равенству $\mathbf{w}_i(1)/\mathbf{w}_i(2) \approx \hat{\mathbf{w}}_{\text{mle}}(1)/\hat{\mathbf{w}}_{\text{mle}}(2)$, а потому близки к направлению решающей границы максимального правдоподобия. Точки отсортированы в порядке возрастания нормы веса (3.16, 4.47, 5.83, 7.62).

Чтобы решение было единственным, мы используем (сферическое) гауссово априорное распределение с центром в начале координат, $\mathcal{N}(\mathbf{w}|\mathbf{0}, \sigma^2\mathbf{I})$. Значение σ^2 определяет силу априорного распределения. Если положить $\sigma^2 = \infty$, то оценка MAP будет равна $\mathbf{w} = \mathbf{0}$; это дает максимально неопределенные предсказания, потому что все точки \mathbf{x} порождают прогнозное распределение вида $p(y = 1|\mathbf{x}) = 0.5$. Если положить $\sigma^2 = 0$, то оценка MAP совпадает с MLE, что дает минимально неопределенные предсказания. (В частности, для всех точек с положительными метками будет $p(y = 1|\mathbf{x}) = 1.0$, а для всех точек с отрицательными метками – $p(y = 1|\mathbf{x}) = 0.0$, потому что данные разделимы.) В качестве компромисса (чтобы получить красивую иллюстрацию) мы выбрали значение $\sigma^2 = 100$.

Умножение этого априорного распределения на правдоподобие дает ненормированное апостериорное распределение, показанное на рис. 10.13c. Оценка MAP показана красной точкой. Аппроксимация Лапласа этого апостериорного распределения показана на рис. 10.13d. Мы видим, что оно дает правильную моду (по построению), но форма распределения несколько искажена. (Направление с юго-запада на северо-восток отражает неопределенность абсолютной величины \mathbf{w} , а направление с юго-востока на северо-запад – неопределенность ориентации решающей границы.)

На рис. 10.14 показаны линии уровня апостериорного прогнозного распределения. На рис. 10.14a изображена подстановочная аппроксимация с использованием оценки MAP. Мы видим, что в предсказании решающей границы нет никакой неопределенности, хотя мы генерировали вероятностное распределение меток. На рис. 10.14b показано, что происходит, если подставить примеры, выбранные из гауссова апостериорного распределения. Теперь имеется значительная неопределенность в ориентации «наилучшей» решающей границы. На рис. 10.14c показано среднее этих примеров. Усредняя по нескольким предсказаниям, мы видим, что неопределенность решающей границы «расширяется» по мере того, как мы удаляемся от обучающих данных. На рис. 10.14d показано, что результаты пробит-аппроксимации очень похожи на аппроксимацию Монте-Карло.

10.5.2. Аппроксимация апостериорного прогнозного распределения

Апостериорное распределение $p(\mathbf{w}|\mathcal{D})$ сообщает нам все, что известно о параметрах модели при имеющихся данных. Однако в приложениях машинного обучения наибольший интерес обычно представляет предсказание выхода y для заданного входа \mathbf{x} , а не попытка понять параметры модели. Таким образом, нам нужно вычислить **апостериорное прогнозное распределение**:

$$p(y|\mathbf{x}, \mathcal{D}) = \int p(y|\mathbf{x}, \mathbf{w})p(\mathbf{w}|\mathcal{D})d\mathbf{w}. \quad (10.108)$$

В разделе 4.6.7.1 мы обсуждали простой подход к решению этой задачи: сначала вычислить точечную оценку $\hat{\mathbf{w}}$ параметров, например оценку MLE

или MAP, а затем игнорировать всю апостериорную неопределенность, предположив, что $p(\mathbf{w}|\mathcal{D}) = \delta(\mathbf{w} - \hat{\mathbf{w}})$. В данном случае приведенный выше интеграл сводится к следующей подстановочной аппроксимации:

$$p(y|\mathbf{x}, \mathcal{D}) \approx \int p(y|\mathbf{x}, \mathbf{w})\delta(\mathbf{w} - \hat{\mathbf{w}})d\mathbf{w} = p(y|\mathbf{x}, \hat{\mathbf{w}}). \quad (10.109)$$

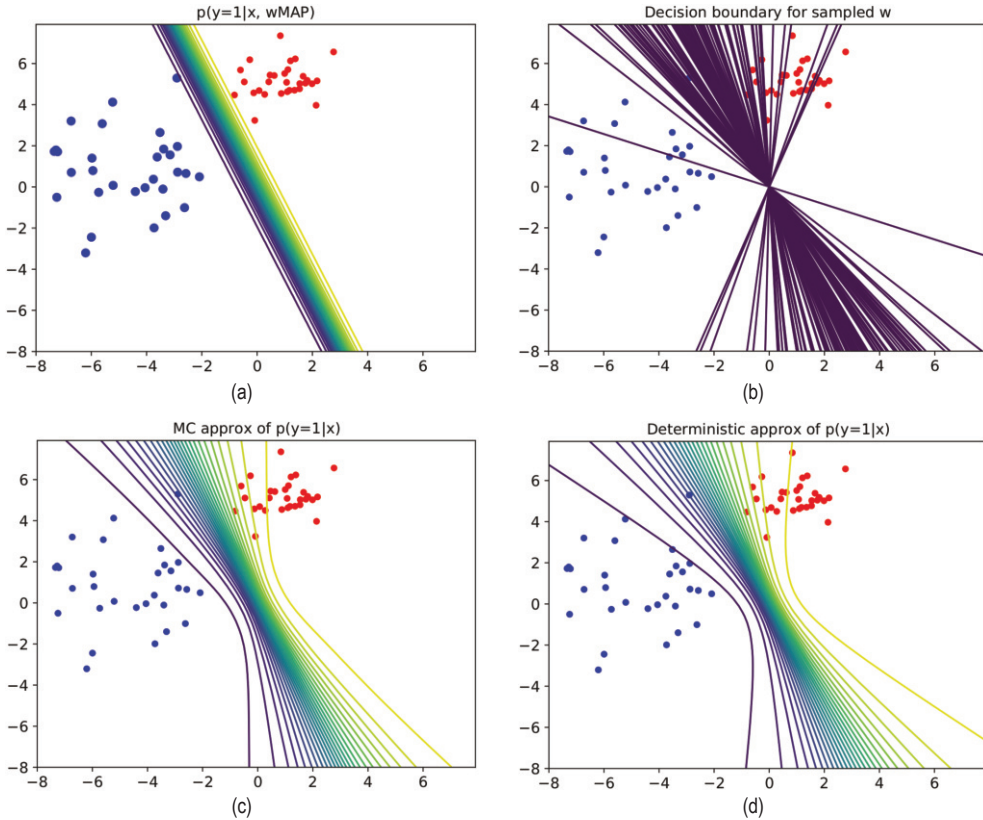


Рис. 10.14 ❖ Апостериорное прогнозное распределение для модели логистической регрессии в двумерном случае. (a) Линии уровня $p(y = 1|\mathbf{x}; \hat{\mathbf{w}}_{map})$. (b) Выборка из апостериорного прогнозного распределения. (c) Усреднение примеров из этой выборки. (d) Модерированный выход (пробит-аппроксимация). На основе рисунка Марка Джиролами. Построено программой по адресу figures.probl.ai/book1/10.14

Однако если мы хотим вычислить неопределенность предсказаний, то следует использовать невырожденное апостериорное распределение. Как мы увидим, обычно в качестве такового берут гауссово распределение. Но все равно нужно приближенно вычислить интеграл (10.108). Ниже мы обсудим некоторые подходы к решению этой задачи.

10.5.2.1. Аппроксимация Монте-Карло

Проще всего аппроксимировать интеграл методом Монте-Карло. Это означает, что мы производим выборку объема S из апостериорного распределения, $\mathbf{w}_s \sim p(\mathbf{w}|\mathcal{D})$, а затем вычисляем

$$p(y = 1|\mathbf{x}, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S \sigma(\mathbf{w}_s^\top \mathbf{x}). \quad (10.110)$$

10.5.2.2. Пробит-аппроксимация

Аппроксимация Монте-Карло проста, но работает медленно, потому что *на этапе тестирования* мы должны выбирать S примеров для каждого входа \mathbf{x} . По счастью, если $p(\mathbf{w}|\mathcal{D}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, то существует простая, но точная детерминированная аппроксимация, впервые предложенная в работе [SL90]. Мы объясним, в чем ее суть, следуя изложению в книге [Bis06, стр. 219]. Ключевое наблюдение заключается в том, что сигмоидная функция $\sigma(a)$ по форме похожа на функцию гауссова распределения (см. раздел 2.6.1) $\Phi(a)$. В частности, $\sigma(a) \approx \Phi(\lambda a)$, где выбор $\lambda^2 = \pi/8$ гарантирует, что обе функции будут иметь одинаковый наклон в начале координат. Это полезно, потому что мы можем точно проинтегрировать функцию гауссова распределения по плотности гауссова распределения:

$$\int \Phi(\lambda a) \mathcal{N}(a|m, v) da = \Phi\left(\frac{m}{(\lambda^{-2} + v)^{\frac{1}{2}}}\right) = \Phi\left(\frac{\lambda m}{(1 + \lambda^2 v)^{\frac{1}{2}}}\right) \approx \sigma(\kappa(v)m), \quad (10.111)$$

где по определению

$$\kappa(v) \triangleq (1 + \pi v/8)^{-\frac{1}{2}}. \quad (10.112)$$

Таким образом, если положить $a = \mathbf{x}^\top \mathbf{w}$, то

$$p(y = 1|\mathbf{x}, \mathcal{D}) \approx \sigma(\kappa(v)m); \quad (10.113)$$

$$m = \mathbb{E}[a] = \mathbf{x}^\top \boldsymbol{\mu}; \quad (10.114)$$

$$v = \mathbb{V}[a] = \mathbb{V}[\mathbf{x}^\top \mathbf{w}] = \mathbf{x}^\top \boldsymbol{\Sigma} \mathbf{x}, \quad (10.115)$$

где в последней строке мы воспользовались формулой (2.165). Поскольку Φ обратна к пробит-функции, то эта формула называется **пробит-аппроксимацией**.

Применение формулы (10.113) дает предсказания, менее экстремальные (в смысле уровня доверия), чем подстановочная оценка. Чтобы убедиться в этом, заметим, что $0 < \kappa(v) < 1$ и, значит, $\kappa(v)m < m$, так что $\sigma(\kappa(v)m)$ ближе к 0.5, чем $\sigma(m)$. Однако на саму решающую границу это не влияет. Действи-

тельно, решающая граница – это множество точек \mathbf{x} , для которых $p(y = 1|\mathbf{x}, \mathcal{D}) = 0.5$. Отсюда следует, что $\kappa(v)m = 0$, т. е. $m = \bar{\mathbf{w}}^\top \mathbf{x} = 0$; но это то же самое, что решающая граница из подстановочной оценки. Таким образом, «байесовость» не изменяет частоту неправильной классификации (в данном случае), зато изменяет оценки доверия к модели, что бывает важно, как мы покажем в разделе 10.5.1.

В случае нескольких классов можно воспользоваться **обобщенной про-бит-аппроксимацией** [Gib97]:

$$p(y = c|\mathbf{x}, \mathcal{D}) \approx \frac{\exp(\kappa(v_c)m_c)}{\sum_{c'} \exp(\kappa(v_{c'})m_{c'})}; \quad (10.116)$$

$$m_c = \bar{\mathbf{m}}_c^\top \mathbf{x}; \quad (10.117)$$

$$v_c = \mathbf{x}^\top \mathbf{V}_{c,c} \mathbf{x}, \quad (10.118)$$

где κ определено формулой (10.112). В отличие от бинарного случая учет апостериорной ковариации дает не такие предсказания, как при подстановочном подходе (см. упражнение 3.10.3 в [RW06]).

Другие аппроксимации гауссовых интегралов в сочетании с сигмоидной функцией и функцией softmax, см. в работе [Dau17].

10.6. УПРАЖНЕНИЯ

Упражнение 10.1 [градиент и гессиан логарифмического правдоподобия для мультиномиальной регрессии].

- а. Обозначим $\mu_{ik} = \mathcal{S}(\boldsymbol{\eta}_i)_k$, где $\boldsymbol{\eta}_i = \mathbf{w}^\top \mathbf{x}_i$. Покажите, что якобиан функции softmax равен

$$\frac{\partial \mu_{ik}}{\partial \eta_{ij}} = \mu_{ik}(\delta_{kj} - \mu_{ij}), \quad (10.119)$$

где $\delta_{kj} = I(k = j)$.

- б. Покажите далее, что градиент NLL равен

$$\nabla_{\mathbf{w}_c} \ell = \sum_i (y_{ic} - \mu_{ic}) \mathbf{x}_i. \quad (10.120)$$

Указание: воспользуйтесь правилом дифференцирования сложной функции и тем фактом, что $\sum_c y_{ic} = 1$.

- с. Покажите, что блочная подматрица гессиана для классов c и c' имеет вид:

$$\mathbf{H}_{c,c'} = -\sum_i \mu_{ic}(\delta_{c,c'} - \mu_{i,c'}) \mathbf{x}_i \mathbf{x}_i^\top. \quad (10.121)$$

Зная это, покажите, что гессиан NLL положительно определен.

Упражнение 10.2 [регуляризация отдельных членов двумерной логистической регрессии*].

(Источник: Яаккола.)

- а. Рассмотрим данные на рис. 10.15а, где мы обучаем модель $p(y = 1 | \mathbf{x}, \mathbf{w}) = \sigma(w_0 + w_1x_1 + w_2x_2)$. Предположим, что при обучении модели используется оценка максимального правдоподобия, т. е. мы минимизируем функцию

$$J(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}), \quad (10.122)$$

где $\ell(\mathbf{w}, \mathcal{D}_{\text{train}})$ – логарифмическое правдоподобие на обучающем наборе. Прикиньте график возможной решающей границы, соответствующей $\hat{\mathbf{w}}$. (Сначала скопируйте рисунок (грубого эскиза достаточно), а затем наложите свой ответ на копию, так как понадобится несколько вариантов этого рисунка). Является ли ваш ответ (решающая граница) единственным? Сколько ошибок классификации делает ваш метод на обучающем наборе?

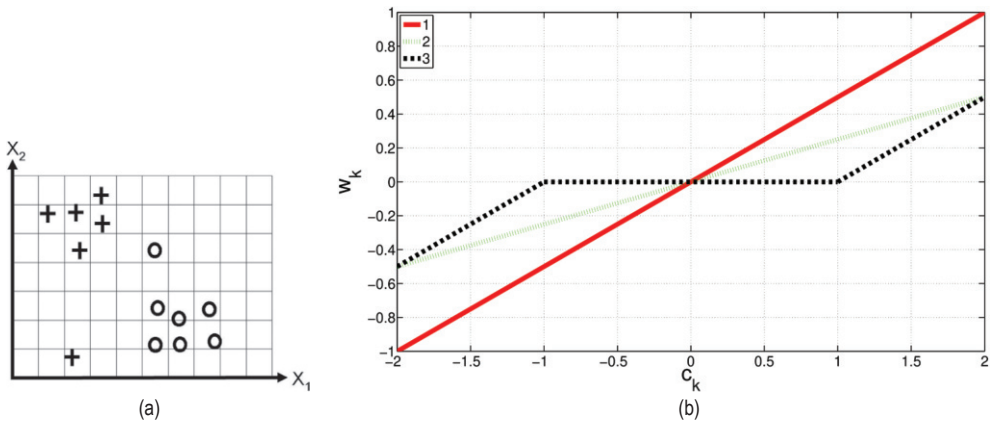


Рис. 10.15 ❖ (а) Данные к вопросу о логистической регрессии.

(б) График зависимости \hat{w}_k от величины корреляции c_k для трех разных оценок

- б. Теперь предположим, что регуляризируется только параметр w_0 , т. е. мы минимизируем функцию:

$$J_0(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) + \lambda w_0^2. \quad (10.123)$$

Предположим, что λ – очень большое число, поэтому мы регуляризируем w_0 , возможно, уменьшая до 0, а остальные параметры оставляем нерегуляризованными. Сколько ошибок классификации делает ваш метод на обучающем наборе? *Указание:* рассмотрите поведение простой линейной регрессии, $w_0 + w_1x_1 + w_2x_2$, когда $x_1 = x_2 = 0$.

- с. Теперь предположим, что мы уделяем внимание только регуляризации параметра w_1 , т. е. минимизируем функцию:

$$J_1(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) + \lambda w_1^2. \quad (10.124)$$

Прикиньте форму возможной решающей границы. Сколько ошибок классификации делает ваш метод на обучающем наборе?

- d. Теперь будем регуляризовать только параметр w_2 . Прикиньте форму возможной решающей границы. Сколько ошибок классификации делает ваш метод на обучающем наборе?

Упражнение 10.3 [сравнение логистической регрессии и LDA/QDA*].

(Источник: Яаккола.)

Допустим, что мы обучаем следующие бинарные классификаторы с помощью максимального правдоподобия.

- GaussI: порождающий классификатор с гауссовыми условными плотностями классов и обеими ковариационными матрицами, равными \mathbf{I} (единичной матрице), т. е. $p(\mathbf{x}|y = c) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \mathbf{I})$. Предполагается, что распределение $p(y)$ равномерное.
- GaussX: то же, что GaussI, но на ковариационные матрицы не налагается ограничений, т. е. $p(\mathbf{x}|y = c) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$.
- LinLog: модель логистической регрессии с линейными признаками.
- QuadLog: модель логистической регрессии с линейными и квадратичными признаками (т. е. разложение по полиномиальному базису степени 2).

После обучения мы вычисляем качество каждой модели M на обучающем наборе следующим образом:

$$L(M) = \frac{1}{n} \sum_{i=1}^n \log p(y_i | \mathbf{x}_i, \hat{\boldsymbol{\theta}}, M). \quad (10.125)$$

(Отметим, что это условное логарифмическое правдоподобие $p(y|\mathbf{x}, \hat{\boldsymbol{\theta}})$, а не совместное логарифмическое правдоподобие $p(y, \mathbf{x}|\hat{\boldsymbol{\theta}})$. Теперь мы хотим сравнить качество каждой модели. Будем писать $L(M) \leq L(M')$, если модель M должна иметь меньшее (или равное) логарифмическое правдоподобие (на обучающем наборе), чем M' , для любого обучающего набора (иными словами, M хуже M' , по крайней мере, с точки зрения логарифмического правдоподобия). Для каждой из следующих моделей укажите, какие утверждения верны: $L(M) \leq L(M')$, $L(M) \geq L(M')$ или ни то ни другое (т. е. M иногда лучше M' , а иногда хуже). В каждом случае кратко (одной-двумя фразами) объяснить свой ответ.

- GaussI, LinLog.
- GaussX, QuadLog.
- LinLog, QuadLog.
- GaussI, QuadLog.
- Теперь будем измерять качество в терминах средней частоты неправильной классификации на обучающем наборе:

$$R(M) = \frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}(\mathbf{x}_i)). \quad (10.126)$$

Верно ли в общем случае, что из $L(M) > L(M')$ следует, что $R(M) < R(M')$? Объясните свой ответ.

Глава 11

Линейная регрессия

11.1. ВВЕДЕНИЕ

В этом разделе мы обсудим **линейную регрессию** – очень широко используемый метод предсказания вещественного выходного значения (называемого также **зависимой переменной** или **целевой величиной**) $y \in \mathbb{R}$ по вектору вещественных входных значений (называемых также **независимыми переменными**, **объясняющими переменными** или **ковариатами**) $\mathbf{x} \in \mathbb{R}^D$. Основное свойство модели заключается в том, что ожидаемое выходное значение предполагается линейной функцией входов, $\mathbb{E}[y|\mathbf{x}] = \mathbf{w}^T \mathbf{x}$, что упрощает интерпретацию и обучение модели. Нелинейные обобщения мы обсудим ниже в этой книге.

11.2. ЛИНЕЙНАЯ РЕГРЕССИЯ ПО МЕТОДУ НАИМЕНЬШИХ КВАДРАТОВ

В этом разделе мы обсудим наиболее распространенную форму модели линейной регрессии.

11.2.1. Терминология

Термин «линейная регрессия» обычно относится к модели следующего вида:

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|w_0 + \mathbf{w}^T \mathbf{x}, \sigma^2), \quad (11.1)$$

где $\boldsymbol{\theta} = (w_0, \mathbf{w}, \sigma^2)$ – все параметры модели. (В статистике параметры w_0 и \mathbf{w} обычно обозначаются β_0 и $\boldsymbol{\beta}$.)

Вектор параметров $\mathbf{w}_{1:D}$ называется **весами**, или **коэффициентами регрессии**. Каждый коэффициент w_d описывает ожидаемое изменение выхода при изменении соответствующего входного признака x_d на одну единицу измерения. Например, пусть x_1 – возраст человека, x_2 – уровень образования

(представленный вещественным числом), а y – его доход. Тогда w_1 соответствует ожидаемому увеличению дохода, когда человек становится старше на один год (и, стало быть, опытнее), а w_2 – ожидаемому увеличению дохода при повышении уровня образования на одну ступень. Член w_0 называется **смещением** и определяет выходное значение, когда все входы равны 0. Он улавливает безусловное среднее отклика, $w_0 = \mathbb{E}[y]$, и играет роль точки отсчета. Обычно предполагается, что вектор \mathbf{x} записан в виде $[1, x_1, \dots, x_D]$, так что смещение w_0 включено в вектор весов \mathbf{w} .

Если входные данные одномерные ($D = 1$), то модель имеет вид $f(x; \mathbf{w}) = ax + b$, где $b = w_0$ – свободный член, а $a = w_1$ – угловой коэффициент. Это называется **простой линейной регрессией**. Если входные данные многомерные, $\mathbf{x} \in \mathbb{R}^D$, где $D > 1$, то метод называется **многофакторной линейной регрессией**. Если выход тоже многомерный, $\mathbf{y} \in \mathbb{R}^J$, где $J > 1$, то он называется **многомерной линейной регрессией**:

$$p(\mathbf{y}|\mathbf{x}, \mathbf{W}) = \prod_{j=1}^J \mathcal{N}(y_j | \mathbf{w}_j^T \mathbf{x}, \sigma_j^2). \quad (11.2)$$

Простой численный пример см. в упражнении 11.1.

В общем случае прямая линия недостаточно хорошо аппроксимирует большинство наборов данных. Однако мы всегда можем применить к входным признакам нелинейное преобразование, заменив \mathbf{x} на $\boldsymbol{\phi}(\mathbf{x})$, и получить

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y | \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}), \sigma^2). \quad (11.3)$$

При условии, что параметры **экстрактора признаков $\boldsymbol{\phi}$** фиксированы, модель остается *линейной относительно параметров*, но становится нелинейной относительно входов. (О способах обучения экстрактора признаков и результирующего линейного отображения мы поговорим в части III.)

В качестве простого примера нелинейного преобразования рассмотрим случай **полиномиальной регрессии**, описанной в разделе 1.2.2.2. Если входные данные одномерные и мы используем разложение по полиномам степени d , то $\boldsymbol{\phi}(x) = [1, x, x^2, \dots, x^d]$, см. пример на рис. 11.1 (см. также обсуждение сплайнов в разделе 11.5).

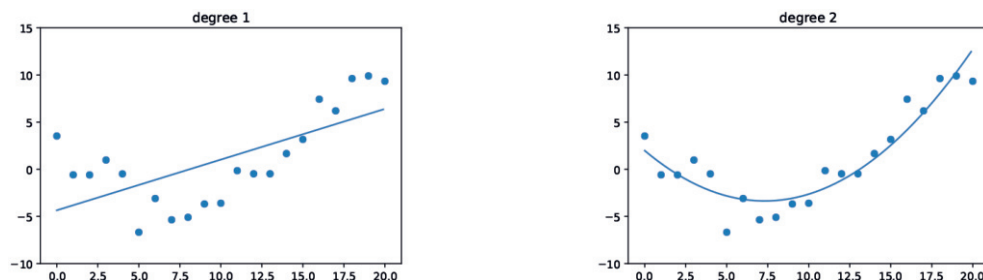


Рис. 11.1 ❖ Аппроксимация 21 точки полиномами степени 1 и 2.
Построено программой по адресу figures.problml.ai/book1/11.1

11.2.2. Оценивание по методу наименьших квадратов

Чтобы аппроксимировать данные линейной моделью, мы будем минимизировать отрицательное логарифмическое правдоподобие на обучающем наборе. Целевая функция имеет вид:

$$\text{NLL}(\mathbf{w}, \sigma^2) = -\sum_{n=1}^N \log \left[\left(\frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp \left(-\frac{1}{2\sigma^2} (y_n - \mathbf{w}^\top \mathbf{x}_n)^2 \right) \right] \quad (11.4)$$

$$= \frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \hat{y}_n)^2 + \frac{N}{2} \log(2\pi\sigma^2), \quad (11.5)$$

где предсказанный отклик по определению равен $\hat{y}_n \triangleq \mathbf{w}^\top \mathbf{x}_n$. Оценка максимального правдоподобия (MLE) – это точка, в которой $\nabla_{\mathbf{w}, \sigma} \text{NLL}(\mathbf{w}, \sigma^2) = 0$. Мы можем сначала оптимизировать относительно \mathbf{w} , а затем найти оптимальное σ .

В этом разделе мы остановимся на оценивании весов \mathbf{w} . В этом случае NLL равно (с точностью до несущественных постоянных) **сумме квадратов невязок** (residual sum of squares – RSS), равной

$$\text{RSS}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N (y_n - \mathbf{w}^\top \mathbf{x}_n)^2 = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 = \frac{1}{2} (\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y}). \quad (11.6)$$

Как ее оптимизировать, мы обсудим ниже.

11.2.2.1. Обыкновенный метод наименьших квадратов

Пользуясь формулой (7.264), можно показать, что градиент равен

$$\nabla_{\mathbf{w}} \text{RSS}(\mathbf{w}) = \mathbf{X}^\top \mathbf{X} \mathbf{w} - \mathbf{X}^\top \mathbf{y}. \quad (11.7)$$

Приравнявая градиент нулю и решая получившееся уравнение, имеем

$$\mathbf{X}^\top \mathbf{X} \mathbf{w} = \mathbf{X}^\top \mathbf{y}. \quad (11.8)$$

Эти уравнения называются **нормальными**, потому что в точке оптимума вектор $\mathbf{y} - \mathbf{X}\mathbf{w}$ нормален (ортогонален) образу \mathbf{X} , как будет объяснено в разделе 11.2.2.2. Соответствующее решение $\hat{\mathbf{w}}$ называется решением по **методу обыкновенных наименьших квадратов** (ordinary least squares – OLS) и имеет вид:

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}. \quad (11.9)$$

Матрица $\mathbf{X}^\dagger = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ является (левой) псевдообратной для (неквадратной) матрицы \mathbf{X} (детали см. в разделе 7.5.3).

Убедиться в единственности решения можно, показав, что гессиан положительно определен. В данном случае гессиан имеет вид:

$$\mathbf{H}(\mathbf{w}) = \frac{\partial^2}{\partial \mathbf{x}^2} \text{RSS}(\mathbf{w}) = \mathbf{X}^T \mathbf{X}. \quad (11.10)$$

Если \mathbf{X} имеет полный ранг (т. е. столбцы \mathbf{X} линейно независимы), то \mathbf{H} положительно определен, потому что для любого $\mathbf{v} > \mathbf{0}$ имеем

$$\mathbf{v}^T (\mathbf{X}^T \mathbf{X}) \mathbf{v} = (\mathbf{X} \mathbf{v})^T (\mathbf{X} \mathbf{v}) = \|\mathbf{X} \mathbf{v}\|^2 > 0. \quad (11.11)$$

Следовательно, в случае матрицы полного ранга целевая функция метода наименьших квадратов имеет единственный глобальный минимум (см. иллюстрацию на рис. 11.2).

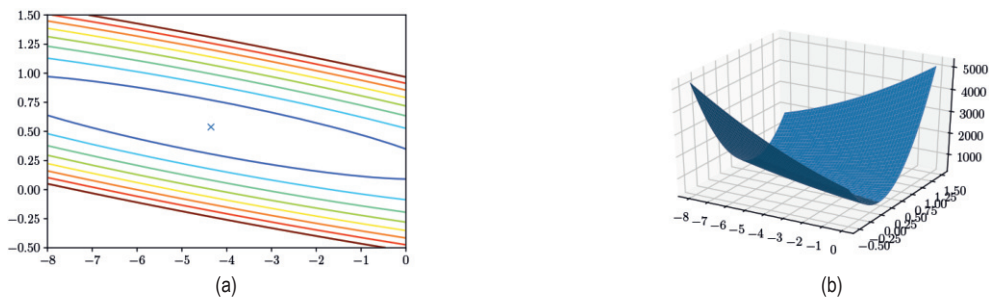


Рис. 11.2 ❖ (a) Линии уровня на поверхности RSS для примера на рис. 11.1а. Синий крестик представляет MLE. (b) Соответствующий график поверхности. Построено программой по адресу figures.problml.ai/book1/11.2

11.2.2.2. Геометрическая интерпретация метода наименьших квадратов

У нормальных уравнений имеется элегантная геометрическая интерпретация, вытекающая, как мы сейчас объясним, из раздела 7.7. Будем предполагать, что $N > D$, т. е. наблюдений больше, чем неизвестных (такая система уравнений называется **переопределенной**). Мы ищем вектор $\hat{\mathbf{y}} \in \mathbb{R}^N$, который принадлежал бы линейному подпространству, натянутому на \mathbf{X} , и был максимально близок к \mathbf{y} , т. е.

$$\underset{\hat{\mathbf{y}} \in \text{span}(\{\mathbf{x}_{:,1}, \dots, \mathbf{x}_{:,d}\})}{\text{argmin}} \quad \|\mathbf{y} - \hat{\mathbf{y}}\|_2, \quad (11.12)$$

где $\mathbf{x}_{:,d}$ — d -й столбец \mathbf{X} . Поскольку $\hat{\mathbf{y}} \in \text{span}(\mathbf{X})$, существует такой вектор весов \mathbf{w} , что

$$\hat{\mathbf{y}} = w_1 \mathbf{x}_{:,1} + \dots + w_n \mathbf{x}_{:,D} = \mathbf{X} \mathbf{w}. \quad (11.13)$$

Чтобы норма невязки, $\mathbf{y} - \hat{\mathbf{y}}$, была минимальной, необходимо, чтобы вектор невязок был ортогонален каждому столбцу \mathbf{X} . Отсюда

$$\mathbf{x}_{:,d}^T (\mathbf{y} - \hat{\mathbf{y}}) = 0 \Rightarrow \mathbf{X}^T (\mathbf{y} - \mathbf{X} \mathbf{w}) = 0 \Rightarrow \mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}. \quad (11.14)$$

Следовательно, $\hat{\mathbf{y}}$ имеет вид:

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \quad (11.15)$$

Это не что иное, как **ортогональная проекция** \mathbf{y} на пространство столбцов \mathbf{X} . Например, рассмотрим случай, когда имеется $N = 3$ обучающих примеров размерности $D = 2$. Обучающие данные определяют двумерное линейное подпространство, натянутое на два столбца \mathbf{X} , каждому из которых соответствует точка в трехмерном пространстве. Мы проецируем \mathbf{y} , который также представлен точкой в трехмерном пространстве, на это подпространство, как показано на рис. 11.3.

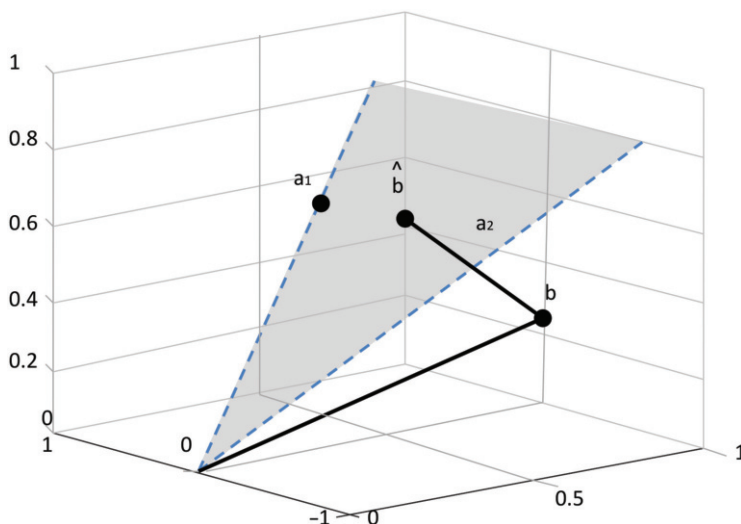


Рис. 11.3 ❖ Графическая интерпретация метода наименьших квадратов для $m = 3$ уравнений и $n = 2$ неизвестных в системе $\mathbf{Ax} = \mathbf{b}$. \mathbf{a}_1 и \mathbf{a}_2 – столбцы \mathbf{A} , определяющие двумерное линейное подпространство, погруженное в \mathbb{R}^3 . Вектор \mathbf{b} принадлежит \mathbb{R}^3 , его ортогональная проекция на линейное подпространство обозначена $\hat{\mathbf{b}}$. Отрезок прямой, соединяющий \mathbf{b} с $\hat{\mathbf{b}}$, – вектор невязок, норму которого мы хотим минимизировать

Матрицу проекции

$$\text{Proj}(\mathbf{X}) \triangleq \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \quad (11.16)$$

в англоязычной литературе иногда называют **hat matrix** (матрица крышки), потому что $\hat{\mathbf{y}} = \text{Proj}(\mathbf{X})\mathbf{y}$. В частном случае, когда $\mathbf{X} = \mathbf{x}$ является вектором-столбцом, ортогональная проекция \mathbf{y} на прямую \mathbf{x} принимает вид:

$$\text{Proj}(\mathbf{x})\mathbf{y} = \mathbf{x} \frac{\mathbf{x}^T\mathbf{y}}{\mathbf{x}^T\mathbf{x}}. \quad (11.17)$$

11.2.2.3. Алгоритмические проблемы

Напомним, что решение по методу OLS имеет вид:

$$\hat{\mathbf{w}} = \mathbf{X}^T \mathbf{y} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}. \quad (11.18)$$

Однако, даже если теоретически возможно вычислить псевдообратную матрицу $\mathbf{X}^T \mathbf{X}$, делать этого не следует, потому что $\mathbf{X}^T \mathbf{X}$ может оказаться плохо обусловленной или сингулярной.

Лучший (и более общий) подход – вычислить псевдообратную матрицу с помощью сингулярного разложения (SVD). Действительно, заглянув в исходный код функции `sklearn.linear_model.fit`, вы увидите, что в нем используется функция `scipy.linalg.lstsq`, которая в свою очередь вызывает `DGELSD` – основанный на SVD решатель, реализованный в библиотеке LAPACK, написанной на языке Fortran¹.

Однако если матрица \mathbf{X} высокая и узкая (т. е. $N \gg D$), то может оказаться быстрее использовать QR-разложение (раздел 7.6.2). Для этого положим $\mathbf{X} = \mathbf{QR}$, где $\mathbf{Q}^T \mathbf{Q} = \mathbf{I}$. В разделе 7.7 мы показали, что метод OLS эквивалентен нахождению такого решения системы линейных уравнений $\mathbf{X}\mathbf{w} = \mathbf{y}$, которое минимизирует ошибку $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$. (Если $N = D$ и матрица \mathbf{X} полного ранга, то система имеет единственное решение и ошибка будет равна 0.) Применив QR-разложение, мы сможем переписать эту систему уравнений в виде:

$$(\mathbf{QR})\mathbf{w} = \mathbf{y}; \quad (11.19)$$

$$\mathbf{Q}^T \mathbf{QR}\mathbf{w} = \mathbf{Q}^T \mathbf{y}; \quad (11.20)$$

$$\mathbf{w} = \mathbf{R}^{-1}(\mathbf{Q}^T \mathbf{y}). \quad (11.21)$$

Поскольку \mathbf{R} верхнетреугольная, последнюю систему уравнений можно решить методом обратного хода, избежав тем самым обращения матрицы (см. код по адресу code.problm.ai/book1/linsys_solve_demo).

Альтернативой использованию прямых методов, основанных на матричных разложениях (таких как SVD и QR), является применение итеративных решателей, например метода **сопряженных градиентов** (в котором предполагается, что \mathbf{X} симметричная и положительно определенная) или **GMRES** (обобщенного метода минимизации невязки), который работает для матриц \mathbf{X} общего вида. (В библиотеке SciPy он реализован в модуле `sparse.linalg.gmres`.) Для этих методов требуется только умение умножать матрицу на вектор (т. е. реализация линейного **оператора**), поэтому они хорошо подходят для задач, где \mathbf{X} разреженная или обладает выраженной структурой. Детали см., например, в работе [TB97].

¹ Заметим, что значительная часть научного кода на Python – на самом деле лишь тонкая обертка кода, написанного на Fortran или C++. Так сделано из соображений производительности. Но из-за этого изменять базовые алгоритмы довольно трудно. С другой стороны, библиотеки для научных расчетов на языке Julia написаны на самом этом языке, что делает их понятнее не в ущерб быстродействию.

И последний важный вопрос – обычно крайне важно перед обучением модели **стандартизовать** входные признаки, т. е. сделать среднее нулевым, а дисперсию единичной. Это можно сделать, пользуясь формулой (10.51).

11.2.2.4. Метод взвешенных наименьших квадратов

Иногда желательно связать с каждым примером вес. Например, в случае **гетероскедастичной регрессии** дисперсия зависит от входных данных, так что модель имеет вид:

$$p(y|\mathbf{x}; \boldsymbol{\theta}) = N(y|\mathbf{w}^\top \mathbf{x}, \sigma^2(\mathbf{x})) = \frac{1}{\sqrt{2\pi\sigma^2(\mathbf{x})}} \exp\left(-\frac{1}{2\sigma^2(\mathbf{x})}(y - \mathbf{w}^\top \mathbf{x})^2\right). \quad (11.22)$$

Таким образом,

$$p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w}, \boldsymbol{\Lambda}^{-1}), \quad (11.23)$$

где $\boldsymbol{\Lambda} = \text{diag}(1/\sigma^2(\mathbf{x}_n))$. Это называется **взвешенной линейной регрессией**. Можно показать, что MLE в этом случае равна

$$\mathbf{w} = (\mathbf{X}^\top \boldsymbol{\Lambda} \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{\Lambda} \mathbf{y}. \quad (11.24)$$

Она называется оценкой по методу **взвешенных наименьших квадратов**.

11.2.3. Другие подходы к вычислению MLE

В этом разделе мы обсудим другие подходы к вычислению MLE.

11.2.3.1. Нахождение смещения и углового коэффициента по отдельности

Обычно мы используем модель вида $p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|w_0 + \mathbf{w}^\top \mathbf{x}, \sigma^2)$, где w_0 – смещение. Мы можем вычислить (w_0, \mathbf{w}) одновременно, добавив в \mathbf{X} столбец единиц и вычислив MLE, как описано выше. Но можно вместо этого искать \mathbf{w} и w_0 по отдельности. (Это будет полезно позднее.) Точнее, можно показать, что

$$\hat{\mathbf{w}} = (\mathbf{X}_c^\top \mathbf{X}_c)^{-1} \mathbf{X}_c^\top \mathbf{y}_c = \left[\sum_{i=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^\top \right]^{-1} \left[\sum_{i=1}^N (y_n - \bar{y})(\mathbf{x}_n - \bar{\mathbf{x}}) \right]; \quad (11.25)$$

$$\hat{w}_0 = \frac{1}{N} \sum_n y_n - \frac{1}{N} \sum_n \mathbf{x}_n^\top \hat{\mathbf{w}} = \bar{y} - \bar{\mathbf{x}}^\top \hat{\mathbf{w}}, \quad (11.26)$$

где \mathbf{X}_c – центрированная входная матрица, строками которой являются векторы $\mathbf{x}_n^c = \mathbf{x}_n - \bar{\mathbf{x}}$, а $\mathbf{y}_c = \mathbf{y} - \bar{y}$ – центрированный выходной вектор. Таким образом, мы можем сначала вычислить $\hat{\mathbf{w}}$ на центрированных данных, а затем оценить w_0 , используя $\bar{y} - \bar{\mathbf{x}}^\top \hat{\mathbf{w}}$.

11.2.3.2. Простая линейная регрессия (одномерные входные данные)

В случае одномерных (скалярных) входных данных результаты из раздела 11.2.3.1 сводятся к простой форме, знакомой по начальному курсу статистики:

$$\hat{w}_1 = \frac{\sum_n (x_n - \bar{x})(y_n - \bar{y})}{\sum_n (x_n - \bar{x})^2} = \frac{C_{xy}}{C_{xx}}; \quad (11.27)$$

$$\hat{w}_0 = \bar{y} - \hat{w}_1 \bar{x} = \mathbb{E}[y] - w_1 \mathbb{E}[x], \quad (11.28)$$

где $C_{xy} = \text{Cov}[X, Y]$ и $C_{xx} = \text{Cov}[X, X] = \mathbb{V}[X]$. Этот результат понадобится нам ниже.

11.2.3.3. Частная регрессия

Из формулы (11.27) можно вычислить коэффициент регрессии Y от X :

$$R_{YX} \triangleq \frac{\partial}{\partial x} \mathbb{E}[Y|X = x] = w_1 = \frac{C_{xy}}{C_{xx}}. \quad (11.29)$$

Это угловой коэффициент предсказанной линейной зависимости Y от X .

Теперь рассмотрим случай, когда имеется два входа, т. е. $Y = w_0 + w_1 X_1 + w_2 X_2 + \varepsilon$, где $\mathbb{E}[\varepsilon] = 0$. Можно показать, что оптимальный коэффициент регрессии w_1 равен $R_{YX_1 \cdot X_2}$, это **частный коэффициент регрессии** Y на X_1 при постоянном X_2 :

$$w_1 = R_{YX_1 \cdot X_2} = \frac{\partial}{\partial x} \mathbb{E}[Y|X_1 = x, X_2]. \quad (11.30)$$

Отметим, что эта величина инвариантна относительно конкретного значения X_2 , по которому производится обусловливание.

Вывести w_2 можно аналогично. На самом деле вывод можно обобщить на несколько входных переменных. В любом случае оптимальные коэффициенты равны частным коэффициентам регрессии. Это означает, что j -й коэффициент \hat{w}_j можно интерпретировать как изменение выходного значения y , ожидаемое при изменении входного значения x_j на одну единицу, когда все остальные входы постоянны.

11.2.3.4. Рекурсивное вычисление MLE

OLS – это пакетный метод вычисления MLE. В некоторых приложениях данные поступают непрерывным потоком, поэтому нужно вычислять оценку в онлайн-овом режиме, или **рекурсивно**, как обсуждалось в разделе 4.4.2. В этом разделе мы покажем, как это делается в случае простой (одномерной) линейной регрессии.

Напомним (см. раздел 11.2.3.2), что пакетная MLE для простой линейной регрессии имеет вид:

$$\hat{w}_1 = \frac{\sum_n (x_n - \bar{x})(y_n - \bar{y})}{\sum_n (x_n - \bar{x})^2} = \frac{C_{xy}}{C_{xx}}, \quad (11.31)$$

$$\hat{w}_0 = \bar{y} - \hat{w}_1 \bar{x}, \quad (11.32)$$

где $C_{xy} = \text{Cov}[X, Y]$ и $C_{xx} = \text{Cov}[X, X] = \mathbb{V}[X]$.

Обсудим, как вычислить эти величины рекурсивно. Для этого определим следующие достаточные статистики:

$$\bar{x}^{(n)} = \frac{1}{n} \sum_{i=1}^n x_i, \quad \bar{y}^{(n)} = \frac{1}{n} \sum_{i=1}^n y_i; \quad (11.33)$$

$$C_{xx}^{(n)} = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2, \quad C_{xy}^{(n)} = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}), \quad C_{yy}^{(n)} = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2. \quad (11.34)$$

Мы можем обновлять средние в онлайн-режиме по формуле:

$$\bar{x}^{(n+1)} = \bar{x}^{(n)} + \frac{1}{n+1} (x_{n+1} - \bar{x}^{(n)}), \quad \bar{y}^{(n+1)} = \bar{y}^{(n)} + \frac{1}{n+1} (y_{n+1} - \bar{y}^{(n)}). \quad (11.35)$$

Для обновления членов ковариации сначала перепишем $C_{xy}^{(n)}$ в виде

$$C_{xy}^{(n)} = \frac{1}{n} \left[\left(\sum_{i=1}^n x_i y_i \right) + \left(\sum_{i=1}^n \bar{x}^{(n)} \bar{y}^{(n)} \right) - \bar{x}^{(n)} \left(\sum_{i=1}^n y_i \right) - \bar{y}^{(n)} \left(\sum_{i=1}^n x_i \right) \right] \quad (11.36)$$

$$= \frac{1}{n} \left[\left(\sum_{i=1}^n x_i y_i \right) + n \bar{x}^{(n)} \bar{y}^{(n)} - \bar{x}^{(n)} n \bar{y}^{(n)} - \bar{y}^{(n)} n \bar{x}^{(n)} \right] \quad (11.37)$$

$$= \frac{1}{n} \left[\left(\sum_{i=1}^n x_i y_i \right) - n \bar{x}^{(n)} \bar{y}^{(n)} \right]. \quad (11.38)$$

Отсюда

$$\sum_{i=1}^n x_i y_i = n C_{xy}^{(n)} + n \bar{x}^{(n)} \bar{y}^{(n)}; \quad (11.39)$$

$$C_{xy}^{(n+1)} = \frac{1}{n+1} \left[x_{n+1} y_{n+1} + n C_{xy}^{(n)} + n \bar{x}^{(n)} \bar{y}^{(n)} - (n+1) \bar{x}^{(n+1)} \bar{y}^{(n+1)} \right]. \quad (11.40)$$

Обновление $C_{xx}^{(n+1)}$ выводится аналогично.

На рис. 11.4 показана простая иллюстрация этих формул для модели одномерной регрессии. Для обобщения их на случай D -мерных входных данных проще всего использовать СГС. В результате получается **алгоритм наименьших средних квадратов**, детали см. в разделе 8.4.2.

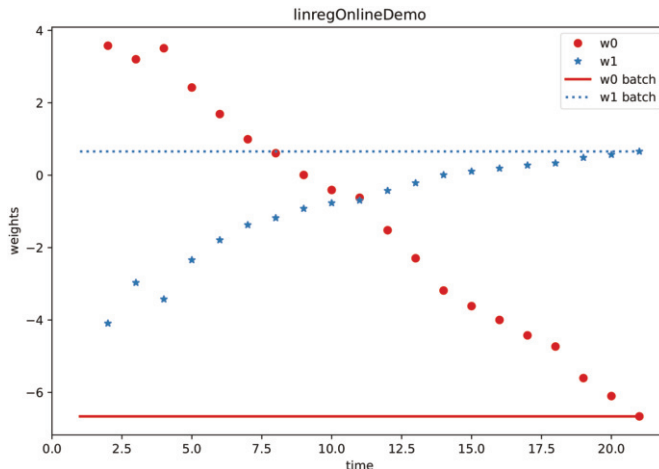


Рис. 11.4. Зависимость коэффициентов регрессии от времени для одномерной модели на рис. 1.7а. Построено программой по адресу figures.problml.ai/book1/11.4

11.2.3.5. Вывод MLE с порождающей точки зрения

Линейная регрессия – это дискриминантная модель вида $p(y|\mathbf{x})$. Однако для регрессии можно использовать также порождающие модели по аналогии с тем, как мы использовали такие модели для классификации в главе 9. Наша цель – вычислить условное математическое ожидание:

$$f(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}] = \int y p(y|\mathbf{x}) dy = \frac{\int y p(\mathbf{x}, y) dy}{\int p(\mathbf{x}, y) dy}. \quad (11.41)$$

Пусть требуется аппроксимировать $p(\mathbf{x}, y)$ многомерным нормальным распределением. Оценки MLE параметров совместного распределения – это эмпирические средние и ковариации (доказательство см. в разделе 4.2.6):

$$\boldsymbol{\mu}_x = \frac{1}{N} \sum_n \mathbf{x}_n; \quad (11.42)$$

$$\mu_y = \frac{1}{N} \sum_n y_n; \quad (11.43)$$

$$\boldsymbol{\Sigma}_{xx} = \frac{1}{N} \sum_n (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^\top = \frac{1}{N} \mathbf{X}_c^\top \mathbf{X}_c; \quad (11.44)$$

$$\boldsymbol{\Sigma}_{xy} = \frac{1}{N} \sum_n (\mathbf{x}_n - \bar{\mathbf{x}})(y_n - \bar{y}) = \frac{1}{N} \mathbf{X}_c^\top \mathbf{y}_c. \quad (11.45)$$

Следовательно, из формулы (3.28) имеем

$$\mathbb{E}[y|\mathbf{x}] = \mu_y + \Sigma_{xy}^T \Sigma_{xx}^{-1} (\mathbf{x} - \mu_x). \quad (11.46)$$

Этот результат можно переписать в виде $\mathbb{E}[y|\mathbf{x}] = w_0 + \mathbf{w}^T \mathbf{x}$, положив

$$w_0 = \mu_y - \mathbf{w}^T \mu_x = \bar{y} - \mathbf{w}^T \bar{\mathbf{x}}; \quad (11.47)$$

$$\mathbf{w} = \Sigma_{xx}^{-1} \Sigma_{xy} = (\mathbf{X}_c^T \mathbf{X}_c)^{-1} \mathbf{X}_c^T \mathbf{y}_c. \quad (11.48)$$

Это совпадает с оценками MLE для дискриминантной модели, найденными в разделе 11.2.3.1. Таким образом, мы видим, что обучение совместной модели с последующим обусловливанием дает такой же результат, как обучение условной модели. Однако это справедливо только для гауссовых моделей (дальнейшее обсуждение этого момента см. в разделе 9.4).

11.2.3.6. Вывод MLE для Σ^2

Оценив $\hat{\mathbf{w}}_{\text{mle}}$ одним из описанных выше методов, мы можем оценить дисперсию шума. Легко показать, что MLE имеет вид:

$$\hat{\sigma}_{\text{mle}}^2 = \underset{\sigma^2}{\operatorname{argmin}} \operatorname{NLL}(\hat{\mathbf{w}}, \sigma^2) = \frac{1}{N} \sum_{n=1}^N (y_n - \mathbf{x}_n^T \hat{\mathbf{w}})^2. \quad (11.49)$$

Это в точности совпадает со среднеквадратической ошибкой, что согласуется с интуицией.

11.2.4. Измерение степени согласия оценки

В этом разделе мы обсудим простые способы оценки соответствия модели регрессии данным (так называемой **степени согласия**).

11.2.4.1. Графики невязок

Для одномерных входных данных разумность модели можно проверить, построив график зависимости невязок, $r_n = y_n - \hat{y}_n$, от входа x_n . Он так и называется – **график невязок**. Модель предполагает, что невязки имеют распределение $\mathcal{N}(0, \sigma^2)$, так что график невязок должен представлять собой облако точек, расположенных примерно поровну под и над горизонтальной осью без каких-либо очевидных тенденций.

В качестве примера на рис. 11.5а приведен график невязок для линейной модели на рис. 1.7а. Как видим, в структуре невязок наблюдается некоторая искривленность, свидетельствующая о плохой аппроксимации. На рис. 11.5б показан график невязок для квадратичной модели на рис. 1.7б. Видно, что аппроксимация гораздо лучше.

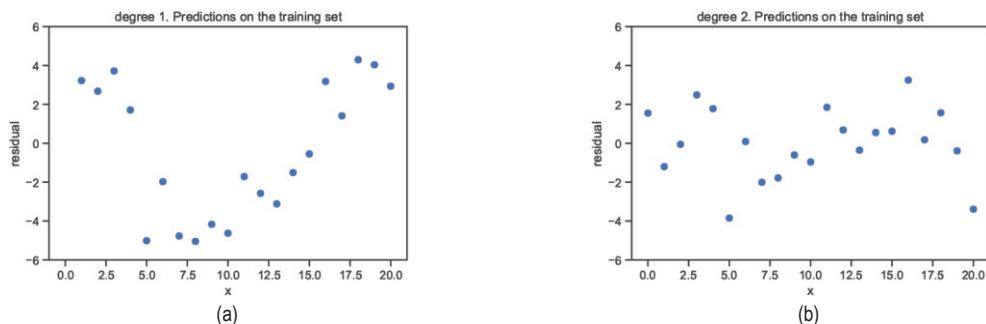


Рис. 11.5. График невязок полиномиальной регрессии степени 1 и 2 для функций на рис. 1.7а–b. Построено программой по адресу figures.problm.ai/book1/11.5

Чтобы обобщить этот подход на многомерные входные данные, мы можем построить график зависимости предсказаний \hat{y}_n от истинных выходов y_n , а не от x_n . Если модель хорошая, то точки будут лежать на диагональной прямой. Примеры см. на рис. 11.6.

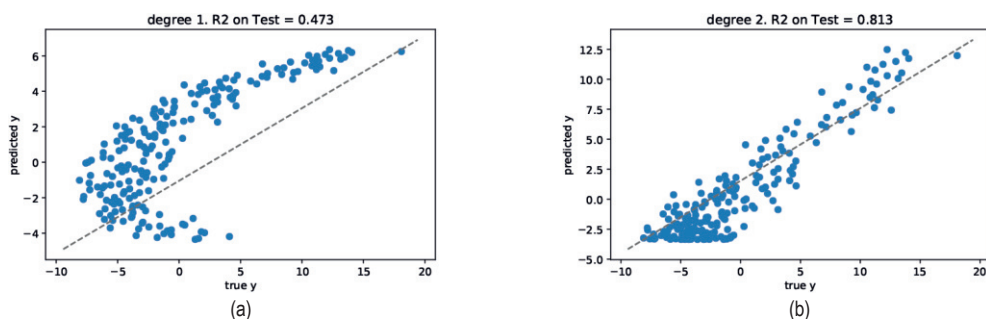


Рис. 11.6 ❖ Предсказанные и истинные точки для полиномиальной регрессии степени 1 и 2 для функций на рис. 1.7а–b. Построено программой по адресу figures.problm.ai/book1/11.6

11.2.4.2. Точность предсказания и R^2

Количественно оценить точность аппроксимации можно, вычислив сумму квадратов невязок (RSS) на наборе данных: $RSS(\mathbf{w}) = \sum_{n=1}^N (y_n - \mathbf{w}^T \mathbf{x}_n)^2$. Чем меньше RSS, тем лучше модель аппроксимирует данные. Другая мера – **коэффициент из среднеквадратической ошибки** (root mean squared error – **RMSE**):

$$RMSE(\mathbf{w}) \triangleq \sqrt{\frac{1}{N} RSS(\mathbf{w})}. \quad (11.50)$$

Для интерпретации лучше подходит **коэффициент детерминации**, обозначаемый R^2 :

$$R^2 \triangleq 1 - \frac{\sum_{n=1}^N (\hat{y}_n - y_n)^2}{\sum_{n=1}^N (\bar{y}_n - y_n)^2} = 1 - \frac{\text{RSS}}{\text{TSS}}, \quad (11.51)$$

где $\bar{y} = (1/N) \sum_{n=1}^N y_n$ – эмпирическое среднее выходов, $\text{RSS} = \sum_{n=1}^N (y_n - \hat{y}_n)^2$ – сумма квадратов невязок, а $\text{TSS} = \sum_{n=1}^N (y_n - \bar{y}_n)^2$ – полная сумма квадратов. Таким образом, мы видим, что R^2 измеряет дисперсию предсказаний относительно простого постоянного предсказания $\hat{y}_n = \bar{y}$. Можно показать, что $0 \leq R^2 \leq 1$, причем чем больше значение, тем меньше дисперсия (т. е. аппроксимация лучше). Это показано на рис. 11.6.

11.3. Гребневая регрессия

Оценка максимального правдоподобия может приводить к переобучению, о чем шла речь в разделе 1.2.2.2. Простое решение этой проблемы – использовать оценку MAP для априорного гауссова распределения весов с нулевым средним, $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \lambda^{-1}\mathbf{I})$, как в разделе 4.5.3. Это называется **гребневой регрессией**.

Точнее, оценка MAP вычисляется следующим образом:

$$\hat{\mathbf{w}}_{\text{map}} = \underset{\mathbf{w}}{\text{argmin}} \frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) + \frac{1}{2\tau^2} \mathbf{w}^\top \mathbf{w} \quad (11.52)$$

$$= \underset{\mathbf{w}}{\text{argmin}} \text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2, \quad (11.53)$$

где $\lambda \triangleq \sigma^2/\tau^2$ пропорционально силе априорного распределения, а

$$\|\mathbf{w}\|_2 \triangleq \sqrt{\sum_{d=1}^D |w_d|^2} = \sqrt{\mathbf{w}^\top \mathbf{w}} \quad (11.54)$$

– ℓ_2 -норма вектора \mathbf{w} . Таким образом, мы штрафует за слишком большие веса. В общем случае эта техника называется **ℓ_2 -регуляризацией** или **уменьшением весов** и применяется очень широко (см. иллюстрацию на рис. 4.5).

Отметим, что член смещения w_0 не штрафует, потому что он влияет только на глобальное среднее выхода и не дает никакого вклада в переобучение (см. упражнение 11.2).

11.3.1. Вычисление оценки MAP

В этом разделе мы обсудим алгоритмы вычисления оценки MAP.

Оценка MAP соответствует минимизации следующей целевой функции со штрафом:

$$J(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2, \quad (11.55)$$

где $\lambda = \sigma^2/\tau^2$ – сила регуляризатора. Производная равна

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = 2(\mathbf{X}^T \mathbf{X} \mathbf{w} - \mathbf{X}^T \mathbf{y} + \lambda \mathbf{w}), \quad (11.56)$$

откуда

$$\hat{\mathbf{w}}_{\text{map}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_D)^{-1} \mathbf{X}^T \mathbf{y} = \left(\sum_n \mathbf{x}_n \mathbf{x}_n^T + \lambda \mathbf{I}_D \right)^{-1} \left(\sum_n y_n \mathbf{x}_n \right). \quad (11.57)$$

11.3.1.1. Решение с использованием QR-разложения

Наивное вычисление оценки $\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$ путем обращения матрицы – неудачная идея, поскольку такое вычисление может оказаться медленным и численно неустойчивым. В этом разделе мы опишем, как свести задачу к стандартной задаче наименьших квадратов, к которой можно применить QR-разложение, как обсуждалось в разделе 11.2.2.3.

Мы предполагаем, что априорное распределение имеет вид $p(\mathbf{w}) = \mathcal{N}(0, \Lambda^{-1})$, где Λ – матрица точности. В случае гребневой регрессии $\Lambda = (1/\tau^2) \mathbf{I}$. Это априорное распределение можно смоделировать, добавив «виртуальные данные» в обучающий набор:

$$\tilde{\mathbf{X}} = \begin{pmatrix} \mathbf{X}/\sigma \\ \sqrt{\Lambda} \end{pmatrix}, \quad \tilde{\mathbf{y}} = \begin{pmatrix} \mathbf{y}/\sigma \\ \mathbf{0}_{D \times 1} \end{pmatrix}, \quad (11.58)$$

где $\Lambda = \sqrt{\Lambda} \sqrt{\Lambda}^T$ – разложение Холески матрицы Λ . Мы видим, что $\tilde{\mathbf{X}}$ – матрица размера $(N + D) \times D$, где дополнительные строки представляют псевдоданные из априорного распределения.

Теперь покажем, что применение RSS к этим расширенным данным эквивалентно применению RSS со штрафом к исходным данным:

$$f(\mathbf{w}) = (\tilde{\mathbf{y}} - \tilde{\mathbf{X}} \mathbf{w})^T (\tilde{\mathbf{y}} - \tilde{\mathbf{X}} \mathbf{w}) \quad (11.59)$$

$$= \left(\begin{pmatrix} \mathbf{y}/\sigma \\ \mathbf{0} \end{pmatrix} - \begin{pmatrix} \mathbf{X}/\sigma \\ \sqrt{\Lambda} \end{pmatrix} \mathbf{w} \right)^T \left(\begin{pmatrix} \mathbf{y}/\sigma \\ \mathbf{0} \end{pmatrix} - \begin{pmatrix} \mathbf{X}/\sigma \\ \sqrt{\Lambda} \end{pmatrix} \mathbf{w} \right) \quad (11.60)$$

$$= \begin{pmatrix} \frac{1}{\sigma} (\mathbf{y} - \mathbf{X} \mathbf{w}) \\ -\sqrt{\Lambda} \mathbf{w} \end{pmatrix}^T \begin{pmatrix} \frac{1}{\sigma} (\mathbf{y} - \mathbf{X} \mathbf{w}) \\ -\sqrt{\Lambda} \mathbf{w} \end{pmatrix} \quad (11.61)$$

$$= \frac{1}{\sigma^2} (\mathbf{y} - \mathbf{X} \mathbf{w})^T (\mathbf{y} - \mathbf{X} \mathbf{w}) + (\sqrt{\Lambda} \mathbf{w})^T (\sqrt{\Lambda} \mathbf{w}) \quad (11.62)$$

$$= \frac{1}{\sigma^2} (\mathbf{y} - \mathbf{X} \mathbf{w})^T (\mathbf{y} - \mathbf{X} \mathbf{w}) + \mathbf{w}^T \Lambda \mathbf{w}. \quad (11.63)$$

Следовательно, оценка MAP имеет вид:

$$\hat{\mathbf{w}}_{\text{map}} = (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \tilde{\mathbf{y}}, \quad (11.64)$$

а это уравнение можно решить стандартным методом OLS. Именно, мы можем вычислить QR-разложение $\tilde{\mathbf{X}}$, а затем действовать, как в разделе 11.2.2.3. Это занимает время $O((N + D)D^2)$.

11.3.1.2. Решение с использованием сингулярного разложения

В этом разделе предполагается, что $D > N$, как обычно и бывает при использовании гребневой регрессии. В этом случае быстрее вычислить сингулярное, а не QR-разложение. Разберемся, как это работает. Пусть $\mathbf{X} = \mathbf{USV}^T$ – сингулярное разложение \mathbf{X} , где $\mathbf{V}^T\mathbf{V} = \mathbf{I}_N$, $\mathbf{UU}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}_N$, а \mathbf{S} – диагональная матрица размера $N \times N$. Обозначим $\mathbf{R} = \mathbf{US}$ матрицу $N \times N$. Можно показать (см. упражнение 18.4 в [HTF09]), что

$$\tilde{\mathbf{w}}_{\text{map}} = \mathbf{V}(\mathbf{R}^T\mathbf{R} + \lambda\mathbf{I}_N)^{-1}\mathbf{R}^T\mathbf{y}. \quad (11.65)$$

Иными словами, мы можем заменить D -мерные векторы \mathbf{x}_i N -мерными векторами \mathbf{r}_i и выполнить аппроксимацию со штрафом, как и раньше. Общее время работы теперь составляет $O(DN^2)$ – меньше $O(D^3)$ при $D > N$.

11.3.2. Связь между гребневой регрессией и PCA

В этом разделе мы обсудим интересную связь между гребневой регрессией и методом главных компонент (PCA) (который будем рассматривать в разделе 20.1), чтобы глубже понять, почему гребневая регрессия работает так хорошо. Наше изложение основано на книге [HTF09, стр. 66].

Пусть $\mathbf{X} = \mathbf{USV}^T$ – сингулярное разложение \mathbf{X} , где $\mathbf{V}^T\mathbf{V} = \mathbf{I}_N$, $\mathbf{UU}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}_N$ и \mathbf{S} – диагональная матрица размера $N \times N$. Из формулы (11.65) видно, что предсказания гребневой регрессии на обучающем наборе имеют вид:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}_{\text{map}} = \mathbf{USV}^T\mathbf{V}(\mathbf{S}^2 + \lambda\mathbf{I})^{-1}\mathbf{SU}^T\mathbf{y}; \quad (11.66)$$

$$= \mathbf{USU}^T\mathbf{y} = \sum_{j=1}^D \mathbf{u}_j \tilde{S}_{jj} \mathbf{u}_j^T \mathbf{y}, \quad (11.67)$$

где

$$\tilde{S}_{jj} \triangleq [\mathbf{S}(\mathbf{S}^2 + \lambda\mathbf{I})^{-1}\mathbf{S}]_{jj} = \frac{\sigma_j^2}{\sigma_j^2 + \lambda} \quad (11.68)$$

и σ_j – сингулярные числа \mathbf{X} . Отсюда

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}_{\text{map}} \tilde{S}_{jj} = \sum_{j=1}^D \mathbf{u}_j \frac{\sigma_j^2}{\sigma_j^2 + \lambda} \mathbf{u}_j^T \mathbf{y}. \quad (11.69)$$

С другой стороны, предсказание методом наименьших квадратов имеет вид:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}_{\text{mle}} = (\mathbf{USV}^T)(\mathbf{VS}^{-1}\mathbf{U}^T\mathbf{y}) = \mathbf{UU}^T\mathbf{y} = \sum_{j=1}^D \mathbf{u}_j \mathbf{u}_j^T \mathbf{y}. \quad (11.70)$$

Если σ_j^2 мало по сравнению с λ , то направление \mathbf{u}_j не окажет заметного влияния на предсказание. Памятуя об этом, определим эффективное число **степеней свободы** модели следующим образом:

$$\text{dof}(\lambda) = \sum_{j=1}^D \frac{\sigma_j^2}{\sigma_j^2 + \lambda}. \quad (11.71)$$

При $\lambda = 0$ $\text{dof}(\lambda) = D$, а при $\lambda \rightarrow \infty$ $\text{dof}(\lambda) \rightarrow 0$.

Попытаемся понять, почему такое поведение желательно. В разделе 11.7 мы покажем, что $\text{Cov}[\mathbf{w}|\mathcal{D}] \propto (\mathbf{X}^T\mathbf{X})^{-1}$, если в качестве априорного распределения \mathbf{w} выбрать равномерное. Поэтому направления наибольшей неопределенности \mathbf{w} описываются собственными векторами матрицы $(\mathbf{X}^T\mathbf{X})^{-1}$ с наибольшими собственными значениями, как показано на рис. 7.6; им соответствуют собственные векторы $\mathbf{X}^T\mathbf{X}$ с наименьшими собственными значениями. В разделе 7.5.2 мы показали, что квадраты сингулярных чисел σ_j^2 равны собственным значениям $\mathbf{X}^T\mathbf{X}$. Поэтому малые сингулярные числа σ_j соответствуют направлениям с высокой апостериорной дисперсией. Это те направления, в которых гребневая регрессия дает наибольшую усадку.

Это процесс показан на рис. 11.7. Горизонтальный параметр w_1 плохо определен данными (его апостериорная дисперсия высока), а вертикальный параметр w_2 определен хорошо. Поэтому $w_{\text{map}}(2)$ близка к $w_{\text{mle}}(2)$, но $w_{\text{map}}(1)$ сильно сдвинута в сторону априорного среднего, равного 0. Таким образом, плохо определенные параметры уменьшаются по величине. Это называется **усадкой** (shrinkage).

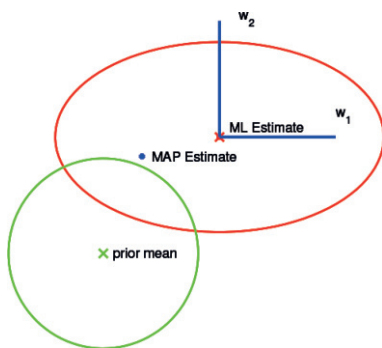


Рис. 11.7 ❖ Геометрия гребневой регрессии. Правдоподобие представлено эллипсом, а априорное распределение – окружностью с центром в начале координат. На основе рис. 3.15 из [Bis06]. Построено программой по адресу figures.probml.ai/book1/11.7

Существует родственная, но иная техника, называемая **регрессией главных компонент**, – вариант PCA с учителем. Идея такова: сначала применить

РСА, чтобы снизить размерность до K , а затем воспользоваться этими низкоразмерными признаками как входными данными для регрессии. Однако этот метод работает хуже гребневой регрессии с точки зрения точности предсказаний [НТФ01, стр. 70]. Причина в том, что регрессия главных компонент сохраняет только первые K (выведенных) измерений, а остальные $D - K$ полностью игнорирует. Напротив, гребневая регрессия выполняет «мягкое» взвешивание всех измерений.

11.3.3. Выбор силы регуляризатора

Чтобы найти оптимальное значение λ , мы можем попробовать конечное число значений и применить перекрестную проверку для оценки ожидаемой потери на каждом (см. раздел 4.5.5.2). Пример приведен на рис. 4.5d.

Этот подход обходится дорого, если выбор возможных значений велик. По счастью, зачастую можно прибегнуть к **теплому запуску** процедуры оптимизации, воспользовавшись значением $\hat{\mathbf{w}}(\lambda_k)$ в качестве инициализатора $\hat{\mathbf{w}}(\lambda_{k+1})$, где $\lambda_{k+1} < \lambda_k$; иными словами, в начале на модель налагаются сильные ограничения (сильный регуляризатор), а затем они постепенно ослабляются (уменьшается степень регуляризации). Множество параметров $\hat{\mathbf{w}}_k$, промежуточное в процессе оптимизации, называется **путем регуляризации** (см. пример на рис. 11.10a).

Мы можем также использовать для выбора λ эмпирический байесовский подход. Именно, гиперпараметр выбирается путем вычисления $\hat{\lambda} = \arg\max_{\lambda} \log p(\mathcal{D}|\lambda)$, где $p(\mathcal{D}|\lambda)$ – маргинальное правдоподобие. На рис. 4.7b показано, что оценка при этом получается почти такой же, как при перекрестной проверке. Однако у байесовского подхода есть несколько преимуществ: для вычисления $p(\mathcal{D}|\lambda)$ достаточно обучить одну модель, тогда как при перекрестной проверке одна и та же модель обучается K раз; кроме того, $p(\mathcal{D}|\lambda)$ – гладкая функция от λ , так что можно использовать градиентную оптимизацию вместо дискретного поиска.

11.4. РЕГРЕССИЯ LASSO

В разделе 11.3 мы предполагали для коэффициентов регрессии гауссово априорное распределение при обучении моделей линейной регрессии. Часто это хорошее решение, поскольку поощряет выбор малых весов и, следовательно, предотвращает переобучение. Но иногда мы хотим, чтобы параметры были не просто малыми, а точно равными нулю, т. е. чтобы вектор $\hat{\mathbf{w}}$ был **разреженным**, поэтому минимизируем **L0-норму**:

$$\|\mathbf{w}\|_0 = \sum_{d=1}^D \mathbb{I}(|w_d| > 0). \quad (11.72)$$

Это полезно, потому что может быть использовано для **отбора признаков**. Чтобы убедиться в этом, заметим, что предсказание имеет вид $f(\mathbf{x}; \mathbf{w}) =$

$\sum_{d=1}^D w_d x_d$, поэтому если какой-то вес $w_d = 0$, то соответствующий признак x_d игнорируется. (Та же идея применима к нелинейным моделям, например ГНС, – для этого мы поощряем разреженность весов первого слоя.)

11.4.1. Оценка MAP с априорным распределением Лапласа (ℓ_1 -регуляризация)

Есть много способов вычисления таких разреженных оценок (см., например, [Bha+19]). В этом разделе мы займемся оценкой MAP с априорным распределением Лапласа (которое обсудим в разделе 11.6.1):

$$p(\mathbf{w}|\lambda) = \prod_{d=1}^D \text{Lap}(w_d|0, 1/\lambda) \propto \prod_{d=1}^D e^{-\lambda|w_d|}, \quad (11.73)$$

где λ – параметр разреженности и

$$\text{Lap}(w|\mu, b) \triangleq \frac{1}{2b} \exp\left(-\frac{|w - \mu|}{b}\right). \quad (11.74)$$

Здесь μ – параметр сдвига, а $b > 0$ – масштабный параметр. На рис. 2.15 показано, что $\text{Lap}(w|0, b)$ сосредотачивает больше плотности в окрестности 0, чем $\mathcal{N}(w|0, \sigma^2)$, даже если дисперсия одна и та же.

Чтобы вычислить оценку MAP для модели линейной регрессии с этим априорным распределением, мы должны минимизировать следующую целевую функцию:

$$\text{PNLL}(\mathbf{w}) = -\log p(\mathcal{D}|\mathbf{w}) - \log p(\mathbf{w}|\lambda) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_1, \quad (11.75)$$

где $\|\mathbf{w}\|_1 \triangleq \sum_{d=1}^D |w_d|$ – ℓ_1 -норма вектора \mathbf{w} . Этот метод называется **lasso**, что расшифровывается как «least absolute shrinkage and selection operator» [Tib96] (почему он так назван, мы объясним позже). Вообще, оценка апостериорного максимума (MAP) с априорным распределением Лапласа называется **ℓ_1 -регуляризацией**.

Заметим также, что можно было бы использовать другие нормы вектора весов. В общем случае q -норма определяется так:

$$\|\mathbf{w}\|_q = \left(\sum_{d=1}^D |w_d|^q \right)^{1/q}. \quad (11.76)$$

Для $q < 1$ можно получить еще более разреженные решения. В пределе, когда $q = 0$, мы имеем **ℓ_0 -норму**:

$$\|\mathbf{w}\|_0 = \sum_{d=1}^D \mathbb{I}(|w_d| > 0). \quad (11.77)$$

Однако можно показать, что для любого $q < 1$ задача становится невыпуклой (см., например, [HTW15]). Таким образом, ℓ_1 -норма – самое сильное из выпуклых ослаблений ℓ_0 -нормы.

11.4.2. Почему ℓ_1 -регуляризация дает разреженные решения?

Теперь объясним, почему ℓ_1 -регуляризация приводит к разреженным решениям, а ℓ_2 -регуляризация – нет. Ограничимся случаем линейной регрессии, хотя похожие рассуждения проходят и для других моделей. В случае регрессии lasso целевая функция негладкая и имеет такой вид (обсуждение гладкости см. в разделе 8.1.4):

$$\min_{\mathbf{w}} \text{NLL}(\mathbf{w}) + \lambda \|\mathbf{w}\|_1. \quad (11.78)$$

Это лагранжиан для следующей задачи квадратичного программирования (см. раздел 8.5.4):

$$\min_{\mathbf{w}} \text{NLL}(\mathbf{w}) \quad \text{при условии } \|\mathbf{w}\|_1 \leq B, \quad (11.79)$$

где B – верхняя граница ℓ_1 -нормы весов: малой (точной) границе B соответствует большой штраф λ и наоборот.

Целевую функцию гребневой регрессии $\min_{\mathbf{w}} \text{NLL}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2$ тоже можно записать в форме с ограничением:

$$\min_{\mathbf{w}} \text{NLL}(\mathbf{w}) \quad \text{при условии } \|\mathbf{w}\|_2^2 \leq B. \quad (11.80)$$

На рис. 11.8 изображены линии уровня целевой функции NLL, а также поверхности ограничений по нормам ℓ_2 и ℓ_1 . Из теории условной оптимизации (раздел 8.5) мы знаем, что оптимальное решение имеет место в точке, где самая нижняя линия уровня целевой функции пересекается с поверхностью

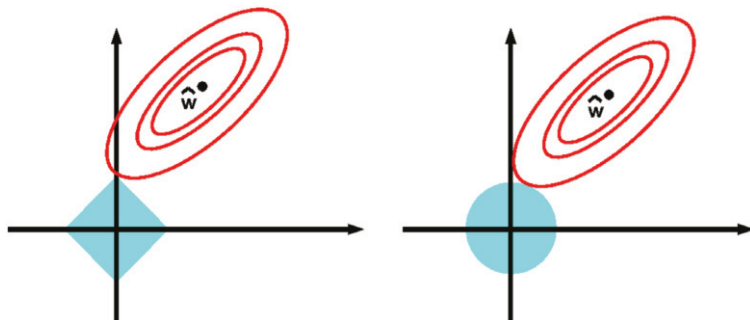


Рис. 11.8 ❖ Сравнение регуляризации метода наименьших квадратов по нормам ℓ_1 и ℓ_2 . На основе рис. 3.12 из [HTF01]

ограничения (в предположении, что ограничение активно). Геометрически ясно, что по мере ослабления ограничения B мы расширяем «шар» по норме ℓ_1 , пока он не коснется целевой функции; эллипс с большей вероятностью пересекут углы «шара», а не его стороны, особенно в пространстве высокой размерности, потому что углы больше «выпирают». Углам соответствуют разреженные решения, расположенные на осях координат. С другой стороны, при расширении шара по норме ℓ_2 точка пересечения с целевой функцией может оказаться где угодно; никаких «углов» нет, поэтому разреженности не отдается предпочтения.

11.4.3. Жесткие и мягкие пороги

Целевая функция регрессии lasso имеет вид $\mathcal{L}(\mathbf{w}) = \text{NLL}(\mathbf{w}) + \lambda \|\mathbf{w}\|_1$. Можно показать (упражнение 11.3), что градиент гладкой части NLL имеет вид:

$$\frac{\partial}{\partial w_d} \text{NLL}(\mathbf{w}) = a_d w_d - c_d; \quad (11.81)$$

$$a_d = \sum_{n=1}^N x_{nd}^2; \quad (11.82)$$

$$c_d = \sum_{n=1}^N x_{nd} (y_n - \mathbf{w}_{-d}^\top \mathbf{x}_{n,-d}), \quad (11.83)$$

где \mathbf{w}_{-d} — это вектор \mathbf{w} без элемента d , и аналогично $\mathbf{x}_{n,-d}$ — вектор признаков без элемента d . Как видим, c_d пропорционально корреляции между d -м столбцом признаков, $\mathbf{x}_{:,d}$, и невязкой, полученной, когда предсказано использование всех остальных признаков, $\mathbf{r}_{-d} = \mathbf{y} - \mathbf{X}_{:,-d} \mathbf{w}_{-d}$. Следовательно, величина c_d показывает, насколько признак d релевантен для предсказания \mathbf{y} относительно других признаков и текущих параметров. Приравнивание градиента нулю дает оптимальное обновление \mathbf{w}_d , когда остальные веса зафиксированы:

$$w_d = c_d / a_d = \frac{\mathbf{x}_{:,d}^\top \mathbf{r}_{-d}}{\|\mathbf{x}_{:,d}\|_2^2}. \quad (11.84)$$

Соответствующее новое предсказание \mathbf{r}_{-d} принимает вид $\hat{\mathbf{r}}_{-d} = w_d \mathbf{x}_{:,d}$, а это не что иное, как ортогональная проекция невязки на вектор-столбец $\mathbf{x}_{:,d}$, что согласуется с формулой (11.15).

Теперь прибавим член ℓ_1 . К сожалению, член $\|\mathbf{w}\|_1$ не дифференцируем, когда $w_d = 0$. Но мы все же можем вычислить субградиент в этой точке. Пользуясь формулой (8.14), находим, что

$$\partial_{w_d} \mathcal{L}(\mathbf{w}) = (a_d w_d - c_d) + \lambda \partial_{w_d} \|\mathbf{w}\|_1 \quad (11.85)$$

$$= \begin{cases} \{a_d w_d - c_d - \lambda\}, & \text{если } w_d < 0 \\ [-c_d - \lambda, -c_d + \lambda], & \text{если } w_d = 0. \\ \{a_d w_d - c_d + \lambda\}, & \text{если } w_d > 0 \end{cases} \quad (11.86)$$

В зависимости от значения c_d следует рассмотреть три случая решения уравнения $\partial_{w_d} \mathcal{L}(\mathbf{w}) = 0$.

1. Если $c_d < -\lambda$, так что между признаком и невязкой существует сильная отрицательная корреляция, то субградиент равен нулю при $\hat{w}_d = (c_d + \lambda)/a_d < 0$.
2. Если $c_d \in [-\lambda, \lambda]$, так что между признаком и невязкой существует лишь слабая корреляция, субградиент равен нулю при $\hat{w}_d = 0$:
3. Если $c_d > \lambda$, так что между признаком и невязкой существует сильная положительная корреляция, то субградиент равен нулю при $\hat{w}_d = (c_d - \lambda)/a_d > 0$.

Короче говоря, имеем

$$\hat{w}_d(c_d) = \begin{cases} (c_d + \lambda)/a_d, & \text{если } c_d < -\lambda \\ 0, & \text{если } c_d \in [-\lambda, \lambda]. \\ (c_d - \lambda)/a_d, & \text{если } c_d > \lambda \end{cases} \quad (11.87)$$

Это можно записать в виде

$$\hat{w}_d = \text{SoftThreshold}\left(\frac{c_d}{a_d}, \lambda/a_d\right), \quad (11.88)$$

где

$$\text{SoftThreshold}(x, \delta) \triangleq \text{sign}(x)(|x| - \delta)_+ \quad (11.89)$$

и $x_+ = \max(x, 0)$ – положительная часть x . Это называется **мягким порогом** (см. также раздел 8.6.2) и показано на рис. 11.9а, где мы построили график зависимости \hat{w}_d от c_d . Черная пунктирная прямая – это место, где $w_d = c_d/a_d$, что соответствует аппроксимации по методу наименьших квадратов. Сплошная красная линия, представляющая регуляризованную оценку \hat{w}_d , сдвинута от пунктирной линии вниз (или вверх) на λ , за исключением участка $-\lambda \leq c_d \leq \lambda$, где она проходит по горизонтальной оси ($w_d = 0$).

На рис. 11.9б показан **жесткий порог**. В этом случае мы устанавливаем w_d в 0, если $-\lambda \leq c_d \leq \lambda$, но не усаживаем значения w_d вне этого интервала. Угловой коэффициент линии мягкого порога не такой же, как к диагонали, т. е. даже большие коэффициенты усаживаются к нулю. Поэтому lasso и означает «least absolute selection and *shrinkage* operator» – оператор выбора наименьшей абсолютной величины и *усадки*. Таким образом, lasso – смещенная оценка (см. раздел 4.7.6.1).

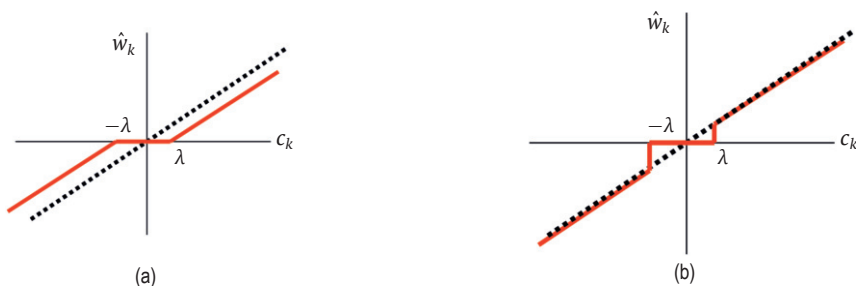


Рис. 11.9 ❖ Слева: задание мягкого порога. Справа: задание жесткого порога. В обоих случаях по горизонтальной оси откладывается невязка в случае предсказания с использованием всех коэффициентов, кроме w_k , а по вертикальной – оценка коэффициента \hat{w}_k , доставляющая минимум этой регуляризированной невязке. Плоский участок в середине – интервал $[-\lambda, +\lambda]$

Простое решение проблемы смещенной оценки – **исключение систематической ошибки** (debiasing) – заключается в использовании двухшагового процесса оценивания: сначала с помощью lasso оценивается носитель вектора весов (т. е. находятся его ненулевые элементы), а затем отобранные коэффициенты переоцениваются методом наименьших квадратов. Пример показан на рис. 11.13.

11.4.4. Путь регуляризации

Если $\lambda = 0$, то мы получаем решение по обыкновенному методу наименьших квадратов, которое будет плотным. При увеличении λ вектор решения $\hat{\mathbf{w}}(\lambda)$ будет становиться все более разреженным. Если λ больше некоторого критического значения, то мы получим $\hat{\mathbf{w}} = \mathbf{0}$. Это происходит тогда, когда градиент NLL оказывается равен градиенту штрафа и они взаимно сокращаются:

$$\lambda_{\max} = \max_d |\nabla_{w_d} \text{NLL}(\mathbf{0})| = \max_d c_d(\mathbf{w} = \mathbf{0}) = \max_d |\mathbf{y}^T \mathbf{x}_{:,d}| = \|\mathbf{X}^T \mathbf{y}\|_{\infty}. \quad (11.90)$$

Можно вместо этого работать с границей B по норме ℓ_1 . При $B = 0$ получаем $\hat{\mathbf{w}} = \mathbf{0}$. По мере увеличения B решение становится плотнее. Наибольшее значение B , при котором хотя бы один компонент равен нулю, равно $B_{\max} = \|\hat{\mathbf{w}}_{\text{mle}}\|_1$.

По мере увеличения λ вектор решения $\hat{\mathbf{w}}$ становится все более разреженным, хотя необязательно монотонно. Мы можем построить графики зависимости значений \hat{w}_d от λ (или от границы B) для каждого признака d ; они называются **путями регуляризации**. Это показано на рис. 11.10b, где мы применяем регрессию lasso к набору данных о раке предстательной железы, взятому из [HTF09]. (Признаки gleason и svi рассматриваются как числовые, а не категориальные.) На рисунке слева при $B = 0$ все коэффициенты нулевые. При увеличении B коэффициенты постепенно «включаются»¹. Анало-

¹ Принято строить зависимость решения от **коэффициента усадки**, определенного как $s(B) = B/B_{\max}$, а не от B . Однако это влияет только на масштаб по горизонтальной оси, но не на форму кривых.

гичный результат для гребневой регрессии показан на рис. 11.10а. Здесь все коэффициенты ненулевые (в предположении, что $\lambda > 0$), поэтому решение не является разреженным.

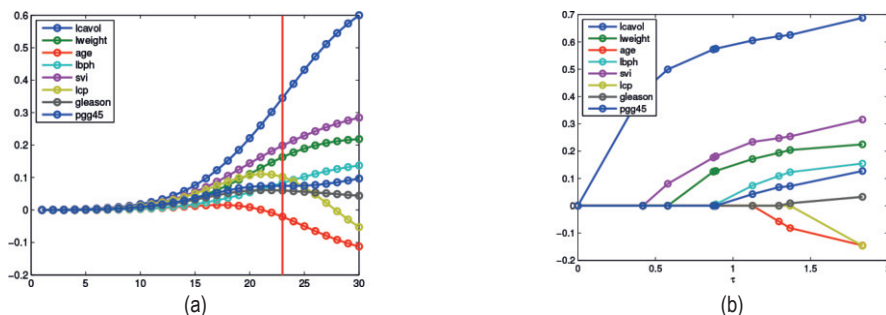


Рис. 11.10 ❖ (а) Профили зависимости коэффициентов гребневой регрессии для примера рака предстательной железы от границы B при ℓ_2 -норме \mathbf{w} , т. е. малые B (большие λ) расположены слева. Вертикальная прямая – значение, выбранное в результате 5-групповой перекрестной проверки по правилу одной стандартной ошибки. На основе рис. 3.8 из [НТФ09]. Построено программой по адресу figures.problml.ai/book1/11.10. (б) То же, что (а), но с использованием нормы ℓ_1 вектора \mathbf{w} . На оси x показаны критические значения $\lambda = 1/B$, в которых путь регуляризации терпит разрывы. На основе рис. 3.10 из [НТФ09]. Построено программой по адресу figures.problml.ai/book1/11.10

Удивительно, но можно показать, что путь регуляризации в случае lasso – кусочно-линейная функция от λ [Efr+04; GL15]. То есть существует множество критических значений λ , в которых активное множество ненулевых коэффициентов изменяется. На участках между этими критическими значениями все ненулевые коэффициенты увеличиваются или уменьшаются линейно. Это показано на рис. 11.10b. Более того, критические значения можно даже найти аналитически [Efr+04]. В табл. 11.1 показаны значения коэффициентов в каждой критической точке на пути регуляризации (последняя строка – решение по методу наименьших квадратов).

Таблица 11.1. Значения коэффициентов для модели линейной регрессии, аппроксимирующей набор данных о раке предстательной железы, при изменении силы ℓ_1 -регуляризаторы. Соответствующие графики показаны на рис. 11.10b

0	0	0	0	0	0	0	0
0.4279	0	0	0	0	0	0	0
0.5015	0.0735	0	0	0	0	0	0
0.5610	0.1878	0	0	0.0930	0	0	0
0.5622	0.1890	0	0.0036	0.0963	0	0	0
0.5797	0.2456	0	0.1435	0.2003	0	0	0.0901
0.5864	0.2572	-0.0321	0.1639	0.2082	0	0	0.1066
0.6994	0.2910	-0.1337	0.2062	0.3003	-0.2565	0	0.2452
0.7164	0.2926	-0.1425	0.2120	0.3096	-0.2890	-0.0209	0.2773

Изменяя λ от λ_{\max} до 0, мы можем перейти от решения, в котором все веса равны нулю, к решению, в котором все веса ненулевые. К сожалению, не любой размер подмножества достигим с помощью lasso. В частности, можно показать, что при $D > N$ оптимальное решение может содержать не более N переменных. В разделе 11.4.8 мы увидим, что если использовать регуляризаторы по норме ℓ_2 и ℓ_1 совместно (этот метод называется эластичной сетью), то можно будет находить разреженные решения, содержащие больше переменных, чем имеется обучающих примеров. Это позволяет задействовать размеры моделей между N и D .

11.4.5. Сравнение методов наименьших квадратов, lasso, гребневой регрессии и выбора подмножеств

В этом разделе мы сравним методы наименьших квадратов, lasso, гребневой регрессии и выбора подмножеств. Для простоты будем предполагать, что все признаки ортонормированы, т. е. $\mathbf{X}^T \mathbf{X} = \mathbf{I}$. В этом случае NLL имеет вид:

$$\text{NLL}(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = \mathbf{y}^T \mathbf{y} + \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{w}^T \mathbf{X}^T \mathbf{y}, \quad (11.91)$$

$$= \text{const} + \sum_d w_d^2 - 2 \sum_d \sum_n w_d x_{nd} y_n, \quad (11.92)$$

и мы видим, что его можно представить в виде суммы членов, по одному на каждое измерение. Поэтому оценки MAP и MLE можно записать аналитически отдельно для каждого веса w_d , как показано ниже.

- **MLE.** Из формулы (11.85) получаем, что решение методом OLS имеет вид:

$$\hat{\mathbf{w}}_d^{\text{mle}} = c_d / a_d = \mathbf{x}_{:d}^T \mathbf{y}, \quad (11.93)$$

где $\mathbf{x}_{:d}$ — d -й столбец \mathbf{X} .

- **Гребневая регрессия.** Можно показать, что оценка гребневой регрессии имеет вид:

$$\hat{w}_d^{\text{ridge}} = \frac{\hat{w}_d^{\text{mle}}}{1 + \lambda}. \quad (11.94)$$

- **Lasso.** Из формулы (11.88) и того факта, что $\hat{\mathbf{w}}_d^{\text{mle}} = c_d / a_d$, имеем

$$\hat{w}_d^{\text{lasso}} = \text{sign}(\hat{w}_d^{\text{mle}}) (|\hat{w}_d^{\text{mle}}| - \lambda)_+. \quad (11.95)$$

Это соответствует заданию мягкого порога, показанному на рис. 11.9а.

- **Выбор подмножества.** Если отобрать K лучших признаков методом выбора подмножества, то оценка параметров принимает вид:

$$\hat{w}_d^{\text{ss}} = \begin{cases} \hat{w}_d^{\text{mle}}, & \text{если } \text{rank}(|\hat{w}_d^{\text{mle}}|) \leq K \\ 0 & \text{в противном случае} \end{cases}, \quad (11.96)$$

где rank – положение в списке весов, отсортированном по величине (ранг). Это соответствует заданию жесткого порога, показанному на рис. 11.9b.

Теперь мы приведем результаты экспериментального сравнения качества предсказания этих методов регрессии на наборе данных о раке предстательной железы, взятые из [HTF09]. (Признаки gleason и svi рассматриваются как числовые, а не категориальные.) На рис. 11.11 показаны оценки коэффициентов при значении λ (или K), выбранном путем перекрестной проверки; как видим, метод выбора подмножества дает самое разреженное решение, а за ним следует lasso. С точки зрения качества предсказаний все методы очень похожи, как показывает рис. 11.12.

Член	OLS	Лучшее подмножество	Гребневая	Lasso
intercept	2.465	2.477	2.467	2.465
lcalvol	0.676	0.736	0.522	0.548
lweight	0.262	0.315	0.255	0.224
age	-0.141	0.000	-0.089	0.000
lbph	0.209	0.000	0.186	0.129
svi	0.304	0.000	0.259	0.186
lcp	-0.287	0.000	-0.095	0.000
gleason	-0.021	0.000	0.025	0.000
pgg45	0.266	0.000	0.169	0.083
Test_error	0.521	0.492	0.487	0.457
Std_error	0.176	0.141	0.157	0.146

Рис. 11.11 ❖ Результаты разных методов для набора данных о раке предстательной железы с 8 признаками и 67 обучающими примерами. Методы: OLS = обыкновенный метод наименьших квадратов, Subset = регрессия на лучшем подмножестве, Ridge = гребневая регрессия, Lasso. В строках представлены коэффициенты; мы видим, что регрессия на подмножестве и lasso дают разреженные решения. В нижней строке приведена среднеквадратическая ошибка на тестовом наборе (30 примеров). На основе табл. 3.3. из [HTF09]. Построено программой по адресу figures.problml.ai/book1/11.11

11.4.6. Согласованность выбора переменных

Обычно ℓ_1 -регуляризацию применяют для оценки множества релевантных переменных; эта процедура называется **выбором переменных**. Метод, способный восстановить истинный набор релевантных переменных (т. е. носитель вектора \mathbf{w}^*) в пределе при $N \rightarrow \infty$, называется **согласованным с выбором модели**. (Это теоретическое понятие, предполагающее, что в основе данных лежит модель.)

Приведем пример. Сначала генерируется разреженный сигнал \mathbf{w}^* размера $D = 4096$, который содержит 160 случайно расположенных пиков, равных ± 1 . Затем генерируется случайная матрица плана \mathbf{X} размера $N \times D$, где $N = 1024$. Наконец, генерируется зашумленное наблюдение $\mathbf{y} = \mathbf{X}\mathbf{w}^* + \boldsymbol{\varepsilon}$, где

$\varepsilon_n \sim \mathcal{N}(0, 0.01^2)$. После этого мы оцениваем \mathbf{w} по \mathbf{y} и \mathbf{X} . Исходный \mathbf{w}^* показан в первом ряду на рис. 11.13. Во втором ряду показана ℓ_1 -оценка $\hat{\mathbf{w}}_{L1}$, полученная при $\lambda = 0.1\lambda_{\max}$. Мы видим, что пики расположены там, где нужно, т. е. модель правильно идентифицировала релевантные переменные. Однако, хотя $\hat{\mathbf{w}}_{L1}$ правильно идентифицировала ненулевые компоненты, они слишком малы из-за усадки. В третьем ряду показаны результаты после устранения систематической ошибки (см. раздел 11.4.3). Видно, что мы смогли восстановить исходный вектор весов. Наконец, в последнем ряду показана плотная оценка, полученная методом OLS. Также видно, что не существует такого одного порога, который можно было бы применить к $\hat{\mathbf{w}}_{\text{mle}}$, чтобы восстановить правильный разреженный вектор весов.

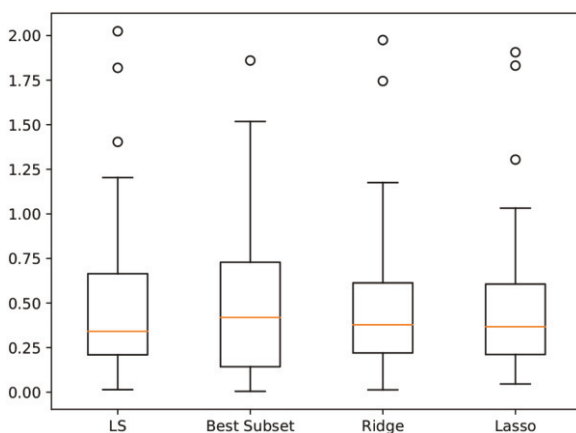


Рис. 11.12 ❖ Коробчатая диаграмма, показывающая абсолютные величины ошибок предсказания на тестовом наборе данных о раке предстательной железы для различных методов регрессии. Построено программой по адресу figures.probl.ai/book1/11.12

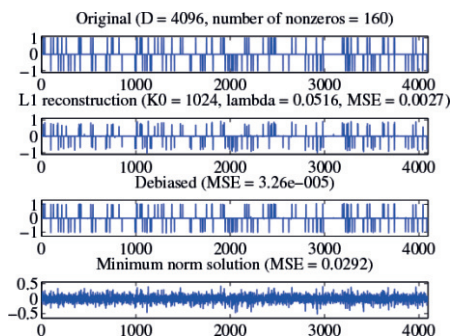


Рис. 11.13 ❖ Пример восстановления разреженного сигнала с помощью lasso. Детали см. в тексте. На основе рис. 1 из работы of [FNW07]. Построено программой по адресу figures.probl.ai/book1/11.13

Чтобы воспользоваться методом lasso для выбора переменных, мы должны задать λ . Для выбора оптимального значения на пути регуляризации часто применяют перекрестную проверку. Однако важно заметить, что перекрестная проверка выбирает значение λ , дающее хорошую точность предсказаний. Обычно оно не совпадает со значением, восстанавливающим «истинную» модель. Чтобы понять, почему это так, вспомним, что ℓ_1 -регуляризация выполняет выбор и усадку, т. е. выбранные коэффициенты сдвигаются ближе к 0. Чтобы предотвратить такую усадку релевантных коэффициентов, перекрестная проверка стремится выбрать не слишком большое значение λ . Конечно, это должно приводить к менее разреженной модели, содержащей нерелевантные переменные (ложноположительные результаты). Действительно, в работе [MB06] доказано, что оптимальное с точки зрения качества предсказаний значение λ не обладает свойством согласованности с выбором модели. Однако были предложены различные модификации базового метода, обеспечивающие согласованность с выбором модели (см., например, [BG11; HTW15]).

11.4.7. Групповое lasso

При стандартной ℓ_1 -регуляризации мы предполагаем, что существует взаимно однозначное соответствие между параметрами и переменными, т. е. $\hat{\mathbf{w}}_d = 0$ означает, что переменная d исключается. Но в более сложных моделях с одной переменной может быть ассоциировано несколько параметров. В частности, с каждой переменной d может быть связан вектор весов \mathbf{w}_d , так что полный вектор весов имеет блочную структуру $\mathbf{w} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_D]$. Если мы хотим исключить переменную d , то должны сделать так, чтобы весь подвектор \mathbf{w}_d обратился в нуль. Это называется **групповой разреженностью**.

11.4.7.1. Приложения

Приведем несколько примеров, когда групповая разреженность бывает полезной.

- Линейная регрессия с категориальными входами: если d -я переменная категориальная и имеет K возможных значений, то она будет представлена унитарным вектором длины K (раздел 1.5.3.1), поэтому, чтобы исключить ее, нужно будет обнулить весь вектор входящих весов.
- Мультиномиальная логистическая регрессия: с d -й переменной ассоциировано C весов, по одному на класс (раздел 10.3), поэтому, чтобы исключить ее, нужно будет обнулить весь вектор исходящих весов.
- Нейронные сети: у k -го нейрона несколько входов, поэтому если мы хотим «выключить нейрон», то должны будем обнулить все входящие веса. Это позволяет использовать групповую разреженность для обучения структуры нейронной сети (детали см., например, в работе [GEN19]).
- Обучение с несколькими задачами: с каждым входным признаком ассоциировано C весов, по одному на выходную задачу. Если мы хотим

использовать некоторый признак для всех задач или ни для одной из них, то должны выбрать веса на уровне групп [OTJ07].

11.4.7.2. Штрафование по норме \boxtimes_2

Для поощрения групповой разреженности мы разбиваем вектор параметров на G групп, $\mathbf{w} = [\mathbf{w}_1, \dots, \mathbf{w}_G]$, а затем минимизируем следующую целевую функцию:

$$\text{PNLL}(\mathbf{w}) = \text{NLL}(\mathbf{w}) + \lambda \sum_{g=1}^G \|\mathbf{w}_g\|_2, \quad (11.97)$$

где $\|\mathbf{w}_g\|_2 = \sqrt{\sum_{d \in g} w_d^2}$ — ℓ_2 -норма вектора групповых весов. Если NLL вычисляется по методу наименьших квадратов, то этот метод называется **групповым lasso** [YL06; Куи+10].

Заметим, что если бы мы использовали в (11.97) квадраты норм, то модель была бы эквивалентна гребневой регрессии, так как

$$\sum_{g=1}^G \|\mathbf{w}_g\|_2^2 = \sum_g \sum_{d \in g} w_d^2 = \|\mathbf{w}\|_2^2. \quad (11.98)$$

Благодаря использованию квадратного корня мы штрафует за радиус шара, содержащего вектор весов группы, а радиус может быть малым, только когда малы все элементы.

По-другому понять, почему вариант с квадратным корнем поощряет разреженность на групповом уровне, можно, рассмотрев градиент целевой функции. Предположим, что имеется только одна группа из двух переменных, поэтому штраф имеет вид $\sqrt{w_1^2 + w_2^2}$. Производная по w_1 равна

$$\frac{\partial}{\partial w_1} (w_1^2 + w_2^2)^{\frac{1}{2}} = \frac{w_1}{\sqrt{w_1^2 + w_2^2}}. \quad (11.99)$$

Если w_2 близко к нулю, то производная стремится к 1, а w_1 также прижимается к нулю с силой, пропорциональной λ . Однако если w_2 велико, то производная стремится к 0, а w_1 тоже может оставаться большим. Поэтому все коэффициенты в группе будут иметь схожий размер.

11.4.7.3. Штрафование по норме \boxtimes_∞

В одном из вариантов описанной техники норма ℓ_2 заменяется нормой ℓ_∞ [TVW05; ZRY05]:

$$\|\mathbf{w}_g\|_\infty = \max_{d \in g} |w_d|. \quad (11.100)$$

Ясно, что это тоже приведет к групповой разреженности, поскольку если самый большой элемент группы принудительно уменьшается, то с меньшими происходит то же самое.

11.4.7.4. Пример

Иллюстрации применения этой техники приведены на рис. 11.14 и 11.15. Имеется истинный сигнал размера $D = 2^{12} = 4096$, разбитый на 64 группы по 64 элемента. Мы случайным образом выбираем 8 групп \mathbf{w} и назначаем им ненулевые значения. На рис. 11.14 значения выбираются из распределения $\mathcal{N}(0, 1)$, а на рис. 11.15 все значения равны 1. Затем мы выбираем случайную матрицу плана \mathbf{X} размера $N \times D$, где $N = 2^{10} = 1024$. Наконец, мы генерируем $\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\varepsilon}$, где $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, 10^{-4}\mathbf{I}_N)$. С такими данными мы оцениваем носитель \mathbf{w} методом lasso или группового lasso по норме ℓ_1 , а затем оцениваем ненулевые значения методом наименьших квадратов (оценка с исключенной систематической ошибкой).

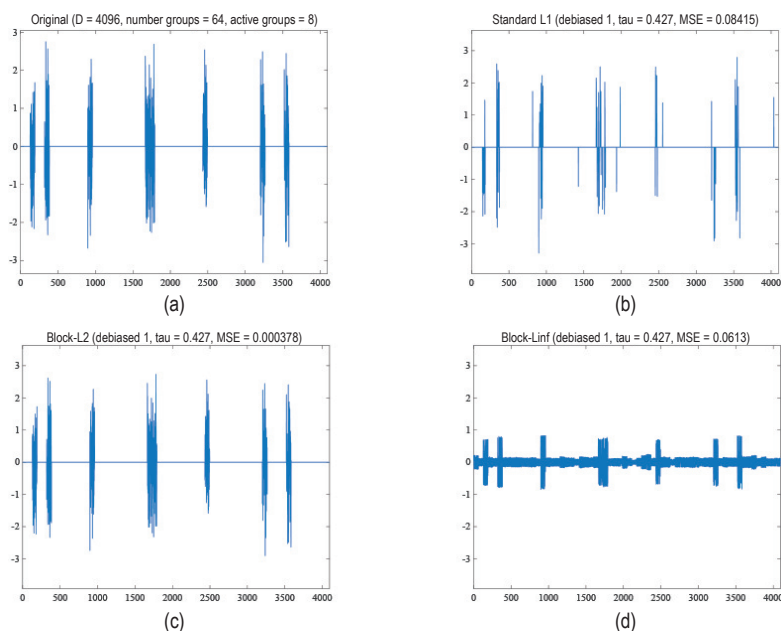


Рис. 11.14 ❖ Иллюстрация группового lasso, когда исходный сигнал – кусочно-гауссова функция. (а) Исходный сигнал. (б) Оценка методом lasso. (с) Оценка методом группового lasso с нормой ℓ_2 на блоках. (д) Оценка методом группового lasso с нормой ℓ_1 на блоках. На основе рис. 3 и 4 из [WNF09]. Построено программой по адресу figures.problm.ai/book1/11.14

Из рисунков видно, что метод группового lasso работает намного лучше, чем обычный метод lasso, поскольку учитывает известную групповую структуру. Также видно, что при использовании нормы ℓ_1 есть тенденция к уравниванию величин всех элементов в одном блоке. Это хорошо во втором примере, но плохо в первом. (Значение λ во всех примерах было одинаково и задано вручную.)

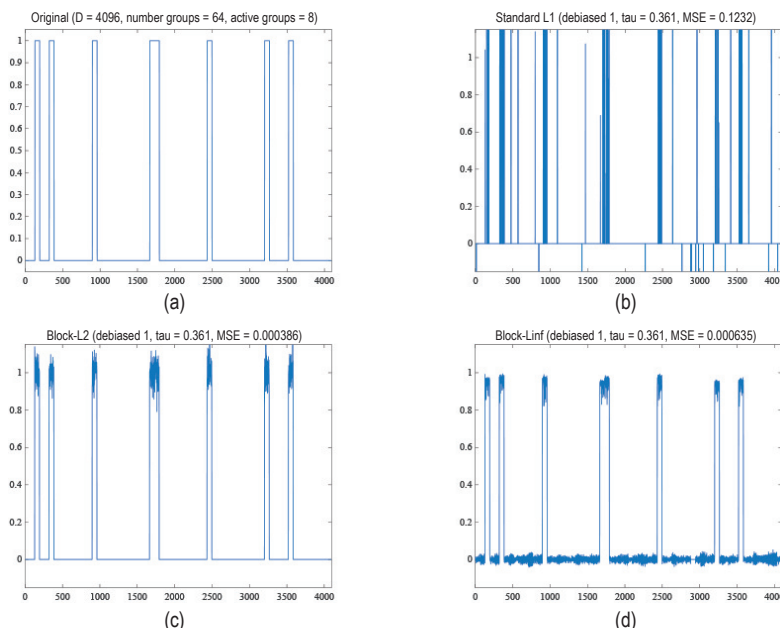


Рис. 11.15 ❖ То же, что рис. 11.14, но исходный сигнал кусочно-постоянный. Построено программой по адресу figures.problm.ai/book1/11.15

11.4.8. Эластичная сеть (комбинация гребневой регрессии и lasso)

Для применения группового lasso мы должны заранее определить групповую структуру. В некоторых задачах она неизвестна, но тем не менее мы хотели бы рассматривать сильно коррелированные коэффициенты как неявную группу. Один из способов достижения этой цели, предложенный в работе [ZH05], – использовать **эластичную сеть**, представляющую собой гибрид гребневой регрессии и lasso-регрессии¹. Это соответствует минимизации следующей целевой функции:

$$\mathcal{L}(\mathbf{w}, \lambda_1, \lambda_2) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda_2 \|\mathbf{w}\|_2^2 + \lambda_1 \|\mathbf{w}\|_1. \quad (11.101)$$

Эта функция строго выпукла (в предположении, что $\lambda_2 > 0$), поэтому существует единственный глобальный минимум, даже если матрица \mathbf{X} неполного ранга. Можно показать [ZH05], что любой строго выпуклый штраф, налагаемый на \mathbf{w} , будет обладать **группирующим эффектом**, т. е. коэффициенты регрессии сильно коррелированных переменных будут близки. В частности, если два признака тождественно равны, т. е. $\mathbf{X}_{:j} = \mathbf{X}_{:k}$, то можно показать, что

¹ Метод назван «эластичная сеть», потому что он работает как «растяжимая рыболовная сеть, которая удерживает всю крупную рыбу» [ZH05].

их оценки тоже равны, $\hat{\mathbf{w}}_j = \hat{\mathbf{w}}_k$. С другой стороны, при использовании lasso может случиться так, что $\hat{\mathbf{w}}_j = 0$, а $\hat{\mathbf{w}}_k \neq 0$ или наоборот, что дает не столь устойчивые оценки.

Помимо такой мягкой группировки, эластичная сеть обладает и другими достоинствами. В частности, если $D > N$, то максимальное число ненулевых элементов, которое можно выбрать (за исключением оценки MLE, имеющей D ненулевых элементов), равно N . А эластичная сеть может выбрать более N ненулевых элементов на пути к плотной оценке и тем самым исследовать больше возможных подмножеств переменных.

11.4.9. Алгоритмы оптимизации

Для решения задачи оптимизации lasso и других ℓ_1 -регуляризованных целевых функций было предложено много алгоритмов. В этом разделе мы кратко рассмотрим некоторые наиболее популярные.

11.4.9.1. Покоординатный спуск

Иногда бывает трудно оптимизировать все переменные сразу, но легко по одной. Именно, мы можем искать j -й коэффициент, зафиксировав все остальные:

$$\mathbf{w}_j^* = \operatorname{argmin}_{\rho} \mathcal{L}(\mathbf{w} + \rho \mathbf{e}_j), \quad (11.102)$$

где \mathbf{e}_j – j -й единичный вектор. Это называется **покоординатным спуском**. Мы можем либо перебирать координаты детерминированно, либо выбирать их случайным образом, либо выбирать ту координату, по которой градиент убывает быстрее всего.

Этот метод особенно привлекателен, если каждую из одномерных задач оптимизации можно решить аналитически, как в случае lasso (см. формулу (11.87)). Это так называемый **алгоритм стрельбы** [Fu98; WL08]. (Слово «стрельба» – отсылка к ковбойской теме, навеянной термином «lasso».) Детали приведены в алгоритме 4.

Метод покоординатного спуска был распространен на обобщенные линейные модели в работе [FHT10] и лежит в основе популярной библиотеки **glmnet**.

Алгоритм 4. Покоординатный спуск для lasso (алгоритм стрельбы)

```

1  Инициализировать  $\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$ ;
2  repeat
3      for  $d = 1, \dots, D$  do
4           $a_d = \sum_{n=1}^N x_{nd}^2$ ;
5           $c_d = \sum_{n=1}^N x_{nd} (y_n - \mathbf{w}^T \mathbf{x}_n + w_d x_{nd})$ ;
6           $w_d = \text{SoftThreshold}(c_d/a_d, \lambda/a_d)$ ;
7  until сошелся;
```

11.4.9.2. Спроецированный градиентный спуск

В этом разделе мы преобразуем недифференцируемый штраф по норме ℓ_1 в гладкий регуляризатор. Для этого сначала воспользуемся **расщеплением переменной** (split variable trick) и запишем $\mathbf{w} = \mathbf{w}^+ - \mathbf{w}^-$, где $\mathbf{w}^+ = \max\{\mathbf{w}, 0\}$ и $\mathbf{w}^- = -\min\{\mathbf{w}, 0\}$. Теперь можно заменить $\|\mathbf{w}\|_1$ суммой $\sum_d (w_d^+ + w_d^-)$. Также необходимо заменить $\text{NLL}(\mathbf{w})$ на $\text{NLL}(\mathbf{w}_d^+ + \mathbf{w}_d^-)$. Таким образом, мы получаем гладкую задачу оптимизации, правда, условной:

$$\min_{\mathbf{w}^+ \geq 0, \mathbf{w}^- \geq 0} \text{NLL}(\mathbf{w}^+ - \mathbf{w}^-) + \lambda \sum_{d=1}^D (w_d^+ + w_d^-). \quad (11.103)$$

Для решения этой задачи можно воспользоваться спроецированным градиентным спуском (раздел 8.6.1). Именно, мы можем навязать ограничение, спроецировав на положительный ортант $w_d := \max(w_d, 0)$; эта операция обозначается P_+ . Тогда обновление спроецированного градиента принимает вид:

$$\begin{pmatrix} \mathbf{w}_{t+1}^+ \\ \mathbf{w}_{t+1}^- \end{pmatrix} = P_+ \left[\begin{pmatrix} \mathbf{w}_t^+ - \rho_t \nabla \text{NLL}(\mathbf{w}_t^+ - \mathbf{w}_t^-) - \rho_t \lambda \mathbf{e} \\ \mathbf{w}_t^- + \rho_t \nabla \text{NLL}(\mathbf{w}_t^+ - \mathbf{w}_t^-) - \rho_t \lambda \mathbf{e} \end{pmatrix} \right], \quad (11.104)$$

где \mathbf{e} – вектор, состоящий из всех единиц.

11.4.9.3. Проксимальный градиентный спуск

В разделе 8.6 мы познакомились с методом проксимального градиентного спуска, который можно использовать для оптимизации гладких функций с негладкими штрафами, например по норме ℓ_1 . В разделе 8.6.2 мы показали, что проксимальный оператор для ℓ_1 -штрафа соответствует мягкому порогу. Таким образом, обновление проксимального градиентного спуска можно записать в виде

$$\mathbf{w}_{t+1} = \text{SoftThreshold}(\mathbf{w}_t - \rho_t \nabla \text{NLL}(\mathbf{w}_t), \rho_t \lambda), \quad (11.105)$$

где оператор мягкого порога (формула (8.134)) применяется поэлементно. Это называется **итеративным алгоритмом мягкого порога** (iterative soft thresholding – **ISTA**) [DDDM04; Don95]. Если объединить его с ускорением Нестерова, то получится метод «быстрый ISTA», или **FISTA** [BT09], который широко применяется для обучения разреженных линейных моделей.

11.4.9.4. LARS

В этом разделе мы обсудим методы, которые могут порождать множество решений для разных значений λ , начиная с пустого множества, т. е. вычисляют полный путь регуляризации (раздел 11.4.4). В этих алгоритмах используется тот факт, что можно быстро вычислить $\hat{\mathbf{w}}(\lambda_k)$ по $\hat{\mathbf{w}}(\lambda_{k-1})$, если $\lambda_k \approx \lambda_{k-1}$; это называется **теплым запуском**. На самом деле, даже если нам нужно только решение для одного значения λ , назовем его λ_* , иногда вычислительно эф-

фективнее вычислить множество решений, от λ_{\max} до λ_* , применяя теплый запуск; это называется **методом продолжения**, или **методом гомотопии**. Часто это гораздо быстрее, чем «холодный запуск» с точки λ_* , особенно когда λ_* мало.

Алгоритм **LARS** [Efr+04] (least angle regression and shrinkage) – пример метода гомотопии для задачи lasso. Он умеет эффективно вычислять $\hat{\mathbf{w}}(\lambda)$ для всех возможных значений λ . (Похожий алгоритм был независимо открыт в работах [OPT00b; OPT00a]).

LARS работает следующим образом. Он начинает с настолько большого значения λ , что выбирается только переменная, которая сильнее всего коррелирована с выходным вектором \mathbf{y} . Затем λ уменьшается, до тех пор пока не будет найдена вторая переменная, имеющая такую же (по величине) корреляцию с текущей невязкой, как и первая переменная, где невязка на k -м шаге пути определяется как $\mathbf{r}_k = \mathbf{y} - \mathbf{X}_{:,F_k} \mathbf{w}_k$, где F_k – текущее **активное множество** (см. формулу (11.83)). Удивительно, но это новое значение λ можно найти аналитически с помощью геометрического рассуждения (отсюда и слова «least angle» (наименьший угол) в названии алгоритма). Это позволяет алгоритму быстро «прыгнуть» к следующей точке на пути регуляризации, в которой активное множество изменяется. Процедура повторяется, пока не будут добавлены все переменные.

Необходимо оставить возможность исключать переменные из текущего активного множества даже при увеличении λ , если мы хотим, чтобы последовательность решений соответствовала пути регуляризации в методе lasso. Если запретить исключение переменных, то получится слегка отличающийся алгоритм, который называется **методом наименьших углов** (least angle regression – LAR). LAR очень похож на **жадный прямой выбор** (greedy forward selection) и на метод, известный под названием **бустинг наименьших квадратов** (least squares boosting) (см., например, [HTW15]).

11.5. РЕГРЕССИОННЫЕ СПЛАЙНЫ*

Мы видели, как использовать разложение по полиномиальному базису для создания нелинейных отображений входа в выход, даже когда модель остается линейной по параметрам. Проблема с полиномами заключается в том, что они дают глобальную аппроксимацию функции. Мы можем добиться большей гибкости с помощью серии локальных аппроксимаций. Для этого нужно только определить набор базисных функций, имеющих локальный носитель. Понятие «локальности» трудно определить в пространствах входов большой размерности, из-за чего в этом разделе мы ограничимся одномерными выходами. Тогда мы сможем построить аппроксимацию функции вида

$$f(x; \boldsymbol{\theta}) = \sum_{i=1}^m w_i B_i(x), \quad (11.106)$$

где B_i – i -я базисная функция.

В качестве базисных функций часто используют **В-сплайны** (буква «В» означает «basis», а «сплайн» – это гибкая линейка, с помощью которого художники рисуют кривые линии, проходящие через заданные точки плоскости).

11.5.1. В-сплайны в качестве базисных функций

Сплайном называется кусочно-полиномиальная функция степени D , для которой положения кусков определяются множеством **узлов**, $t_1 < \dots < t_m$. Точнее, полином определен на каждом из интервалов $(-\infty, t_1)$, $[t_1, t_2]$, \dots , $[t_m, 1)$. Эта функция непрерывна и имеет непрерывные производные порядка 1, \dots , $D - 1$ в узлах. Часто используют **кубические сплайны**, для которых $D = 3$. Такие функции непрерывны, и их первая и вторая производные также непрерывны в каждом узле.

Мы опустим детали вычисления В-сплайнов, поскольку для наших целей они несущественны. Достаточно знать, что можно вызвать функцию `patsy.bs`, которая преобразует матрицу данных \mathbf{X} размера $N \times 1$ в матрицу плана \mathbf{B} размера $N \times (K + D + 1)$, где K – число узлов, а D – степень. (Или можно задать желаемое число базисных функций и позволить `patsy` самой найти число и местоположения узлов.)

Этот подход показан на рис. 11.16, где используются В-сплайны степени 0, 1 и 3 с тремя узлами. Взвешенные комбинации этих базисных функций позволяют получать все более и более гладкие функции, как показано в нижнем ряду.

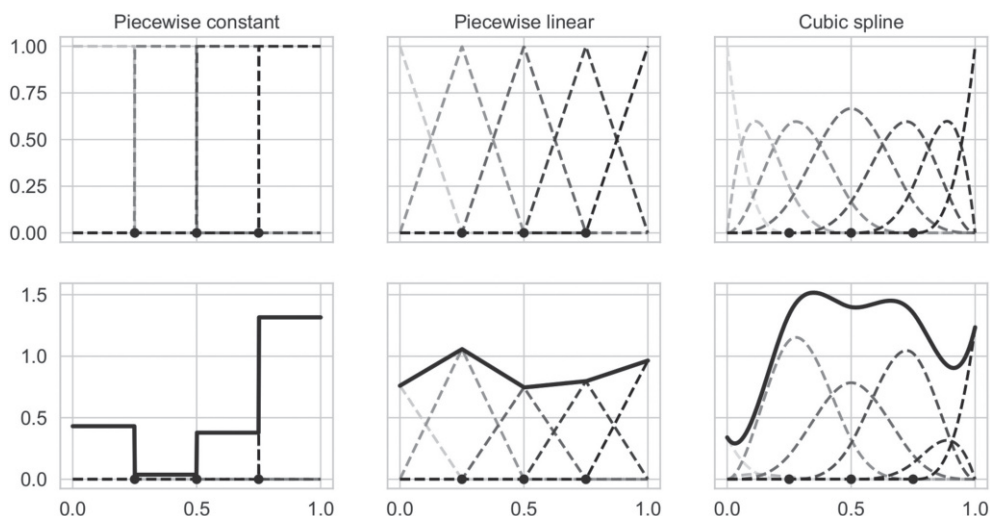


Рис. 11.16 ❖ В-сплайны степени 0, 1 и 3. Верхний ряд: невзвешенные базисные функции. Точками обозначены места трех внутренних узлов – [0.25, 0.5, 0.75]. Нижний ряд: взвешенные комбинации базисных функций со случайными весами. Построено программой по адресу figures.problm.ai/book1/11.16. На основе рис. 5.4 из работы [MKL11]. Печатается с разрешения Освальдо Мартина

На рис. 11.16 видно, что у каждой базисной функции имеется локальный носитель. В любой входной точке x «активны» только $D + 1$ базисных функций. Это станет более очевидным, если нарисовать саму матрицу плана \mathbf{B} . Сначала рассмотрим кусочно-постоянный сплайн, показанный на рис. 11.17a. Первый B-сплайн (столбец 1) равен 1 для первых пяти наблюдений и 0 для всех остальных. Второй B-сплайн (столбец 0) равен 0 для первых пяти наблюдений, 1 для следующих пяти, а затем снова 0 и т. д. Теперь рассмотрим кусочно-линейный сплайн на рис. 11.17b. Первый B-сплайн (столбец 0) изменяется от 1 до 0, следующие три сплайна изменяются от 0 до 1 и снова до 0; последний сплайн (столбец 4) изменяется от 0 до 1; все эти сплайны представлены треугольными формами, показанными на среднем рисунке в верхнем ряду. Наконец, рассмотрим кубический сплайн на рис. 11.17c. Здесь активации более гладкие, так что и результирующая модель тоже будет более гладкой.

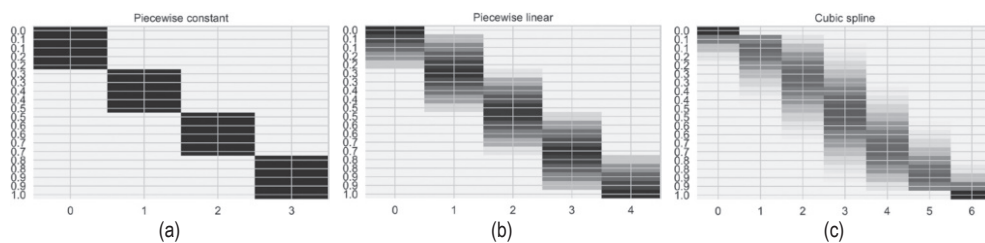


Рис. 11.17 ❖ Матрица плана для B-сплайнов степени (a) 0, (b) 1 и (c) 3. Сплайны вычислены для 20 входных точек в диапазоне от 0 до 1. Построено программой по адресу figures.problml.ai/book1/11.17. На основе рис. 5.6 из работы [MKL11].

Печатается с разрешения Освальдо Мартина

11.5.2. Обучение линейно модели с помощью сплайнового базиса

Вычислив матрицу плана \mathbf{B} , мы можем использовать ее для обучения линейной модели методом наименьших квадратов или гребневой регрессии. (Обычно лучше использовать какую-то регуляризацию.) Например, рассмотрим набор данных из работы [McE20, раздел 4.5], содержащий температуры в начале сезона цветения сакуры в Японии по годам (нам этот набор данных интересен своей полупериодической структурой). Мы аппроксимируем его кубическим сплайном с 15 узлами, выбранным в соответствии с квантилями данных. Результат показан на рис. 11.18. Как видим, аппроксимация выглядит разумно. Увеличение числа узлов улучшит качество аппроксимации, но в конце концов приведет к переобучению. Для выбора числа узлов можно применить какой-нибудь метод выбора модели, например поиск на сетке в сочетании с перекрестной проверкой.

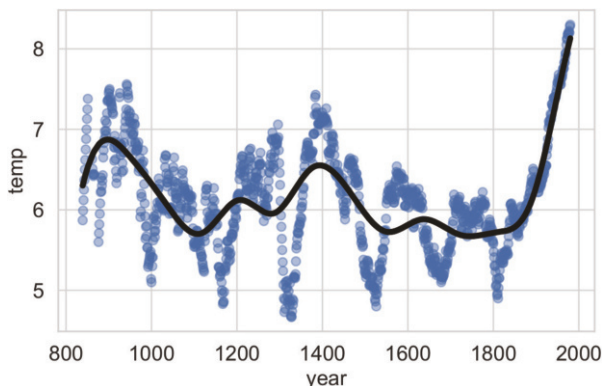


Рис. 11.18 ❖ Аппроксимация одномерного набора данных кубическим сплайном с 15 узлами. Построено программой по адресу figures.problai/book1/11.18. На основе рис. 5.6 из работы [McE20]

11.5.3. Сглаживающие сплайны

Сглаживающие сплайны связаны с регрессионными, но число узлов N совпадает с числом точек данных. Таким образом, это непараметрические модели, потому что число параметров растет вместе с размером данных, а не фиксируется заранее. Во избежание переобучения при использовании сглаживающих сплайнов применяется ℓ_2 -регуляризация. Эта техника тесно связана с регрессией на основе гауссовых процессов, обсуждаемой в разделе 17.2.

11.5.4. Обобщенные аддитивные модели

Обобщенная аддитивная модель (generalized additive model – **GAM**) расширяет сплайновую регрессию на случай многомерных входов [HT90]. Для этого игнорируются взаимодействия между входами и предполагается, что функция имеет следующую аддитивную форму:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \alpha + \sum_{d=1}^D f_d(x_d), \quad (11.107)$$

где каждая функция f_d – регрессионный или сглаживающий сплайн. Эту модель можно обучить методом **настройки с возвращениями** (backfitting), который итеративно подгоняет каждую f_d к частичным невязкам, порождаемым другими членами. GAM-модели можно применять не только к регрессии (но, например, и к классификации), воспользовавшись функцией связи, как в обобщенных линейных моделях (глава 12).

11.6. РОБАСТНАЯ ЛИНЕЙНАЯ РЕГРЕССИЯ*

Очень часто шум в моделях регрессии моделируют с помощью гауссова распределения с нулевым средним и постоянной дисперсией, $r_n \sim \mathcal{N}(0, \sigma^2)$, где $r_n = y_n - \mathbf{w}^T \mathbf{x}_n$. Как мы видели, в этом случае максимизация правдоподобия эквивалентна минимизации суммы квадратов невязок. Однако наличие в данных **выбросов** может приводить к плохой аппроксимации, как показано на рис. 11.19а. (Выбросами являются точки в нижней части рисунка.) Это объясняется тем, что штраф пропорционален квадрату отклонения, поэтому точки, оказавшиеся далеко от прямой, оказывают большее влияние, чем точки вблизи прямой.

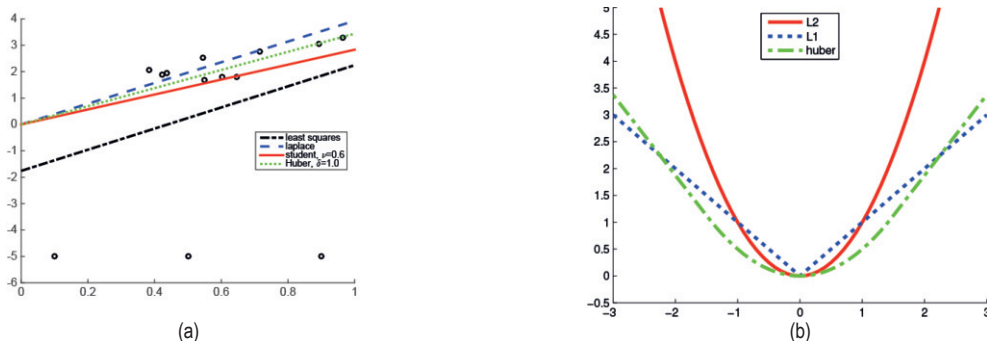


Рис. 11.19 ❖ (а) Робастная линейная регрессия. Построено программой по адресу figures.problml.ai/book1/11.19. (б) Функции потерь ℓ_2 , ℓ_1 и Хьюбера при $\delta = 1.5$. Построено программой по адресу figures.problml.ai/book1/11.19

Один из способов добиться **робастности** к выбросам – заменить гауссово распределение выходной переменной распределением с **тяжелыми хвостами**, которое назначает более высокое правдоподобие выбросам, не прибегая к возмущению прямой линии, чтобы «объяснить» их. Ниже мы обсудим несколько распределений вероятностей выходной переменной, пригодных для этой цели, см. перечень в табл. 11.2.

11.6.1. Правдоподобие Лапласа

В разделе 2.7.3 мы отмечали, что распределение Лапласа тоже робастно к выбросам. Если использовать его в качестве модели наблюдений для регрессии, то получится следующее правдоподобие:

$$p(y|\mathbf{x}, \mathbf{w}, b) = \text{Lap}(y|\mathbf{w}^T \mathbf{x}, b) \propto \exp\left(-\frac{1}{b}|y - \mathbf{w}^T \mathbf{x}|\right). \quad (11.108)$$

Таблица 11.2. Сводка правдоподобий, априорных и апостериорных распределений, применяемых для линейной регрессии. Столбец «Правдоподобие» относится к форме распределения $p(y|x, w, \sigma^2)$, столбец «Априорное» – к форме распределения $p(w)$, а столбец «Апостериорное» – к форме распределения $p(w|D)$. «Точечное» означает вырожденное распределение $\delta(w - \hat{w})$, где \hat{w} – оценка MAP. Оценка MLE эквивалентна точечному апостериорному и равномерному априорному распределению

Правдоподобие	Априорное	Апостериорное	Название	Раздел
Гауссово	Равномерное	Точечное	Наименьших квадратов	11.2.2
Стюдента	Равномерное	Точечное	Робастная регрессия	11.6.2
Лапласа	Равномерное	Точечное	Робастная регрессия	11.6.1
Гауссово	Гауссово	Точечное	Гребневая	11.3
Гауссово	Лапласа	Точечное	Lasso	11.4
Гауссово	Гауссово-Гамма	Гауссово-Гамма	Байесовская линейная регрессия	11.7

Причиной робастности является использование $|y - \mathbf{w}^T \mathbf{x}|$ вместо $(y - \mathbf{w}^T \mathbf{x})^2$. На рис. 11.19а приведен пример этого метода в действии.

11.6.1.1. Вычисление MLE методами линейного программирования

Мы можем вычислить MLE для этой модели средствами линейного программирования. В разделе 8.5.3 мы говорили, что это метод решения задач условной оптимизации вида

$$\operatorname{argmin}_{\mathbf{v}} \mathbf{c}^T \mathbf{v} \quad \text{при условии} \quad \mathbf{A} \mathbf{v} \leq \mathbf{b}, \quad (11.109)$$

где $\mathbf{v} \in \mathbb{R}^n$ – множество n неизвестных параметров, $\mathbf{c}^T \mathbf{v}$ – линейная целевая функция, подлежащая минимизации, а $\mathbf{a}_i^T \mathbf{v} \leq b_i$ – множество m линейных ограничений, которые необходимо удовлетворить. Чтобы применить эту идею к нашей задаче, определим $\mathbf{v} = (w_1, \dots, w_D, e_1, \dots, e_N) \in \mathbb{R}^{D+N}$, где $e_i = |y_i - \hat{y}_i|$ – невязка для примера i . Мы хотим минимизировать сумму невязок, поэтому определим $\mathbf{c} = (0, \dots, 0, 1, \dots, 1) \in \mathbb{R}^{D+N}$, где первые D элементов равны 0, а последние N элементов равны 1.

Мы должны наложить ограничение $e_i = |y_i - \hat{y}_i|$. На самом деле достаточно потребовать, чтобы $|\mathbf{w}^T \mathbf{x}_i - y_i| \leq e_i$, потому что минимизация суммы e_i «натолкнется» на это ограничение и сделает его точным. Поскольку $|a| \leq b \Rightarrow -b \leq a \leq b$, мы можем расписать условие $|\mathbf{w}^T \mathbf{x}_i - y_i| \leq e_i$ в виде двух линейных ограничений:

$$e_i \geq \mathbf{w}^T \mathbf{x}_i - y_i; \quad (11.110)$$

$$e_i \geq -(\mathbf{w}^T \mathbf{x}_i - y_i). \quad (11.111)$$

Условие (11.110) можно записать в виде

$$(\mathbf{x}_i, 0, \dots, 0, -1, 0, \dots, 0)^T \mathbf{v} \leq y_i \quad (11.112)$$

где первые D элементов заполнены \mathbf{x}_i , а $-1 - (D + i)$ -й элемент вектора. Аналогично условие (11.111) можно записать в виде

$$(-\mathbf{x}_i, 0, \dots, 0, -1, 0, \dots, 0)^T \mathbf{v} \leq -y_i. \quad (11.113)$$

Эти ограничения можно записать в форме $\mathbf{A}\mathbf{v} \leq \mathbf{b}$, определив $\mathbf{A} \in \mathbb{R}^{2N \times (N+D)}$ следующим образом:

$$\mathbf{A} = \begin{pmatrix} \mathbf{x}_1 & -1 & 0 & 0 \dots & 0 \\ -\mathbf{x}_1 & -1 & 0 & 0 \dots & 0 \\ \mathbf{x}_2 & 0 & -1 & 0 \dots & 0 \\ -\mathbf{x}_2 & 0 & -1 & 0 \dots & 0 \\ & & \vdots & & \end{pmatrix}, \quad (11.114)$$

а $\mathbf{b} \in \mathbb{R}^{2N}$ – так:

$$\mathbf{b} = (y_1, -y_1, y_2, -y_2, \dots, y_N, -y_N). \quad (11.115)$$

11.6.2. t-правдоподобие Стьюдента

В разделе 2.7.1 мы обсуждали свойства робастности распределения Стьюдента. Чтобы использовать это в контексте регрессии, мы можем просто сделать среднее линейной функцией входов, как предложено в работе [Zel76]:

$$p(y|\mathbf{x}, \mathbf{w}, \sigma^2, \nu) = \mathcal{T}(y|\mathbf{w}^T \mathbf{x}, \sigma^2, \nu). \quad (11.116)$$

Эту модель можно обучить методом СГС или ЕМ (детали см. в [Mur22]).

11.6.3. Функция потерь Хьюбера

Альтернативой минимизации NLL с помощью правдоподобия Лапласа или Стьюдента является использование **функции потерь Хьюбера**, которая определяется следующим образом:

$$\ell_{\text{hyber}}(r, \delta) = \begin{cases} r^2/2, & \text{если } |r| \leq \delta \\ \delta|r| - \delta^2/2, & \text{если } |r| > \delta \end{cases}. \quad (11.117)$$

Она эквивалентна норме ℓ_2 для ошибок, меньших δ , и норме ℓ_1 для больших ошибок. Ее график показан на рис. 5.3.

Преимущество этой функции потерь заключается в том, что она всюду дифференцируема. Следовательно, оптимизировать потерю Хьюбера будет гораздо быстрее, чем правдоподобие Лапласа, потому что можно использовать стандартные методы гладкой оптимизации (например, СГС), а не линейное программирование. На рис. 11.19 показана функция потерь Хьюбера в действии. Результаты качественно похожи на полученные методами Лапласа и Стьюдента.

Параметр δ , управляющий степень робастности, обычно задается вручную или с помощью перекрестной проверки. Однако в работе [Bar19] показано, как аппроксимировать функцию потерь Хьюбера, так чтобы можно было оптимизировать δ градиентными методами.

11.6.4. RANSAC

В компьютерном зрении для робастной регрессии часто применяют метод **RANSAC** (random sample consensus – консенсус на случайной выборке) [FB81]. Он работает следующим образом: производим выборку небольшого начального множества точек, обучаем на нем модель, идентифицируем выбросы относительно этой модели (примеры с большими невязками), исключаем выбросы, а затем переобучаем модель на регулярных точках. Повторяем эту процедуру для многих случайно выбранных начальных множеств и выбираем наилучшую модель.

Детерминированную альтернативу RANSAC дает следующая итеративная схема: сначала предполагаем, что все точки регулярные, и обучаем модель $\hat{\mathbf{w}}_0$; затем на каждой итерации t идентифицируем выбросы как точки с большой невязкой относительно модели $\hat{\mathbf{w}}_t$, исключаем их и переобучаем модель на оставшихся точках – в результате получается модель $\hat{\mathbf{w}}_{t+1}$. Несмотря на то что из-за такой схемы жестких порогов задача становится невыпуклой, можно доказать, что эта простая схема быстро сходится к оптимальной оценке при некоторых разумных предположениях [Muk+19; Sug+19].

11.7. БАЙЕСОВСКАЯ ЛИНЕЙНАЯ РЕГРЕССИЯ*

Мы видели, как вычисляются оценки MLE и MAP для моделей линейной регрессии с различными априорными распределениями. В этом разделе мы обсудим, как вычислять апостериорное распределение параметров, $p(\boldsymbol{\theta}|\mathcal{D})$. Для простоты будем предполагать, что дисперсия известна, так что нам просто нужно вычислить $p(\mathbf{w}|\mathcal{D}, \sigma^2)$. Общий случай рассмотрен во втором томе этой книги, [Mur22].

11.7.1. Априорные распределения

Для простоты будем использовать гауссово априорное распределение:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\bar{\mathbf{w}}, \bar{\Sigma}). \quad (11.118)$$

Это небольшое обобщение априорного распределения, используемого в гребневой регрессии (раздел 11.3). Обсуждение других априорных распределений см. во втором томе этой книги, [Mur22].

11.7.2. Апостериорные распределения

Мы можем переписать правдоподобие в терминах многомерного гауссова распределения в виде

$$p(\mathcal{D}|\mathbf{w}, \sigma^2) = \prod_{n=1}^N p(y_n|\mathbf{w}^\top \mathbf{x}, \sigma^2) = \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w}, \sigma^2 \mathbf{I}_N), \quad (11.119)$$

где \mathbf{I}_N – единичная матрица размера $N \times N$. Затем можно с помощью формулы Байеса для гауссовых распределений (3.37) вывести апостериорное распределение:

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}, \sigma^2) \propto N(\mathbf{w}|\tilde{\mathbf{w}}, \tilde{\Sigma})N(\mathbf{y}|\mathbf{X}\mathbf{w}, \sigma^2 \mathbf{I}_N) = N(\mathbf{w}|\hat{\mathbf{w}}, \hat{\Sigma}); \quad (11.120)$$

$$\hat{\mathbf{w}} \triangleq \tilde{\Sigma}(\tilde{\Sigma}^{-1}\tilde{\mathbf{w}} + \frac{1}{\sigma^2}\mathbf{X}^\top \mathbf{y}); \quad (11.121)$$

$$\hat{\Sigma} \triangleq (\tilde{\Sigma}^{-1} + \frac{1}{\sigma^2}\mathbf{X}^\top \mathbf{X})^{-1}, \quad (11.122)$$

где $\hat{\mathbf{w}}$ – апостериорное среднее, а $\hat{\Sigma}$ – апостериорная ковариация.

Если $\tilde{\mathbf{w}} = \mathbf{0}$ и $\tilde{\Sigma} = \tau^2 \mathbf{I}$, то апостериорное среднее принимает вид $\hat{\mathbf{w}} = (1/\sigma^2)\hat{\Sigma} \mathbf{X}^\top \mathbf{y}$. Если положить $\lambda = \sigma^2/\tau^2$, то мы вернемся к оценке гребневой регрессии, $\hat{\mathbf{w}} = (\lambda \mathbf{I} + \mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$, что совпадает с формулой (11.57).

11.7.3. Пример

Пусть имеется модель одномерной регрессии вида $f(x, \mathbf{w}) = w_0 + w_1 x_1$, где истинные параметры равны $w_0 = -0.3$ и $w_1 = 0.5$. Произведем вывод $p(\mathbf{w}|\mathcal{D})$ и визуализируем двумерное априорное и апостериорное распределение при увеличении размера обучающего набора N .

На рис. 11.20 показаны графики правдоподобия, апостериорного распределения и аппроксимации апостериорного прогнозного распределения¹. В каждом ряду показан график одного из этих распределений при увеличении количества обучающих данных N . Дадим необходимые пояснения.

- В первом ряду $N = 0$, т. е. апостериорное распределение совпадает с априорным. В этом случае наши предсказания «размазаны по всему пространству», потому что априорное распределение, по сути дела, равномерное.
- Во втором ряду $N = 1$, поэтому мы видим одну точку данных (синий кружочек на графике в третьем столбце). Наше апостериорное распределение оказывается ограниченным соответствующим правдоподобием, и предсказания лежат близко к наблюдаемым данным. Однако

¹ Для аппроксимации мы выбрали несколько точек из апостериорного распределения, $\mathbf{w}_s \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, а затем построили график $\mathbb{E}[y|x, \mathbf{w}_s]$, где x пробегает отрезок $[-1, 1]$, для каждого выбранного значения параметра.

апостериорное распределение имеет гребневидную форму, это говорит о том, что существует много возможных решений с разными угловыми коэффициентами и свободными членами. Это согласуется со здравым смыслом, потому что невозможно однозначно вывести два параметра (w_0 и w_1) из одного наблюдения.

- В третьем ряду $N = 2$. В этом случае апостериорное распределение гораздо уже, потому что правдоподобие дает два ограничения. Все наши предсказания будущего теперь расположены ближе к обучающим данным.

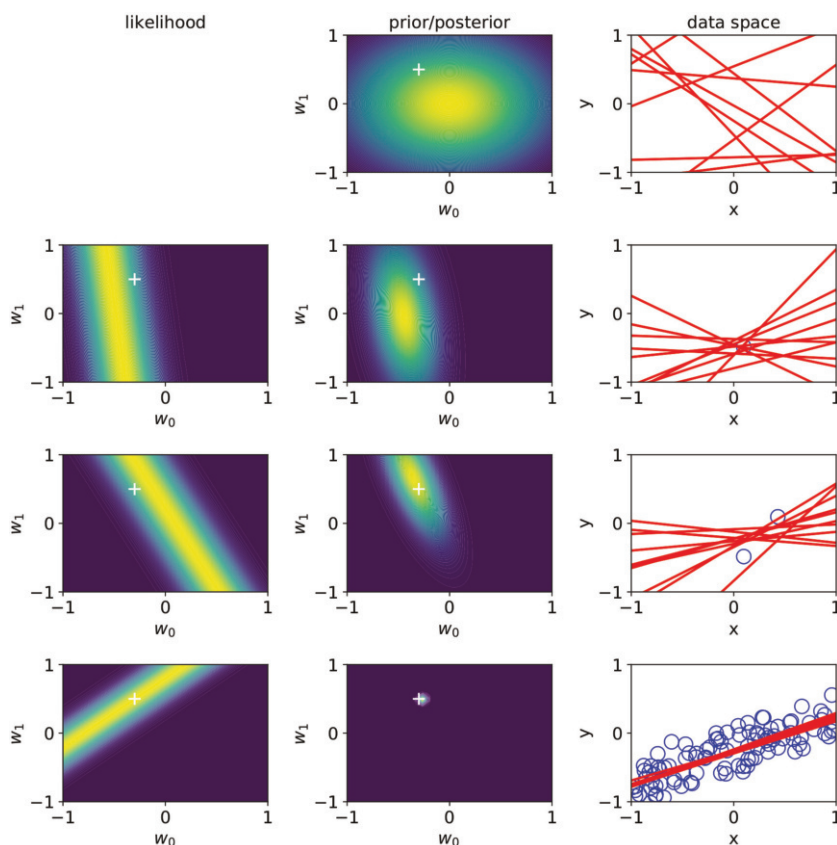


Рис. 11.20 ❖ Последовательный байесовский вывод параметров модели линейной регрессии $p(y|x) = \mathcal{N}(y|w_0 + w_1x_1, \sigma^2)$. Левый столбец: функция правдоподобия для текущей точки данных. Средний столбец: апостериорное распределение при условии N точек данных, $p(w_0, w_1|x_{1:N}, y_{1:N}, \sigma^2)$. Правый столбец: выборка из текущего апостериорного прогнозного распределения. Ряд 1: априорное распределение ($N = 0$). Ряд 2: после 1 точки данных. Ряд 3: после 2 точек данных. Ряд 4: после 100 точек данных. Белый крестик в столбцах 1 и 2 представляет истинное значение параметра; Мы видим, что мода апостериорного распределения быстро сходится к этой точке. Синие кружочки в столбце 3 – наблюдаемые точки данных. На основе рис. 3.7 из [Bis06]. Построено программой по адресу figures.probml.ai/book1/11.20

- В четвертом (последнем) ряду $N = 100$. Теперь апостериорное распределение является, по существу, дельта-функцией, сосредоточенной в истинном значении $\mathbf{w}_* = (-0.3, 0.5)$, отмеченном белым кружочком на графиках во втором и третьем столбцах. Разброс предсказаний связан с неустранимым гауссовым шумом с дисперсией σ^2 .

Этот пример показывает, что при увеличении количества данных оценка апостериорного среднего, $\hat{\mu} = E[\mathbf{w}|\mathcal{D}]$, сходится к истинному значению \mathbf{w}_* , с которым генерировались данные. Поэтому мы говорим, что байесовская оценка является состоятельной (детали см. в разделе 5.3.2). Мы также видим, что неопределенность апостериорного распределения со временем уменьшается. Именно это имеют в виду, когда говорят, что параметры «обучаются» по мере увеличения объема наблюдаемых данных.

11.7.4. Вычисление апостериорного прогнозного распределения

Мы обсудили, как вычислять неопределенность параметров модели, $p(\mathbf{w}|\mathcal{D})$. Но как насчет неопределенности предсказаний будущих выходов? Пользуясь формулой (3.38), можно показать, что апостериорное прогнозное распределение в тестовой точке \mathbf{x} также является гауссовым:

$$p(y|\mathbf{x}, \mathcal{D}, \sigma^2) = \int \mathcal{N}(y|\mathbf{x}^\top \mathbf{w}, \sigma^2) \mathcal{N}(\mathbf{w}|\hat{\mu}, \hat{\Sigma}) d\mathbf{w}; \quad (11.123)$$

$$= \mathcal{N}(y|\hat{\mu}^\top \mathbf{x}, \hat{\sigma}^2(\mathbf{x})), \quad (11.124)$$

где $\hat{\sigma}^2(\mathbf{x}) \triangleq \sigma^2 + \mathbf{x}^\top \hat{\Sigma} \mathbf{x}$ – дисперсия апостериорного прогнозного распределения в точке \mathbf{x} после наблюдения N обучающих примеров. Предсказанная дисперсия зависит от двух членов: дисперсии шума наблюдений σ^2 и дисперсии параметров $\hat{\Sigma}$. Последняя транслируется в дисперсию наблюдений способом, зависящим от близости \mathbf{x} к обучающим данным \mathcal{D} . Это показано на рис. 11.21b, где мы видим, что «усы» (величина ошибки) становятся длиннее по мере удаления от обучающих точек, что является признаком возрастающей неопределенности. Это может оказаться важным в некоторых приложениях, например активном обучении, где мы можем выбирать, откуда собирать обучающие данные (см. раздел 19.4).

Иногда численное вычисление апостериорного распределения параметров, $p(\mathbf{w}|\mathcal{D})$, невозможно. В таких случаях мы можем взять точечную оценку, $\hat{\mathbf{w}}$, а затем воспользоваться подстановочной аппроксимацией. Это дает

$$p(y|\mathbf{x}, \mathcal{D}, \sigma^2) = \int \mathcal{N}(y|\mathbf{x}^\top \mathbf{w}, \sigma^2) \delta(\mathbf{w} - \hat{\mathbf{w}}) d\mathbf{w} = p(y|\mathbf{x}^\top \hat{\mathbf{w}}, \sigma^2). \quad (11.125)$$

Как видим, апостериорная прогнозная дисперсия постоянна и не зависит от данных, что показано на рис. 11.21a. Производя выборку параметра из этого апостериорного распределения, мы всегда будем восстанавливать одну и ту же функцию, как показано на рис. 11.21c. Напротив, при выборке из истинного апостериорного распределения, $\mathbf{w}_s \sim p(\mathbf{w}|\mathcal{D}, \sigma^2)$, мы получим диа-

пазон различных функций, показанный на рис. 11.21d, что точнее отражает неопределенность.

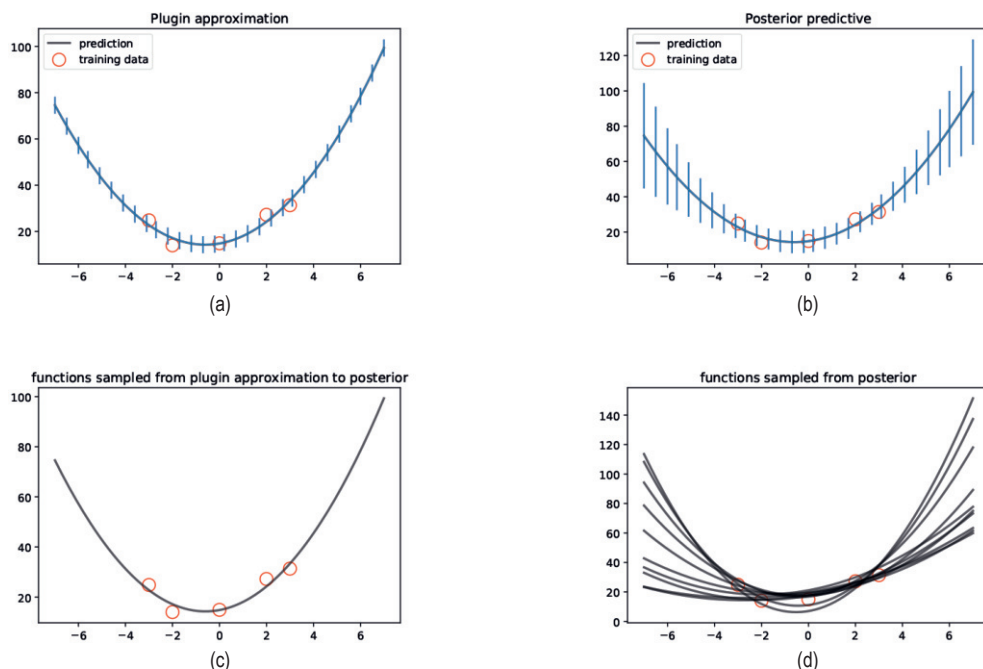


Рис. 11.21 ❖ (a) Подстановочная аппроксимация плотности прогнозного распределения (подставляется MLE параметров) при аппроксимации одномерных данных полиномом второй степени. (b) Плотность апостериорного прогнозного распределения, полученная путем исключения параметров с помощью интегрирования. Черная кривая – апостериорное среднее, длина усов равна двум стандартным отклонениям от плотности. (c) 10 примеров, выбранных из подстановочной аппроксимации апостериорного прогнозного распределения. (d) 10 примеров, выбранных из истинного апостериорного прогнозного распределения. Построено программой по адресу figures.problm.ai/book1/11.21

11.7.5. Преимущество центрирования

Внимательный читатель, возможно, заметил, что двумерное апостериорное распределение на рис. 11.20 имеет форму вытянутого эллипса (который при $N \rightarrow \infty$ схлопывается в точку). Отсюда следует, что существует сильная апостериорная корреляция между обоими параметрами, которая может вызвать вычислительные трудности.

Чтобы понять, почему это происходит, заметим, что каждая точка данных индуцирует функцию правдоподобия, соответствующую прямой, проходящей через эту точку. Глядя на все данные вместе, мы видим, что предсказания с максимальным правдоподобием должны соответствовать прямым, проходящим через среднее данных, (\bar{x}, \bar{y}) . Таких прямых много, но, увели-

чивая угловой коэффициент, мы должны уменьшать свободный член. Поэтому можно считать, что прямые с высокой вероятностью вращаются вокруг среднего, как колесо фортуны¹. Корреляция между w_0 и w_1 объясняет, почему апостериорное распределение имеет форму диагональной прямой. (Гауссово априорное распределение преобразует ее в вытянутый эллипс, но апостериорная корреляция все же сохраняется до тех пор, пока размер выборки не станет настолько велик, что апостериорное распределение схлопнется в точку.)

Вычислять такие вытянутые апостериорные распределения бывает трудно. Простое решение – центрировать входные данные, положив $x'_n = x_n - \bar{x}$. Теперь прямые могут вращаться вокруг начала координат, что уменьшает апостериорную корреляцию между w_0 и w_1 (см. иллюстрацию на рис. 11.22). (Можно также разделить каждое x_n на стандартное отклонение этого признака, как предлагалось в разделе 10.2.8.)

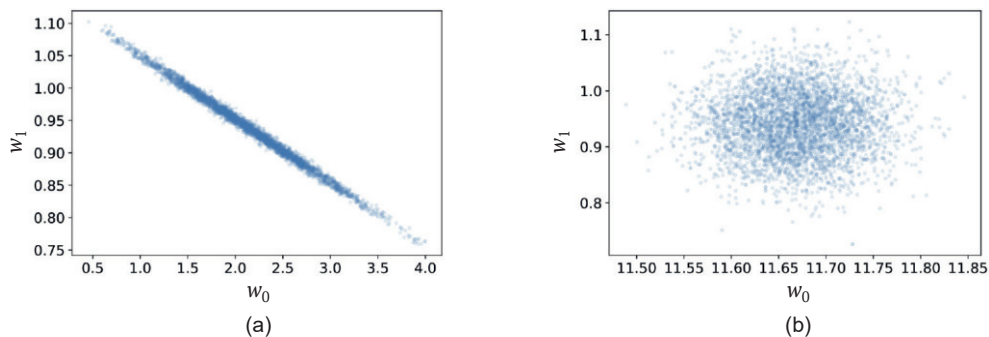


Рис. 11.22 ❖ Выборка из апостериорного распределения $p(w_0, w_1 | \mathcal{D})$ для одномерной модели регрессии $p(y | x, \theta) = \mathcal{N}(y | w_0 + w_1 x, \sigma^2)$ с гауссовым априорным распределением. (a) Исходные данные. (b) Центрированные данные. Построено программой по адресу figures.problml.ai/book1/11.22

Мы можем преобразовать апостериорное распределение, выведенное путем аппроксимации центрированных данных, в первоначальную систему координат, заметив, что

$$y' = w'_0 + w'_1 x' = w'_0 + w'_1 (x - \bar{x}) = (w'_0 - w'_1 \bar{x}) + w'_1 x. \quad (11.126)$$

Таким образом, параметры нецентрированных данных равны $w_0 = w'_0 - w'_1 \bar{x}$ и $w_1 = w'_1$.

11.7.6. Мультиколлинеарность

Во многих наборах данных входные переменные могут быть сильно коррелированы. Включение их всех в общем случае не ухудшает верность предсказаний (при условии, что используется подходящее априорное распределение

¹ Эта аналогия взята из работы [Mar18, стр. 96].

или регуляризатор, предотвращающий переобучение). Однако это может затруднить интерпретацию коэффициентов.

Для иллюстрации воспользуемся искусственным примером из работы [McE20, раздел 6.1]. Пусть имеется набор данных о N лицах, в котором представлены сведения об их росте h_i , а также длине левой и правой ноги, l_i и r_i . Предположим, что $h_i \sim \mathcal{N}(10, 2)$, так что средний рост равен $h^- = 10$ (единица измерения не указана). И пусть длины ног, измеренные в долях от роста, имеют равномерное распределение $\rho_i \sim \text{Unif}(0.4, 0.5)$ плюс небольшой гауссов шум: $l_i \sim \mathcal{N}(\rho_i h_i, 0.02)$ и $r_i \sim \mathcal{N}(\rho_i h_i, 0.02)$.

Пусть требуется предсказать рост человека по длинам его ног (я же говорил, что это искусственный пример!). Поскольку длины левой и правой ноги – зашумленные измерения неизвестной величины, полезно учитывать обе. Поэтому применим линейную регрессию для аппроксимации $p(h|l, r) = \mathcal{N}(h|\alpha + \beta_l l + \beta_r r, \sigma^2)$. Мы используем широкие распределения $\alpha, \beta_l, \beta_r \sim \mathcal{N}(0, 100)$ и $\sigma \sim \text{Expn}(1)$.

Поскольку средняя длина ноги равна $\bar{l} = 0.45\bar{h} = 4.5$, можно было бы ожидать, что оба коэффициента β будут близки к $\bar{h}/\bar{l} = 10/4.5 = 2.2$. Однако маргинальные апостериорные распределения, показанные на рис. 11.23, говорят о другом: мы видим, что апостериорное среднее β_l близко к 2.6, но β_r близко к -0.6 . Таким образом, складывается впечатление, что данные о правой ноге не нужны. А все дело в том, что коэффициент регрессии для признака j кодирует ценность знания x_j при условии, что все остальные признаки x_{-j} уже известны; мы обсуждали это в разделе 11.2.2.1. Если мы уже знаем длину левой ноги, то маргинальная ценность знаний о длине правой ноги невелика. Однако если повторить этот пример, немного изменив данные, то можно прийти к противоположному выводу и отдать предпочтение правой ноге перед левой.

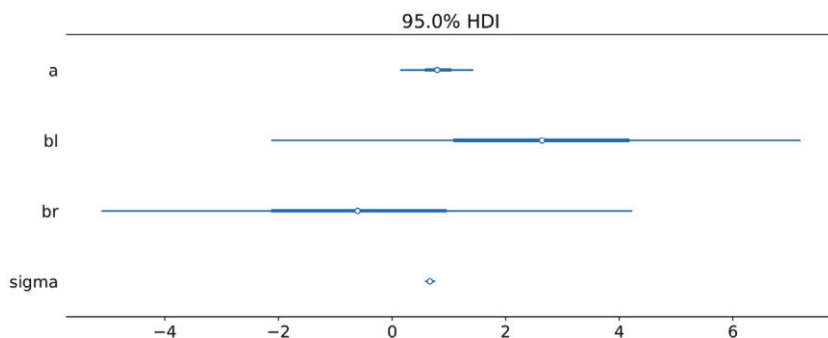


Рис. 11.23 ❖ Апостериорные маргинальные распределения параметров в примере с двумя ногами.

Построено программой по адресу figures.probml.ai/book1/11.23

Получить более ясную картину можно, взглянув на совместное распределение $p(\beta_l, \beta_r|D)$, показанное на рис. 11.24а. Мы видим, что параметры очень сильно коррелированы, так что если β_r велико, то β_l мало, и наоборот. Марги-

нальное распределение каждого параметра не улавливает этот факт. Однако оно все же показывает, что неопределенность каждого параметра велика, т. е. они неидентифицируемы. Вместе с тем, их сумма корректно определена, что видно из рис. 11.24b, где представлен график $p(\beta_l + \beta_r|\mathcal{D})$; он центрирован относительно точки 2.2, как и следовало ожидать.

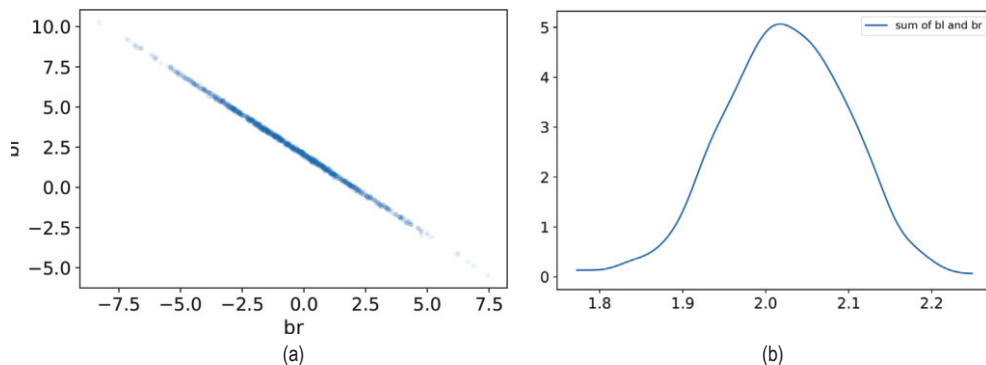


Рис. 11.24 ❖ Апостериорные распределения в примере с двумя ногами. (а) Совместное апостериорное распределение $p(\beta_l, \beta_r|\mathcal{D})$ (б) Апостериорное распределение $p(\beta_l + \beta_r|\mathcal{D})$. Построено программой по адресу figures.probl.ai/book1/11.24

Этот пример показывает, что нужно проявлять осторожность, пытаясь интерпретировать значимость оценок отдельных коэффициентов модели, поскольку порознь они могут нести мало информации.

11.7.7. Автоматическое определение релевантности (ARD)*

Рассмотрим модель линейной регрессии с известным шумом наблюдений, но неизвестными весами, $\mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I})$. Предположим, что для весов используется гауссово априорное распределение, $w_j \sim \mathcal{N}(0, 1/\alpha_j)$, где α_j – точность j -го параметра. Предположим также, что априорные оценки точностей имеют вид:

$$\hat{\alpha} = \underset{\alpha}{\operatorname{argmax}} p(\mathbf{y}|\mathbf{X}, \alpha), \quad (11.127)$$

где

$$p(\mathbf{y}|\mathbf{X}, \alpha) = \int p(\mathbf{y}|\mathbf{X}\mathbf{w}, \sigma^2)p(\mathbf{w}|\mathbf{0}, \operatorname{diag}(\alpha)^{-1})d\mathbf{w} \quad (11.128)$$

– маргинальное правдоподобие. Это пример эмпирического байесовского метода, потому что мы оцениваем априорное распределение по данным. Можно рассматривать это как вычислительное упрощение полного байесов-

ского подхода. Однако у него есть и дополнительные преимущества. Именно, предположим, что после оценивания α мы вычисляем оценку MAP:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \mathcal{N}(\mathbf{w} | \mathbf{0}, \hat{\alpha}^{-1}). \quad (11.129)$$

Это дает разреженную оценку $\hat{\mathbf{w}}$, что может показаться удивительным, если учесть, что гауссово априорное распределение \mathbf{w} не поощряет разреженности. Причины этого явления объясняются во втором томе этой книги.

Этот прием называется **разреженным байесовским обучением** [Tip01], или **автоматическим определением релевантности** (automatic relevancy determination – **ARD**) [Mac95; Nea96]. Первоначально он был разработан для нейронных сетей (где разреженность важна для весов первого слоя), но здесь мы применили его к линейным моделям (см. также раздел 17.4.1, где он применяется к ядерным линейным моделям).

11.8. УПРАЖНЕНИЯ

Упражнение 11.1 [линейная регрессия с несколькими выходами*].
(Источник: Яаккола.)

Рассмотрим линейную модель регрессии с двумерным выходным вектором $\mathbf{y}_i \in \mathbb{R}^2$. Пусть имеются бинарные входные данные, $x_i \in \{0, 1\}$. Обучающие данные имеют вид:

x	y
0	$(-1, -1)^T$
0	$(-1, -2)^T$
0	$(-2, -1)^T$
1	$(1, 1)^T$
1	$(1, 2)^T$
1	$(2, 1)^T$

Погрузим каждое x_i в двумерное пространство с помощью следующей базисной функции:

$$\phi(0) = (1, 0)^T, \phi(1) = (0, 1)^T. \quad (11.130)$$

Модель принимает вид:

$$\hat{\mathbf{y}} = \mathbf{W}^T \phi(x), \quad (11.131)$$

где \mathbf{W} – матрица 2×2 . Вычислите MLE матрицы \mathbf{W} по приведенным выше данным.

Упражнение 11.2 [центрирование и гребневая регрессия].

Предположим, что $\bar{\mathbf{x}} = 0$, т. е. входные данные уже центрированы. Покажите, что оптимизатор

$$J(\mathbf{w}, w_0) = (\mathbf{y} - \mathbf{X}\mathbf{w} - w_0\mathbf{1})^T(\mathbf{y} - \mathbf{X}\mathbf{w} - w_0\mathbf{1}) + \lambda \mathbf{w}^T \mathbf{w} \quad (11.132)$$

дает

$$\hat{w}_0 = \bar{y}; \quad (11.133)$$

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}. \quad (11.134)$$

Упражнение 11.3 [частная производная RSS*].

Пусть $RSS(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$ – сумма квадратов невязок.

а. Покажите, что

$$\frac{\partial}{\partial w_k} RSS(\mathbf{w}) = a_k w_k - c_k; \quad (11.135)$$

$$a_k = 2 \sum_{i=1}^n x_{ik}^2 = 2 \|\mathbf{x}_{:,k}\|^2; \quad (11.136)$$

$$c_k = 2 \sum_{i=1}^n x_{ik} (y_i - \mathbf{w}_{-k}^T \mathbf{x}_{i,-k}) = 2 \mathbf{x}_{:,k}^T \mathbf{r}_k, \quad (11.137)$$

где \mathbf{w}_{-k} – \mathbf{w} без k -го элемента, $\mathbf{x}_{i,-k}$ – \mathbf{x}_i без k -го элемента и $\mathbf{r}_k = \mathbf{y} - \mathbf{w}_{-k}^T \mathbf{x}_{:, -k}$ – невязка вследствие использования всех признаков, кроме k -го. *Указание:* разбейте веса на включающие и не включающие k -й.

б. Покажите, что если $\frac{\partial}{\partial w_k} RSS(\mathbf{w}) = 0$, то

$$\hat{w}_k = \frac{\mathbf{x}_{:,k}^T \mathbf{r}_k}{\|\mathbf{x}_{:,k}\|^2}. \quad (11.138)$$

Следовательно, при последовательном добавлении признаков оптимальный вес k -го признака вычисляется путем ортогонального проецирования $\mathbf{x}_{:, -k}$ на текущую невязку.

Упражнение 11.2 [сведение эластичной сети к lasso].

Определим

$$J_1(\mathbf{w}) = |\mathbf{y} - \mathbf{X}\mathbf{w}|^2 + \lambda_2 |\mathbf{w}|^2 + \lambda_1 |\mathbf{w}|_1 \quad (11.139)$$

и

$$J_2(\mathbf{w}) = |\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\tilde{\mathbf{w}}|^2 + c\lambda_1 |\mathbf{w}|_1, \quad (11.140)$$

где $c = (1 + \lambda_2)^{-1/2}$ и

$$\tilde{\mathbf{X}} = c \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda_2} \mathbf{I}_d \end{pmatrix}, \quad \tilde{\mathbf{y}} = \begin{pmatrix} \mathbf{y} \\ \mathbf{0}_{d \times 1} \end{pmatrix}. \quad (11.141)$$

Покажите, что

$$\operatorname{argmin} J_1(\mathbf{w}) = c(\operatorname{argmin} J_2(\mathbf{w})), \quad (11.142)$$

т. е.

$$J_1(c\mathbf{w}) = J_2(\mathbf{w}), \quad (11.143)$$

и, следовательно, задачу эластичной сети можно решить, применив регрессию lasso к модифицированным данным.

Упражнение 11.5 [усадка в линейной регрессии*].

(Источник: Яаккола.)

Рассмотрим линейную регрессию с ортонормированной матрицей плана, т. е. $\|\mathbf{x}_{:, -k}\|_2^2 = 1$ для каждого столбца (признака) k и $\mathbf{x}_{:, k}^\top \mathbf{x}_{:, j} = 0$, так что каждый параметр w_k можно оценивать отдельно.

На рис. 10.15b показаны графики зависимости $\hat{\mathbf{w}}_k$ от $c_k = 2\mathbf{y}^\top \mathbf{x}_{:, k}$, корреляции k -го признака с выходом, для трех разных методов оценивания: обыкновенных наименьших квадратов (OLS), гребневой регрессии с параметром λ_2 и lasso с параметром λ_1 .

- К несчастью, мы забыли пометить графики. Каким методом построена сплошная (1), пунктирная (2) и штриховая (3) линия?
- Каково значение λ_1 ?
- Каково значение λ_2 ?

Упражнение 11.6 [ЕМ-алгоритм для смеси методов линейной регрессии].

Выведите уравнения ЕМ для аппроксимации смесью методов линейной регрессии.

Глава 12

Обобщенные линейные модели*

12.1. ВВЕДЕНИЕ

В главе 10 мы обсуждали логистическую регрессию, которая в бинарном случае соответствует модели $p(y|\mathbf{x}, \mathbf{w}) = \text{Ber}(y|\sigma(\mathbf{w}^\top \mathbf{x}))$. В главе 11 мы обсуждали линейную регрессию, соответствующую модели $p(y|\mathbf{x}, \mathbf{w}) = \mathcal{N}(y|\mathbf{w}^\top \mathbf{x}, \sigma^2)$. Очевидно, что они очень похожи. В частности, в обоих случаях среднее выхода, $\mathbb{E}[y|\mathbf{x}, \mathbf{w}]$, является линейной функцией от входов \mathbf{x} .

Оказывается, существует широкое семейство моделей, обладающих этим свойством. Они называются **обобщенными линейными моделями** (generalized linear model – **GLM**) [MN89].

GLM – это условная версия экспоненциального семейства (раздел 3.4), в которой естественные параметры являются линейными функциями входов. Точнее, модель имеет следующий вид:

$$p(y_n|\mathbf{x}_n, \mathbf{w}, \sigma^2) = \exp\left\{\frac{y_n \eta_n - A(\eta_n)}{\sigma^2} + \log h(y_n, \sigma^2)\right\}, \quad (12.1)$$

где $\eta_n \triangleq \mathbf{w}^\top \mathbf{x}_n$ – естественный (зависящий от входных данных) параметр, $A(\eta_n)$ – логарифмический нормализатор, $\mathcal{T}(y) = y$ – достаточная статистика, а σ^2 – рассеяние¹.

Мы будем обозначать отображение линейных входных данных в среднее выхода $\mu_n = \ell^{-1}(\eta_n)$, где функция ℓ называется **функцией связи**, а ℓ^{-1} – **функцией среднего**.

На основе результатов из раздела 3.4.3 можно показать, что среднее и дисперсия выходной переменной равны:

¹ Строго говоря, в GLM используется небольшое обобщение естественного экспоненциального семейства, называемое **экспоненциальным семейством рассеяния** (exponential dispersion family). Для скалярной переменной оно имеет вид $p(y|\eta, \sigma^2) = h(y, \sigma^2) \exp[(\eta y - A(\eta))/\sigma^2]$. Здесь σ^2 называется **параметром рассеяния**. При фиксированном σ^2 это естественное экспоненциальное семейство.

$$\mathbb{E}[y_n|\mathbf{x}_n, \mathbf{w}, \sigma^2] = A'(\eta_n) \triangleq \ell^{-1}(\eta_n): \quad (12.2)$$

$$\mathbb{V}[y_n|\mathbf{x}_n, \mathbf{w}, \sigma^2] = A''(\eta_n)\sigma^2. \quad (12.3)$$

12.2. ПРИМЕРЫ

В этом разделе мы приведем примеры широко используемых GLM.

12.2.1. Линейная регрессия

Напомним, что линейная регрессия имеет вид:

$$p(y_n|\mathbf{x}_n, \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_n - \mathbf{w}^\top \mathbf{x}_n)^2\right). \quad (12.4)$$

Отсюда

$$\log p(y_n|\mathbf{x}_n, \mathbf{w}, \sigma^2) = -\frac{1}{2\sigma^2}(y_n - \eta_n)^2 - \frac{1}{2}\log(2\pi\sigma^2), \quad (12.5)$$

где $\eta_n = \mathbf{w}^\top \mathbf{x}_n$. Это можно записать в форме GLM следующим образом:

$$\log p(y_n|\mathbf{x}_n, \mathbf{w}, \sigma^2) = \frac{y_n\eta_n - \frac{\eta_n^2}{2}}{\sigma^2} - \frac{1}{2}\left(\frac{y_n^2}{\sigma^2} + \log(2\pi\sigma^2)\right). \quad (12.6)$$

Как видим, $A(\eta_n) = \eta_n^2/2$, откуда

$$\mathbb{E}[y_n] = \eta_n = \mathbf{w}^\top \mathbf{x}_n; \quad (12.7)$$

$$\mathbb{V}[y_n] = \sigma^2. \quad (12.8)$$

12.2.2. Биномиальная регрессия

Если выходная переменная – количество успехов в N_n испытаниях, $y_n \in \{0, \dots, N_n\}$, то можно использовать **биномиальную регрессию**, определяемую следующим образом:

$$p(y_n|\mathbf{x}_n, N_n, \mathbf{w}) = \text{Bin}(y_n|\sigma(\mathbf{w}^\top \mathbf{x}_n), N_n). \quad (12.9)$$

Мы видим, что бинарная логистическая регрессия – частный случай, соответствующий $N_n = 1$.

Логарифм плотности распределения вероятностей имеет вид:

$$\log p(y_n|\mathbf{x}_n, N_n, \mathbf{w}) = y_n \log \mu_n + (N_n - y_n) \log(1 - \mu_n) + \log \binom{N_n}{y_n} \quad (12.10)$$

$$= y_n \log \left(\frac{\mu_n}{1 - \mu_n} \right) + N_n \log(1 - \mu_n) + \log \left(\frac{N_n}{y_n} \right), \quad (12.11)$$

где $\mu_n = \sigma(\eta_n)$. Определим

$$\eta_n \triangleq \log \left(\frac{\mu_n}{1 - \mu_n} \right) = \log \left[\frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}_n}} \frac{1 + e^{-\mathbf{w}^\top \mathbf{x}_n}}{e^{-\mathbf{w}^\top \mathbf{x}_n}} \right] = \log \frac{1}{e^{-\mathbf{w}^\top \mathbf{x}_n}} = \mathbf{w}^\top \mathbf{x}_n. \quad (12.12)$$

Следовательно, биномиальная регрессия записывается в форме GLM следующим образом:

$$\log p(y_n | \mathbf{x}_n, N_n, \mathbf{w}) = y_n \eta_n - A(\eta_n) + h(y_n), \quad (12.13)$$

где $h(y_n) = \log \left(\frac{N_n}{y_n} \right)$ и

$$A(\eta_n) = -N_n \log(1 - \mu_n) = N_n \log(1 + e^{\eta_n}). \quad (12.14)$$

Отсюда

$$\mathbb{E}[y_n] = \frac{dA}{d\eta_n} = \frac{N_n e^{\eta_n}}{1 + e^{\eta_n}} = \frac{N_n}{1 + e^{-\eta_n}} = N_n \mu_n \quad (12.15)$$

и

$$\mathbb{V}[y_n] = \frac{d^2 A}{d\eta_n^2} = N_n \mu_n (1 - \mu_n). \quad (12.16)$$

12.2.3. Регрессия Пуассона

Если выходная переменная – целочисленный счетчик, $y_n \in \{0, 1, \dots\}$, то можно воспользоваться **регрессией Пуассона**, определяемой следующим образом:

$$p(y_n | \mathbf{x}_n, \mathbf{w}) = \text{Poi}(y_n | \exp(\mathbf{w}^\top \mathbf{x}_n)), \quad (12.17)$$

где

$$\text{Poi}(y | \mu) = e^{-\mu} \frac{\mu^y}{y!} \quad (12.18)$$

– распределение Пуассона. Регрессия Пуассона широко используется в биостатистике, где y_n может представлять количество заболеваний у данного человека или в данной местности либо количество чтений некоторой позиции в геноме в контексте высокопроизводительного секвенирования (см., например, [Kua+09]).

Логарифм плотности распределения вероятностей имеет вид:

$$\log p(y_n | \mathbf{x}_n, \mathbf{w}) = y_n \log \mu_n - \mu_n - \log(y_n!), \quad (12.19)$$

где $\mu_n = \exp(\mathbf{w}^T \mathbf{x}_n)$. В форме GLM имеем

$$\log p(y_n | \mathbf{x}_n, \mathbf{w}) = y_n \eta_n - A(\eta_n) + h(y_n), \quad (12.20)$$

где $\eta_n = \log(\mu_n) = \mathbf{w}^T \mathbf{x}_n$, $A(\eta_n) = \mu_n = e^{\eta_n}$ и $h(y_n) = -\log(y_n!)$. Отсюда

$$\mathbb{E}[y_n] = \frac{dA}{d\eta_n} = e^{\eta_n} = \mu_n \quad (12.21)$$

и

$$\mathbb{V}[y_n] = \frac{d^2 A}{d\eta_n^2} = e^{\eta_n} = \mu_n. \quad (12.22)$$

12.3. GLM с НЕКАНОНИЧЕСКИМИ ФУНКЦИЯМИ СВЯЗИ

Мы видели, что средние параметры выходного распределения описываются формулой $\mu = \ell^{-1}(\eta)$, где ℓ – функция связи. Эту функцию можно выбрать несколькими способами, которые мы сейчас и обсудим.

Каноническая функция связи ℓ обладает тем свойством, что $\theta = \ell(\mu)$, где θ – канонические (естественные) параметры. Отсюда

$$\theta = \ell(\mu) = \ell(\ell^{-1}(\eta)) = \eta. \quad (12.23)$$

Это то, что мы предполагали до сих пор. Например, для распределения Бернулли канонический параметр – логарифм отношения шансов $\theta = \log(\mu/(1 - \mu))$, который возвращается функцией **logit**:

$$\theta = \ell(\mu) = \text{logit}(\mu) = \log\left(\frac{\mu}{1 - \mu}\right). \quad (12.24)$$

Обратной к этой функции является сигмоида, или логистическая функция $\mu = \sigma(\theta) = 1/(1 + e^{-\theta})$.

Однако никто не запрещает использовать другие функции связи. Например, **пробит-функция связи** имеет вид:

$$\eta = \ell(\mu) = \Phi^{-1}(\mu). \quad (12.25)$$

Еще одна функция связи, которую иногда используют для бинарных выходов, – **дополнительный двойной логарифм** (complementary log-log):

$$\eta = \ell(\mu) = \log(-\log(1 - \mu)). \quad (12.26)$$

Она применяется в приложениях, где мы либо наблюдаем 0 событий (обозначается $y = 0$) или одно и более (обозначается $y = 1$), а появление события описывается распределением Пуассона с коэффициентом λ . Пусть E – число

событий. Предположение Пуассона означает, что $p(E = 0) = \exp(-\lambda)$ и, следовательно,

$$p(y = 0) = (1 - \mu) = p(E = 0) = \exp(-\lambda). \quad (12.27)$$

Таким образом, $\lambda = -\log(1 - \mu)$. Если λ является функцией ковариат, то нужно гарантировать ее положительность, поэтому мы полагаем $\lambda = e^\eta$, откуда

$$\eta = \log(\lambda) = \log(-\log(1 - \mu)). \quad (12.28)$$

12.4. ОЦЕНКА МАКСИМАЛЬНОГО ПРАВДОПОДОБИЯ

Для обучения GLM можно использовать методы, похожие на те, что мы применяли для обучения модели логистической регрессии. В частности, отрицательное логарифмическое правдоподобие имеет вид (постоянные члены игнорируются):

$$\text{NLL}(\mathbf{w}) = -\log p(\mathcal{D}|\mathbf{w}) = -\frac{1}{\sigma^2} \sum_{n=1}^N \ell_n, \quad (12.29)$$

где

$$\ell_n \triangleq \eta_n y_n - A(\eta_n), \quad (12.30)$$

где $\eta_n = \mathbf{w}^\top \mathbf{x}_n$. Чтобы упростить обозначения, будем предполагать, что $\sigma^2 = 1$. Градиент одного члена вычисляется следующим образом:

$$\mathbf{g}_n \triangleq \frac{\partial \ell_n}{\partial \mathbf{w}} = \frac{\partial \ell_n}{\partial \eta_n} \frac{\partial \eta_n}{\partial \mathbf{w}} = (y_n - A'(\eta_n)) \mathbf{x}_n = (y_n - \mu_n) \mathbf{x}_n, \quad (12.31)$$

где $\mu_n = f(\mathbf{w}^\top \mathbf{x}_n)$, а f – обратная функция связи, которая отображает канонические параметры в средние. Например, в случае логистической регрессии $f(\eta_n) = \sigma(\eta_n)$, поэтому мы возвращаемся к формуле (10.21). Это выражение градиента можно очевидным образом использовать в методе СГС или любом другом градиентном методе.

Гессиян имеет вид:

$$\mathbf{H} = \frac{\partial^2}{\partial \mathbf{w} \partial \mathbf{w}^\top} \text{NLL}(\mathbf{w}) = -\sum_{n=1}^N \frac{\partial \mathbf{g}_n}{\partial \mathbf{w}^\top}, \quad (12.32)$$

где

$$\frac{\partial \mathbf{g}_n}{\partial \mathbf{w}^\top} = \frac{\partial \mathbf{g}_n}{\partial \mu_n} \frac{\partial \mu_n}{\partial \mathbf{w}^\top} = -\mathbf{x}_n f'(\mathbf{w}^\top \mathbf{x}_n) \mathbf{x}_n^\top. \quad (12.33)$$

Отсюда

$$\mathbf{H} = \sum_{n=1}^N f'(\eta_n) \mathbf{x}_n \mathbf{x}_n^\top. \quad (12.34)$$

Например, в случае логистической регрессии $f(\eta_n) = \sigma(\eta_n)$ и $f'(\eta_n) = \sigma(\eta_n)(1 - \sigma(\eta_n))$, так что мы возвращаемся к формуле (10.23). В общем случае мы видим, что гессиан положительно определен, так как $f'(\eta_n) > 0$; следовательно, функция отрицательного логарифмического правдоподобия выпукла, поэтому MLE для GLM единственна (в предположении, что $f(\eta_n) > 0$ для всех n).

На основании вышеизложенных результатов мы можем обучить GLM, применяя градиентные методы, – очень похоже на то, как обучали модели логистической регрессии.

12.5. РАБОЧИЙ ПРИМЕР: ПРЕДСКАЗАНИЕ ОБРАЩЕНИЙ ЗА СТРАХОВЫМИ ВЫПЛАТАМИ

В этом разделе мы приведем пример предсказания обращений за страховыми выплатами с применением линейной регрессии и регрессии Пуассона¹. Цель состоит в том, чтобы предсказать ожидаемое ежегодное количество обращений за выплатами после ДТП. Набор данных включает 678 000 примеров с 9 признаками, в том числе возраст водителя, возраст транспортного средства, мощность двигателя и т. д. Предсказать требуется частоту обращений, т. е. количество обращений по договору страхования, деленное на экспозицию (т. е. срок договора в годах).

На рис. 12.1а показан тестовый набор. Мы видим, что для 94 % договоров никаких обращений не было, поэтому данные содержат много нулей, что типично для данных о счетчиках и частотах. Средняя частота обращений равна 10 %. Это можно преобразовать в фиктивную модель, которая всегда предсказывает такую постоянную. Результат подобного предсказания показан на рис. 12.1б. Но хотелось бы добиться лучшего.

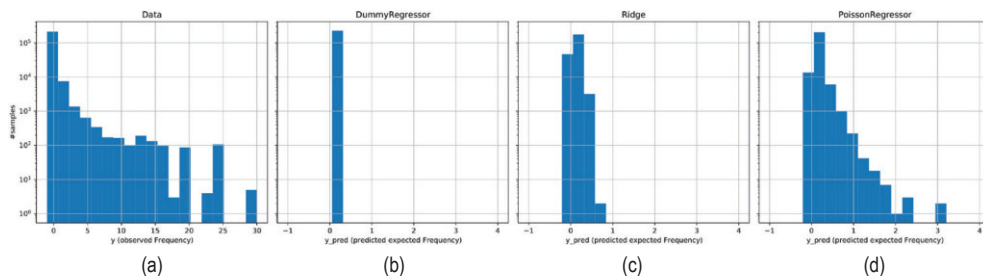


Рис. 12.1. Предсказания частоты обращений за страховыми выплатами на тестовом наборе. (а) Данные. (б) Постоянный предиктор. (с) Линейная регрессия. (д) Регрессия Пуассона. Построено программой по адресу figures.probml.ai/book1/12.1

¹ Пример взят из https://scikit-learn.org/stable/auto_examples/linear_model/plot_poisson_regression_non_normal_loss.html.

Можно поступить по-простому – использовать линейную регрессию в сочетании с каким-нибудь простым методом конструирования признаков (распределением по интервалам для непрерывных значений или унитарным кодированием для категориальных). (Мы добавили слабую ℓ_2 -регуляризацию, так что, строго говоря, это гребневая регрессия.) Результат показан на рис. 12.1с. Лучше, чем эталон, но все равно не очень хорошо. В частности, эта модель может предсказывать отрицательное значение и не улавливает длинный хвост.

Можно добиться лучшего с помощью регрессии Пуассона, используя те же самые признаки, но логарифмическую функцию связи. Результаты показаны на рис. 12.1d. Как видим, предсказания гораздо лучше.

Возникает интересный вопрос – как количественно охарактеризовать качество модели в такого рода задачах. Если использовать среднеквадратическую ошибку или среднее абсолютное отклонение, то из табл. 12.1 можно сделать вывод, что гребневая регрессия лучше, чем регрессия Пуассона, хотя это, очевидно, неверно, как показывает рис. 12.1. Поэтому качество чаще измеряют с помощью **девиации**, которая определяется следующим образом:

$$D(\mathbf{y}, \hat{\boldsymbol{\mu}}) = 2 \sum_i (\log p(y_i | \mu_i^*) - \log p(y_i | \mu_i)), \quad (12.35)$$

где μ_i – предсказанные параметры для i -го примера (на основе входных признаков \mathbf{x}_i и обучающего набора \mathcal{D}), а μ_i^* – оптимальный параметр, оцененный путем обучения модели только на истинном выходе y_i . (Это так называемая **насыщенная модель**, которая идеально аппроксимирует тестовый набор.) В случае регрессии Пуассона имеем $\mu_i^* = y_i$. Отсюда

$$D(\mathbf{y}, \boldsymbol{\mu}) = 2 \sum_i [(y_i \log y_i - y_i - \log(y_i!)) - (y_i \log \hat{\mu}_i - \hat{\mu}_i - \log(y_i!))] \quad (12.36)$$

$$= 2 \sum_i \left[y_i \log \frac{y_i}{\hat{\mu}_i} + \hat{\mu}_i - y_i \right]. \quad (12.37)$$

Если судить по этому показателю, то модель Пуассона, безусловно, лучше (см. последний столбец в табл. 12.1).

Таблица 12.1. Показатели качества на тестовом наборе.

СКО = среднеквадратическая ошибка. **САО** = среднее абсолютное отклонение.

Девиация = девиация Пуассона

Название	СКО	САО	Девиация
Фиктивная	0.564	0.189	0.625
Гребневая	0.560	0.177	0.601
Пуассона	0.560	0.186	0.594

Мы можем также вычислить **калибровочную кривую**, которая отражает зависимость фактической частоты от предсказанной. Для этого мы разбиваем множество предсказаний на интервалы и подсчитываем эмпирическую частоту обращений для всех примеров, предсказанная частота которых по-

падает в данный интервал. Результаты показаны на рис. 12.2. Мы видим, что постоянная эталонная модель хорошо откалибрована, но, конечно, не очень точна. Гребневая модель плохо откалибрована на низких частотах. В частности, она занижает оценку общего числа обращений в тестовом наборе – 10 693 против истинного числа 11 935. Модель регрессии Пуассона откалибрована лучше (т. е. когда она предсказывает высокую частоту обращений, частота действительно оказывается высокой), а общее число предсказанных ей обращений равно 11 930.

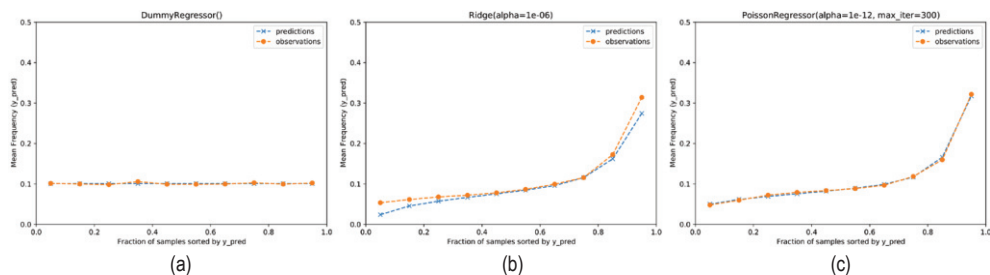


Рис. 12.2 ❖ Калибровочная кривая для предсказания обращений за страховыми выплатами. Построено программой по адресу figures.problml.ai/book1/12.2

Часть III



ГЛУБОКИЕ НЕЙРОННЫЕ СЕТИ

Глава 13

Нейронные сети для структурированных данных

13.1. ВВЕДЕНИЕ

В части II мы обсуждали линейные модели регрессии и классификации. В частности, в главе 10 обсуждалась логистическая регрессия, которой в бинарном случае соответствует модель $p(y|\mathbf{x}, \mathbf{w}) = \text{Ber}(y|\sigma(\mathbf{w}^T\mathbf{x}))$, а в многоклассовом случае – модель $p(y|\mathbf{x}, \mathbf{W}) = \text{Cat}(y|\mathcal{S}(\mathbf{W}\mathbf{x}))$. В главе 11 мы обсуждали линейную регрессию, которой соответствует модель $p(y|\mathbf{x}, \mathbf{w}) = \mathcal{N}(y|\mathbf{w}^T\mathbf{x}, \sigma^2)$. А в главе 12 рассматривались обобщенные линейные модели, которые обобщают вышеупомянутые модели на другие виды выходных распределений, например распределение Пуассона. Однако во всех этих моделях делается сильное предположение о линейности отображения входных данных в выходные.

Простой способ повысить гибкость таких моделей заключается в преобразовании признаков путем замены \mathbf{x} на $\boldsymbol{\phi}(\mathbf{x})$. Например, можно использовать полиномиальное преобразование, которое в одномерном случае имеет вид $\boldsymbol{\phi}(x) = [1, x, x^2, x^3, \dots]$, как обсуждалось в разделе 1.2.2.2. Иногда оно называется **разложением по базисным функциям**. Тогда модель принимает вид:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}\boldsymbol{\phi}(\mathbf{x}) + \mathbf{b}. \quad (13.1)$$

Она все еще линейна относительно параметров $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$, что упрощает обучение (поскольку функция отрицательного логарифмического правдоподобия выпуклая). Однако необходимость вручную задавать преобразование признаков сильно ограничивает наши возможности.

Естественный следующий шаг заключается в том, чтобы снабдить экстрактор признаков собственными параметрами, $\boldsymbol{\theta}_2$:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}\boldsymbol{\phi}(\mathbf{x}; \boldsymbol{\theta}_2) + \mathbf{b}, \quad (13.2)$$

где $\theta = (\theta_1, \theta_2)$ и $\theta_1 = (\mathbf{W}, \mathbf{b})$. Очевидно, этот процесс можно повторять рекурсивно, создавая все более и более сложные функции. Композиция L функций выглядит следующим образом:

$$f(\mathbf{x}; \theta) = f_L(f_{L-1}(\dots(f_1(\mathbf{x})) \dots)), \quad (13.3)$$

где $f_\ell(\mathbf{x}) = f(\mathbf{x}, \theta_\ell)$ – функция слоя ℓ . Именно эта ключевая идея и лежит в основе **глубоких нейронных сетей** (ГНС, англ. DNN).

Термином «ГНС» на самом деле объединяется большое семейство моделей, которые komponуют дифференцируемые функцию в произвольный ориентированный ациклический граф (орграф, англ. DAG), отображающий вход в выход. Формула (13.3) – простейший пример, в котором орграф представляет собой линейную цепочку. Она получила название **нейронной сети прямого распространения** (англ. FFNN), или **многослойного перцептрона** (МСП, англ. MLP).

МСП предполагает, что на вход подается вектор фиксированной размерности, скажем $\mathbf{x} \in \mathbb{R}^D$. Такие данные принято называть **структурированными**, или **табличными**, поскольку они часто хранятся в виде матрицы плана размера $N \times D$, каждый столбец (признак) которой имеет свою семантику, например: рост, вес, возраст и т. д. В последующих главах мы обсудим другие виды ГНС, более подходящие для представления «**неструктурированных данных**», например изображений и текста, когда размер данных переменный, а каждый отдельный элемент (например, пиксель или слово) сам по себе не несет никакого смысла¹. В частности, в главе 14 обсуждаются **сверточные нейронные сети** (СНС, англ. CNN), предназначенные для работы с изображениями, а в главе 15 – **рекуррентные нейронные сети** (РНС, англ. RNN) и **трансформеры**, предназначенные для работы с последовательностями. Наконец, в главе 23 обсуждаются **графовые нейронные сети** (graph neural networks – GNN) для работы с графами.

Хотя ГНС могут показывать хорошие результаты, обычно для достижения высокого качества нужно позаботиться о многочисленных технических деталях. Некоторые из них обсуждаются в дополнительном материале к книге на сайте probml.ai. Есть и другие книги, где эта тема рассматривается более подробно (например, [Zha+20; Cho21; Gér19; GBC16]), а также разнообразные онлайн-курсы. Читателям, интересующимся теорией, рекомендуем, например, [Ber+21; Cal20; al99].

¹ Термин «неструктурированные данные» не вполне корректен, потому что и изображения, и текст имеют структуру. Например, соседние пиксели изображения сильно коррелированы, как и соседние слова в тексте. На самом деле именно эта структура и используется в сверточных (СНС) и рекуррентных (РНС) нейронных сетях. С другой стороны, МСП не делает никаких предположений о входных данных. Это полезно в приложениях, основанных на табличных данных, где структура (зависимости между столбцами) обычно не очевидна, поэтому для ее выявления необходимо обучение. МСП можно применить к изображениям и тексту, и мы скоро это увидим, но качество обычно получается хуже по сравнению со специализированными моделями типа СНС и РНС. (Есть, правда, исключения, например неструктурированная модель **MLP-mixer** [Tol+21], которую можно обучить работе с изображениями и текстовыми данными, но, чтобы компенсировать отсутствие индуктивного смещения, таким моделям нужно очень много данных.)

13.2. Многослойные перцептроны (МСП)

В разделе 10.2.5 мы объяснили, что перцептрон – это детерминированный вариант логистической регрессии. Точнее, это отображение вида

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbb{I}(\mathbf{w}^T \mathbf{x} + \mathbf{b} \geq 0) = H(\mathbf{w}^T \mathbf{x} + \mathbf{b}), \quad (13.4)$$

где $H(a)$ – **ступенчатая функция Хевисайда**, называемая также **линейной пороговой функцией**. Поскольку решающие границы, представленные перцептроном, линейны, их возможности крайне ограничены. В 1969 году Марвин Мински и Сеймур Пейперт опубликовали знаменитую книгу «Перцептроны» [MP69], в которой привели много примеров задач распознавания образов, которые перцептроны решить не в состоянии. Ниже приведен один такой пример, после чего мы обсудим, как все-таки решить задачу.

13.2.1. Задача XOR

Один из самых знаменитых примеров из книги «Перцептроны» – **задача XOR**. В ней цель – обучить функцию, которая вычисляет ИСКЛЮЧАЮЩЕЕ ИЛИ двух своих двоичных входов. Таблица истинности для этой функции приведена в табл. 13.1. На рис. 13.1а приведена ее визуализация. Ясно, что данные не являются линейно разделимыми, поэтому представить такое отображение перцептроном невозможно.

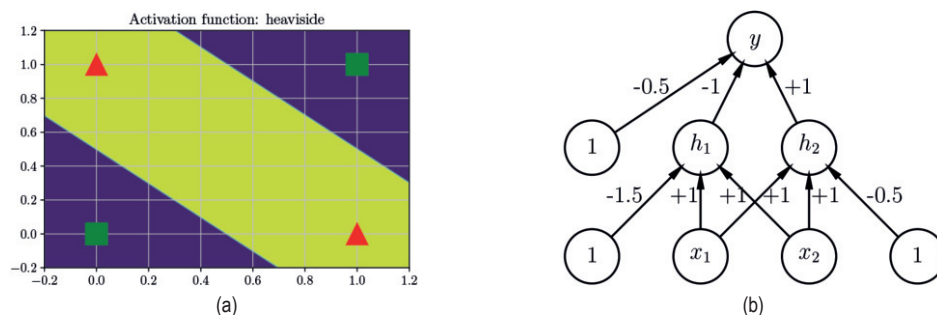


Рис. 13.1 ❖ (а) Иллюстрация того, что функция XOR не является линейно разделимой, но может быть разделена двухслойной моделью с функциями активации Хевисайда. На основе рис. 10.6 из работы [Gér19]. Построено программой по адресу figures.problml.ai/book1/13.1. (б) Нейронная сеть с одним скрытым слоем, веса которой были подобраны вручную для реализации функции XOR. Здесь h_1 – функция AND, а h_2 – функция OR. Члены смещения реализованы с помощью весов из постоянных узлов со значением 1

Однако проблему можно решить, составив перцептроны в стопку, – так называемый **многослойный перцептрон (МСП)**. Так, для решения задачи XOR годится МСП, показанный на рис. 13.1б. Он состоит из трех перцептронов, обозначенных h_1 , h_2 и y . Узлы, обозначенные буквой x , – входы, а узлы

с цифрой 1 представляют постоянные члены. Узлы h_1 и h_3 называются **скрытыми блоками**, потому что их значения не присутствуют среди обучающих данных.

Таблица 13.1. Таблица истинности для функции XOR (ИСКЛЮЧАЮЩЕЕ ИЛИ), $y = x_1 \vee x_2$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Первый скрытый блок вычисляет $h_1 = x_1 \wedge x_2$, используя специально подобранный набор весов (здесь \wedge обозначает операцию AND). Именно, его входам x_1 и x_2 сопоставлены веса 1.0, но имеется еще член смещения -1.5 (он реализован «проводом» с весом -1.5 , исходящим из фиктивного узла с фиксированным значением 1). Таким образом, h_1 активируется тогда и только тогда, когда x_1 и x_2 включены, так как в этом случае

$$\mathbf{w}_1^T \mathbf{x} - b_1 = [1.0, 1.0]^T [1, 1] - 1.5 = 0.5 > 0. \quad (13.5)$$

Второй скрытый блок вычисляет $h_2 = x_1 \vee x_2$, где \vee обозначает операцию OR, а третий – окончательный результат $y = \bar{h}_1 \wedge h_2$, где $\bar{h} = -h$ – операция NOT (логическое отрицание). Таким образом,

$$y = f(x_1, x_2) = \overline{(x_1 \wedge x_2)} \wedge (x_1 \vee x_2). \quad (13.6)$$

Это выражение эквивалентно функции XOR.

Обобщая этот пример, можно показать, что МСП способен представить любую логическую функцию. Однако мы, очевидно, хотели бы избежать задания весов и смещений вручную. Далее в этой главе мы поговорим о том, как обучать эти параметры на основе данных.

13.2.2. Дифференцируемые МСП

МСП, рассмотренный в разделе 13.2.1, был определен как стек перцептронов, включающий недифференцируемую функцию Хевисайда. Такие модели трудно обучать, поэтому они никогда широко не использовались. Но допустим, что мы заменили функцию Хевисайда $H: \mathbb{R} \rightarrow \{0, 1\}$ дифференцируемой **функцией активации** $\varphi: \mathbb{R} \rightarrow \mathbb{R}$. Точнее, определим скрытые блоки \mathbf{z}_l на каждом уровне l как линейное преобразование скрытых блоков предыдущего уровня, применяемое поэлементно с помощью этой функции активации:

$$\mathbf{z}_l = f_l(\mathbf{z}_{l-1}) = \varphi_l(\mathbf{b}_l + \mathbf{W}_l \mathbf{z}_{l-1}) \quad (13.7)$$

или в скалярной форме:

$$z_{kl} = \varphi_l \left(b_{kl} + \sum_{j=1}^{K_{l-1}} w_{jkl} z_{jl-1} \right). \quad (13.8)$$

Величина, передаваемая функции активации, называется **преактивацией**:

$$a_l = b_l + \mathbf{W}_l \mathbf{z}_{l-1}, \quad (13.9)$$

так что $\mathbf{z}_l = \varphi_l(\mathbf{a}_l)$.

Если теперь составить композицию L этих функций, как в формуле (13.3), то мы сможем вычислить градиент выхода по параметрам в каждом слое с помощью правила дифференцирования сложной функции, что в этом контексте называют также **обратным распространением**, как будет объяснено в разделе 13.3. (Это справедливо для любой дифференцируемой функции активации, хотя одни работают лучше, чем другие, как обсуждается в разделе 13.2.3.) Затем градиент можно передать оптимизатору и таким образом минимизировать некоторую целевую функцию (см. раздел 13.4). По этой причине термином «МСП» иногда называют эту дифференцируемую форму модели, а не исторически появившуюся раньше недифференцируемую версию со ступенчатыми блоками.

13.2.3. Функции активации

Мы вправе выбрать любую дифференцируемую функцию активации в каждом слое. Однако если использовать линейную функцию активации $\varphi_l(a) = c_l a$, то все сводится к регулярной линейной модели. Чтобы убедиться в этом, заметим, что формула (13.3) принимает вид:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}_L c_L (\mathbf{W}_{L-1} c_{L-1} (\dots (\mathbf{W}_1 \mathbf{x}) \dots)) \propto \mathbf{W}_L \mathbf{W}_{L-1} \dots \mathbf{W}_1 \mathbf{x} = \mathbf{W}' \mathbf{x}, \quad (13.10)$$

где для простоты обозначений члены смещения опущены. Поэтому важно, чтобы функции активации были нелинейны.

На заре развития нейронных сетей часто выбирали сигмоидную (логистическую) функцию, которую можно рассматривать как гладкую аппроксимацию функции Хевисайда, применяемой в перцептроне:

$$\sigma(a) = \frac{1}{1 + e^{-a}}. \quad (13.11)$$

Однако, как показывает рис. 13.2а, сигмоидная функция **насыщается** (выходит на асимптоту), принимая значения, близкие к 1 для больших положительных входов и близкие к 0 для больших отрицательных входов. Еще одна популярная функция – гиперболический тангенс \tanh , который имеет похожую форму, но асимптотические значения равны -1 и $+1$ (см. рис. 13.2б).

В областях насыщения градиент выхода по входу близок к нулю, поэтому любой сигнал градиента с верхних уровней не сможет распространиться на предшествующие уровни. Из-за этой **проблемы исчезающего градиента**

трудно обучить модели методом градиентного спуска (детали см. в разделе 13.4.2). Один из ключей к обучению очень глубоких моделей – использовать ненасыщаемые функции активации. Самой популярной из них является **блок линейной ректификации** (rectified linear unit – **ReLU**), предложенный в работах [GBB11; KSH12]. Он определяется следующим образом:

$$\text{ReLU}(a) = \max(a, 0) = a\mathbb{I}(a > 0). \quad (13.12)$$

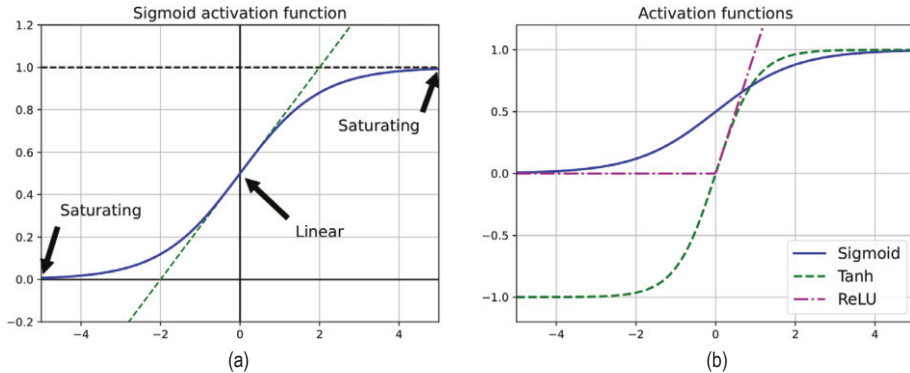


Рис. 13.2 ❖ (а) Сигмоидная функция линейна в окрестности 0, но насыщается для больших положительных и отрицательных входов. На основе рис. 11.1 из [Gér19]. (б) Графики некоторых функций активации нейронной сети. Построено программой по адресу figures.probml.ai/book1/13.2

Функция ReLU просто «выключает» отрицательные входы, а положительные передает без изменения. Ее график показан на рис. 13.2b, а дополнительные сведения приведены в разделе 13.4.3.

13.2.4. Примеры моделей

МСП можно использовать для классификации и регрессии данных самых разных типов. Ниже приведено несколько примеров.

13.2.4.1. МСП для классификации двумерных данных по двум категориям

На рис. 13.3 показан МСП с двумя скрытыми слоями, применяемый к вектору двумерных входов, которые соответствуют точкам на плоскости, принадлежащим двум concentric circles. Модель имеет вид:

$$p(y|\mathbf{x}; \boldsymbol{\theta}) = \text{Ber}(y|\sigma(a_3)); \quad (13.13)$$

$$a_3 = \mathbf{w}_3^T \mathbf{z}_2 + b_3; \quad (13.14)$$

$$\mathbf{z}_2 = \varphi(\mathbf{W}_2 \mathbf{z}_1 + \mathbf{b}_2); \quad (13.15)$$

$$\mathbf{z}_1 = \varphi(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1). \quad (13.16)$$

Здесь a_3 – финальная оценка логита, которая преобразуется в вероятность сигмоидной (логистической) функцией. Значение a_3 вычислено как линейная комбинация двух скрытых блоков в слое 2 по формуле $a_3 = \mathbf{w}_3^T \mathbf{z}_2 + b_3$. В свою очередь слой 2 вычисляется как нелинейная комбинация четырех скрытых блоков слоя 1 по формуле $\mathbf{z}_2 = \varphi(\mathbf{W}_2 \mathbf{z}_1 + \mathbf{b}_2)$. Наконец, слой 1 вычисляется как нелинейная комбинация трех входных блоков по формуле $\mathbf{z}_1 = \varphi(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$. Подбирая параметры $\theta = (\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \mathbf{w}_3, b_3)$ с целью минимизации отрицательного логарифмического правдоподобия, мы можем очень хорошо аппроксимировать обучающие данные, несмотря на сильную нелинейность решающей границы. (Интерактивный вариант этого рисунка имеется на странице <http://playground.tensorflow.org>.)

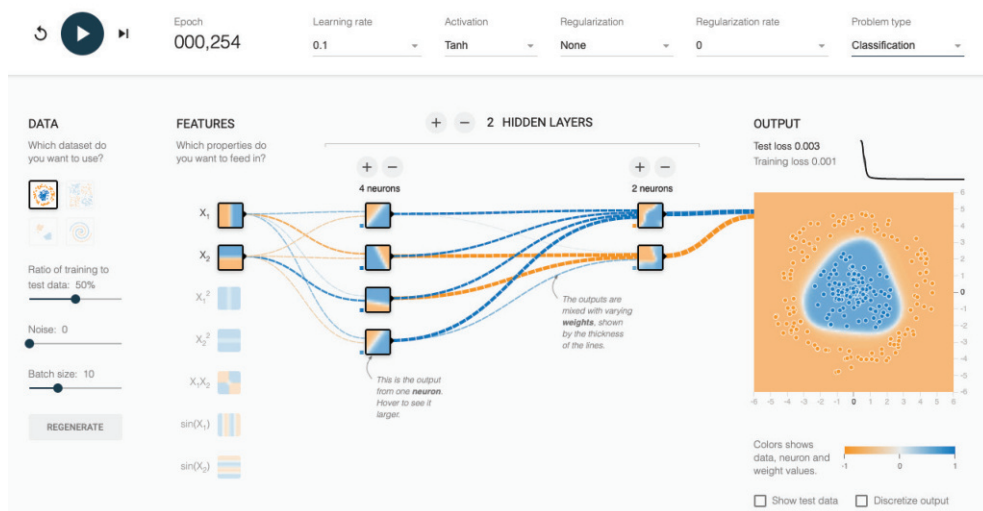


Рис. 13.3 ❖ МСП с двумя скрытыми слоями, применяемый к набору двумерных точек из 2 классов, показанному в левом верхнем углу. Для каждого скрытого блока показана решающая граница в этой части сети. Окончательный выход показан справа. На вход подаются данные $\mathbf{x} \in \mathbb{R}^2$, функциям активации первого слоя – $\mathbf{z}_1 \in \mathbb{R}^4$, функциям активации второго слоя – $\mathbf{z}_2 \in \mathbb{R}^2$, а функция последнего слоя получает данные $a_3 \in \mathbb{R}$, которые преобразуются в вероятности сигмоидной функцией. Этот снимок экрана взят из интерактивной демонстрации на сайте <http://playground.tensorflow.org>

13.2.4.2. МСП для классификации изображений

Чтобы применить МСП к классификации изображений, мы должны «распрямить» (flatten) двумерные входные данные в одномерный вектор. Затем можно будет использовать архитектуру прямого распространения типа той, что была описана в разделе 13.2.4.1. Например, рассмотрим МСП для классификации цифр из набора данных MNIST (раздел 3.5.2). Это $28 \times 28 =$

784-мерный набор. Если взять два скрытых слоя по 128 блоков каждый, за которыми следует последний 10-путевой слой softmax, то получится модель, показанная в табл. 13.2.

Таблица 13.2. Структура МСП для классификации набора данных MNIST.

Заметим, что $100\,480 = (784 + 1) \times 128$ и $16\,512 = (128 + 1) \times 128$.

Построено программой `mlp_mnist_tf.ipynb`

Модель: «sequential»

Слой (тип)	Форма выхода	Параметров
flatten (Распрямление)	(None, 784)	0
dense (Плотный)	(None, 128)	100480
dense_1 (Плотный)	(None, 128)	16512
dense_2 (Плотный)	(None, 10)	1290

Всего параметров: 118 282
Обучаемых параметров: 118 282
Необучаемых параметров: 0

На рис. 13.4 показано несколько предсказаний, сделанных этой моделью. Мы обучали ее только на протяжении двух «эпох» (проходов по обучающему набору), но модель уже работает вполне неплохо, показывая на тестовом наборе верность 97.1 %. Да и ошибки не чудовищные, например 9 принято за 3. Еще несколько эпох обучения – и верность на тестовом наборе улучшилась бы.

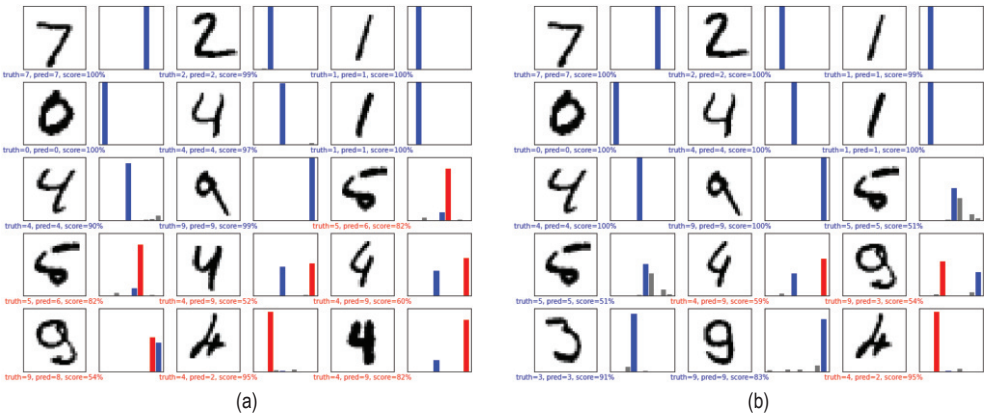


Рис. 13.4 ❖ Результат применения МСП (с двумя скрытыми слоями по 128 блоков и одним выходным слоем с 10 блоками) к нескольким изображениям из набора MNIST (специально выбранным, чтобы продемонстрировать некоторые ошибки). Красным цветом обозначены ошибки, синим – правильные предсказания. (a) После одной эпохи обучения. (b) После двух эпох. Построено программой по адресу figures.probml.ai/book1/13.4

В главе 14 мы обсудим другой вид модели – сверточную нейронную сеть, которая лучше подходит для изображений. Она дает еще лучшее качество

и нуждается в меньшем числе параметров, но зато использует априорную информацию о пространственной структуре изображений. Напротив, при работе с МСП мы можем случайным образом перетасовывать пиксели, и это никак не скажется на выходе (при условии, что все входные данные перетасовываются одинаково).

13.2.4.3. МСП для классификации текстов

Чтобы применить МСП к классификации текстов, нам понадобится преобразовать последовательность слов переменной длины v_1, \dots, v_T (где каждый v_t – унитарный вектор длины V , а V – размер словаря) в вектор x фиксированной размерности. Проще всего сделать это следующим образом. Сначала рассмотрим вход как неупорядоченный мешок слов (раздел 1.5.4.1), $\{v_t\}$. Первый слой модели – матрица погружений W_1 размера $E \times V$, которая преобразует любой разреженный V -мерный вектор в плотное E -мерное погружение, $e_t = W_1 v_t$ (подробнее о погружениях слов см. раздел 20.5). Затем мы преобразуем это множество T E -мерных погружений в вектор фиксированной длины, применяя **пулинг глобальным усреднением**, $\bar{e} = (1/T) \sum_{t=1}^T e_t$. Результат уже можно передать на вход МСП. Например, если используется один скрытый слой и логистический выход (для бинарной классификации), то получаем:

$$p(y|x; \theta) = \text{Ber}(y|\sigma(w^T_3 h + b_3)), \quad (13.17)$$

$$h = \varphi(W_2 \bar{e} + b_2); \quad (13.18)$$

$$\bar{e} = \frac{1}{T} \sum_{t=1}^T e_t; \quad (13.19)$$

$$e_t = W_1 v_t. \quad (13.20)$$

Если размер словаря $V = 1000$, размер погружения $E = 16$, а скрытый слой содержит 16 блоков, то мы получаем модель, показанную в табл. 13.3. Применив ее к набору данных IMDB, содержащему отзывы о фильмах, с целью классификации по эмоциональной окраске (см. раздел 1.5.2.1), мы получим верность 86 % на тестовом наборе.

Из табл. 13.3 видно, что число параметров модели очень велико, что может привести к переобучению, поскольку в обучающем наборе IMDB всего 25 000 примеров. Однако видно и то, что большинство параметров принадлежит матрице погружений, поэтому, вместо того чтобы обучать ее с учителем, можно выполнить предварительное обучение моделей погружения слов без учителя, как обсуждается в разделе 20.5. Если матрица погружений W_1 фиксирована, то нужно будет настроить только параметры в слоях 2 и 3 для этой конкретной задачи с помеченными примерами, что требует гораздо меньше данных (см. также главу 19, где описаны общие методы обучения при ограниченном объеме помеченных данных).

Таблица 13.3. Структура МСП для классификации обзоров из набора данных IMDB. Размер словаря $V = 1000$, размер погружения $E = 16$, число блоков в скрытом слое 16. Размер матрицы погружений W_1 равен 10000×16 , а длина смещения b_2 равна 16 (отметим, что $16 \times 16 + 16 = 272$), в последнем слое (обозначенном «dense_1») вектор весов w_3 имеет длину 16, а смещение b_3 – длину 1. Слой глобального пулинга усреднением не имеет свободных параметров. Построено программой `mlp_imdb_tf.ipynb`

Модель: «sequential»

Слой (тип)	Форма выхода	Параметров
embedding (Погружение)	(None, (None, 16))	160 000
global_average_pooling1d (Глобальный пулинг усреднением)	(None, 16)	0
dense (Плотный)	(None, 16)	272
dense_1 (Плотный)	(None, 1)	17

Всего параметров: 169 289
Обучаемых параметров: 169 289
Необучаемых параметров: 0

13.2.4.4. МСП для гетероскедастической регрессии

Мы также можем использовать МСП для регрессии. На рис. 13.5 показано, как создать модель для гетероскедастической нелинейной регрессии. (Термин «гетероскедастический» означает просто, что дисперсия предсказанного выхода зависит от входных данных, см. раздел 2.6.3.) У этой функции два выхода: $f_\mu(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}, \boldsymbol{\theta}]$ и $f_\sigma(\mathbf{x}) = \sqrt{\mathbb{V}[y|\mathbf{x}, \boldsymbol{\theta}]}$. Мы можем разделить большую часть слоев (а значит, и параметров) между этими двумя функциями, воспользовавшись общим «скелетом» и двумя выходными «головами», как показано на рис. 13.5. Для головы μ используется линейная функция активации, $\varphi(a) = a$, а для головы σ – функция softplus, $\varphi(a) = \sigma_+(a) = \log(1 + e^a)$. Если используются линейные головы и нелинейный скелет, модель в целом имеет вид:

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{w}_\mu^T f(\mathbf{x}; \mathbf{w}_{\text{shared}}), \sigma_+(\mathbf{w}_\sigma^T f(\mathbf{x}; \mathbf{w}_{\text{shared}}))). \tag{13.21}$$

На рис. 13.6 показано преимущество модели такого вида на наборе данных, среднее которого растет со временем линейно, с сезонными колебаниями, а дисперсия возрастает квадратично. Это простой пример **модели стохастической волатильности**; ее можно использовать для моделирования финансовых данных, а также глобальной температуры Земли, для которой (в связи с изменением климата) среднее и дисперсия возрастают. Мы видим, что модель регрессии, в которой выходная дисперсия σ^2 рассматривается как фиксированный (не зависящий от входных данных) параметр, иногда оказывается недостаточно уверенной, потому что вынуждена подстраиваться под общий уровень шума и не может адаптироваться к уровню шума в каждой точке пространства входов.

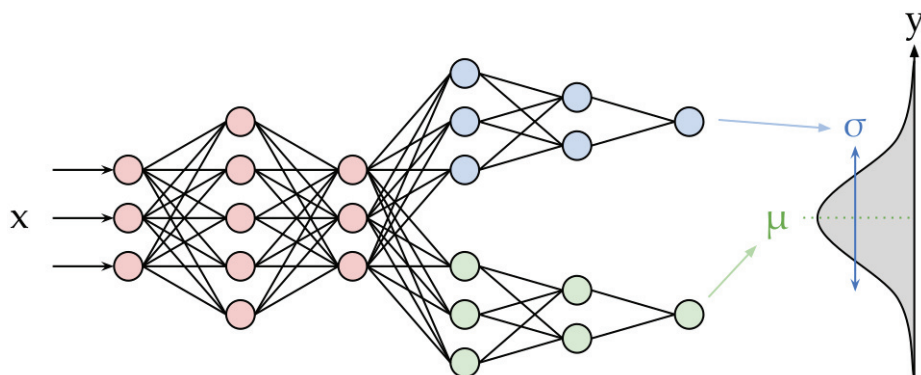


Рис. 13.5 ❖ МСП с разделяемым «скелетом» и kdevz выходными «головками», одна из которых предсказывает среднее, а другая дисперсию. Взято со страницы <https://brendanhasz.github.io/2019/07/23/bayesian-density-net.html>. Печатается с разрешением Брендана Хасса

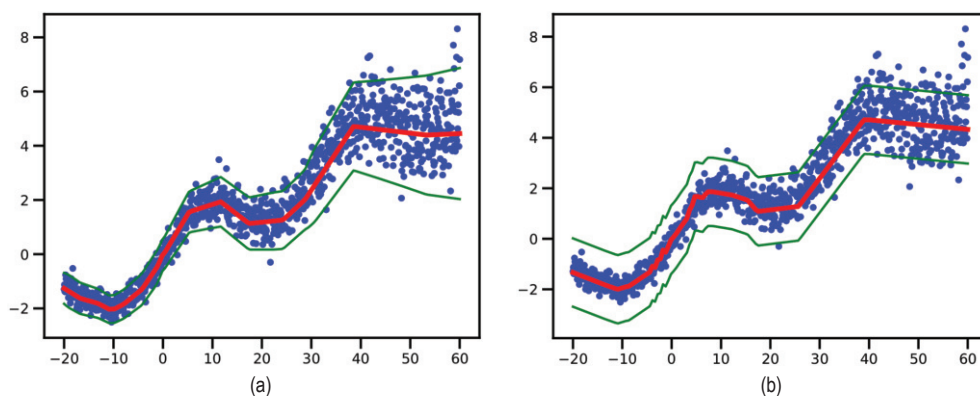


Рис. 13.6 ❖ Предсказания МСП, обученного с использованием MLE на одномерном наборе данных регрессии с возрастающим шумом. (а) Выходная дисперсия не зависит от входных данных, как на рис. 13.5. (б) Среднее вычисляется с помощью той же модели, что в (а), но выходная дисперсия рассматривается как фиксированный параметр σ^2 , для которого вычисляется оценка MLE после обучения, как в разделе 11.2.3.6. Построено программой по адресу figures.problml.ai/book1/13.6

13.2.5. Важность глубины

Можно показать, что МСП с одним скрытым слоем является **универсальным аппроксиматором функций**, т. е. он способен смоделировать любую достаточно гладкую функцию с любой точностью при наличии достаточного числа скрытых блоков [HSW89; Sub89; Hor91]. Интуитивно понятно, что причина этого заключается в том, что каждый скрытый блок может описать полуплоскость, а при достаточно большом числе таких плоскостей можно

«вырезать» любую область пространства, с которой мы можем ассоциировать произвольный отклик (это особенно хорошо видно, если используются кусочно-линейные функции активации, как показано на рис. 13.7).

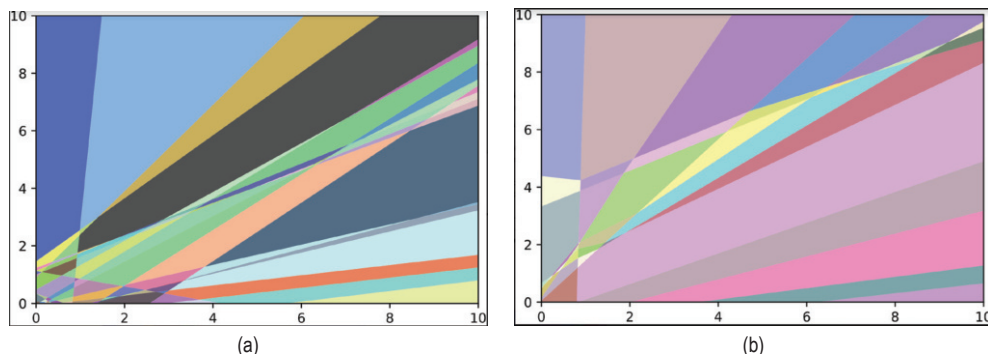


Рис. 13.7 ❖ Разбиение \mathbb{R}^2 на конечное множество линейных решающих регионов, порожденное МСП с ReLU-активациями с (а) одним скрытым слоем с 25 скрытыми блоками и (б) двумя скрытыми слоями. На основе рис. 1 из работы [HAB19]. Печатается с разрешения Максима Андрущенко

Однако существуют аргументы, как экспериментальные, так и теоретические (например, [Has87; Mon+14; Rag+17; Pog+17]), в пользу того, что глубокие сети работают лучше мелких. Причина в том, что последующие слои могут пользоваться признаками, выявленными предыдущими слоями, т. е. мы имеем функцию, определенную в виде **композиции** или **иерархически**. Например, предположим, что мы хотим классифицировать строки, описывающие геном, и с положительным классом ассоциировано регулярное выражение `*AA??CGCG??AA*`. Хотя его можно было бы аппроксимировать моделью с одним скрытым слоем, интуитивно понятно, что обучить модель было бы проще, если бы она сначала обнаруживала «мотивы» AA и CG, используя скрытые блоки в слое 1, а затем использовала эти признаки для определения простого линейного классификатора в слое 2 – по аналогии с тем, как мы решили задачу XOR в разделе 13.2.1.

13.2.6. Революция глубокого обучения

Хотя идеям, стоящим за ГНС, уже несколько десятков лет, лишь в 2010-х годах они начали получать широкое распространение. И первой областью, воспринявшей эти методы, стало автоматическое распознавание речи (automatic speech recognition – ASR), основанное на прорывных результатах работы [Dah+11]. Этот подход быстро стал стандартной парадигмой и был с готовностью принят как в академических кругах, так и в промышленности [Hin+12].

Однако момент наивысшего торжества настал, когда в работе [KSH12] было показано, что сверточные нейронные сети (СНС) способны значительно улучшить качество распознавания на эталонной задаче классификации

изображений ImageNet – частота ошибок уменьшилась с 26 до 16 % всего за один год (см. рис. 1.14b); это был гигантский скачок, по сравнению с предшествующими темпами прогресса, – примерно 2 % в год.

К взрывообразному росту использования ГНС привело несколько факторов. Первый – доступность дешевых **графических процессоров (GPU)**; первоначально они разрабатывались, чтобы ускорить рендеринг изображений в видеоиграх, но оказалось, что могут резко уменьшить время обучения больших ГНС, для которого нужны похожие вычисления с матрицами и векторами. Второй фактор – появление все новых и новых больших размеченных наборов данных, которые позволяют обучать сложные аппроксиматоры функций со многими параметрами без переобучения. (Например, набор ImageNet состоит из 1.3 млн помеченных изображений и используется для обучения моделей с миллионами параметров.) На самом деле если системы глубокого обучения считать «ракетами», то большие наборы данных следует назвать топливом для них¹.

Вдохновленные выдающимся эмпирическим успехом ГНС, различные компании стали проявлять интерес к этой технологии. Это привело к разработке высококачественных библиотек с открытым исходным кодом, например: Tensorflow (Google), PyTorch (Facebook) и MXNet (Amazon). Эти библиотеки поддерживают автоматическое дифференцирование (см. раздел 13.3) и масштабируемую градиентную оптимизацию (см. раздел 8.4) сложных дифференцируемых функций. Мы будем пользоваться некоторыми из них в разных местах книги для реализации различных моделей, а не только ГНС².

Дополнительные сведения об истории «революции глубокого обучения» можно найти, например, в работах [Sej18; Met21].

13.2.7. Связи с биологией

В этом разделе мы обсудим связи между теми видами нейронных сетей, которые обсуждались выше и называются **искусственными нейронными сетями (ИНС)** и реальными сетями нейронов. Детали того, как работает живой биологический мозг, весьма сложны (см., например, [Kan+12]), но можно дать упрощенную картинку, своего рода комикс.

Начнем с рассмотрения модели одного нейрона. В первом приближении можно считать, что состояние нейрона k (возбуждение или торможение), обозначаемое $h_k \in \{0, 1\}$, зависит от активности его входов, обозначаемых $\mathbf{x} \in \mathbb{R}^D$, а также от силы входящих связей, обозначаемой $\mathbf{w}_k \in \mathbb{R}^D$. Мы можем вычислить взвешенную сумму входов по формуле $a_k = \mathbf{w}_k^T \mathbf{x}$. Эти веса можно рассматривать как «провода», соединяющие входы x_d с нейроном h_k , – ана-

¹ Эта популярная аналогия принадлежит Эндрю Ыну, который привел ее в основном докладе на конференции GPU Technology Conference (GTC) в 2015 году. Его слайды доступны по адресу <https://bit.ly/38RTxzH>.

² Заметим, однако, что, по мнению некоторых авторов (см., например, [BI19]), современные библиотеки недостаточно гибкие и слишком сильно акцентируют внимание на методах, основанных на умножении плотных матриц на векторы, в ущерб более общим алгоритмическим примитивам.

логи **дендритов** реального нейрона (см. рис. 13.8). Затем эта взвешенная сумма сравнивается с пороговым значением b_k , и если активация превосходит порог, то нейрон возбуждается; можно считать, что нейрон генерирует электрический сигнал или **потенциал действия**. Таким образом, мы можем смоделировать поведение нейрона с помощью функции $h_k(\mathbf{x}) = H(\mathbf{w}_k^T \mathbf{x} - b_k)$, где $H(a) = \mathbb{I}(a > 0)$ – функция Хевисайда. Эта так называемая **модель Мак-Каллока-Питтса** была предложена в 1943 году в работе [MP43].

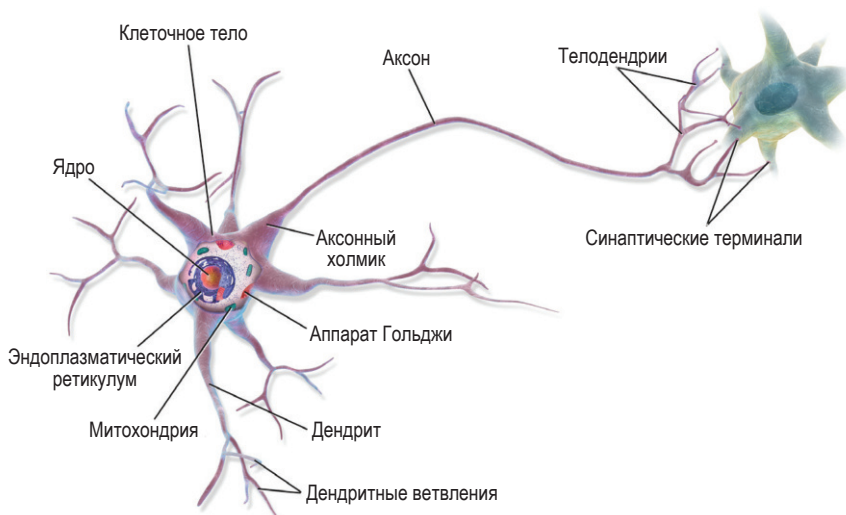


Рис. 13.8 ❖ Два нейрона, связанных в «цепь». Выходной аксон левого нейрона образует синаптическую связь с дендритами клетки справа. Электрические заряды в виде потока ионов позволяют клеткам взаимодействовать. Взято из статьи <https://en.wikipedia.org/wiki/Neuron>. Печатается с разрешения автора «Википедии» BruceBlau

Несколько таких нейронов можно объединить в ИНС. Иногда это рассматривалось как модель мозга. Однако ИНС отличается от биологического мозга во многих отношениях, например:

- в большинстве ИНС для модификации силы связей используется обратное распространение (см. раздел 13.3). Однако в реальном мозге ничего такого нет, потому что мозг не может посылать информацию по аксону в обратном направлении [Ben+15b; BS16; KH19]. Вместо этого для корректировки силы синапса используются локальные правила обновления;
- большинство ИНС – сети только прямого распространения, но в реальном мозге есть много обратных связей. Считается, что обратная связь играет роль априорного распределения, которое в сочетании с правдоподобиями, поступающими от сенсорной системы, может быть использована для вычисления апостериорного распределения скрытых состояний мира и далее для принятия оптимального решения (см., например, [Doy+07]);

- в большинстве ИНС используются упрощенные нейроны, состоящие из взвешенной суммы, которая передается на выход через нелинейность, тогда как биологические нейроны имеют разветвленную древовидную структуру с дендритами (рис. 13.8) и сложной пространственно-временной динамикой;
- ИНС в большинстве своем меньше по размеру и числу связей, чем биологические нейроны (см. рис. 13.9). Конечно, ИНС с каждой неделей становятся все больше, подпитываемые различными **аппаратными ускорителями** типа GPU и TPU (тензорными процессорами) и т. д. Но, даже если ИНС сравниваются с биологическим мозгом по числу блоков, сравнение все равно неправомерно, потому что обрабатывающая способность биологического нейрона гораздо выше, чем искусственного (см. предыдущий пункт);

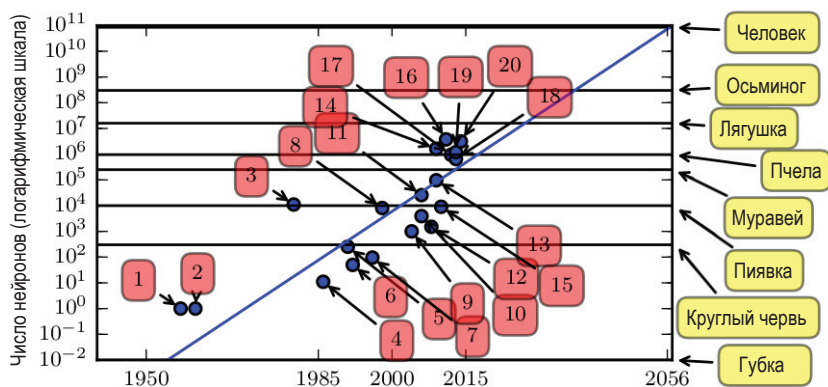


Рис. 13.9 ❖ Изменение размеров нейронных сетей со временем. Модели 1, 2, 3 и 4 соответствуют перцептрону [Ros58], адаптивному линейному блоку [WH60], неокогнитрону [Fuk80] и первым МСП, обученным методом обратного распространения [RHW86]. Приближенное число нейронов в некоторых биологических организмах, показанное на шкале слева (у губки число нейронов равно 0), основано на данных из статьи https://en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons. На основе рис. 1.11 из книг [GBC16]. Печатается с разрешения Яна Гудфеллоу

- большинство ИНС предназначено для моделирования какой-то одной функции, например сопоставления изображению метки или отображения одной последовательности слов в другую. Напротив, биологический мозг — исключительно сложная система, состоящая из нескольких специализированных взаимодействующих модулей, которые реализуют разные виды функций и поведения, как то: восприятие, управление, память, язык и т. д. (см., например, [Sha88; Kan+12]).

Конечно, были попытки создать реалистичные модели биологического мозга (например, проект **Blue Brain** [Mar06; Yon19]). Но возникает интересный вопрос: полезно ли изучение мозга на таком уровне детализации для «решения проблемы ИИ». Принято считать, что низкоуровневые детали био-

логического мозга не имеют значения, если целью является создание «интеллектуальных машин». Ведь самолеты не машут крыльями и все-таки летают. Однако есть предположение, что ИИ будет следовать «законам интеллекта», похожим на те, которым подчиняются биологические агенты, точно так же как самолеты и птицы подчиняются одним и тем же законам аэродинамики.

К сожалению, мы пока не знаем, как выглядят «законы интеллекта», да и есть ли они вообще. В этой книге мы будем предполагать, что любой интеллектуальный агент должен следовать базовым принципам обработки информации и байесовской теории принятия решений, которая, как известно, является оптимальным способом принятия решений в условиях неопределенности (см. раздел 5.1).

На практике оптимальный байесовский подход зачастую вычислительно нереализуем. В природе биологические агенты выработали различные алгоритмические «короткие пути» к оптимальному решению; так можно объяснить многие **эвристики**, применяемые людьми в повседневных рассуждениях [KST82; GTA00; Gri20]. По мере того как задачи, которые мы поручаем машинам, становятся все сложнее, мы можем позаимствовать у нейробиологии и когнитивистики опыт приближенного решения задач (см., например, [MWK16; Has+17; Lak+17]). Однако следует также помнить о том, что системы ИИ и МО все чаще используются для приложений с особыми требованиями к безопасности, когда мы ожидаем, что машина справится лучше человека. В таких случаях нам нужно не просто эвристическое решение, которое работает «в большинстве случаев»; необходимы доказуемо надежные методы, подобные тем, что применяются инженерами (дополнительное обсуждение см. в разделе 1.6.3).

13.3. ОБРАТНОЕ РАСПРОСТРАНЕНИЕ

Этот раздел написан в соавторстве с Мэтью Блонделем.

В этом разделе мы опишем знаменитый алгоритм обратного распространения, который может быть полезен для вычисления градиента выхода сети по параметрам в каждом слое. Затем этот градиент можно передать алгоритму градиентной оптимизации, как обсуждается в разделе 13.4.

Алгоритм обратного распространения впервые был описан в работе [BH69], а затем независимо открыт в работе [Wer74]. Однако только работа [RHW86] привлекла к нему внимание сообщества «классического» машинного обучения. Подробнее об истории вопроса см. статью в «Википедии»¹.

Для начала предположим, что граф вычислений представляет собой простой линейный граф последовательно расположенных слоев, как в МСП. В этом случае алгоритм обратного распространения эквивалентен повторному применению правила дифференцирования сложной функции, известного из математического анализа (см. формулу (7.261)). Однако его можно обобщить на произвольные ориентированные ациклические графы (DAG),

¹ <https://en.wikipedia.org/wiki/Backpropagation#History>.

как будет описано в разделе 13.3.4. Эта общая процедура называется **автоматическим дифференцированием**.

13.3.1. Прямой и обратный режим дифференцирования

Рассмотрим отображение вида $\mathbf{o} = \mathbf{f}(\mathbf{x})$, где $\mathbf{x} \in \mathbb{R}^n$ и $\mathbf{o} \in \mathbb{R}^m$. Предположим, что \mathbf{f} определена как композиция функций:

$$\mathbf{f} = \mathbf{f}_4 \circ \mathbf{f}_3 \circ \mathbf{f}_2 \circ \mathbf{f}_1, \quad (13.22)$$

где $\mathbf{f}_1 : \mathbb{R}^n \rightarrow \mathbb{R}^{m_1}$, $\mathbf{f}_2 : \mathbb{R}^{m_1} \rightarrow \mathbb{R}^{m_2}$, $\mathbf{f}_3 : \mathbb{R}^{m_2} \rightarrow \mathbb{R}^{m_3}$, $\mathbf{f}_4 : \mathbb{R}^{m_3} \rightarrow \mathbb{R}^m$. Для вычисления $\mathbf{o} = \mathbf{f}(\mathbf{x})$ необходимо выполнить промежуточные шаги: $\mathbf{x}_2 = \mathbf{f}_1(\mathbf{x})$, $\mathbf{x}_3 = \mathbf{f}_2(\mathbf{x}_2)$, $\mathbf{x}_4 = \mathbf{f}_3(\mathbf{x}_3)$, $\mathbf{o} = \mathbf{f}_4(\mathbf{x}_4)$.

Мы можем вычислить якобиан $\mathbf{J}_f(\mathbf{x}) = \partial \mathbf{o} / \partial \mathbf{x}^T \in \mathbb{R}^{m \times n}$, применив правило дифференцирования сложной функции:

$$\frac{\partial \mathbf{o}}{\partial \mathbf{x}} = \frac{\partial \mathbf{o}}{\partial \mathbf{x}_4} \frac{\partial \mathbf{x}_4}{\partial \mathbf{x}_3} \frac{\partial \mathbf{x}_3}{\partial \mathbf{x}_2} \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}} = \frac{\partial \mathbf{f}_4(\mathbf{x}_4)}{\partial \mathbf{x}_4} \frac{\partial \mathbf{f}_3(\mathbf{x}_3)}{\partial \mathbf{x}_3} \frac{\partial \mathbf{f}_2(\mathbf{x}_2)}{\partial \mathbf{x}_2} \frac{\partial \mathbf{f}_1(\mathbf{x})}{\partial \mathbf{x}} \quad (13.23)$$

$$= \mathbf{J}_{f_4}(\mathbf{x}_4) \mathbf{J}_{f_3}(\mathbf{x}_3) \mathbf{J}_{f_2}(\mathbf{x}_2) \mathbf{J}_{f_1}(\mathbf{x}). \quad (13.24)$$

Теперь поговорим о том, как вычислить якобиан $\mathbf{J}_f(\mathbf{x})$ эффективно. Напомним, что

$$\mathbf{J}_f(\mathbf{x}) = \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \nabla f_1(\mathbf{x})^T \\ \vdots \\ \nabla f_m(\mathbf{x})^T \end{pmatrix} = \left(\frac{\partial \mathbf{f}}{\partial x_1}, \dots, \frac{\partial \mathbf{f}}{\partial x_n} \right) \in \mathbb{R}^{m \times n}, \quad (13.25)$$

где $\nabla f_i(\mathbf{x})^T \in \mathbb{R}^{1 \times n}$ – i -я строка ($i = 1 \dots m$), а $\partial \mathbf{f} / \partial x_j \in \mathbb{R}^m$ – j -й столбец ($j = 1 \dots n$). Заметим, что в наших обозначениях при $m = 1$ градиент, обозначаемый $\nabla \mathbf{f}(\mathbf{x})$, имеет такую же форму, как \mathbf{x} . То есть он является вектором-столбцом, тогда как $\mathbf{J}_f(\mathbf{x})$ – вектор-строка. Поэтому технически имеем $\nabla \mathbf{f}(\mathbf{x}) = \mathbf{J}_f(\mathbf{x})^T$.

Мы можем выделить i -ю строку из $\mathbf{J}_f(\mathbf{x})$, воспользовавшись произведением вектора на якобиан (vector Jacobian product – VJP) вида $\mathbf{e}_i^T \mathbf{J}_f(\mathbf{x})$, где $\mathbf{e}_i \in \mathbb{R}^m$ – единичный базисный вектор. Аналогично j -й столбец можно выделить из $\mathbf{J}_f(\mathbf{x})$, воспользовавшись произведением якобиана на вектор (Jacobian vector product – JVP) вида $\mathbf{J}_f(\mathbf{x}) \mathbf{e}_j$, где $\mathbf{e}_j \in \mathbb{R}^n$. Таким образом, вычисление $\mathbf{J}_f(\mathbf{x})$ сводится к вычислению либо n JVP, либо m VJP.

Если $n < m$, то более эффективно находить $\mathbf{J}_f(\mathbf{x})$ для каждого столбца $j = 1 \dots n$, вычисляя JVP справа налево. Произведение справа на вектор-столбец \mathbf{v} имеет вид:

$$\mathbf{J}_f(\mathbf{x}) \mathbf{v} = \underbrace{\mathbf{J}_{f_4}(\mathbf{x}_4)}_{m \times m_3} \underbrace{\mathbf{J}_{f_3}(\mathbf{x}_3)}_{m_3 \times m_2} \underbrace{\mathbf{J}_{f_2}(\mathbf{x}_2)}_{m_2 \times m_1} \underbrace{\mathbf{J}_{f_1}(\mathbf{x}_1)}_{m_1 \times n} \underbrace{\mathbf{v}}_{n \times 1}. \quad (13.26)$$

Его можно вычислить, применив **дифференцирование в прямом режиме**, см. псевдокод в алгоритме 5. В предположении, что $m = 1$ и $n = m_1 = m_2 = m_3$, стоимость вычисления $\mathbf{J}_f(\mathbf{x})$ равна $O(n^3)$.

Алгоритм 5. Дифференцирование в прямом режиме

```

1   $\mathbf{x}_1 := \mathbf{x}$ 
2   $\mathbf{v}_j := \mathbf{e}_j \in \mathbb{R}^n$  для  $j = 1 : n$ 
3  for  $k = 1 : K$  do
4     $\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k)$ 
5     $\mathbf{v}_j := \mathbf{J}_{f_k}(\mathbf{x}_k)\mathbf{v}_j$  для  $j = 1 : n$ 
6  Вернуть  $\mathbf{o} = \mathbf{x}_{K+1}$ ,  $[\mathbf{J}_f(\mathbf{x})]_{:,j} = \mathbf{v}_j$  для  $j = 1 \dots n$ 
  
```

Если $n > m$ (например, когда выход скалярный), то эффективнее искать $\mathbf{J}_f(\mathbf{x})$ для каждой строки $i = 1 \dots m$, вычисляя VJP слева направо. Произведение слева на вектор-строку \mathbf{u}^\top имеет вид:

$$\mathbf{u}^\top \mathbf{J}_f(\mathbf{x}) = \underbrace{\mathbf{u}^\top}_{1 \times m} \underbrace{\mathbf{J}_{f_4}(\mathbf{x}_4)}_{m \times m_3} \underbrace{\mathbf{J}_{f_3}(\mathbf{x}_3)}_{m_3 \times m_2} \underbrace{\mathbf{J}_{f_2}(\mathbf{x}_2)}_{m_2 \times m_1} \underbrace{\mathbf{J}_{f_1}(\mathbf{x}_1)}_{m_1 \times n}. \quad (13.27)$$

Для этого можно применить **дифференцирование в обратном режиме**, см. псевдокод в алгоритме 6. В предположении, что $m = 1$ и $n = m_1 = m_2 = m_3$, стоимость вычисления $\mathbf{J}_f(\mathbf{x})$ равна $O(n^2)$.

Алгоритм 6. Дифференцирование в обратном режиме

```

1   $\mathbf{x}_1 := \mathbf{x}$ 
2  for  $k = 1 : K$  do
3     $\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k)$ 
4     $\mathbf{u}_i := \mathbf{e}_i \in \mathbb{R}^m$  для  $i = 1 : m$ 
5    for  $k = K : 1$  do
6       $\mathbf{u}_i^\top := \mathbf{u}_i^\top \mathbf{J}_{f_k}(\mathbf{x}_k)$  для  $i = 1 : m$ 
7  Вернуть  $\mathbf{o} = \mathbf{x}_{K+1}$ ,  $[\mathbf{J}_f(\mathbf{x})]_{i,:} = \mathbf{u}_i^\top$  для  $i = 1 \dots m$ 
  
```

Оба алгоритма 5 и 6 можно модифицировать, так чтобы они вычисляли JVP и VJP для любого набора входных векторов, если принимать на входе $\{\mathbf{v}_j\}_{j=1,\dots,n}$ и $\{\mathbf{u}_i\}_{i=1,\dots,m}$ соответственно. Инициализация этих векторов стандартным базисом особенно полезна, когда на выходе должен быть полный якобиан.

13.3.2. Дифференцирование в обратном режиме для многослойных перцептронов

В предыдущем разделе мы рассмотрели простую линейную модель прямого распространения, когда ни в одном слое нет обучаемых параметров. В этом разделе в каждом слое могут быть (необязательные) параметры $\theta_1, \dots, \theta_4$ (см. иллюстрацию на рис. 13.10). Нас будет интересовать случай, когда отобра-

532 ❖ Нейронные сети для структурированных данных

жение имеет вид $\mathcal{L} : \mathbb{R}_n \rightarrow \mathbb{R}$, т. е. выходом является скаляр. Например, рассмотрим потерю ℓ_2 для МСП с одним скрытым слоем:

$$\mathcal{L}((\mathbf{x}, \mathbf{y}), \boldsymbol{\theta}) = \frac{1}{2} \|\mathbf{y} - \mathbf{W}_2 \varphi(\mathbf{W}_1 \mathbf{x})\|_2^2. \quad (13.28)$$

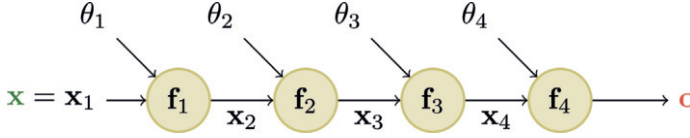


Рис. 13.10 ❖ Простая линейная модель прямого распространения с четырьмя слоями. Здесь \mathbf{x} – вход, а \mathbf{o} – выход. Из работы [Blo20]

Ее можно представить в виде следующей модели прямого распространения:

$$\mathcal{L} = f_4 \circ f_3 \circ f_2 \circ f_1; \quad (13.29)$$

$$\mathbf{x}_2 = f_1(\mathbf{x}, \boldsymbol{\theta}_1) = \mathbf{W}_1 \mathbf{x}; \quad (13.30)$$

$$\mathbf{x}_3 = f_2(\mathbf{x}_2, \emptyset) = \varphi(\mathbf{x}_2); \quad (13.31)$$

$$\mathbf{x}_4 = f_3(\mathbf{x}_3, \boldsymbol{\theta}_3) = \mathbf{W}_2 \mathbf{x}_3; \quad (13.32)$$

$$\mathcal{L} = f_4(\mathbf{x}_4, \mathbf{y}) = \frac{1}{2} \|\mathbf{x}_4 - \mathbf{y}\|^2. \quad (13.33)$$

Мы будем использовать нотацию $f_k(\mathbf{x}_k, \boldsymbol{\theta}_k)$ для обозначения функции в слое k , где \mathbf{x}_k – предыдущий выход, а $\boldsymbol{\theta}_k$ – необязательные параметры в этом слое.

В этом примере последний слой возвращает скаляр, поскольку соответствует функции потерь $\mathcal{L} \in \mathbb{R}$. Поэтому при вычислении векторов градиента эффективнее использовать дифференцирование в обратном режиме.

Сначала обсудим, как вычислить градиент скалярного выхода по параметрам в каждом слое. Легко вычислить градиент по предсказаниям в последнем слое $\partial \mathcal{L} / \partial \mathbf{x}_4$. Для вычисления градиентов по параметрам в предыдущих слоях воспользуемся правилом дифференцирования сложной функции:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_3} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_4} \frac{\partial \mathbf{x}_4}{\partial \boldsymbol{\theta}_3}; \quad (13.34)$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_2} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_4} \frac{\partial \mathbf{x}_4}{\partial \mathbf{x}_3} \frac{\partial \mathbf{x}_3}{\partial \boldsymbol{\theta}_2}; \quad (13.35)$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_4} \frac{\partial \mathbf{x}_4}{\partial \mathbf{x}_3} \frac{\partial \mathbf{x}_3}{\partial \mathbf{x}_2} \frac{\partial \mathbf{x}_2}{\partial \boldsymbol{\theta}_1}, \quad (13.36)$$

где каждая частная производная $\partial \mathcal{L} / \partial \boldsymbol{\theta}_k = (\nabla_{\boldsymbol{\theta}_k} \mathcal{L})^\top$ – d_k -мерный вектор-строка градиента, а d_k – количество параметров в слое k . Как видим, эти градиенты

можно вычислить рекурсивно, умножая вектор-строку градиента в слое k на якобиан $\partial \mathbf{x}_k / \partial \mathbf{x}_{k-1}$, являющийся матрицей $n_k \times n_{k-1}$, где n_k – количество скрытых блоков в слое k (см. псевдокод в алгоритме 7).

Этот алгоритм вычисляет градиент потери по параметрам в каждом слое. Также он вычисляет градиент потери по входу, $\nabla_{\mathbf{x}} \mathcal{L} \in \mathbb{R}^n$, где n – размерность входа. Эта последняя величина не нужна для обучения параметров, но может быть полезна для генерирования входных данных модели (см. некоторые приложения в разделе 14.6).

Алгоритм 7. Обратное распространение для МСП с K слоями

```

1 // Прямой проход
2  $\mathbf{x}_1 := \mathbf{x}$ 
3 for  $k = 1 : K$  do
4    $\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k, \boldsymbol{\theta}_k)$ 
5 // Обратный проход
6  $\mathbf{u}_{K+1} := 1$ 
7 for  $k = K : 1$  do
8    $\mathbf{g}_k := \mathbf{u}_{k+1}^\top \partial \mathbf{f}_k(\mathbf{x}_k, \boldsymbol{\theta}_k) / \partial \boldsymbol{\theta}_k$ 
9    $\mathbf{u}_k^\top := \mathbf{u}_{k+1}^\top \partial \mathbf{f}_k(\mathbf{x}_k, \boldsymbol{\theta}_k) / \partial \mathbf{x}_k$ 
10 // Выход
11 Вернуть  $\mathcal{L} = \mathbf{x}_{K+1}$ ,  $\nabla_{\mathbf{x}} \mathcal{L} = \mathbf{u}_1$ ,  $\{\nabla_{\boldsymbol{\theta}_k} \mathcal{L} = \mathbf{g}_k : k = 1 : K\}$ 

```

Остается только определить, как вычисляется произведение вектора на якобиан (VJP) во всех поддерживаемых слоях. Это зависит от формы функции в каждом слое. Ниже рассматривается несколько примеров.

13.3.3. Произведение вектора на якобиан для типичных слоев

Напомним, что якобиан для слоя вида $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ определяется по формуле

$$\mathbf{J}_f(\mathbf{x}) = \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \nabla f_1(\mathbf{x})^\top \\ \vdots \\ \nabla f_m(\mathbf{x})^\top \end{pmatrix} = \left(\frac{\partial \mathbf{f}}{\partial x_1}, \dots, \frac{\partial \mathbf{f}}{\partial x_n} \right) \in \mathbb{R}^{m \times n}, \quad (13.37)$$

где $\nabla f_i(\mathbf{x})^\top \in \mathbb{R}^n$ – i -я строка ($i = 1 \dots m$), а $\partial \mathbf{f} / \partial x_j \in \mathbb{R}^m$ – j -й столбец ($j = 1 \dots n$). В этом разделе мы опишем, как вычислять VJP $\mathbf{u}^\top \mathbf{J}_f(\mathbf{x})$ для часто встречающихся слоев.

13.3.3.1. Слой перекрестной энтропии

Рассмотрим слой с перекрестной энтропией в качестве потери, который принимает логиты \mathbf{x} и целевые метки \mathbf{u} на входе и возвращает скаляр:

$$\begin{aligned}
z = f(\mathbf{x}) &= \text{CrossEntropyWithLogits}(\mathbf{y}, \mathbf{x}) = -\sum_c y_c \log(\mathcal{S}(\mathbf{x})_c) \\
&= -\sum_c y_c \log p_c,
\end{aligned} \tag{13.38}$$

где $\mathbf{p} = \mathcal{S}(\mathbf{x}) = \frac{e^{x_c}}{\sum_{c'=1}^C e^{x_{c'}}}$ – предсказанные вероятности классов, а \mathbf{y} – истинное распределение меток (часто унитарный вектор). Якобиан по входным данным равен

$$\mathbf{J} = \frac{\partial z}{\partial \mathbf{x}} = (\mathbf{p} - \mathbf{y})^\top \in \mathbb{R}^{1 \times C}. \tag{13.39}$$

Чтобы убедиться в этом, предположим, что целевая метка – класс c . Имеем

$$z = f(\mathbf{x}) = -\log(p_c) = -\log\left(\frac{e^{x_c}}{\sum_j e^{x_j}}\right) = \log\left(\sum_j e^{x_j}\right) - x_c. \tag{13.40}$$

Отсюда

$$\frac{\partial z}{\partial x_i} = \frac{\partial}{\partial x_i} \log \sum_j e^{x_j} - \frac{\partial}{\partial x_i} x_c = \frac{e^{x_j}}{\sum_j e^{x_j}} - \frac{\partial z}{\partial x_i} x_c = p_i - \mathbb{I}(i = c). \tag{13.41}$$

Положив $\mathbf{y} = [\mathbb{I}(i = c)]$, получаем формулу (13.39). Отметим, что якобиан этого слоя является вектором-строкой, поскольку выход скалярный.

13.3.3.2. Поэлементная нелинейность

Рассмотрим слой, в котором нелинейность применяется поэлементно, $\mathbf{z} = \mathbf{f}(\mathbf{x}) = \varphi(\mathbf{x})$, так что $z_i = \varphi(x_i)$. Элемент в позиции (i, j) якобиана равен

$$\frac{\partial z_i}{\partial x_j} = \begin{cases} \varphi'(x_i), & \text{если } i = j \\ 0 & \text{в противном случае} \end{cases}, \tag{13.42}$$

где $\varphi'(a) = d\varphi(a)/da$. Иными словами, якобиан по входу равен

$$\mathbf{J} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \text{diag}(\varphi'(\mathbf{x})). \tag{13.43}$$

Для произвольного вектора \mathbf{u} мы можем вычислить $\mathbf{u}^\top \mathbf{J}$ путем поэлементного умножения диагональных элементов \mathbf{J} на \mathbf{u} . Например, если

$$\varphi(a) = \text{ReLU}(a) = \max(a, 0), \tag{13.44}$$

то имеем

$$\varphi'(a) = \begin{cases} 0 & a < 0 \\ 1 & a > 0 \end{cases}. \tag{13.45}$$

Субпроизводная (раздел 8.1.4.1) в точке $a = 0$ может быть любым значением из диапазона $[0, 1]$. Часто ее полагают равной 0. Тогда

$$\text{ReLU}'(a) = H(a), \quad (13.46)$$

где H – ступенчатая функция Хевисайда.

13.3.3.3. Линейный слой

Теперь рассмотрим линейный слой, $\mathbf{z} = f(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x}$, где $\mathbf{W} \in \mathbb{R}^{m \times n}$, так что $\mathbf{x} \in \mathbb{R}^n$ и $\mathbf{z} \in \mathbb{R}^m$. Мы можем вычислить якобиан по входному вектору, $\mathbf{J} = \partial \mathbf{z} / \partial \mathbf{x} \in \mathbb{R}^{m \times n}$, следующим образом. Заметим, что

$$z_i = \sum_{k=1}^n W_{ik} x_k. \quad (13.47)$$

Тогда элемент якобиана в позиции (i, j) равен

$$\frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_{k=1}^n W_{ik} x_k = \sum_{k=1}^n W_{ik} \frac{\partial}{\partial x_j} x_k = W_{ij}, \quad (13.48)$$

так как $\partial x_k / \partial x_j = \mathbb{I}(k = j)$. Отсюда якобиан по входу равен

$$\mathbf{J} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \mathbf{W}. \quad (13.49)$$

Произведение вектора $\mathbf{u}^\top \in \mathbb{R}^{1 \times m}$ на якобиан $\mathbf{J} \in \mathbb{R}^{m \times n}$ равно

$$\mathbf{u}^\top \frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \mathbf{u}^\top \mathbf{W} \in \mathbb{R}^{1 \times n}. \quad (13.50)$$

Теперь рассмотрим якобиан по матрице весов, $\mathbf{J} = \partial \mathbf{z} / \partial \mathbf{W}$. Его можно представить в виде матрицы $m \times (m \times n)$, работать с которой трудно. Поэтому возьмем вместо этого градиент по одному весу, W_{ij} . Его вычислить легче, потому что $\partial \mathbf{z} / \partial W_{ij}$ – вектор. Для его вычисления заметим, что

$$z_k = \sum_{l=1}^m W_{kl} x_l; \quad (13.51)$$

$$\frac{\partial z_k}{\partial W_{ij}} = \sum_{l=1}^m x_l \frac{\partial}{\partial W_{ij}} W_{kl} = \sum_{l=1}^m x_l \mathbb{I}(i = k \text{ и } j = l). \quad (13.52)$$

Отсюда

$$\frac{\partial \mathbf{z}}{\partial W_{ij}} = (0 \quad \dots \quad 0 \quad x_j \quad 0 \quad \dots \quad 0)^\top, \quad (13.53)$$

где ненулевой элемент находится в позиции i . Произведение вектора $\mathbf{u}^\top \in \mathbb{R}^{1 \times m}$ на якобиан $\partial \mathbf{z} / \partial \mathbf{W} \in \mathbb{R}^{m \times (m \times n)}$ можно представить матрицей формы $1 \times (m \times n)$. Заметим, что

$$\mathbf{u}^\top \frac{\partial \mathbf{z}}{\partial \mathbf{W}_{ij}} = \sum_{k=1}^m u_k \frac{\partial z_k}{\partial \mathbf{W}_{ij}} = u_i x_j. \quad (13.54)$$

Следовательно,

$$\left[\mathbf{u}^\top \frac{\partial \mathbf{z}}{\partial \mathbf{W}} \right]_{1,:} = \mathbf{u} \mathbf{x}^\top \in \mathbb{R}^{m \times n}. \quad (13.55)$$

13.3.3.4. Соберем все вместе

В упражнении 13.1 вам будет предложено собрать все вышеизложенное вместе.

13.3.4. Графы вычислений

МСП – простейший вид ГНС, в которой каждый слой напрямую подает данные на вход следующему, так что образуется цепная структура, показанная на рис. 13.10. Однако в современных ГНС дифференцируемые компоненты могут комбинироваться более сложными способами, образуя **граф вычислений**, по аналогии с тем, как программисты строят из элементарных функций более сложные. (На самом деле были предложения вместо термина «глубокое обучение» использовать «**дифференцируемое программирование**».) Единственное ограничение заключается в том, что граф вычислений должен быть **ориентированным ациклическим графом (DAG)**, каждая вершина которого представляет функцию, дифференцируемую по всем своим аргументам.

В качестве примера рассмотрим функцию:

$$f(x_1, x_2) = x_2 e^{x_1} \sqrt{x_1 + x_2 e^{x_1}}. \quad (13.56)$$

Мы можем вычислить ее, воспользовавшись графом на рис. 13.11 со следующими промежуточными функциями:

$$x_3 = f_3(x_1) = e^{x_1}; \quad (13.57)$$

$$x_4 = f_4(x_2, x_3) = x_2 x_3; \quad (13.58)$$

$$x_5 = f_5(x_1, x_4) = x_1 + x_4; \quad (13.59)$$

$$x_6 = f_6(x_5) = \sqrt{x_5}; \quad (13.60)$$

$$x_7 = f_7(x_4, x_6) = x_4 x_6. \quad (13.61)$$

Заметим, что узлы пронумерованы в топологическом порядке (родители всегда предшествуют потомкам). На обратном проходе, поскольку граф уже не цепной, может возникнуть необходимость в суммировании градиентов по нескольким путям. Например, поскольку x_4 влияет на x_5 и x_7 , имеем

$$\frac{\partial o}{\partial \mathbf{x}_4} = \frac{\partial o}{\partial \mathbf{x}_5} \frac{\partial \mathbf{x}_5}{\partial \mathbf{x}_4} + \frac{\partial o}{\partial \mathbf{x}_7} \frac{\partial \mathbf{x}_7}{\partial \mathbf{x}_4}. \quad (13.62)$$

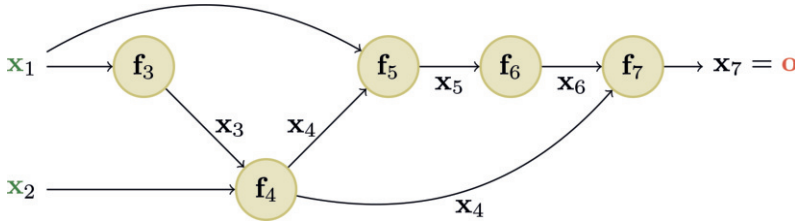


Рис. 13.11 ❖ Пример графа вычислений с тремя (скалярными) входами и одним (скалярным) выходом. Взято из работы [Blo20]

Мы можем избежать повторных вычислений, если будем работать в обратном топологическом порядке. Например,

$$\frac{\partial o}{\partial \mathbf{x}_7} = \frac{\partial \mathbf{x}_7}{\partial \mathbf{x}_7} = \mathbf{I}_m; \quad (13.63)$$

$$\frac{\partial o}{\partial \mathbf{x}_6} = \frac{\partial o}{\partial \mathbf{x}_7} \frac{\partial \mathbf{x}_7}{\partial \mathbf{x}_6}; \quad (13.64)$$

$$\frac{\partial o}{\partial \mathbf{x}_5} = \frac{\partial o}{\partial \mathbf{x}_6} \frac{\partial \mathbf{x}_6}{\partial \mathbf{x}_5}; \quad (13.65)$$

$$\frac{\partial o}{\partial \mathbf{x}_4} = \frac{\partial o}{\partial \mathbf{x}_5} \frac{\partial \mathbf{x}_5}{\partial \mathbf{x}_4} + \frac{\partial o}{\partial \mathbf{x}_7} \frac{\partial \mathbf{x}_7}{\partial \mathbf{x}_4}. \quad (13.66)$$

В общем случае пользуемся формулой

$$\frac{\partial o}{\partial \mathbf{x}_j} = \sum_{k \in \text{children}(j)} \frac{\partial o}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k}{\partial \mathbf{x}_j}, \quad (13.67)$$

где суммирование производится по всем дочерним вершинам k вершины j , как показано на рис. 13.12. Вектор градиента $\partial o / \partial \mathbf{x}_k$ уже был вычислен для каждой дочерней вершины k ; эта величина называется **адьюнктом**. Он умножается на якобиан $\partial \mathbf{x}_k / \partial \mathbf{x}_j$ каждой дочерней вершины.

Граф вычислений можно построить заранее, пользуясь API для определения **статического графа** (так это работало в Tensorflow 1). А можно вместо этого вычислять его «**своевременно**», **трассируя** выполнение функции от входного аргумента (так работает безотлагательный режим в Tensorflow, а также библиотеки JAX и PyTorch). Второй подход упрощает работу с **динамическим графом**, форма которого может изменяться в зависимости от значений, вычисленных функцией.

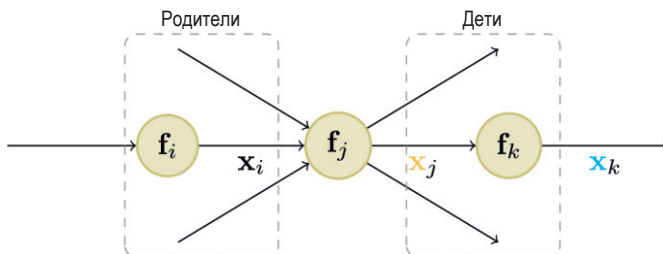


Рис. 13.12 ❖ Нотация автоматического дифференцирования в вершине j графа вычислений. Взято из работы [Blo20]

На рис. 13.13 показан граф вычислений, соответствующий МСА с одним скрытым слоем с уменьшением весов. Точнее, модель вычисляет линейные преактивации $\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$, скрытые активации $\mathbf{h} = \phi(\mathbf{z})$, линейные выходы $\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$, потерю $L = \ell(\mathbf{o}, \mathbf{y})$, регуляризатор $s = \lambda/2(\|\mathbf{W}^{(1)}\|_F^2 + \|\mathbf{W}^{(2)}\|_F^2)$ и полную потерю $J = L + s$.

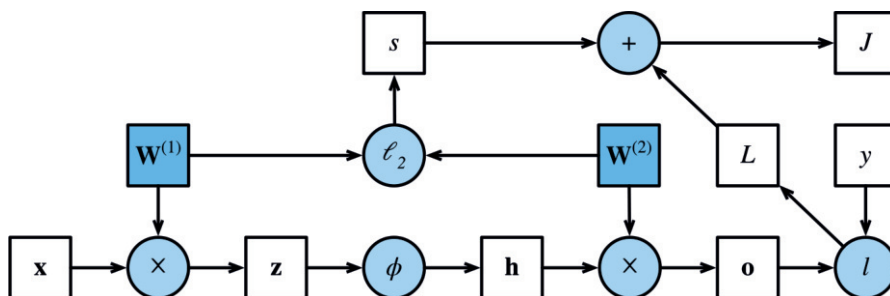


Рис. 13.13 ❖ Граф вычислений для МСП со входом \mathbf{x} , скрытым слоем \mathbf{h} , выходом \mathbf{o} , функцией потерь $L = \ell(\mathbf{o}, \mathbf{y})$, ℓ_2 -регуляризатором s , зависящим от весов, и общей потерей $J = L + s$. На основе рис. 4.7.1 из работы [Zha+20]. Печатается с разрешения Астона Чжана

13.4. ОБУЧЕНИЕ НЕЙРОННЫХ СЕТЕЙ

В этом разделе мы обсудим, как аппроксимировать данные глубокими нейронными сетями. Стандартный подход – использовать оценку максимального правдоподобия, минимизируя отрицательное логарифмическое правдоподобие (NLL):

$$\mathcal{L}(\theta) = -\log p(\mathcal{D}|\theta) = -\sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n; \theta). \quad (13.68)$$

Также часто прибавляют регуляризатор (например, отрицательное логарифмическое априорное распределение), но это мы обсудим в разделе 13.5.

В принципе, мы можем просто вычислить градиент этой потери, воспользовавшись алгоритмом обратного распространения (раздел 13.3), и передать его какому-нибудь готовому оптимизатору из числа обсуждавшихся в главе 8. (Популярным выбором является оптимизатор Adam из раздела 8.4.6.3, поскольку он хорошо масштабируется на большие наборы данных (т. е. является алгоритмом типа СГС) и довольно быстро сходится (благодаря использованию диагонального преобусловливания и момента). Но на практике это может работать плохо. В этом разделе мы поговорим о том, какие возможны проблемы, а также обсудим некоторые решения. Дополнительные сведения о практических вопросах обучения ГНС см. такие книги, как [HG20; Zha+20; Gér19].

Помимо практических проблем, есть и теоретические. В частности, отметим, что потеря ГНС не является выпуклой целевой функцией, поэтому в общем случае мы не сможем найти ее глобальный оптимум. Тем не менее СГС находит на удивление хорошие решения. Ученые до сих пор пытаются понять, почему это так; обзор недавних работ на эту тему см. в [Bah+20].

13.4.1. Настройка скорости обучения

Важно правильно настроить скорость обучения (размер шага), гарантирующую сходимость к хорошему решению. Этот вопрос обсуждался в разделе 8.4.3.

13.4.2. Исчезающие и взрывные градиенты

При обучении очень глубоких моделей градиент может стать либо очень малым (это называется **проблемой исчезающего градиента**), либо очень большим (это называется **проблемой взрывного градиента**), поскольку сигнал ошибки передается через последовательность слоев, которые уменьшают или увеличивают его [Нос+01]. (В разделе 15.2.6 мы увидим, что похожая проблема возникает при обучении рекуррентных нейронных сетей на очень длинных последовательностях.)

Чтобы лучше разобраться в существе проблемы, рассмотрим градиент потери по узлу в слое 1:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}_l} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_{l+1}} \frac{\partial \mathbf{z}_{l+1}}{\partial \mathbf{z}_l} = \mathbf{J}_l \mathbf{g}_{l+1}, \quad (13.69)$$

где $\mathbf{J}_l = \partial \mathbf{z}_{l+1} / \partial \mathbf{z}_l$ – матрица Якоби, а $\mathbf{g}_{l+1} = \partial \mathcal{L} / \partial \mathbf{z}_{l+1}$ – градиент в следующем слое. Если \mathbf{J}_l не зависит от слоя, то ясно, что вклад градиента в последнем слое, \mathbf{g}_L , в слой 1 равен $\mathbf{J}^{L-1} \mathbf{g}_L$. Таким образом, поведение системы зависит от собственных векторов \mathbf{J} .

Хотя \mathbf{J} – вещественная матрица, в общем случае она не симметрична, поэтому собственные значения и собственные векторы могут быть комплексными, а мнимые компоненты соответствуют колебательному поведению. Обозначим λ **спектральный радиус** \mathbf{J} , равный максимуму абсолютных величин

собственных значений. Если он больше 1, то градиент будет взрывным, а если меньше 1, то исчезающим. (Аналогично спектральный радиус матрицы \mathbf{W} , связывающей \mathbf{z}_l с \mathbf{z}_{l+1} , определяет устойчивость динамической системы при работе в прямом режиме.)

Проблему взрывного градиента можно сгладить, применив **обрезание градиента**, когда модуль градиента уменьшается, если оказывается слишком большим, т. е. мы полагаем

$$\mathbf{g}' = \min\left(1, \frac{c}{\|\mathbf{g}\|}\right) \mathbf{g}. \quad (13.70)$$

В таком случае норма \mathbf{g}' никогда не сможет стать больше c , но направление вектора будет таким же, как у \mathbf{g} .

Но справиться с проблемой исчезающего градиента труднее. Имеются различные решения, например:

- изменить функции активации в каждом слое, чтобы градиент не мог стать слишком большим или слишком малым; см. раздел 13.4.3;
- изменить архитектуру, так чтобы обновления были аддитивными, а не мультипликативными; см. раздел 13.4.4;
- изменить архитектуру, стандартизовав активации в каждом слое, так чтобы распределение активаций по набору данных оставалось постоянным в процессе обучения; см. раздел 14.2.4.1;
- тщательно выбирать начальные значения параметров; см. раздел 13.4.5.

13.4.3. Функции активации без насыщения

В разделе 13.2.3 мы упоминали, что сигмоидная функция активации стремится к 0 при больших отрицательных значениях аргумента и к 1 – при больших положительных. В этих режимах сигнал градиента равен 0, что препятствует обратному распространению.

Чтобы понять, почему градиент исчезает, рассмотрим слой, который вычисляет $\mathbf{z} = \sigma(\mathbf{W}\mathbf{x})$, где

$$\varphi(a) = \sigma(a) = \frac{1}{1 + \exp(-a)}. \quad (13.71)$$

Если веса инициализированы слишком большими значениями (положительными или отрицательными), то вектор $\mathbf{a} = \mathbf{W}\mathbf{x}$ легко может принять большие значения, а следовательно, \mathbf{z} будет испытывать насыщение, т. е. асимптотически стремиться к 0 или к 1, поскольку именно так ведет себя сигмоида (рис. 13.14а). Теперь рассмотрим градиент потери по входам \mathbf{x} (от предшествующего слоя) и параметрам \mathbf{W} . Производная функции активации равна

$$\varphi'(a) = \sigma(a)(1 - \sigma(a)). \quad (13.72)$$

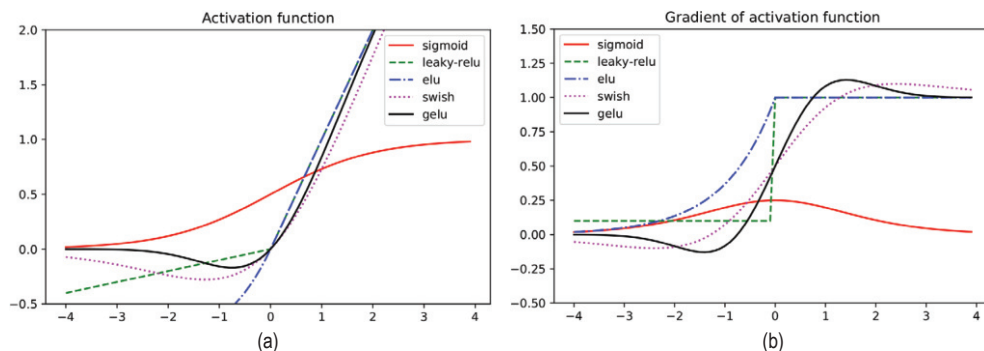


Рис. 13.14 ❖ (a) Некоторые популярные функции активации. (b) Графики их градиентов. Построено программой по адресу figures.problml.ai/book1/13.14

Ее график показан на рис. 13.14b. В разделе 13.3.3 было показано, что градиент потери по входам равен

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \mathbf{W}^T \boldsymbol{\delta} = \mathbf{W}^T \mathbf{z}(1 - \mathbf{z}), \quad (13.73)$$

а градиент потери по параметрам равен

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \boldsymbol{\delta} \mathbf{x}^T = \mathbf{z}(1 - \mathbf{z}) \mathbf{x}^T. \quad (13.74)$$

Поэтому, если \mathbf{z} близок к 0 или 1, то градиент стремится к 0.

При обучении очень глубоких моделей чрезвычайно важно использовать **функции активации без насыщения**. Был предложен ряд других функций, их сводка приведена в табл. 13.4, а более полные сведения можно найти по адресу <https://mlfromscratch.com/activation-functions-explained>.

Таблица 13.4. Некоторые популярные функции активации в нейронных сетях

Название	Определение	Диапазон	Ссылка
Сигмоида	$\sigma(a) = 1/(1 + e^{-a})$	$[0, 1]$	
Гиперболический тангенс	$\tanh(a) = 2\sigma(2a) - 1$	$[-1, 1]$	
Softplus	$\sigma_+(a) = \log(1 + e^a)$	$[0, \infty]$	[GBB11]
Блок линейной ректификации	$\text{ReLU}(a) = \max(a, 0)$	$[0, \infty]$	[GBB11; KSH12]
ReLU с утечкой	$\max(a, 0) + \alpha \min(a, 0)$	$[-\infty, \infty]$	[MHN13]
Экспоненциальный линейный блок	$\max(a, 0) + \min(\alpha(e^a - 1), 0)$	$[-\infty, \infty]$	[CUH16]
Swish	$a\sigma(a)$	$[-\infty, \infty]$	[RZL17]
GELU	$a\Phi(a)$	$[-\infty, \infty]$	[HG16]

13.4.3.1. ReLU

Самой распространенной функцией активации является **блок линейной ректификации**, или **ReLU**, предложенный в работах [GBB11; KSH12]. Она определена следующим образом:

$$\text{ReLU}(a) = \max(a, 0) = a\mathbb{I}(a > 0). \quad (13.75)$$

Функция ReLU просто «отключает» отрицательные входы, а положительные передает без изменения. Градиент имеет вид:

$$\text{ReLU}'(a) = \mathbb{I}(a > 0). \quad (13.76)$$

Теперь предположим, что мы используем ее в слое для вычисления $\mathbf{z} = \text{ReLU}(\mathbf{W}\mathbf{x})$. В разделе 13.3.3 было показано, что градиент по входам равен

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \mathbf{W}^T \mathbb{I}(\mathbf{z} > 0), \quad (13.77)$$

а по параметрам:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbb{I}(\mathbf{z} > 0) \mathbf{x}^T. \quad (13.78)$$

Поэтому градиент не исчезает ни при каких положительных \mathbf{z} .

К сожалению, если веса инициализированы большими отрицательными значениями, то некоторые компоненты $\mathbf{a} = \mathbf{W}\mathbf{x}$ легко могут принять большие отрицательные значения, а тогда \mathbf{z} будет стремиться к 0. Из-за этого градиент по весам тоже будет стремиться к 0. Алгоритм никогда не сможет выбраться из этой ситуации, поэтому скрытые блоки (компоненты \mathbf{z}) навечно останутся выключенными. Это называется проблемой «**мертвого ReLU**» [Lu+19].

13.4.3.2. ReLU без насыщения

Проблему мертвых ReLU можно решить, воспользовавшись вариантами ReLU без насыщения. Одна из альтернатив – **ReLU с утечкой**, предложенный в работе [MHN13]. Он определяется следующим образом:

$$\text{LReLU}(a; \alpha) = \max(\alpha a, a), \quad (13.79)$$

где $0 < \alpha < 1$. Угловой коэффициент этой функции равен 1 для положительных входов и α для отрицательных, поэтому какой-то сигнал будет передан предшествующим слоям, даже если на вход подано отрицательное значение (см. график на рис. 13.14b). Вариант ReLU с утечкой, в котором параметр α сделан обучаемым, а не фиксированным, называется **параметрическим ReLU** [He+15].

Еще один популярный выбор – функция **ELU**, предложенная в работе [CUH16]. Она определяется следующим образом:

$$\text{ELU}(a; \alpha) = \begin{cases} \alpha(e^a - 1), & \text{если } a \leq 0 \\ a, & \text{если } a > 0 \end{cases}. \quad (13.80)$$

По сравнению с ReLU с утечкой ее преимущество в гладкости (см. график на рис. 13.14а). Небольшая модификация ELU, называемая **SELU** (самонормируемый ELU), была предложена в работе [Kla+17]. Она имеет вид:

$$\text{SELU}(a; \alpha, \lambda) = \lambda \text{ELU}(a; \alpha). \quad (13.81)$$

Как ни удивительно это звучит, авторы доказали, что при тщательном выборе α и λ эта функция гарантирует, что выход каждого слоя будет стандартизован (при условии, что вход тоже стандартизован) даже без применения таких методов, как пакетная нормировка (раздел 14.2.4.1). Это может оказаться полезным для аппроксимации данных моделью.

13.4.3.3. Другие варианты

Можно не заниматься ручным подбором хороших функций активации, а воспользоваться оптимизацией методом черного ящика для поиска в пространстве функциональных форм. Такой подход был применен в работе [RZL17], и авторы обнаружили, что функция, названная ими **swish**, хорошо ведет себя на некоторых эталонных тестах классификации изображений. Определяется она так:

$$\text{swish}(a; \beta) = a\sigma(\beta a). \quad (13.82)$$

(Та же функция под названием **SiLU** (сигмоидный линейный блок) была независимо предложена в работе [HG16].) Смотрите график на рис. 13.14а.

Еще одной популярной функцией активации является **GELU** (Gaussian Error Linear Unit – линейный блок с гауссовой ошибкой) [HG16]. Она определена следующим образом:

$$\text{GELU}(a) = a\Phi(a), \quad (13.83)$$

где $\Phi(a)$ – функция распределения для стандартного нормального распределения:

$$\Phi(a) = \Pr(\mathcal{N}(0, 1) \leq a) = \frac{1}{2}(1 + \text{erf}(a/\sqrt{2})). \quad (13.84)$$

На рис. 13.14 видно, что эта функция не является ни выпуклой, ни монотонной, в отличие от большинства других функций активации.

GELU можно рассматривать как «мягкий» вариант ReLU, поскольку она заменяет ступенчатую функцию $\mathbb{I}(a > 0)$ гауссовой функцией распределения, $\Phi(a)$. Можно также считать GELU адаптивным вариантом прореживания (раздел 13.5.4), когда мы умножаем вход на двоичную скалярную маску, $m \sim \text{Ber}(\Phi(a))$, где вероятность прореживания равна $1 - \Phi(a)$. Таким образом, математическое ожидание выхода равно

$$\mathbb{E}[a] = \Phi(a) \times a + (1 - \Phi(a)) \times 0 = a\Phi(a). \quad (13.85)$$

Мы можем аппроксимировать GELU функцией swish с подходящими параметрами, а именно

$$\text{GELU}(a) \approx a\sigma(1.702a). \quad (13.86)$$

13.4.4. Остаточные связи

Одно из решений проблемы исчезающего градиента в ГНС состоит в использовании **остаточной сети**, или **ResNet** [He+16a]. Это сеть прямого распространения, в которой каждый слой имеет форму **остаточного блока**, определенного как

$$\mathcal{F}'_l(\mathbf{x}) = \mathcal{F}_l(\mathbf{x}) + \mathbf{x}, \quad (13.87)$$

где \mathcal{F}_l – стандартное неглубокое нелинейное отображение (например, линейный–активация–линейный). Внутренняя функция \mathcal{F}_l вычисляет остаточный член, или дельту, который необходимо прибавить к входу \mathbf{x} , чтобы получить желаемый выход; часто проще обучиться генерировать небольшое возмущение входных данных, чем напрямую предсказывать выход. (Остаточные связи обычно используются в сочетании с СНС, как обсуждается в разделе 14.3.4, но могут быть использованы и в МСП.)

У модели с остаточными связями столько же параметров, сколько у модели без таких связей. Причина в том, что градиенты могут напрямую передаваться от выходного к предшествующим слоям, как схематически показана на рис 13.15b. Чтобы убедиться в этом, заметим, что активации в выходном слое можно выразить в терминах любого предшествующего слоя l в виде

$$\mathbf{z}_L = \mathbf{z}_l + \sum_{i=l}^{L-1} \mathcal{F}_i(\mathbf{z}_i; \theta_i). \quad (13.88)$$

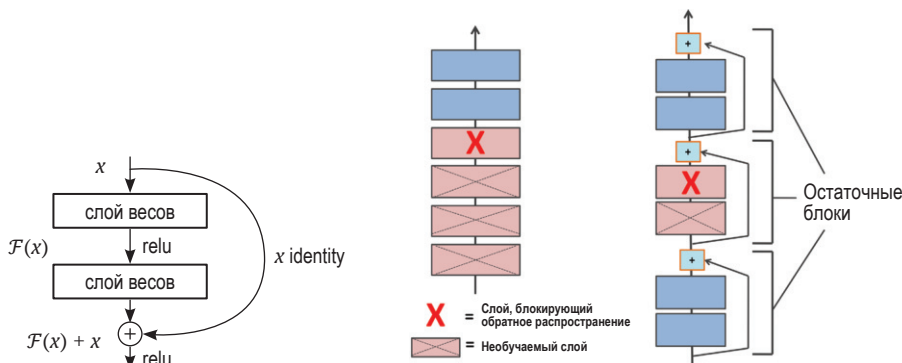


Рис. 13.15 ❖ (а) Остаточный блок.

(b) Почему добавление остаточных связей может помочь при обучении очень глубокой модели. На основе рис. 14.16 из работы [Gér19]

Поэтому мы можем вычислить градиент потери по параметра l -го слоя следующим образом:

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \frac{\partial \mathbf{z}_l}{\partial \theta_l} \frac{\partial \mathcal{L}}{\partial \mathbf{z}_l} \quad (13.89)$$

$$= \frac{\partial \mathbf{z}_l}{\partial \boldsymbol{\theta}_l} \frac{\partial \mathcal{L}}{\partial \mathbf{z}_L} \frac{\partial \mathbf{z}_L}{\partial \mathbf{z}_l} \quad (13.90)$$

$$= \frac{\partial \mathbf{z}_l}{\partial \boldsymbol{\theta}_l} \frac{\partial \mathcal{L}}{\partial \mathbf{z}_L} \left(1 + \sum_{i=l}^{L-1} \frac{\partial F_i(\mathbf{z}_i; \boldsymbol{\theta}_i)}{\partial \mathbf{z}_l} \right) \quad (13.91)$$

$$= \frac{\partial \mathbf{z}_l}{\partial \boldsymbol{\theta}_l} \frac{\partial \mathcal{L}}{\partial \mathbf{z}_L} + \text{другие члены.} \quad (13.92)$$

Таким образом, мы видим, что градиент в слое l зависит непосредственно от градиента в слое L , и в этой зависимости глубина сети вообще не фигурирует.

13.4.5. Инициализация параметров

Поскольку целевая функция для обучения ГНС невыпукла, от инициализации параметров сети может сильно зависеть найденное в конечном итоге решение, а также простота обучения функции (т. е. легкость, с которой информация передается вперед и назад в модели). Далее в этом разделе мы опишем некоторые эвристические методы, часто применяемые для инициализации параметров.

13.4.5.1. Эвристические схемы инициализации

В работе [GB10] показано, что если параметры выбираются из стандартного нормального распределения с фиксированной дисперсией, то могут наблюдаться взрывные активации или градиенты. Чтобы понять, почему это так, рассмотрим линейный блок без функции активации вида $o_i = \sum_{j=1}^{n_{\text{in}}} w_{ij} x_j$; предположим, что $w_{ij} \sim \mathcal{N}(0, \sigma^2)$ и $\mathbb{E}[x_j] = 0$, $\mathbb{V}[x_j] = 2$, где x_j не зависят от w_{ij} . Среднее и дисперсия выхода равны

$$\mathbb{E}[o_i] = \sum_{j=1}^{n_{\text{in}}} \mathbb{E}[w_{ij} x_j] = \sum_{j=1}^{n_{\text{in}}} \mathbb{E}[w_{ij}] \mathbb{E}[x_j] = 0; \quad (13.93)$$

$$\mathbb{V}[o_i] = \mathbb{E}[o_i^2] - (\mathbb{E}[o_i])^2 = \sum_{j=1}^{n_{\text{in}}} \mathbb{E}[w_{ij}^2 x_j^2] - 0 = \sum_{j=1}^{n_{\text{in}}} \mathbb{E}[w_{ij}^2] \mathbb{E}[x_j^2] = n_{\text{in}} \sigma^2 \gamma^2. \quad (13.94)$$

Чтобы удержать дисперсию от взрывного роста, нужно гарантировать, что $n_{\text{in}} \sigma^2 = 1$ (или какой-то другой постоянной), где n_{in} – **разветвление блока по входу** (количество входящих связей).

Теперь рассмотрим обратный проход. Рассуждая аналогично, мы видим, что дисперсия градиентов может расти взрывообразно, если не выполнено условие $n_{\text{out}} \sigma^2 = 1$, где n_{out} – **разветвление блока по выходу** (количество исходящих связей). Чтобы удовлетворить обоим требованиям сразу, положим $\frac{1}{2}(n_{\text{in}} + n_{\text{out}}) \sigma^2 = 1$ или эквивалентно

$$\sigma^2 = \frac{1}{n_{\text{in}} + n_{\text{out}}}. \quad (13.95)$$

Это называется **инициализацией Ксавье** или **инициализацией Глорота** по имени первого автора работы [GB10].

Специальный случай возникает, если $\sigma^2 = 1/n_{\text{in}}$; это известно как **инициализация Лекуна** в честь Яна Лекуна, который предложил ее в 1990-х годах. Она эквивалентна инициализации Глорота при $n_{\text{in}} = n_{\text{out}}$. Если $\sigma^2 = 2/n_{\text{in}}$, то этот метод называется **инициализацией Хе** по имени Симины Хе, предложившего его в работе [He+15].

Заметим, что использовать гауссово распределение необязательно. В самом деле, в приведенном выше выводе участвовали только первые два момента (среднее и дисперсия) и не делалось никаких предположений о гауссовости. Например, предположим, что веса выбраны из равномерного распределения, $w_{ij} \sim \text{Unif}(-a, a)$. Среднее равно 0, а дисперсия $\sigma^2 = a^2/3$.

Следовательно, необходимо положить $a = \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}$.

Хотя в этом выводе предполагается линейный выходной блок, сам метод эмпирически показывает хорошие результаты и для нелинейных блоков. Какой метод инициализации выбрать, зависит от используемой функции активации. Для линейной функции, \tanh , логистической функции и softmax рекомендуется инициализация Глорота, а для ReLU и его вариантов – инициализация Хе. Для SELU рекомендуется инициализация Лекуна. Другие эвристики см. в работе [Gér19].

13.4.5.2. Инициализации, управляемые данными

Мы также можем учитывать данные при инициализации параметров. Например, в работе [MM16] предложена простая, но эффективная схема, известная под названием **последовательно-послойная инициализация с единичной дисперсией** (layer-sequential unit-variance – LSUV) и работающая следующим образом. Сначала мы инициализируем веса в каждом (полносвязном или сверточном) слое с помощью ортонормированных матриц, как предложено в работе [SMG14]. (Это можно сделать, выбрав вектор весов $\mathbf{w} \sim \mathcal{N}(0, \mathbf{I})$, перематрицевав его в матрицу \mathbf{W} и затем вычислив ортонормированный базис посредством QR- или SVD-разложения.) Затем для каждого слоя l вычисляем дисперсию v_l активаций по мини-пакету; после этого масштабируем по формуле $\mathbf{W}_l := \mathbf{W}_l / \sqrt{v_l}$. Эту схему можно рассматривать как ортонормированную инициализацию в сочетании с пакетной нормировкой, выполняемой только для первого мини-пакета. Это быстрее, чем нормировка по полному пакету, но результат иногда получается не хуже.

13.4.6. Параллельное обучение

Обучать модели на больших наборах данных очень долго. Ускорить этот процесс позволяет специализированное оборудование, например **графи-**

ческие процессоры (GPU), позволяющие весьма эффективно перемножать матрицы. Если есть несколько GPU, то иногда можно добиться еще большего ускорения. Существует два основных подхода: **распараллеливание модели**, когда модель разносится на несколько машин, и **распараллеливание по данным**, когда на каждой машине имеется своя копия модели, но применяется она к разным наборам данных.

Распараллеливание модели может оказаться сложным делом, потому что для вычисления правильного ответа машины должны тесно взаимодействовать. Далее мы не будем обсуждать этот подход. Распараллеливание по данным в общем случае проще, поскольку налицо **естественный параллелизм**. Чтобы воспользоваться им для ускорения обучения, мы на каждом шаге обучения t производим следующие действия: 1) распределяем мини-пакет по K машинам, получая части \mathcal{D}_t^k ; 2) каждая машина k вычисляет свой собственный градиент, $\mathbf{g}_t^k = \nabla_{\theta} \mathcal{L}(\theta, \mathcal{D}_t^k)$; 3) собираем все локальные градиенты на центральной машине (например, на устройстве 0) и суммируем их, вычисляя $\mathbf{g}_t = \sum_{k=1}^K \mathbf{g}_t^k$; 4) рассылаем сумму градиентов всем устройствам, так что $\tilde{\mathbf{g}}_t^k = \mathbf{g}_t$; 5) каждая машина обновляет свою копию параметров по формуле $\theta_t^k := \theta_t^k - \rho_t \tilde{\mathbf{g}}_t^k$ (см. иллюстрацию на рис. 13.16).

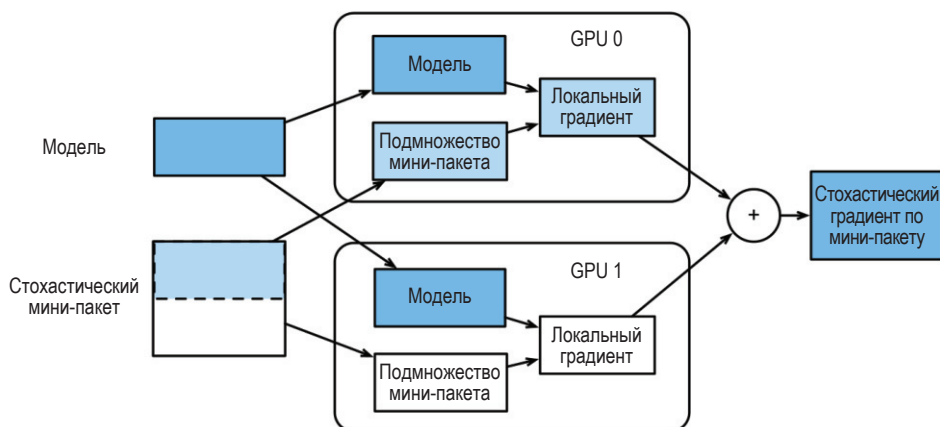


Рис. 13.16 ❖ Вычисление стохастического градиента по мини-пакету с применением распараллеливания по данным и двух GPU. На основе рис. 12.5.2 из работы [Zha+20]. Печатается с разрешения Астона Чжана

Заметим, что шаги 3 и 4 обычно объединяются в один атомарный шаг; это называется операцией **полной редукции** (all-reduce) (для сведения множества всех векторов градиента к одному используется суммирование). Если выполнение на каждой машине блокируется до получения агрегированного градиента от центральной машины, \mathbf{g}_t , то метод называется **синхронным обучением**. Он дает такие же результаты, как обучение на одной машине (с большим размером пакета), только быстрее (в предположении, что все слои нормировки пакета игнорируются). Если каждой машине разрешено обновлять свои параметры, пользуясь собственной локальной оценкой градиента, а не дожидаться, пока другие машины пошлют свои результаты и централь-

ная машина вычислит агрегированную оценку, то метод называется **асинхронным обучением**. Его правильность не гарантируется, так как машины могут рассинхронизироваться и будут обновлять разные версии параметров; поэтому в работе [Niu+11] такой подход назван **диким обучением** (hogwild training) [Niu+11]¹. Однако если обновления разреженные, т. е. все машины модифицируют разные части вектора параметров, то можно доказать, что дикое обучение ведет себя как стандартный синхронный ГСГ.

13.5. РЕГУЛЯРИЗАЦИЯ

В разделе 13.4 мы обсуждали вычислительные проблемы, возникающие при обучении больших нейронных сетей. В этом разделе мы обсудим статистические проблемы. И прежде всего нас будет интересовать, как избежать переобучения. Это очень важно, так как большая нейронная сеть может содержать миллионы параметров.

13.5.1. Ранняя остановка

Самый простой способ предотвратить переобучение называется **ранней остановкой**. Под этим понимается остановка процедуры обучения тогда, как ошибка на контрольном наборе начинает увеличиваться (см. пример на рис. 4.8). Этот метод работает, потому что мы ограничиваем способность алгоритма оптимизации переносить информацию от обучающих примеров к параметрам (см. объяснение в работе [AS19]).

13.5.2. Уменьшение весов

Распространенный подход к уменьшению переобучения – наложить на параметры априорное распределение, а потом воспользоваться оценкой MAP. Принято использовать гауссово априорное распределение для весов, $\mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^2\mathbf{I})$, и смещений, $\mathcal{N}(\mathbf{b}|\mathbf{0}, \beta^2\mathbf{I})$. Это эквивалентно ℓ_2 -регуляризации целевой функции. В литературе по нейронным сетям этот подход называют **уменьшением весов**, потому что он поощряет малые веса, а значит, и более простые модели, как в гребневой регрессии (раздел 11.3).

13.5.3. Разреженные ГНС

Поскольку в нейронной сети много весов, часто бывает полезно поощрить разреженность. Это позволит произвести **сжатие модели** и тем самым сэкономить память и время. Для этого можно использовать ℓ_1 -регуляризацию

¹ Буквально «кабанье обучение». Вероятно, имелась в виду непредсказуемость и необузданность диких свиней. – Прим. перев.

(как в разделе 11.4), ARD (как в разделе 11.7.7) и другие методы (см. недавний обзор [Ное+21]).

В качестве простого примера на рис. 13.17 показан 5-слойный МСП, обученный на данных одномерной регрессии с ℓ_1 -регуляризацией весов. Мы видим, что результирующий граф разрежен. Конечно, есть много других подходов к разреженному оцениванию.

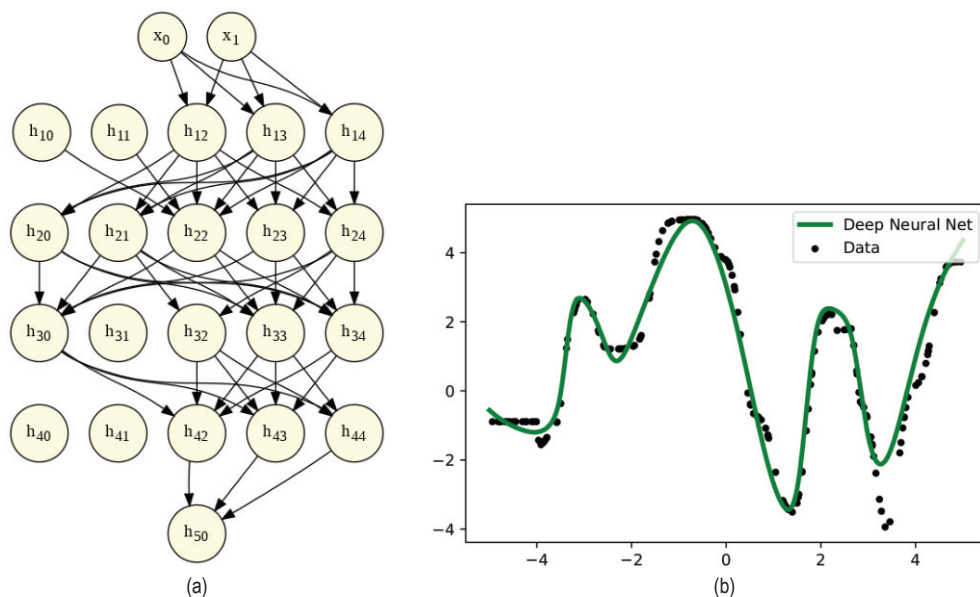


Рис. 13.17 ❖ (а) Глубокая, но разреженная нейронная сеть. Число связей уменьшено с помощью ℓ_1 -регуляризации. На каждом уровне узлы с номером 0 обрезаны по уровню 1, так что исходящие веса соответствуют членам смещения. (b) Предсказания модели на обучающем наборе. Построено программой по адресу figures.problml.ai/book1/13.17

Несмотря на интуитивную привлекательность разреженной топологии, на практике эти методы используются редко, потому что современные GPU оптимизированы для умножения плотных матриц, так что с вычислительной точки зрения у разреженных матриц весов мало преимуществ. Однако если применить методы, поощряющие групповую разреженность, то можно исключить целые слои модели. Это приводит к блочным разреженным матрицам, а значит, к ускорению работы и экономии памяти (см., например, [Sca+17; Wen+16; MAV17; LUW17]).

13.5.4. Прореживание

Предположим, что мы случайным образом (для каждого примера) с вероятностью p отбрасываем исходящие из нейронов связи, как показано на рис. 13.18. Эта техника называется **прореживанием** [Sri+14].

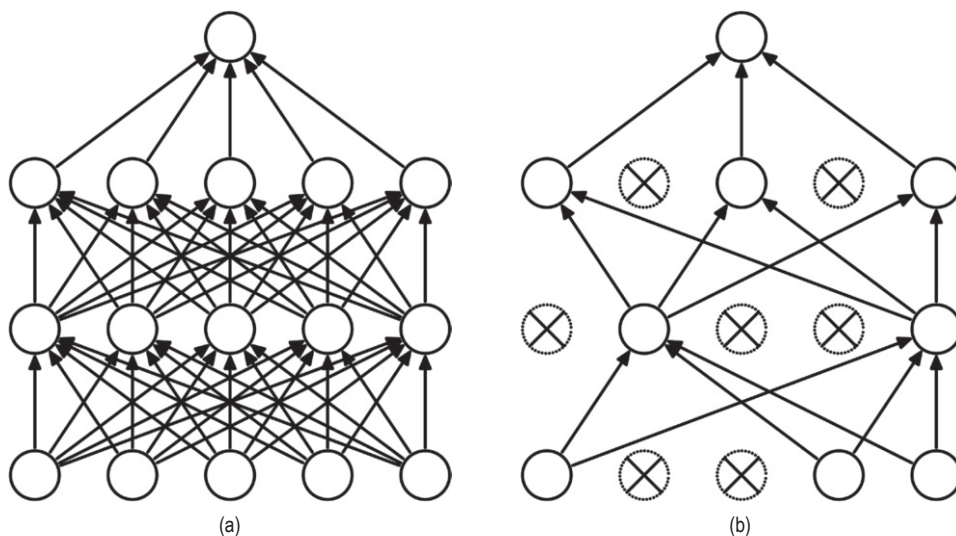


Рис. 13.18 ❖ Прореживание. (а) Стандартная нейронная сеть с двумя скрытыми слоями. (б) Пример истонченной сети, полученной в результате прореживания с $p_0 = 0.5$. Удаленные блоки помечены крестиком. На основе рис. 1 из работы [Sri+14]. Печатается с разрешения Джеффри Хинтона

Прореживание может значительно уменьшить переобучение и применяется очень широко. Интуитивно понятно, что прореживание показывает такие хорошие результаты, потому что предотвращает сложную совместную адаптацию скрытых блоков. Иными словами, каждый блок должен научиться хорошо работать, даже если случайное подмножество других блоков отсутствует. Это не дает блокам обучиться сложным, но хрупким взаимным зависимостям¹. Более формальное объяснение в терминах гауссовых смесей в качестве априорных распределений можно найти в работе [NHLS19].

Мы можем рассматривать прореживание как оценивание зашумленной версии весов, $\theta_{ij} = w_{ij}\varepsilon_{li}$, где $\varepsilon_{li} \sim \text{Ber}(1 - p)$ – член шума, подчиняющийся распределению Бернулли. (Так, если выбрать $\varepsilon_{li} = 0$, то веса всех связей, исходящих из блока i в слое $l - 1$ в любой блок j в слое l будут приравнены 0.) На этапе тестирования мы обычно убираем шум. Чтобы математическое ожидание весов на этапе тестирования было таким же, как на этапе обучения (т. е. входные активации нейронов в среднем были такими же), мы должны при тестировании положить $w_{ij} = \theta_{ij}\mathbb{E}[\varepsilon_{li}]$. В случае бернуллиева шума имеем $\mathbb{E}[\varepsilon] = 1 - p$, поэтому нужно умножить веса на вероятность $1 - p$, перед тем как делать предсказания.

Однако при желании мы можем производить прореживание и во время тестирования. Результатом является **ансамбль** сетей, каждая из которых

¹ Джеффри Хинтон, придумавший прореживание, говорил, что на эту мысль навел его разговор о половом размножении, поощряющем индивидуальную полезность генов (или хотя бы зависимости от небольшого числа других генов) даже в сочетании с другими, случайными, генами.

имеет слегка отличающуюся структуру разреженного графа. Это называется **прореживанием Монте-Карло** [GG16; KG17] и имеет вид:

$$p(y|\mathbf{x}, D) \approx \frac{1}{S} \sum_{s=1}^S p(y|\mathbf{x}, \hat{\mathbf{W}}\varepsilon^s + \hat{\mathbf{b}}), \quad (13.96)$$

где S – число примеров, а запись $\hat{\mathbf{W}}\varepsilon^s$ означает, что мы умножаем все матрицы с оценками весов на выборочный вектор шума. Иногда это дает хорошую аппроксимацию байесовского апостериорного прогнозного распределения $p(y|\mathbf{x}, D)$, особенно если величина шума оптимизирована [GHK17].

13.5.5. Байесовские нейронные сети

При обучении современных ГНС для нахождения единственного набора параметров обычно используют в качестве целевой функции максимальное правдоподобие (со штрафом). Но в случае больших моделей часто бывает, что параметров гораздо больше, чем примеров, поэтому возможно несколько моделей, одинаково хорошо аппроксимирующих обучающие данные, но обобщающихся по-разному. Часто полезно описать индуцированную неопределенность апостериорным прогнозным распределением. Это можно сделать с помощью интегрирования по параметрам, т. е. вычисления:

$$p(y|\mathbf{x}, D) = \int p(y|\mathbf{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|D) d\boldsymbol{\theta}. \quad (13.97)$$

В результате получается **байесовская нейронная сеть (BNN)**. Ее можно рассматривать как бесконечный ансамбль нейронных сетей с разными весами. Исключая параметры путем интегрирования, мы можем избежать переобучения [Mac95]. Байесовская маргинализация представляет серьезные трудности для больших нейронных сетей, но может дать значительный выигрыш в качестве [WI20]. Дополнительные сведения о **байесовском глубоком обучении** см. во втором томе этой книги [Mur22].

13.5.6. Эффекты регуляризации, порождаемые стохастическим градиентным спуском*

Некоторые методы оптимизации (в частности, пакетные методы второго порядка) могут находить «иголки в стоге сена», т. е. узкие, но глубокие «дыры» в поверхности функции потерь, соответствующие конфигурациям параметров с очень низкой потерей. Они называются **острыми минимумами** (см. рис. 13.19 справа). С точки зрения минимизации эмпирической потери оптимизатор достойно справился с работой. Однако такие решения обычно соответствуют переобученной модели. Лучше искать точки **плоских минимумов**, как на рис. 13.19 слева; такие решения более робастные и лучше обобщаются. Чтобы понять, почему это так, заметим, что плоские минимумы соответствуют областям пространства параметров, где велика апостериорная неопределенность, а значит, примеры из таких областей менее склон-

ны точно запоминать несущественную информацию об обучающем наборе [AS17]. СГС часто находит такие плоские минимумы благодаря добавлению шума, который предотвращает «заход» в узкие области поверхности функции потерь (см., например, [SL18]). Это называется **неявной регуляризацией**. Можно также явно поощрить СГС к нахождению таких плоских минимумов, применив метод **энтропийного СГС** [Cha+17], **минимизацию с учетом остроты** [For+21] и другие родственные методы.

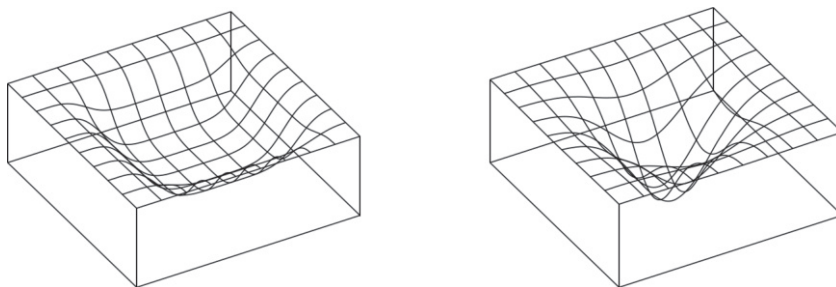


Рис. 13.19 ❖ Плоский и острый минимум. На основе рис. 1 и 2 из работы [HS97a]. Печатается с разрешения Юргена Шмидхубера

Разумеется, поверхность функции потерь зависит не только от значений параметров, но и от данных. Поскольку обычно мы не можем себе позволить градиентный спуск на полном пакете, построим хотя бы набор кривых потерь, по одной на мини-пакет. Если каждая из этих кривых напоминает широкую чашу, как на рис. 13.20а, то мы находимся к точке пространства параметров, робастной к возмущениям, и обобщаемость, вероятно, будет хорошей. Но если итоговая форма широкой чаши является результатом усреднения по многим узким чашам, как показано на рис. 13.20b, то получившаяся оценка вряд ли хорошо обобщается.

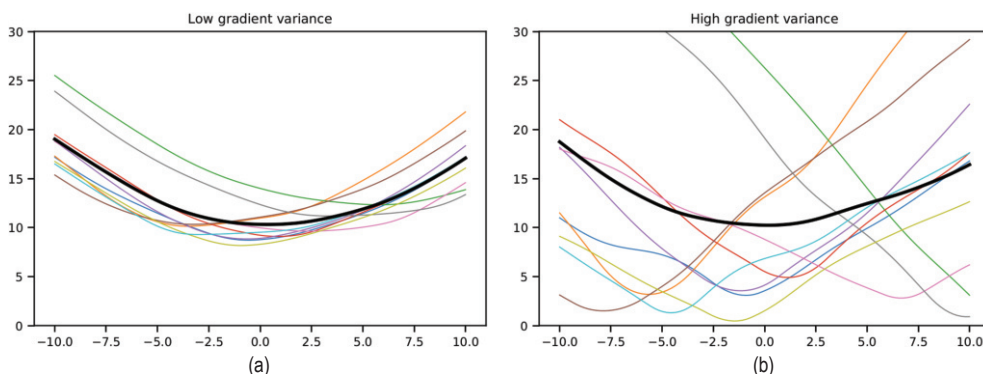


Рис. 13.20 ❖ Каждая кривая показывает, как меняется потеря в зависимости от значений параметров для заданного мини-пакета. (а) Устойчивый локальный минимум. (b) Неустойчивый локальный минимум. Построено программой по адресу figures.problml.ai/book1/13.20. На основе рисунка из статьи <https://bit.ly/3wTc1L6>

Это можно формализовать, проделав анализ, описанный в работах [Smi+21; BD21]. Конкретно авторы рассматривают непрерывный поток градиента, аппроксимирующий поведение (С)ГС. В работе [BD21] рассмотрен ГС на полном пакете и показано, что поток имеет вид $\dot{\mathbf{w}} = -\nabla_{\mathbf{w}} \tilde{\mathcal{L}}_{GD}(\mathbf{w})$, где

$$\tilde{\mathcal{L}}_{GD}(\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \frac{\varepsilon}{4} \|\nabla \mathcal{L}(\mathbf{w})\|^2, \quad (13.98)$$

где $\mathcal{L}(\mathbf{w})$ – исходная потеря, ε – скорость обучения, а второе слагаемое – неявный член регуляризации, который штрафует за большой градиент (высокую кривизну).

В работе [Smi+21] этот анализ распространен на случай ГС. Показано, что поток имеет вид $\dot{\mathbf{w}} = -\nabla_{\mathbf{w}} \tilde{\mathcal{L}}_{SGD}(\mathbf{w})$, где

$$\tilde{\mathcal{L}}_{SGD}(\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \frac{\varepsilon}{4} \sum_{k=1}^m \|\nabla \mathcal{L}_k(\mathbf{w})\|^2, \quad (13.99)$$

где m – число мини-пакетов, а $\mathcal{L}_k(\mathbf{w})$ – потеря на k -м мини-пакете. Сравнивая с потерей в случае ГС на полном пакете, мы видим, что

$$\tilde{\mathcal{L}}_{SGD}(\mathbf{w}) = \tilde{\mathcal{L}}_{GD}(\mathbf{w}) + \frac{\varepsilon}{4} \sum_{k=1}^m \|\nabla \mathcal{L}_k(\mathbf{w}) - \nabla \mathcal{L}(\mathbf{w})\|^2. \quad (13.100)$$

Второе слагаемое оценивает дисперсию градиентов на мини-пакетах, являющуюся мерой устойчивости, а значит, способности к обобщению.

Приведенный анализ показывает, что ГС дает не только вычислительные преимущества (потому что быстрее ГС на полном пакете или методов второго порядка), но и статистические.

13.6. ДРУГИЕ ВИДЫ СЕТЕЙ ПРЯМОГО РАСПРОСТРАНЕНИЯ*

13.6.1. Сети радиально-базисных функций

Рассмотрим нейронную сеть с одним слоем, в которой скрытый слой описывается вектором признаков:

$$\phi(\mathbf{x}) = [\mathcal{K}(\mathbf{x}, \mu_1), \dots, \mathcal{K}(\mathbf{x}, \mu_K)], \quad (13.101)$$

где $\mu_k \in \mathcal{X}$ – множество K **центроидов**, или **эталонов**, а $\mathcal{K}(\mathbf{x}, \mu) \geq 0$ – **ядерная функция**. Более подробно мы опишем ядерные функции в разделе 17.1. Здесь же только приведем пример, а именно **гауссово ядро**:

$$\mathcal{K}_{\text{gauss}}(\mathbf{x}, \mathbf{c}) \triangleq \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{c} - \mathbf{x}\|_2^2\right). \quad (13.102)$$

Параметр σ называется **полосой пропускания** ядра. Заметим, что это ядро инвариантно относительно сдвига, т. е. оно зависит только от расстояния $r = \|\mathbf{x} - \mathbf{c}\|^2$, так что его можно также записать в виде

$$\mathcal{K}_{\text{gauss}}(r) \triangleq \exp\left(-\frac{1}{2\sigma^2}r^2\right). \quad (13.103)$$

Поэтому оно называется **радиально-базисной функцией**, или **RBF-ядром**.

Однослойная нейронная сеть, в которой скрытый слой описывается формулой (13.101) с RBF-ядрами, называется **RBF-сетью** [BL88]. Она имеет вид:

$$p(y|\mathbf{x}; \boldsymbol{\theta}) = p(y|\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})), \quad (13.104)$$

где $\boldsymbol{\theta} = (\boldsymbol{\mu}, \mathbf{w})$. Если центроиды $\boldsymbol{\mu}$ фиксированы, то оптимальные веса \mathbf{w} можно найти (регуляризованным) методом наименьших квадратов, как описано в главе 11. Если же центроиды неизвестны, то их можно оценить, воспользовавшись методом кластеризации без учителя, например методом K средних (раздел 21.3). Вместо этого можно ассоциировать по одному центроиду с каждой точкой в обучающем наборе, т. е. положить $\boldsymbol{\mu}_n = \mathbf{x}_n$, так что $K = N$. Это пример **непараметрической модели**, поскольку число параметров растет (в данном случае линейно) вместе с объемом данных и не зависит от N . Если $K = N$, то модель идеально интерполирует данные и, значит, подвержена переобучению. Однако если мы гарантируем, что вектор выходных весов \mathbf{w} разрежен, то в модели будет использоваться только конечное подмножество входных примеров; это называется **разреженной ядерной машиной**, мы подробно обсудим ее в разделах 17.4.1 и 17.3. Еще один способ избежать переобучения – принять байесовский подход, исключив веса \mathbf{w} с помощью интегрирования; это приводит к модели, называемой **гауссовым процессом**, которую мы обсудим в разделе 17.2.

13.6.1.1. RBF-сеть для регрессии

Мы можем использовать RBF-сети для регрессии, положив $p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}), \sigma^2)$. Например, на рис. 13.22 показан одномерный набор данных, аппроксимированный с помощью $K = 10$ равноотстоящих RBF-прототипов, когда полоса пропускания изменяется от малой до большой. При малых значениях получаются очень извилистые функции, поскольку предсказанная функция будет отлична от нуля только в точках \mathbf{x} , близких к одному из прототипов $\boldsymbol{\mu}_k$. Если же полоса пропускания очень велика, то матрица плана сводится к постоянной матрице, состоящей из одних единиц, т. е. каждая точка одинаково близка к каждому прототипу; поэтому соответствующая функция – просто прямая линия.

13.6.1.2. RBF-сеть для классификации

Мы можем использовать RBF-сети для бинарной классификации, определив $p(y|\mathbf{x}, \boldsymbol{\theta}) = \text{Ber}(\sigma(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})))$. В качестве примера рассмотрим данные, порожден-

ные функцией ИСКЛЮЧАЮЩЕЕ ИЛИ (xor). Это логическая функция с двумя бинарными входами. Ее таблица истинности показана на рис. 13.21а. А на рис. 13.1b показаны некоторые данные, помеченные функцией xor, но, чтобы сделать рисунок нагляднее, мы добавили к точкам небольшую **флуктуацию**¹. Мы видим, что данные невозможно разделить даже полиномом 10-й степени. Однако стоит привлечь RBF-ядро всего с четырьмя прототипами, как задача легко решается, как показано на рис. 13.1с.

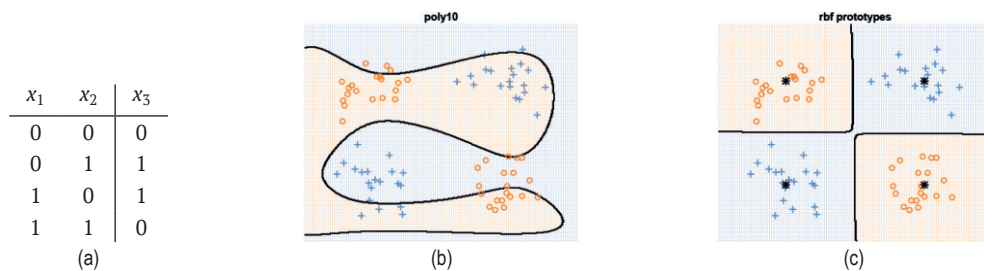


Рис. 13.21 ❖ (а) Таблица истинности xor. (b) Аппроксимация данных логистическим классификатором в виде полинома 10-й степени. (с) Та же модель, но с использованием RBF-ядра с четырьмя центроидами, помеченными черными звездочками. Построено программой по адресу figures.problml.ai/book1/13.21

13.6.2. Смесь экспертов

При рассмотрении задач регрессии обычно предполагают унимодальное распределение выхода, например Гаусса или Стюдента, считая, что среднее и дисперсия являются некоторой функцией входа, т. е.

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{f}_\mu(\mathbf{x}), \text{diag}(\sigma_+(f_\sigma(\mathbf{x})))), \quad (13.105)$$

где в роли функций f могут выступать МСП (возможно, с разделяемыми скрытыми блоками, как на рис. 13.5). Однако эта идея плохо работает для **многозначных функций**, когда одному входу может соответствовать несколько возможных выходов.

На рис. 13.23а приведен пример такой функции. Мы видим, что в средней части графика имеются значения x , которым соответствует два равновероятных значения y . На практике существует много задач такого вида, например предсказание позы человека в трехмерном пространстве по одному изображению [Bo+08], раскрашивание черно-белого изображения [Gua+17], предсказание будущих кадров видеоряда [VT17] и т. д. Любая модель, обученная максимизировать правдоподобие с применением унимодальной плотности выхода – даже если она является гибкой и нелинейной, как, например, ней-

¹ Флуктуация часто применяется в статистике в случаях, когда без нее точки на графике или на экране наложатся бы друг на друга. Заключается она в добавлении равномерно распределенного аддитивного шума, который увеличивает дисперсию.

ронная сеть, – будет плохо работать с такими многозначными функциями, поскольку способна породить только размытый усредненный выход.

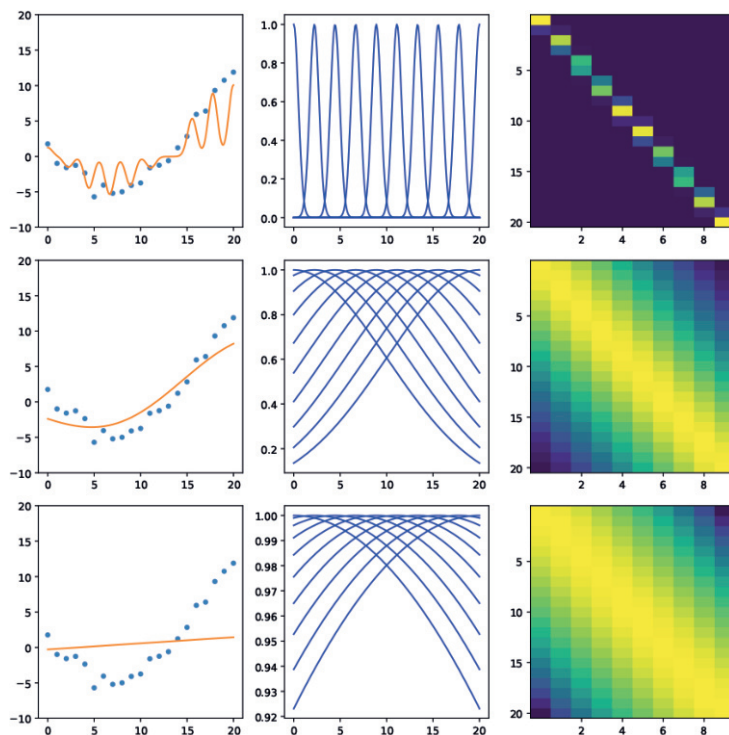


Рис. 13.22 ❖ Линейная регрессия с 10 равноотстоящими радиально-базисными функциями в одномерном случае. Левый столбец: аппроксимированные данные. Средний столбец: базисные функции, вычисленные на сетке. Правый столбец: матрица плана. Сверху вниз показаны различные полосы пропускания для ядерной функции: $\sigma = 0.5, 10, 50$.

Построено программой по адресу figures.probml.ai/book1/13.22

Чтобы обойти эту проблему регрессии к среднему, мы можем воспользоваться **условной смесовой моделью**. То есть будем предполагать, что выход – это взвешенная смесь K разных выходов, соответствующих разным модам выходного распределения для каждого входа \mathbf{x} . В гауссовом случае имеем

$$p(\mathbf{y}|\mathbf{x}) = \sum_{k=1}^K p(\mathbf{y}|\mathbf{x}, z = k)p(z = k|\mathbf{x}); \quad (13.106)$$

$$p(\mathbf{y}|\mathbf{x}, z = k) = \mathcal{N}(\mathbf{y}|\mathbf{f}_{\mu,k}(\mathbf{x}), \text{diag}(\mathbf{f}_{\sigma,k}(\mathbf{x}))); \quad (13.107)$$

$$p(z = k|\mathbf{x}) = \text{Cat}(z|\mathcal{S}(\mathbf{f}_z(\mathbf{x}))). \quad (13.108)$$

Здесь $f_{\mu,k}$ предсказывает среднее k -го гауссова распределения, $f_{\sigma,k}$ предсказывает его дисперсию, а f_z – какие компоненты смеси использовать. Эта модель называется **смесью экспертов** (mixture of experts – **MoE**) [Jas+91; JJ94; YWG12; ME14]. Идея в том, что k -я подмодель $p(y|x, z = k)$ считается «экспертом» в некоторой области пространства входов. Функция $p(z = k|x)$, называемая **привратником** (gating function), решает, какого эксперта использовать в зависимости от выходных значений. Выбирая наиболее правдоподобного эксперта для данного входа x , мы можем «активировать» нужное подмножество модели. Это пример **условного вычисления**, поскольку решение о выборе эксперта основано на результатах вычислений предшествующей сети-привратника [Sha+17].

Мы можем обучить эту модель с помощью СГС или ЕМ-алгоритма (второй метод описан в разделе 8.7.3).

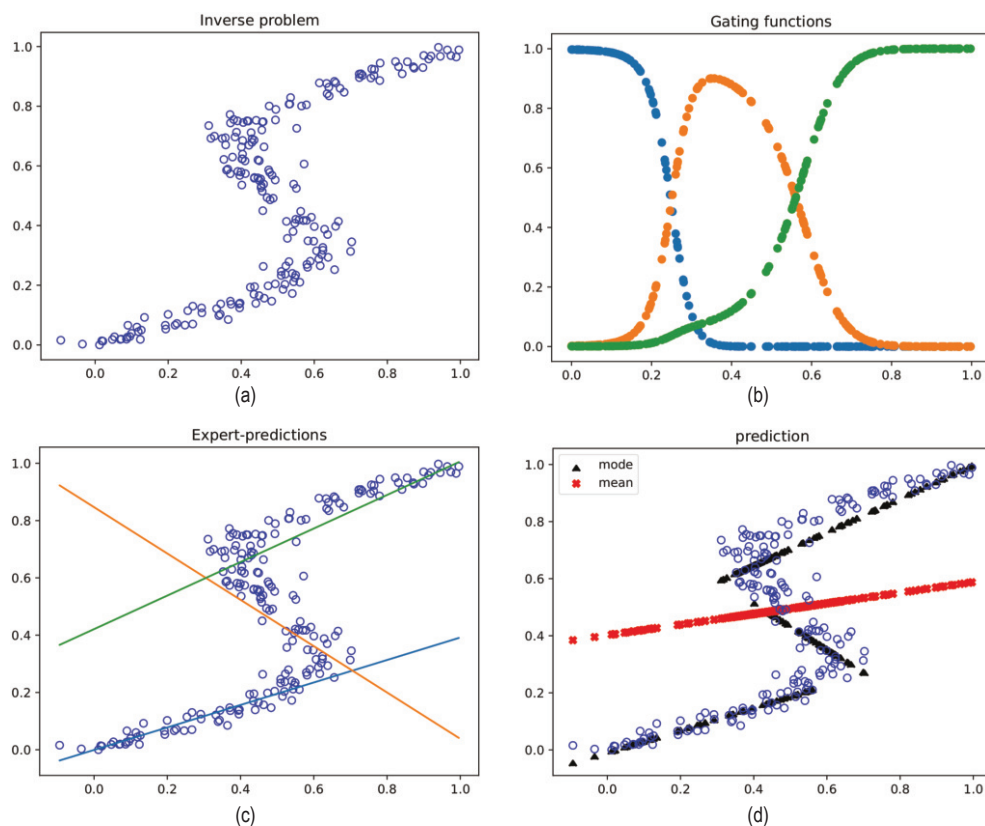


Рис. 13.23 ❖ (а) Данные, описываемые многозначной функцией. (б) Каждый эксперт отвечает за свою область пространства входов. (с) Предсказания каждого эксперта. (д) Общее предсказание. Среднее обозначено красным крестиком, мода черным квадратиком. На основе рисунков 5.20 и 5.21 из работы [Bis06]. Построено программой по адресу figures.problml.ai/book1/13.23

13.6.2.1. Смесь линейных экспертов

В этом разделе мы рассматриваем простой пример, где в роли экспертов выступают модели линейной регрессии, а в роли привратника – линейный классификатор, т. е. модель имеет вид:

$$p(y|\mathbf{x}, z = k, \boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{w}_k^T \mathbf{x}, \sigma_k^2), \quad (13.109)$$

$$p(z|\mathbf{x}, \boldsymbol{\theta}) = \text{Cat}(z|S(\mathbf{V}\mathbf{x})). \quad (13.110)$$

Каждый весовой член $p(z = k|\mathbf{x})$ называется **ответственностью** эксперта k за вход \mathbf{x} . На рис. 13.23b мы видим, как сети-привратники аккуратно делят пространство входов между $K = 3$ экспертами.

Эксперты $p(y|\mathbf{x}, z = k)$ соответствуют моделям линейной регрессии с разными параметрами. Они показаны на рис. 13.23с.

Если взять взвешенную комбинацию экспертов в качестве выхода, то получится красная кривая на рис. 13.23d – очевидно, никуда не годный предиктор. Если же предсказывать только по самому активному эксперту (с наибольшей ответственностью), то получится разрывная черная кривая, дающая куда лучший предиктор.

13.6.2.2. Глубокие сети экспертов

Функция-привратник и эксперты могут быть представлены условной вероятностной моделью любого вида, а не только линейной. Если в качестве таких моделей взять ГНС, то мы получим **сеть на основе смеси моделей разной плотности** (mixture density network – MDN) [Bis94; ZS14] или **глубокой смеси экспертов** [CGG17]. Схематическое изображение такой модели показано на рис. 13.24.

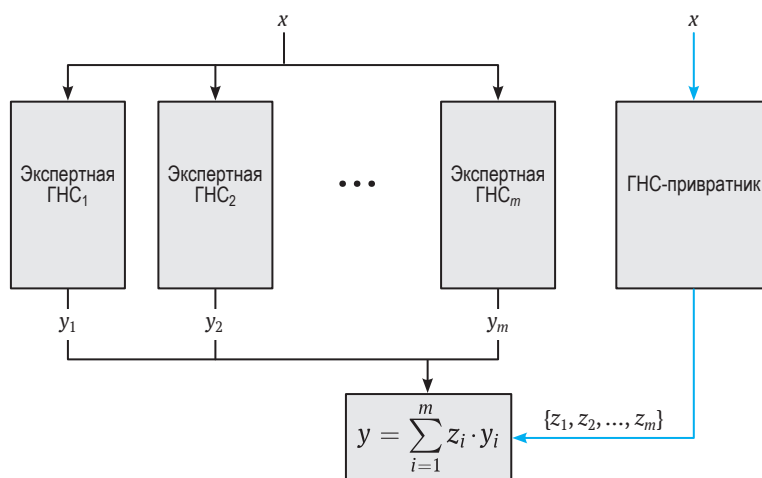


Рис. 13.24 ❖ Глубокая модель m экспертов, представленная в виде нейронной сети. На основе рис. 1 из работы [CGG17]. Печатается с разрешения Джейкоба Голдбергера

13.6.2.3. Иерархические смеси экспертов

Если каждый эксперт сам является МоЕ-моделью, то получающаяся модель называется **иерархической смесью экспертов** (НМЕ) [JJ94]. На рис. 13.25 показан пример такой модели с двухуровневой иерархией.

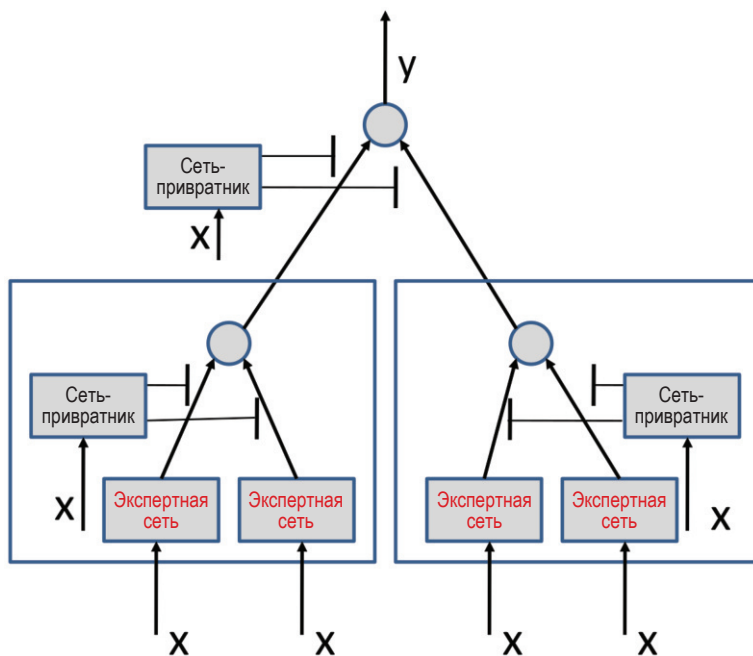


Рис. 13.25 ❖ Двухуровневая иерархическая смесь экспертов как нейронная сеть. Верхняя сеть-привратник выбирает между левым и правым экспертом, представленными большими прямоугольниками; левый и правый эксперты сами выбирают между своими левыми и правыми экспертами следующего уровня

L -уровневую НМЕ можно рассматривать как «мягкое» решающее дерево глубины L , в котором каждый пример проходит по всем ветвям дерева, а конечным предсказанием является взвешенное среднее. (Мы будем обсуждать решающие деревья в разделе 18.1.)

13.7. УПРАЖНЕНИЯ

Упражнение 13.1 [обратное распространение для МСП].

(Основано на упражнении, предложенном Кэвином Кларком.)

Рассмотрим следующий МСП классификации с одним скрытым слоем:

$$\mathbf{x} = \text{input} \in \mathbb{R}^D; \quad (13.111)$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}_1 \in \mathbb{R}^K; \quad (13.112)$$

$$\mathbf{h} = \text{ReLU}(\mathbf{z}) \in \mathbb{R}^K; \quad (13.113)$$

$$\mathbf{a} = \mathbf{V}\mathbf{h} + \mathbf{b}_2 \in \mathbb{R}^C; \quad (13.114)$$

$$\mathcal{L} = \text{CrossEntropy}(\mathbf{y}, \mathcal{S}(\mathbf{a})) \in \mathbb{R}, \quad (13.115)$$

где $\mathbf{x} \in \mathbb{R}^D$, $\mathbf{b}_1 \in \mathbb{R}^K$, $\mathbf{W} \in \mathbb{R}^{K \times D}$, $\mathbf{b}_2 \in \mathbb{R}^C$, $\mathbf{V} \in \mathbb{R}^{C \times K}$, D – размер входа, K – число скрытых блоков, C – число классов. Покажите, что градиенты по параметрам и входам имеют вид:

$$\nabla_{\mathbf{V}} \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial \mathbf{V}} \right]_{1,:} = \mathbf{u}_2 \mathbf{h}^\top \in \mathbb{R}^{C \times K}; \quad (13.116)$$

$$\nabla_{\mathbf{b}_2} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{b}_2} \right)^\top = \mathbf{u}_2 \in \mathbb{R}^C; \quad (13.117)$$

$$\nabla_{\mathbf{W}} \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial \mathbf{W}} \right]_{1,:} = \mathbf{u}_1 \mathbf{x}^\top \in \mathbb{R}^{K \times D}; \quad (13.118)$$

$$\nabla_{\mathbf{b}_1} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{b}_1} \right)^\top = \mathbf{u}_1 \in \mathbb{R}^K; \quad (13.119)$$

$$\nabla_{\mathbf{x}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{x}} \right)^\top = \mathbf{W}^\top \mathbf{u}_1 \in \mathbb{R}^D, \quad (13.120)$$

где градиенты функции потерь по обоим слоям (логитов и скрытому) равны:

$$\mathbf{u}_2 = \nabla_{\mathbf{a}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{a}} \right)^\top = (\mathbf{p} - \mathbf{y}) \in \mathbb{R}^C; \quad (13.121)$$

$$\mathbf{u}_1 = \nabla_{\mathbf{z}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{z}} \right)^\top = (\mathbf{V}^\top \mathbf{u}_2) \odot H(\mathbf{z}) \in \mathbb{R}^K, \quad (13.122)$$

а H – функция Хевисайда. Отметим, что в нашей нотации градиент (имеющий такую же форму, как переменная, по которой мы дифференцируем) равен транспонированной матрице Якоби, когда переменная является вектором, и первым срезом матрицы Якоби, когда переменная является матрицей.

Глава 14

Нейронные сети для изображений

14.1. ВВЕДЕНИЕ

В главе 13 мы обсуждали многослойные перцептроны (МСП) как способ обучить функции отображению «неструктурированных» входных векторов $\mathbf{x} \in \mathbb{R}^D$ в выходные. В этом разделе мы рассмотрим ситуацию, когда вход \mathbf{x} имеет двумерную пространственную структуру. (Похожие идеи применимы к одномерной временной структуре и к трехмерной пространственно-временной структуре.)

Чтобы понять, почему не стоит применять МСП непосредственно к данным изображения, вспомним, что основная операция МСП в каждом скрытом слое – вычисление активаций $\mathbf{z} = \varphi(\mathbf{W}\mathbf{x})$, где \mathbf{x} – вход слоя, \mathbf{W} – веса, а $\varphi()$ – нелинейная функция активации. Таким образом, j -й элемент скрытого слоя равен $z_j = \varphi(\mathbf{w}_j^T \mathbf{x})$. Внутреннюю операцию умножения можно рассматривать как сравнение входа \mathbf{x} с обученным шаблоном или паттерном \mathbf{w}_j ; если совпадение хорошее (большое положительное произведение), то активация этого блока будет велика (в предположении нелинейности ReLU), т. е. j -й паттерн присутствует во входных данных.

Однако это плохо работает, если входом является изображение переменного размера, $\mathbf{x} \in \mathbb{R}^{WHC}$, где W – ширина, H – высота, а C – количество входных каналов (например, $C = 3$ для цветовой модели RGB). Проблема в том, что нам пришлось бы обучать матрицу весов \mathbf{W} для каждого размера входного изображения. Кроме того, даже если бы размер входа был фиксирован, количество параметров оказалось бы запретительно большим для изображений разумного размера, поскольку матрица весов имела бы размер $(W \times H \times C) \times D$, где D – число выходов (скрытых блоков). И последняя проблема заключается в том, что паттерн, встретившийся в одном месте, может не распознаваться в другом – т. е. модель не будет **инвариантной относительно сдвига**, – поскольку никакого разделения весов между позициями нет (см. рис. 14.1).

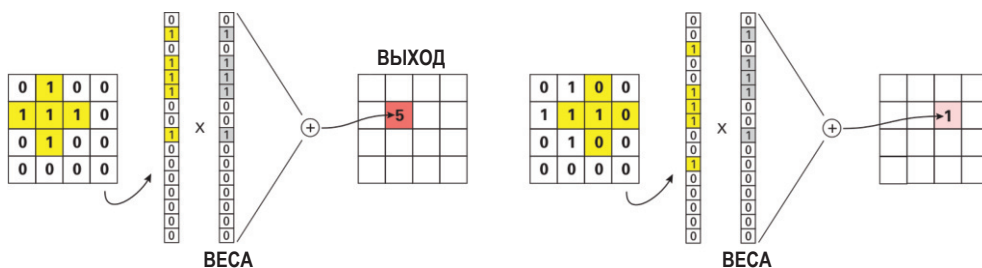


Рис. 14.1 ❖ Выявление паттернов в двумерных изображениях с помощью неструктурированного МСП работает плохо, потому что этот метод не инвариантен относительно сдвигов. Мы можем спроектировать вектор весов, который будет играть роль фильтра для обнаружения крестообразных форм. Он даст сильный отклик 5, если объект расположен слева, но слабый отклик 1, если сдвинуть тот же объект вправо. На основе рис. 7.16 из работы [SAV20]

Для решения этих проблем используются **сверточные нейронные сети (СНС)**, в которых умножение матриц заменено операцией свертки. Подробности мы отложим до раздела 14.2, но основная идея заключается в том, чтобы разбить входное изображение на пересекающиеся двумерные **патчи** и сравнить каждый патч с множеством небольших матриц весов, или **фильтров**, которые представляют части объекта (см. рис. 14.2). Это можно рассматривать как разновидность **сопоставления с шаблоном**. Шаблоны можно обучить на основании данных, как будет показано ниже. Поскольку шаблоны невелики (часто всего-то 3×3 или 5×5), число параметров значительно уменьшается. А поскольку для сравнения с шаблоном используется свертка, а не умножение матриц, модель оказывается инвариантной относительно сдвига. Это полезно в таких задачах, как классификация изображений, в которых наша цель – понять, присутствует объект или нет, независимо от его местоположения.



Рис. 14.2 ❖ Мы можем классифицировать цифру, отыскивая определенные характерные признаки (шаблоны изображений), встречающиеся в нужных (относительных) позициях. На основе рис. 5.1 из работы [Cho17]. Печатается с разрешения Франсуа Шолле

СНС имеют много приложений помимо классификации изображений, и мы обсудим их ниже в этой главе. Их также можно применять к одномерным (см. раздел 15.3) и трехмерным входам, но в этой главе предметом нашего интереса будет в основном двумерный случай.

14.2. НАИБОЛЕЕ УПОТРЕБИТЕЛЬНЫЕ СЛОИ

В этом разделе мы рассмотрим основы СНС.

14.2.1. Сверточные слои

Начнем с описания операции свертки на прямой и на плоскости, после чего расскажем, как она используется в СНС.

14.2.1.1. Свертка в одномерном случае

Сверткой двух функций $f, g : \mathbb{R}^D \rightarrow \mathbb{R}$ называется следующая функция:

$$[f \otimes g](z) = \int_{\mathbb{R}^D} f(u)g(z - u)du. \quad (14.1)$$

Теперь предположим, что вместо функций используются векторы конечной длины, которые будем интерпретировать как функции, определенные на конечном множестве точек. Например, пусть f вычисляется в точках $\{-L, -L + 1, \dots, 0, 1, \dots, L\}$, в результате чего получается вектор весов (называемый также **фильтром** или **ядром**) с элементами от $w_{-L} = f(-L)$ до $w_L = f(L)$. Пусть теперь g вычисляется в точках $\{-N, \dots, N\}$, в результате чего получается вектор признаков с элементами от $x_{-N} = g(-N)$ до $x_N = g(N)$. Тогда приведенная выше формула принимает вид:

$$[\mathbf{w} \otimes \mathbf{x}](i) = w_{-L}x_{i+L} + \dots + w_{-1}x_{i+1} + w_0x_i + w_1x_{i-1} + \dots + w_Lx_{i-L}. \quad (14.2)$$

(Ниже мы обсудим граничные условия (краевые эффекты)). Как видим, вектор весов \mathbf{w} «переворачивается» (т. е. порядок его элементов меняется на противоположный), а затем «протаскивается» по вектору \mathbf{x} с суммированием локальных окон в каждой точке (рис. 14.3).

Существует очень похожая операция, отличающаяся тем, что \mathbf{w} не переворачивается:

$$[\mathbf{w} * \mathbf{x}](i) = w_{-L}x_{i-L} + \dots + w_{-1}x_{i-1} + w_0x_i + w_1x_{i+1} + \dots + w_Lx_{i+L}. \quad (14.3)$$

Она называется **перекрестной корреляцией**. Если вектор весов симметричен, как это часто бывает, то перекрестная корреляция и свертка – одно и то же. В литературе по глубокому обучению под «сверткой» обычно понимается перекрестная корреляция, мы будем придерживаться этого соглашения.

–	–	1	2	3	4	–	–	
7	6	5	–	–	–	–	–	$z_0 = x_0 w_0 = 5$
–	7	6	5	–	–	–	–	$z_1 = x_0 w_1 + x_1 w_0 = 16$
–	–	7	6	5	–	–	–	$z_2 = x_0 w_2 + x_1 w_1 + x_2 w_0 = 34$
–	–	–	7	6	5	–	–	$z_3 = x_1 w_2 + x_2 w_1 + x_3 w_0 = 52$
–	–	–	–	7	6	5	–	$z_4 = x_2 w_2 + x_3 w_1 = 45$
–	–	–	–	–	7	6	5	$z_5 = x_3 w_2 = 28$

Рис. 14.3 ❖ Дискретная свертка $\mathbf{x} = [1, 2, 3, 4]$ с $\mathbf{w} = [5, 6, 7]$ дает $\mathbf{z} = [5, 16, 34, 52, 45, 28]$. Как видим, операция сводится к «переворачиванию» \mathbf{w} и «протаскиванию» его по \mathbf{x} , при этом производится поэлементное умножение и сложение результатов

Мы можем вычислить веса \mathbf{w} на множестве $\{0, 1, \dots, L - 1\}$, а признаки \mathbf{x} – на множестве $\{0, 1, \dots, N - 1\}$, чтобы исключить отрицательные индексы. Тогда приведенная выше формула принимает вид:

$$[\mathbf{w} \circledast \mathbf{x}](i) = \sum_{u=0}^{L-1} w_u x_{i+u}. \quad (14.4)$$

Смотрите пример на рис. 14.4.

Вход								Вход			Выход					
0	1	2	3	4	5	6	*	1	2	=	2	5	8	11	14	17

Рис. 14.4 ❖ Одномерная перекрестная корреляция.
На основе рис. 15.3.2 из работы [Zha+20].
Печатается с разрешения Астона Чжана

14.2.1.2. Свертка в двумерном случае

В двумерном случае формула (14.4) принимает вид:

$$[\mathbf{W} \circledast \mathbf{X}](i, j) = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} w_{u,v} x_{i+u, j+v}, \quad (14.5)$$

где двумерный фильтр \mathbf{W} имеет размер $H \times W$. Например, рассмотрим свертку входа \mathbf{X} размера 3×3 с ядром \mathbf{W} размера 2×2 для вычисления выхода \mathbf{Y} размера 2×2 :

$$\mathbf{Y} = \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix} \circledast \begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{pmatrix} \quad (14.6)$$

$$= \begin{pmatrix} (w_1x_1 + w_2x_2 + w_3x_4 + x_4x_5) & (w_1x_2 + w_2x_3 + w_3x_5 + x_4x_6) \\ (w_1x_4 + w_2x_5 + w_3x_7 + x_4x_8) & (w_1x_5 + w_2x_6 + w_3x_8 + x_4x_9) \end{pmatrix}. \quad (14.7)$$

На рис. 14.5 этот процесс показан наглядно.

Вход				Вход			Выход	
0	1	2	*	0	1	=	19	25
3	4	5		2	3		37	43
6	7	8						

Рис. 14.5 ❖ Двумерная перекрестная корреляция.
Построено программой по адресу figures.probl.ai/book1/14.5.
На основе рис. 6.2.1 из работы [Zha+20]

Двумерную свертку можно рассматривать как **сопоставление с шаблоном**, поскольку выход в точке (i, j) будет велик, если патч с центром в этой точке похож на **W**. Если **W** соответствует ориентированному ребру, то после свертки с ним на выходной **тепловой карте** «загорятся» области с такой же ориентацией, как показано на рис. 14.6. В общем случае свертку можно интерпретировать как форму **выявления признаков**. Поэтому выход $Y = W \otimes X$ называется **картой признаков**.

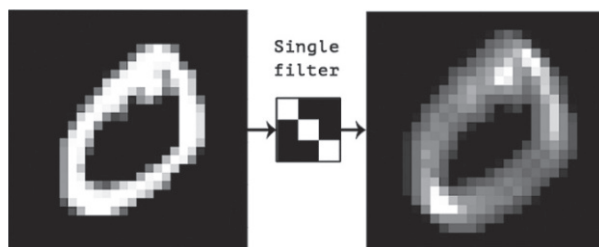


Рис. 14.6 ❖ Свертка двумерного изображения (слева) с фильтром 3×3 (в центре) порождает двумерную карту откликов (справа). Яркие точки на этой карте соответствуют участкам изображения, содержащим диагональные отрезки, направленные из левого верхнего в правый нижний угол. На основе рис. 5.3 из работы [Cho17]. Печатается с разрешения Франсуа Шолле

14.2.1.3. Свертка как умножение матрицы на вектор

Поскольку свертка – линейный оператор, мы можем представить ее в виде умножения на матрицу. Например, рассмотрим формулу (14.7). Мы можем ее переписать как умножение матрицы на вектор, развернув двумерную матрицу **X** в одномерный вектор **x** и умножив его на матрицу **C**, структурно похожую на матрицу Тёплица и выведенную из ядра **W** следующим образом:

$$\mathbf{y} = \mathbf{C}\mathbf{x} = \begin{pmatrix} w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{pmatrix} \quad (14.8)$$

$$= \begin{pmatrix} w_1x_1 + w_2x_2 + w_3x_4 + w_4x_5 \\ w_1x_2 + w_2x_3 + w_3x_5 + w_4x_6 \\ w_1x_4 + w_2x_5 + w_3x_7 + w_4x_8 \\ w_1x_5 + w_2x_6 + w_3x_8 + w_4x_9 \end{pmatrix}. \quad (14.9)$$

Для восстановления формы выхода 2×2 нужно преобразовать вектор \mathbf{y} размера 4×1 обратно в \mathbf{Y}^1 .

Таким образом, мы видим, что СНС похожа на МСП, в котором матрица весов имеет специальную разреженную структуру, а элементы связаны в пространственных направлениях. Тем самым реализуется идея инвариантности относительно сдвига, и число параметров значительно сокращается по сравнению с матрицей весов в стандартном полносвязном, или плотном, слое, используемом в МСП.

14.2.1.4. Граничные условия и дополнение

В формуле (14.7) мы видели, что в результате свертки изображения 3×3 с фильтром 2×2 получается выход размера 2×2 . В общем случае свертка фильтра $f_h \times f_w$ с изображением размера $x_h \times x_w$ порождает выход размера $(x_h - f_h + 1) \times (x_w - f_w + 1)$; это называется **корректной сверткой**, потому что мы применяем фильтр только к «корректным» частям входа, не позволяя ему «вылезать за край». Если мы хотим, чтобы размер выхода был таким же, как размер входа, то можно использовать **дополнение нулями**, т. е. добавление к изображению рамки, состоящей из нулей, как показано на рис. 14.7. Это называется **конгруэнтной сверткой**.

В общем случае, если вход имеет размер $x_h \times x_w$, используется ядро размера $f_h \times f_w$ и с каждой стороны производится дополнение нулями шириной соответственно p_h и p_w , то выход будет иметь следующий размер [DV16]:

$$(x_h + 2p_h - f_h + 1) \times (x_w + 2p_w - f_w + 1). \quad (14.10)$$

Например, рассмотрим рис. 14.8а. Имеем $p = 1$, $f = 3$, $x_h = 5$ и $x_w = 7$, поэтому размер выхода равен

$$(5 + 2 \cdot 1 - 3 + 1) \times (7 + 2 - 3 + 1) = 5 \times 7. \quad (14.11)$$

¹ См. демонстрационную программу по адресу code.problml.ai/book1/conv2d_torch.

Если положить $2p = f - 1$, то размер выхода будет таким же, как размер входа.

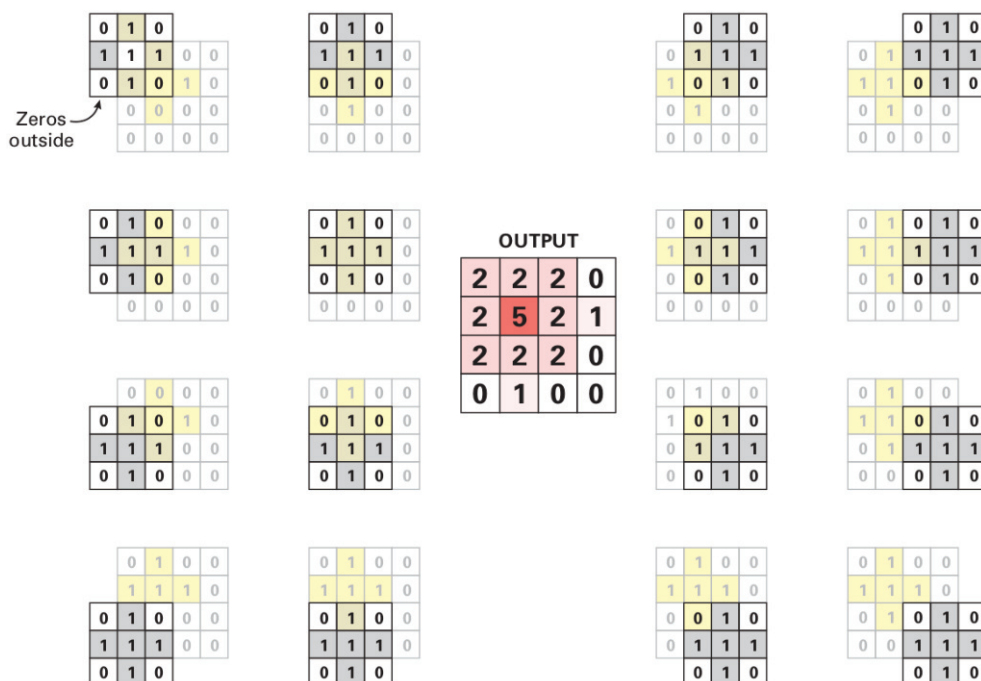


Рис. 14.7 ❖ Конгруэнтная свертка (с дополнением нулями) гарантирует, что выход будет иметь такой же размер, как вход.
На основе рис. 8.3 из работы [SAV20]

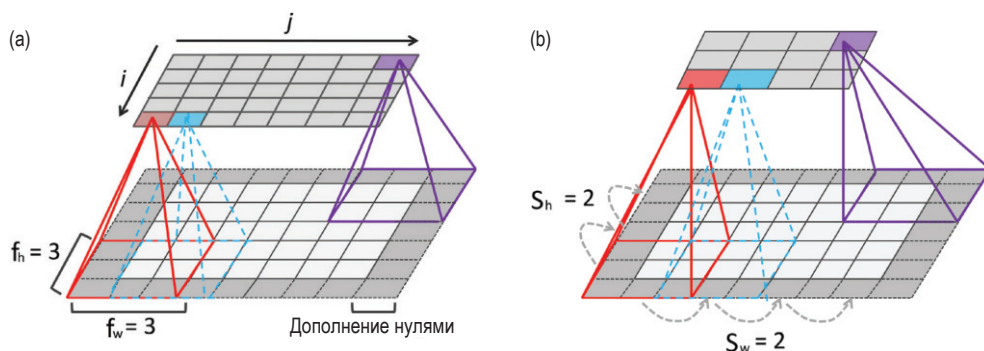


Рис. 14.8 ❖ Дополнения и шаги при двумерной свертке. (a) Мы применяем «конгруэнтную свертку» к входу размером 5×7 (с дополнением нулями) и фильтру 3×3 , получая выход размером 5×7 . (b) Здесь используется шаг 2, поэтому выход имеет размер 3×4 . На основе рис. 14.3–14.4 из работы [Gér19].

14.2.1.5. Свертка с шагом

Поскольку каждый выходной пиксель является взвешенной комбинацией входов в его рецептивном поле (зависящем от размера фильтра), соседние выходные пиксели будут иметь близкие значения, потому что множества входов, на основе которых они вычисляются, пересекаются. Эту избыточность можно уменьшить (а значит, ускорить вычисления), если пропускать каждый s -й вход. Это называется **сверткой с шагом**. Она показана на рис. 14.8b, где вычисляется свертка изображения 5×7 и фильтра 3×3 с шагом 2, в результате чего получается выход размера 3×4 .

В общем случае, если вход имеет размер $x_h \times x_w$, используется ядро размера $f_h \times f_w$, с каждой стороны производится дополнение нулями шириной соответственно p_h и p_w и размеры шага равны s_h и s_w , то выход будет иметь следующий размер [DV16]:

$$\left\lfloor \frac{x_h + 2p_h - f_h + s_h}{s_h} \right\rfloor \times \left\lfloor \frac{x_w + 2p_w - f_w + s_w}{s_w} \right\rfloor. \quad (14.12)$$

Например, рассмотрим рис. 14.8b, на котором размер шага $s = 2$. Теперь выход меньше входа:

$$\left(\left\lfloor \frac{5 + 2 - 3 + 2}{2} \right\rfloor, \left\lfloor \frac{7 + 2 - 3 + 2}{2} \right\rfloor \right) = \left(\left\lfloor \frac{6}{2} \right\rfloor, \left\lfloor \frac{4}{1} \right\rfloor \right) = 3 \times 4. \quad (14.13)$$

14.2.1.6. Несколько входных и выходных каналов

На рис. 14.6 входом являлось черно-белое изображение. В общем случае вход может иметь несколько **каналов** (например, RGB или гиперспектральные полосы на спутниковых изображениях). Мы можем обобщить понятие свертки на этот случай, определив ядро для каждого входного канала; таким образом, теперь \mathbf{W} – трехмерная матрица весов, или **тензор**. Для вычисления выхода мы свертываем канал c входа с ядром $\mathbf{W}_{:, :, c}$, а затем производим суммирование по каналам:

$$z_{i,j} = b + \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} \sum_{c=0}^{C-1} x_{si+u, sj+v, c} w_{u,v, c}, \quad (14.14)$$

где s – шаг (для простоты предполагается, что шаг одинаков в обоих направлениях), а b – смещение. Это показано на рис. 14.9.

Каждая матрица весов может выявить один вид признаков. Обычно мы хотим выявлять признаки разных видов, как показано на рис. 14.2. Это можно сделать, превратив \mathbf{W} в 4-мерную матрицу весов. Фильтр для выявления признаков типа d во входном канале c хранится в подматрице $\mathbf{W}_{:, :, c, d}$. Обобщим определение свертки на этот случай следующим образом:

$$z_{i,j,d} = b_d + \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} \sum_{c=0}^{C-1} x_{si+u, sj+v, c} w_{u,v, c, d}. \quad (14.15)$$

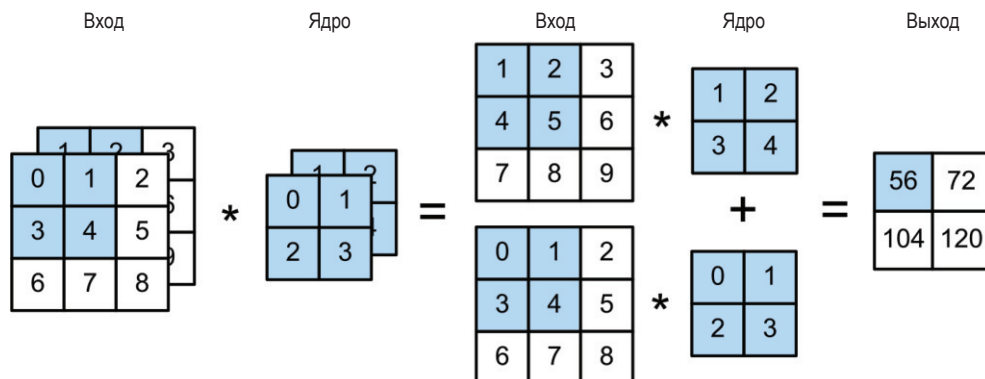


Рис. 14.9 ❖ Двумерная свертка применительно к входу с двумя каналами.
 Построено программой по адресу figures.problml.ai/book1/14.9.
 На основе рис. 6.4.1 из работы [Zha+20]

Это показано на рис. 14.10. Каждый вертикальный цилиндр обозначает множество выходных признаков в данной точке, $z_{i,j,1,D}$; иногда его называют **гиперстолбцом**. Каждый элемент является взвешенной комбинацией C признаков в рецептивном поле каждой из карт признаков в нижележащем слое¹.

14.2.1.7. Свертка 1×1 (поточечная)

Иногда мы просто хотим взять взвешенную комбинацию признаков в данной точке, а не в нескольких точках. Это можно сделать с помощью **свертки 1×1** , или **поточечной свертки**. При этом число каналов меняется с C на D без изменения пространственной размерности:

$$z_{i,j,d} = b_d + \sum_{c=0}^{C-1} x_{i,j,c} w_{0,0,c,d}. \quad (14.16)$$

Это можно рассматривать как однослойный МСП, применяемый параллельно к каждому столбцу признаков.

14.2.2. Пулинговые слои

Свертка сохраняет информацию о положении входных признаков (с учетом уменьшенной разрешающей способности). Это свойство называется **эквивариантностью**. В некоторых случаях нам нужна инвариантность относительно местоположения. Например, в процессе классификации изображений нам иногда нужно только знать, присутствует ли некоторый объект (скажем, лицо) где-нибудь в изображении.

¹ В Tensorflow фильтр для двумерной СНС имеет форму (H, W, C, D) , а мини-пакет карт признаков – форму (размер-пакета, высота-изображения, ширина-изображения, каналы-изображения); такой формат называется **NHWC**. В других системах форматы данных иные.

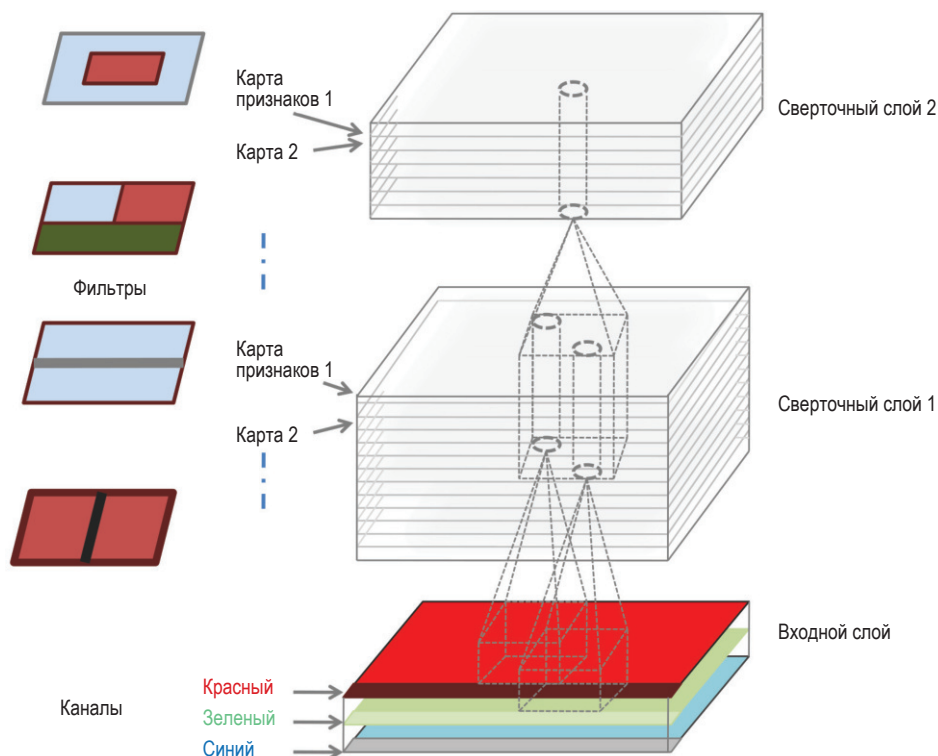


Рис. 14.10 ❖ СНС с двумя сверточными слоями. Вход имеет три цветовых канала. Карты признаков во внутренних слоях имеют несколько каналов. Цилиндры соответствуют гиперстолбцам – векторам признаков в определенной точке. На основе рис. 14.6 из работы [Gér19]

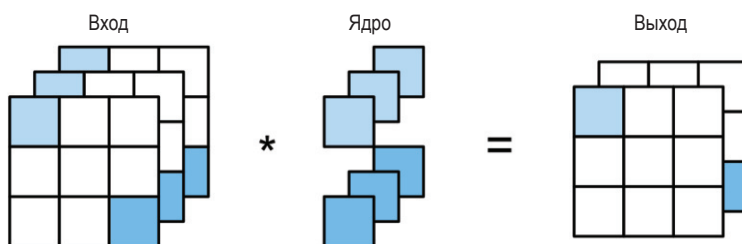


Рис. 14.11 ❖ Отображение трех каналов в два с помощью свертки с фильтром размера $1 \times 1 \times 3 \times 2$. На основе рис. 6.4.2 из работы [Zha+20]

Простой способ решить эту задачу называется **max-пулингом** и заключается в вычислении максимума поданных на вход значений, как показано на рис. 14.12. Альтернатива – использовать **пулинг усреднением**, когда максимум заменяется средним. В любом случае отклик выходного нейрона один и тот же независимо от того, в каком месте его рецептивного поля находится входной паттерн. (Отметим, что пулинг применяется к каждому каналу независимо.)

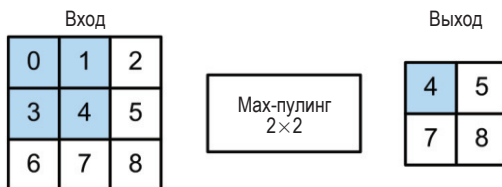


Рис. 14.12 ❖ Мак-пулинг с фильтром 2×2 и шагом 1.
На основе рис. 6.5.1 из работы [Zha+20]

Если усреднение производится по всем позициям карты признаков, то этот метод называется **пулингом глобальным усреднением**. Таким образом, мы можем преобразовать карту признаков размера $H \times W \times D$ в карту размера $1 \times 1 \times D$, трансформировать ее в D -мерный вектор, затем передать этот вектор полносвязному слою и отобразить в C -мерный вектор, который наконец-то передать слою softmax для формирования выхода. Использование пулинга глобальным усреднением означает, что классификатор можно применить к изображению любого размера, поскольку окончательная карта признаков всегда будет преобразована в вектор фиксированной размерности D перед отображением в распределение C классов.

14.2.3. Соберем все вместе

Типичный паттерн проектирования заключается в том, чтобы создать СНС, в которой сверточные слои чередуются с пулинговыми, а в конце находится слой линейной классификации. Это показано на рис. 14.13. (В этом примере мы опустили слои нормировки, поскольку модель не слишком глубокая.) Впервые этот паттерн проектирования был применен в **неокогнитроне** Фукусимы [Fuk75], а в его основу легла модель Хубеля и Визеля, описывающая простые и сложные клетки в зрительной коре головного мозга человека [HW62]. В 1998 году Ян Лекун применил аналогичный дизайн в своей модели **LeNet** [LeC+98], где для оценки параметров использовались обратное распространение и СГС. Этот паттерн по-прежнему пользуется популярностью в моделях распознавания зрительных образов, построенных по образцу биологических нервных систем [RP99], а также в различных практических приложениях (см. разделы 14.3 и 14.5).

14.2.4. Слои нормировки

Базовая схема, изображенная на рис. 14.13, хорошо работает для мелких СНС, но с трудом масштабируется на более глубокие модели из-за проблем исчезающего или взрывного градиента, рассмотренных в разделе 13.4.2. Общепринятое решение этой проблемы – включить в модель дополнительные слои для стандартизации распределения скрытых блоков (т. е. приведения их среднего к нулю, а дисперсии – к единице) – точно так же как мы поступаем с входами многих моделей. Ниже мы обсудим различные виды **слоев нормировки**.

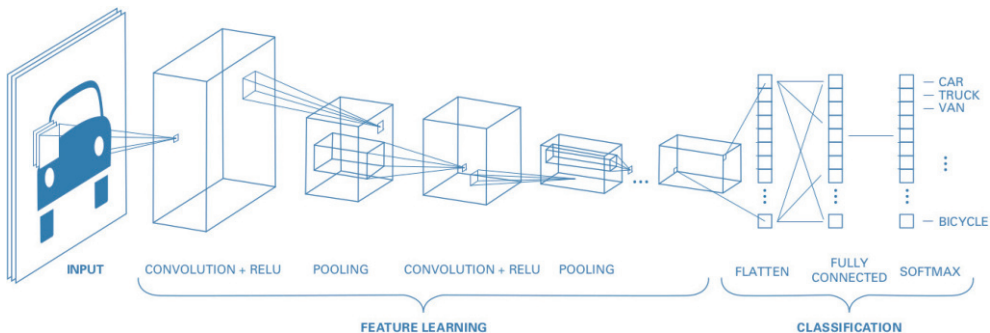


Рис. 14.13 ❖ Простая СНС для классификации изображений.

На основе рисунка из статьи по адресу
<https://blog.floydhub.com/building-your-first-convnet/>

14.2.4.1. Пакетная нормировка

Самый популярный слой нормировки называется **пакетной нормировкой** (batch normalization – BN) [IS15]. Он отвечает за то, чтобы распределение активаций в одном слое имело нулевое среднее и единичную дисперсию при усреднении по всем примерам из мини-пакета. Точнее, мы заменяем вектор активации \mathbf{z}_n (или иногда вектор преактивации \mathbf{a}_n) для примера n (в некотором слое) вектором $\tilde{\mathbf{z}}_n$, вычисляемым следующим образом:

$$\tilde{\mathbf{z}}_n = \gamma \odot \hat{\mathbf{z}}_n + \beta; \quad (14.17)$$

$$\hat{\mathbf{z}}_n = \frac{\mathbf{z}_n - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \varepsilon}}; \quad (14.18)$$

$$\boldsymbol{\mu}_B = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{z} \in \mathcal{B}} \mathbf{z}; \quad (14.19)$$

$$\boldsymbol{\sigma}_B^2 = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{z} \in \mathcal{B}} (\mathbf{z} - \boldsymbol{\mu}_B)^2, \quad (14.20)$$

где \mathcal{B} – мини-пакет, содержащий пример n , $\boldsymbol{\mu}_B$ – среднее активаций по этому пакету¹, $\boldsymbol{\sigma}_B^2$ – соответствующая дисперсия, $\hat{\mathbf{z}}_n$ – стандартизованный вектор активаций, $\tilde{\mathbf{z}}_n$ – он же после сдвига и масштабирования (выход слоя BN), β и γ – обучаемые параметры для этого слоя, а $\varepsilon > 0$ – небольшая постоянная. Поскольку это преобразование дифференцируемое, мы легко можем передать градиенты назад на вход слоя и в параметры BN β и γ .

Применительно к входному слою пакетная нормировка эквивалентна обычной процедуре стандартизации, описанной в разделе Section 10.2. Заме-

¹ Применительно к сверточному слою мы усредняем по пространственным точкам и по примерам, но не по каналам (поэтому длина $\boldsymbol{\mu}$ равна числу каналов). Применительно к полносвязному слою мы усредняем просто по примерам (длина $\boldsymbol{\mu}$ равна ширине слоя).

тим, что среднее и дисперсию входного слоя можно вычислить однократно, поскольку эти данные статические. Однако эмпирические средние и дисперсии внутренних слоев изменяются по мере подбора параметров. (Иногда это называют «**внутренним дрейфом ковариат**» (internal covariate shift).) Именно поэтому необходимо пересчитывать μ и σ^2 для каждого мини-пакета.

На этапе тестирования у нас может быть единственный вход, поэтому вычислить статистику мини-пакета невозможно. Стандартное решение такое: после обучения вычислить μ_i и σ_i^2 по всем примерам из обучающего набора (т. е. на полном пакете), а затем «заморозить» эти параметры и добавить их в список других параметров слоя, т. е. β_i и γ_i . Тогда на этапе тестирования мы можем использовать эти замороженные значения в качестве μ_i и σ_i^2 , а не вычислять статистику по тестовому пакету. Таким образом, если в модели используется BN, то мы должны указывать, применяется она для вывода или для обучения. (Пример кода см. в файле `batchnorm_torch.ipynb`.)

Для ускорения работы мы можем объединить слой пакетной нормировки с заморозкой с предыдущим слоем. Именно, предположим, что предыдущий слой вычисляет $\mathbf{XW} + \mathbf{b}$; его объединение с BN дает $\gamma \odot (\mathbf{XW} + \mathbf{b} - \mu)/\sigma + \beta$. Положив $\mathbf{W}' = \gamma \odot \mathbf{W}/\sigma$ и $\mathbf{b}' = \gamma \odot (\mathbf{b} - \mu)/\sigma + \beta$, мы сможем переписать объединенные слои в виде $\mathbf{W}' + \mathbf{b}'$. Это называется **объединенной пакетной нормировкой** (fused batchnorm). Похожие приемы можно применить для ускорения BN на этапе обучения [Jun+19].

Выигрыш от пакетной нормировки (в плане скорости и устойчивости обучения) может оказаться весьма значительным, особенно для глубоких СНС. Причины этого пока неясны, но похоже, что BN делает поверхность оптимизируемой функции гораздо более гладкой [San+18b]. Кроме того, нормировка уменьшает чувствительность к скорости обучения [ALL18]. Помимо вычислительных, у нее есть и статистические преимущества. В частности, BN действует как регуляризатор; в самом деле, можно показать, что она эквивалентна одной из форм приближенного байесовского вывода [TAS18; Luo+19].

Однако слишком полагаться на мини-пакет данных не стоит, так как это чревато рядом проблем. В частности, при обучении на малых пакетах оценки параметров могут быть неустойчивы, хотя недавний вариант этого метода, **пакетная перенормировка** [Iof17], частично решает эту проблему. Ниже обсуждаются некоторые альтернативы пакетной нормировке.

14.2.4.2. Другие виды слоя нормировки

В разделе 14.2.4.1 мы обсуждали **пакетную нормировку**, которая стандартизирует все активации в данном канале признаков, приводя среднее к нулю, а дисперсию к единице. Это может существенно помочь на этапе обучения и увеличить его скорость (см. пример кода по адресу `code.problml.ai/book1/batchnorm_torch`).

Хотя пакетная нормировка работает хорошо, она сталкивается с трудностями, когда размер пакета мал, потому что в этом случае оценки среднего и дисперсии параметров ненадежны. Возможное решение – вычислять среднее и дисперсию путем пулинга статистик по другим измерениям тензора, а не по примерам из пакета. Точнее, обозначим z_i i -й элемент тензора;

в случае двумерных изображений индекс i состоит из четырех компонент, описывающих пакет, канал, высоту и ширину: $i = (i_N, i_C, i_H, i_W)$. Мы вычисляем среднее и стандартное отклонение для каждого индекса z_i следующим образом:

$$\mu_i = \frac{1}{|\mathcal{S}_i|} \sum_{k \in \mathcal{S}_i} z_k, \quad \sigma_i = \sqrt{\frac{1}{|\mathcal{S}_i|} \sum_{k \in \mathcal{S}_i} (z_k - \mu_i)^2 + \varepsilon}, \quad (14.21)$$

где \mathcal{S}_i – множество элементов, по которым производится усреднение. Затем мы вычисляем $\hat{z}_i = (z_i - \mu_i)/\sigma_i$ и $\tilde{z}_i = \gamma_c \hat{z}_i + \beta_c$, где c – канал, соответствующий индексу i .

В случае пакетной нормировки мы выполняем пулинг по пакету, высоте и ширине, так что \mathcal{S}_i – множество всех элементов тензора с индексом канала i . Чтобы избежать проблем, возникающих для малых пакетов, мы можем вместо этого выполнять пулинг по каналу, высоте и ширине, оставляя фиксированным индекс пакета. Это называется **нормировкой по слою** [BKH16] (см. пример кода по адресу code.problml.ai/book1/layer_norm_torch). Или же можно иметь отдельные параметры нормировки для каждого примера в пакете и для каждого канала. Это называется **индивидуальной нормировкой** [UVL16].

Естественное обобщение рассмотренных выше методов – **групповая нормировка** [WH18], когда пулинг производится по всем точкам, для которых канал принадлежит той же группе, что у i -го компонента. Все варианты показаны на рис. 14.14. Нормировка по слою – частный случай, когда имеется единственная группа, содержащая все каналы. Индивидуальная нормировка – частный случай, когда имеется C групп, по одной на канал. В работе [WH18] экспериментально показано, что может быть выгоднее (с точки зрения скорости обучения, а также верности на этапах обучения и тестирования) использовать группы, более крупные, чем отдельные каналы, но более мелкие, чем все каналы вместе взятые.

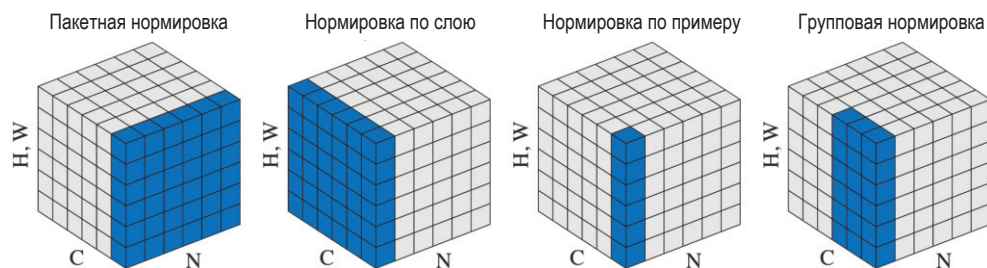


Рис. 14.14 ❖ Различные методы нормировки активаций для СНС. На каждом рисунке показан тензор карты признаков, где N – ось пакета, C – ось каналов, а (H, W) – пространственные оси. Синие пиксели нормированы на одно и тот же среднее и дисперсию, вычисленные путем агрегирования значений этих пикселей. Слева направо: пакетная нормировка, нормировка по слою, нормировка по примеру, групповая нормировка (с двумя группами по три канала). На основе рис. 2 из работы [WH18]. Печатается с разрешения Кайминь Хе

14.2.4.3. Сети без нормировки

В недавней работе [Bro+21] предложен метод, названный **сетями без нормировки**, позволяющий обучать глубокие остаточные сети, не используя пакетную нормировку или еще какой-либо слой нормировки. Идея в том, чтобы заменить нормировку адаптивным отсечением градиента, преследующим ту же цель – избежать неустойчивости обучения. То есть мы используем формулу (13.70), но динамически изменяем порог отсечения. Обучение такой модели происходит быстрее, а результаты оказываются более верными, чем у сопоставимых моделей, обученных с применением пакетной нормировки.

14.3. РАСПРОСТРАНЕННЫЕ АРХИТЕКТУРЫ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ

Общепринято использовать СНС для классификации изображений, т. е. оценивания функции $f: \mathbb{R}^{H \times W \times K} \rightarrow \{0, 1\}^C$, где K – число входных каналов (например, $K = 3$ для RGB-изображений), а C – число классов.

В этом разделе мы дадим краткий обзор различных СНС, разработанных за много лет для решения задач классификации изображений. Более полный обзор СНС см., например, в [Kha+20], а актуальный репозиторий кода и моделей (на PyTorch) – на сайте <https://github.com/rwightman/pytorch-image-models>.

14.3.1. LeNet

Одна из самых ранних СНС, LeNet, была создана в 1998 году [LeC+98] Яном Лекуном. Она проектировалась для классификации рукописных цифр и обучалась на наборе данных MNIST, описанном в разделе 3.5.2. Эта модель показана на рис. 14.15 (см. также рис. 14.16а, где она представлена более компактно). Некоторые предсказания этой модели показаны на рис. 14.17. После всего одной эпохи верность на тестовом наборе уже составляет 98.8 %. Для сравнения, верность МСП из раздела 13.2.4.2 после одной эпохи составляет 95.9 %. Если увеличить число эпох обучения, то верность достигает уровня, при котором качество неотлично от зашумленности меток (примеры коды см. по адресу code.problml.ai/book1/lenet_torch).

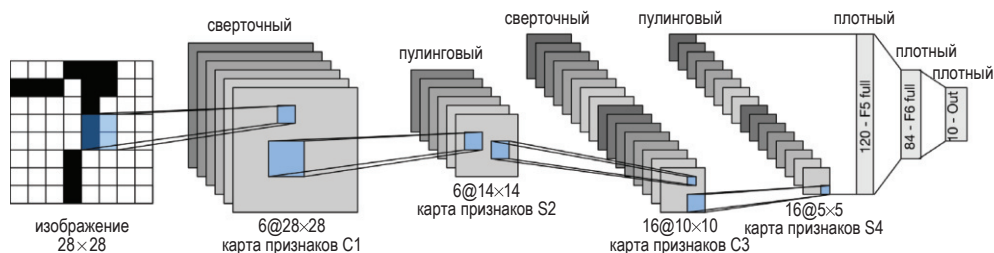


Рис. 14.15 ❖ LeNet5, сверточная нейронная сеть для классификации рукописных цифр. На основе рис. 6.6.1 из работы [Zha+20]. Печатается с разрешения Астона Чжана

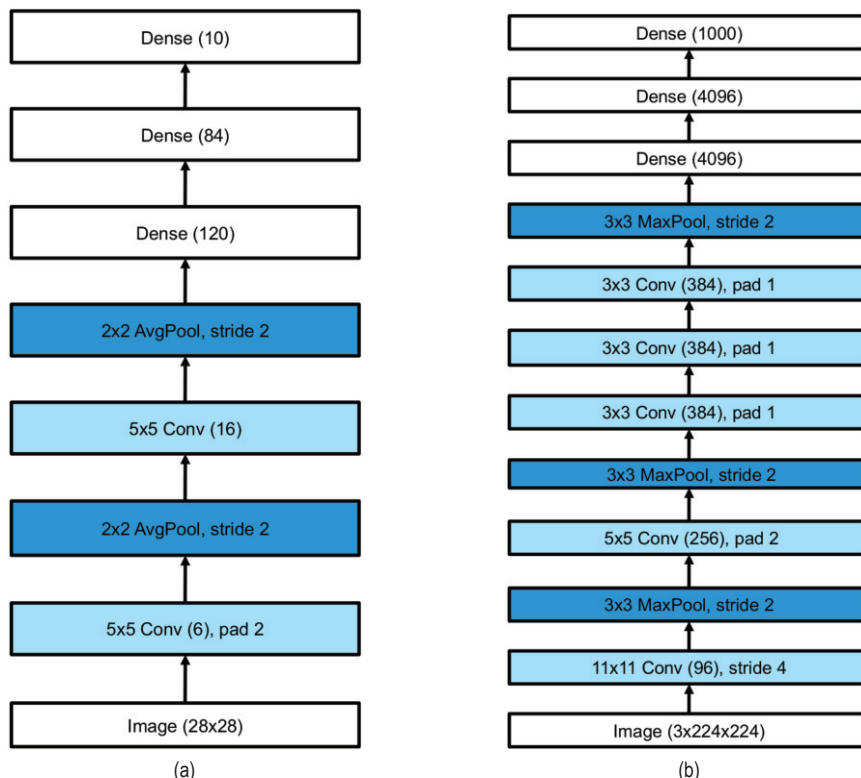


Рис. 14.16 ❖ (a) LeNet5. Предполагается, что размер входа $1 \times 28 \times 28$, как в случае MNIST. На основе рис. 6.6.2 из работы [Zha+20]. Печатается с разрешения Астона Чжана. (b) AlexNet. Предполагается, что размер входа $3 \times 224 \times 224$, как в случае кадрированных и масштабированных изображений из набора ImageNet. На основе рис. 7.1.2 из работы [Zha+20]. Печатается с разрешения Астона Чжана

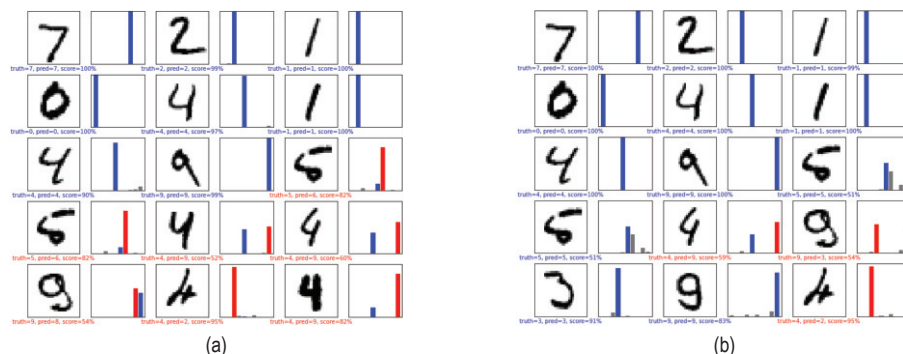


Рис. 14.17 ❖ Результаты применения ЧНС к некоторым изображениям из набора MNIST (выбранным специально, чтобы были видны и ошибки). Красным цветом показано неправильное распознавание, синим – правильное. (a) После одной эпохи обучения. (b) После двух эпох. Построено программой по адресу figures.probl.ai/book1/14.17

Конечно, классификация отдельных цифр имеет ограниченную применимость; люди-то обычно записывают строки цифр или букв. В этом случае требуется не только классификация, но и сегментация. Лекун с сотрудниками придумали, как для решения этой задачи объединить сверточные нейронные сети с моделью, напоминающей условное случайное поле. Эта система была внедрена почтой США. Более подробное описание системы см. в работе [LeC+98].

14.3.2. AlexNet

Хотя СНС к тому времени существовали уже много лет, лишь после выхода работы [KSH12] в 2012 году на них обратили внимание исследователи, занимающиеся компьютерным зрением. Авторы показали, как уменьшить частоту ошибок (на первых пяти результатах) на наборе ImageNet (раздел 1.5.1.2) с предыдущих 26 до 15 % – это считалось очень серьезным улучшением. Модель получила название **AlexNet** по имени ее создателя, Алекса Крижевского.

На рис. 14.16b показана ее архитектура. Модель очень похожа на LeNet (рис. 14.16a) со следующими отличиями: она глубже (восемь слоев с настраиваемыми параметрами, т. е. без учета пулинговых слоев, вместо пяти); используются нелинейности ReLU вместо tanh (объяснение, почему это важно, см. в разделе 13.2.3); для регуляризации используется прореживание (раздел 13.5.4), а не уменьшение весов; несколько сверточных слоев следуют друг за другом подряд, а не чередуются с пулинговыми слоями. Преимущество нескольких сверточных слоев подряд в том, что рецептивное поле становится больше, потому что выход одного слоя подается на вход другого (например, три фильтра 3×3 имеют рецептивное поля размера 7×7). Это лучше, чем использование одного слоя с большим рецептивным полем, потому что между несколькими слоями расположены нелинейности. Кроме того, у трех фильтров 3×3 параметров меньше, чем у одного фильтра 7×7 .

Отметим, что сеть AlexNet имеет 60 млн свободных параметров (гораздо больше, чем количество помеченных примеров – 1 млн) в основном из-за трех полносвязных слоев на выходе. Для обучения этой модели было использовано два GPU (поскольку в то время память GPU была ограничена), и это считается образцом инженерной изобретательности¹. На рис. 1.14a показаны предсказания, сделанные этой моделью для нескольких изображений из набора ImageNet.

¹ Все три автора этой работы (Алекс Крижевски, Илья Суцкевер и Джеффри Хинтов) впоследствии были приняты на работу в Google, хотя Илья ушел оттуда в 2015 году, а Алекс в 2017. Более подробно об этой истории см. в статье <https://en.wikipedia.org/wiki/AlexNet>. Отметим, что AlexNet была не первой СНС, реализованной на GPU; эта честь принадлежит группе из кооперации Microsoft [CPS06], которая добилась 4-кратного ускорения по сравнению с CPU, а затем авторам работы [Cir+11], добившимся 60-кратного ускорения.

14.3.3. GoogLeNet

Компания Google разработала модель под названием **GoogLeNet** [Sze+15b] (название получено объединением слов Google и LeNet). Ее главное отличие от предшествующих моделей заключается в использовании ранее не встречавшегося **начального блока** (inception block)¹, содержащего несколько параллельных путей, на которых используются сверточные фильтры разного размера (см. иллюстрацию на рис. 14.18). Это позволяет модели обучиться оптимальному размеру фильтра на каждом уровне. Модель в целом состоит из 9 начальных блоков, за которыми следует слой пулинга глобальным усреднением (см. иллюстрацию на рис. 14.19). После обнародования этой модели было предложено несколько ее обобщений; детали можно найти в работах [IS15; Sze+15a; SIV17].

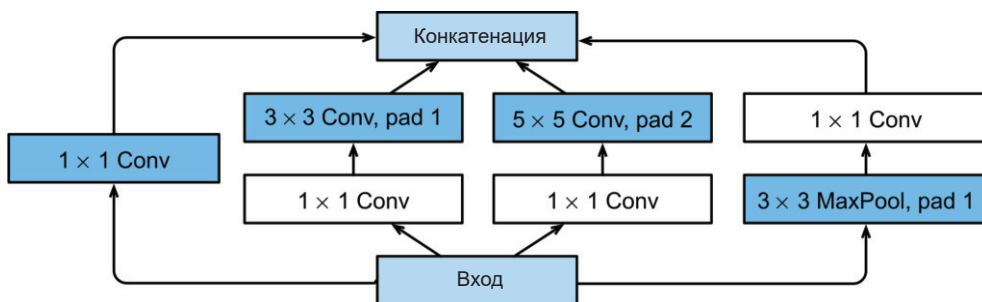


Рис. 14.18 ❖ Начальный модуль. Сверточные слои 1×1 уменьшают количество каналов, сохраняя пространственные измерения. Параллельные пути, содержащие свертки с фильтрами разных размеров, позволяют модели обучиться тому, какой размер лучше всего подходит для каждого слоя. Финальный блок конкатенации объединяет выходы всех путей (их пространственные размеры одинаковы). На основе рис. 7.4.1 из работы [Zha+20]. Печатается с разрешения Астона Чжана

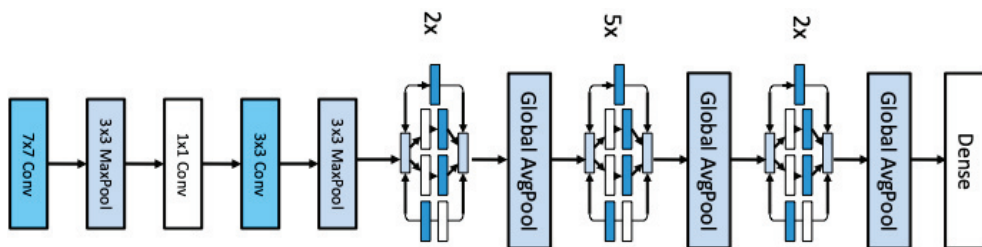


Рис. 14.19 ❖ GoogLeNet (немного модифицированная по сравнению с оригиналом). Входной слой слева. На основе рис. 7.4.2 из работы [Zha+20]. Печатается с разрешения Астона Чжана

¹ Термин происходит от названия фильма «Inception» (в российском прокате – «Начало»), где герой произносит фразу «We need to go deeper» («Нужно идти глубже»). В 2014 году она стала популярным интернет-мемом.

14.3.4. ResNet

Победителем конкурса по классификации набора ImageNet в 2015 году стала команда Microsoft, предложившая модель **ResNet** [He+16a]. Ее основная идея состояла в том, чтобы заменить $\mathbf{x}_{l+1} = \mathcal{F}_l(\mathbf{x}_l)$ на

$$\mathbf{x}_{l+1} = \varphi(\mathbf{x}_l + \mathcal{F}_l(\mathbf{x}_l)). \quad (14.22)$$

Это называется **остаточным блоком**, потому что \mathcal{F}_l обучается только остатку, или разности между входом и выходом слоя, что является более простой задачей. В работе [He+16a] \mathcal{F} имеет вид conv-BN-reluconv-BN, где conv – сверточный слой, а BN – слой пакетной нормировки (раздел 14.2.4.1) (см. иллюстрацию на рис. 14.20 (слева)).

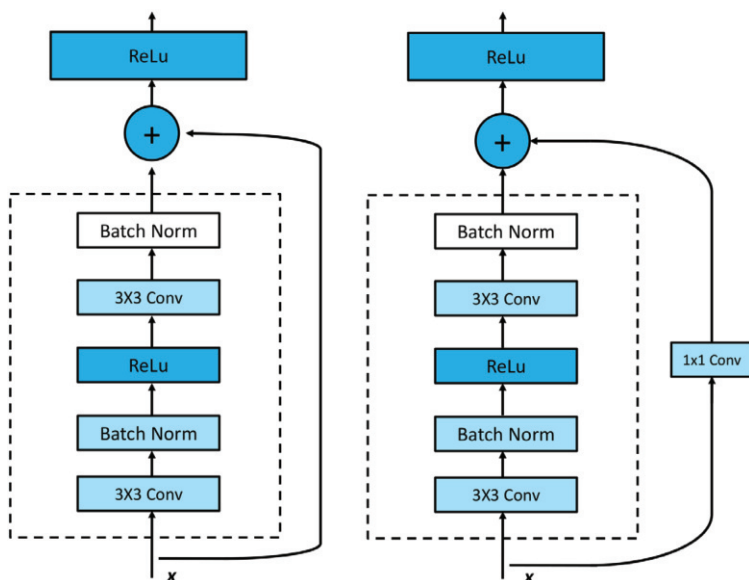


Рис. 14.20 ❖ Остаточный блок для СНС. Слева: стандартная версия. Справа: версия со сверткой 1×1 , позволяющая изменить количество каналов между входом в блок и выходом из него. На основе рис. 7.6.3 из работы [Zha+20]. Печатается с разрешения Астона Чжана

Воспользовавшись дополнением, мы можем сделать так, что пространственные измерения выхода $\mathcal{F}_l(\mathbf{x}_l)$ сверточного слоя будут такими же, как у входного слоя \mathbf{x}_l . Но если мы хотим, чтобы выход мог иметь другое число каналов, то должны добавить сверточный слой 1×1 в прямую связь от \mathbf{x}_l (см. иллюстрацию на рис. 14.20 (справа)).

Использование остаточных блоков позволяет обучать очень глубокие модели. Связано это с тем, что градиент может передаваться напрямую от выходного к предшествующим слоям по прямым связям (см. объяснение причин в разделе 13.4.4).

В работе [He+16a] сеть ResNet с 152 слоями обучена на наборе ImageNet. Однако обычно используют не столь глубокие модели. Например, на рис. 14.21 показана архитектура **ResNet-18** с 18 допускающими обучение слоями: два сверточных слоя 3×3 в каждом из 8 остаточных блоков, а также начальный сверточный слой 7×7 (с шагом 2) и последний полносвязный слой. Символически эту модель можно описать схемой

(Conv : BN : Max) : (R : R) : (R' : R) : (R' : R) : (R' : R) : Avg : FC,

где R – остаточный блок, R' – остаточный блок с прямой связью (из-за изменения количества каналов) с шагом 2, FC – полносвязный (плотный) слой, а : обозначает конкатенацию. Отметим, что пространственный размер входа уменьшается в $2^5 = 32$ раз (в 2 раза на каждом блоке R' плюс в начальном слое Conv-7x7(2) и в слое Max-pool), так что перед подачей на слой пулинга глобальным усреднением изображения 224×224 превращаются в изображения 7×7 .

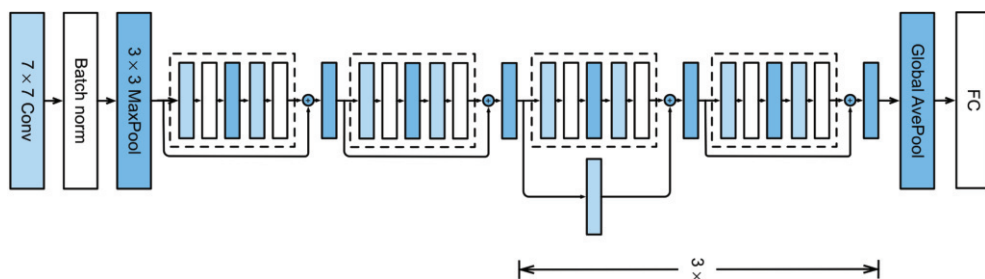


Рис. 14.21 ❖ Архитектура ResNet-18. Каждый обведенный пунктиром модуль – остаточный блок, показанный на рис. 14.20. На основе рис. 7.6.4 из работы [Zha+20]. Печатается с разрешения Астона Чжана

Применяя различные приемы, мы можем достичь верности 89 % на одном лучшем результате на тестовом наборе CIFAR после 20 эпох обучения¹.

В работе [He+16b] показано, что небольшая модификация описанной выше схемы позволяет обучать модели с числом слоев до 1001. Ключевое наблюдение заключается в том, что сигнал, проходящий по прямым связям, все же ослабляется из-за использования нелинейной функции активации после шага сложения $\mathbf{x}_{l+1} = \varphi(\mathbf{x}_l + \mathcal{F}(\mathbf{x}_l))$. Авторы показали, что лучше использовать обновление:

$$\mathbf{x}_{l+1} = \mathbf{x}_l + \varphi(\mathcal{F}_l(\mathbf{x}_l)). \quad (14.23)$$

Такая модель называется **resnet с предактивацией**, или просто **PreResnet**. Теперь сеть может очень легко обучиться тождественной функции в задан-

¹ Приемы такие: приращение данных (раздел 19.1), включающее случайные кадрирования, отражения относительно вертикальной оси и циклический план скоростей обучения (раздел 8.4.3). Используя 50 эпох и стохастическое усреднение весов (раздел 8.4.4), можно довести верность до ~94 % (см. код по адресу code.problm.ai/book1/cifar10_cnn_lightning).

ном слое: если использовать активации ReLU, то надо лишь гарантировать, что $\mathcal{F}_l(\mathbf{x}_l) = 0$, а это можно сделать, положив веса и смещения равными 0.

Альтернативой использованию очень глубокой модели является использование очень «широкой» модели с большим числом каналов признаков в каждом слое. Эта идея лежит в основе модели **широкой resnet** [ZK16], весьма популярной.

14.3.5. DenseNet

В остаточной сети мы прибавляем выход каждой функции к ее входу. Альтернативный подход – конкатенировать выход с входом, как показано на рис. 14.22a. Соединив несколько таких блоков подряд, мы получим архитектуру, показанную на рис. 14.22b. Она называется **DenseNets** [Hua+17a], поскольку каждый слой плотно зависит от всех предыдущих. Поэтому модель сводится к вычислению функции вида

$$\mathbf{x} \rightarrow [\mathbf{x}, f_1(\mathbf{x}), f_2(\mathbf{x}, f_1(\mathbf{x})), f_3(\mathbf{x}, f_1(\mathbf{x}), f_2(\mathbf{x}, f_1(\mathbf{x}))), \dots]. \quad (14.24)$$

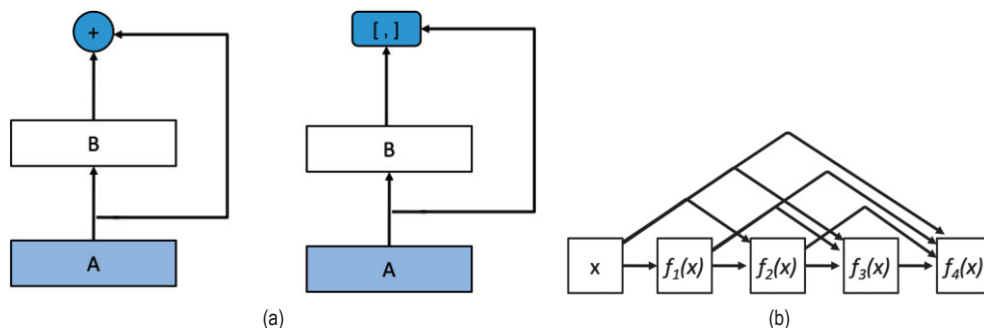


Рис. 14.22 ❖ (a) Слева: остаточный блок прибавляет выход к входу. Справа: блок DenseNet конкатенирует выход с входом. (b) Иллюстрация DenseNet. На основе рис. 7.7.1–7.7.2 из работы [Zha+20]. Печатается с разрешения Астона Чжана

Плотная связность приводит к увеличению числа параметров, так как каналы расположены друг под другом. Это можно компенсировать, добавив между ними сверточные слои 1×1 . Можно также добавить слои с шагом 2, чтобы уменьшить пространственное разрешение (см. пример кода по адресу code.problml.ai/book1/densenet_torch).

DenseNets может показывать лучшие результаты, чем ResNets, поскольку все ранее вычисленные признаки непосредственно доступны выходному слою. Однако с вычислительной точки зрения они дороже.

14.3.6. Поиск архитектуры нейронной сети

Мы видели, что многие СНС похожи по дизайну, а различия заключаются в топологии строительных блоков (например, сверточных и пулинговых

слоев) и настройках параметров (например, шага или количества каналов). Процесс проектирования можно автоматизировать, применяя методы оптимизации черного ящика (без производных) для нахождения архитектур, минимизирующих потерю на контрольном наборе. Это называется **Auto-ML**; в контексте нейронных сетей можно встретить название **поиск архитектуры нейронной сети** (neural architecture search – **NAS**). При выполнении NAS мы можем одновременно оптимизировать несколько целевых функций, например: верность, размер модели, скорость обучения или вывода и т. д. (именно так создана сеть **EfficientNetv2** [TL21]). Основная трудность связана со стоимостью вычисления целевой функции (поскольку для этого требуется обучить каждого кандидата в пространстве моделей). Один из способов уменьшить число вызовов целевой функции – применить байесовскую оптимизацию (см., например, [WNS19]). Другой подход – создать дифференцируемые аппроксимации функции потерь (см., например, [LSY19; Wan+21]). Область NAS очень обширна и все еще растет. Более полный обзор см. в работе [EMH19].

14.4. ДРУГИЕ ФОРМЫ СВЕРТКИ*

В разделе 14.2 мы обсудили основы свертки. В этом разделе мы поговорим о некоторых обобщениях, необходимых для таких приложений, как сегментация и генерирование изображений.

14.4.1. Дырявая свертка

Свертка – это операция, объединяющая значения пикселей в локальной окрестности. Применяя свертку с шагом и последовательное соединение нескольких сверточных слоев, мы можем увеличить рецептивное поле каждого нейрона, т. е. область пространства входов, на которую нейрон реагирует. Однако понадобилось бы много слоев, чтобы у каждого нейрона был достаточный контекст для покрытия всего изображения (если только не использовать очень большие фильтры, но это медленно и количество параметров слишком велико).

В качестве альтернативы можно использовать **свертку с дырками** [Mal99], которую иногда называют французским термином **алгоритм à trous**, а в последнее время – **дырявой сверткой** (dilated convolution) [YK16]. Идея в том, чтобы при выполнении свертки брать каждый r -й входной элемент, где r называется **коэффициентом дырявости**. Например, в одномерном случае свертка с фильтром \mathbf{w} с коэффициентом $r = 2$ эквивалентна регулярной свертке с фильтром $\tilde{\mathbf{w}} = [w_1, 0, w_2, 0, w_3]$, где нули вставлены, чтобы расширить рецептивное поле (отсюда и термин «свертка с дырками»). Это позволяет получить все преимущества широкого рецептивного поля без увеличения числа параметров или объема вычислений (см. иллюстрацию на рис. 14.23).

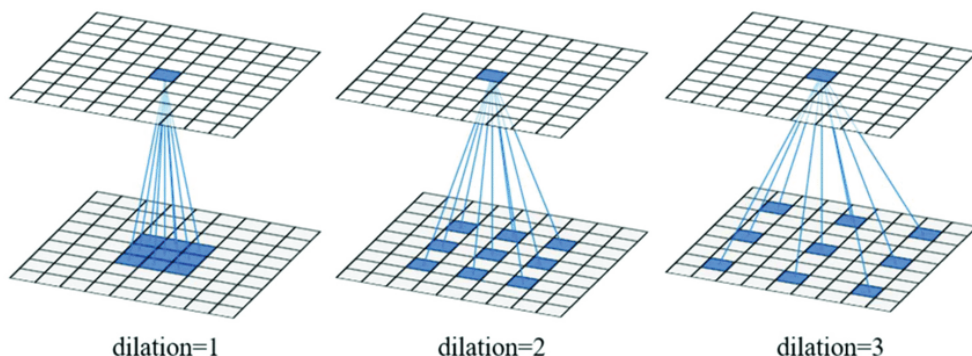


Рис. 14.23 ❖ Дырявая свертка с фильтром 3×3 и коэффициентами 1, 2 и 3. На основе рис. 1 из работы [Cui+19]. Печатается с разрешения Симинь Куя

Точнее, дырявая свертка в двумерном случае определяется следующим образом:

$$z_{i,j,d} = b_d + \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} \sum_{c=0}^{C-1} x_{i+ru, j+rv, c} w_{u,v,c,d}, \quad (14.25)$$

где для простоты мы взяли один и тот же коэффициент r для обоих направлений. Сравните это с формулой (14.15), где используется параметр шага $x_{si+u, sj+v, c}$.

14.4.2. Транспонированная свертка

Свертка сводит большой вход \mathbf{X} к малому выходу \mathbf{Y} за счет того, что вычисляется взвешенная комбинация входных пикселей и сверточного ядра \mathbf{K} . Проще всего это объяснить с помощью кода:

```
def conv(X, K):
    h, w = K.shape
    Y = zeros((X.shape[0] - h + 1, X.shape[1] - w + 1))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            Y[i, j] = (X[i:i + h, j:j + w] * K).sum()
    return Y
```

Транспонированная свертка делает прямо противоположное: порождает большой выход из меньшего входа:

```
def trans_conv(X, K):
    h, w = K.shape
    Y = zeros((X.shape[0] + h - 1, X.shape[1] + w - 1))
    for i in range(X.shape[0]):
        for j in range(X.shape[1]):
            Y[i:i + h, j:j + w] += X[i, j] * K
    return Y
```

Это эквивалентно дополнению входного изображения $(h - 1, w - 1)$ нулями (справа и снизу), где (h, w) – размер ядра, а затем наложения взвешенной копии ядра на каждую позицию входа (в качестве весов используются значения соответствующих пикселей) и суммирования. Этот процесс показан на рис. 14.24. Мы можем рассматривать ядро как «шаблон», который используется для генерирования выхода с модуляцией весами, определяемыми входом.

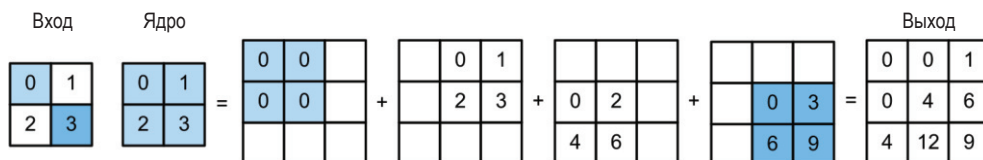


Рис. 14.24 ❖ Транспонированная свертка с ядром 2×2 .

На основе рис. 13.10.1 из работы [Zha+20].

Печатается с разрешения Астона Чжана

Происхождение термина «транспонированная свертка» связано с интерпретацией свертки как умножения матриц (см. обсуждение в разделе 14.2.1.3). Если \mathbf{W} – матрица, полученная из ядра \mathbf{K} применением процесса, описанного формулой (14.9), то можно показать, что $\mathbf{Y} = \text{transposed-conv}(\mathbf{X}, \mathbf{K})$ эквивалентно $\mathbf{Y} = \text{reshape}(\mathbf{W}^T \text{vec}(\mathbf{X}))$ (см. пример кода по адресу code.problml.ai/book1/transposed_conv_torch).

Отметим, что транспонированная свертка иногда называется **антисверткой** (deconvolution), но такое использование термина неправомерно: антисвертка – это процесс «отмены» эффекта свертки с известным фильтром, например фильтром размытия, с целью восстановления исходного входа (см. рис. 14.25).

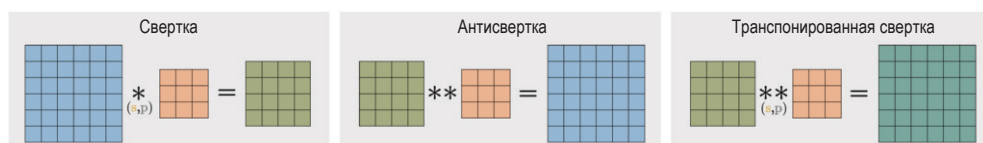


Рис. 14.25 ❖ Свертка, антисвертка и транспонированная свертка.

Здесь s – шаг, а p – дополнение. Взято из статьи <https://tinyurl.com/ynxcxsut>.

Печатается с разрешения Акеля Анвара

14.4.3. Пространственная раздельная свертка

В стандартной свертке используется фильтр размера $H \times W \times C \times D$, поэтому требуется обучать много параметров, а вычисления с ним занимают много времени. Упрощение, известное под названием **пространственная раздельная свертка** (depthwise separable convolution), сначала сворачивает каждый входной канал соответствующим двумерным фильтром \mathbf{w} , а затем отображает эти C каналов в D каналов с помощью свертки \mathbf{w}' размера 1×1 :

$$z_{i,j,d} = b_d + w'_{c,d} \sum_{c=0}^{C-1} \left(\sum_{u=0}^{H-1} \sum_{v=0}^{W-1} x_{i+u,j+v,c} w_{u,v} \right). \quad (14.26)$$

Смотрите иллюстрацию на рис. 14.26.

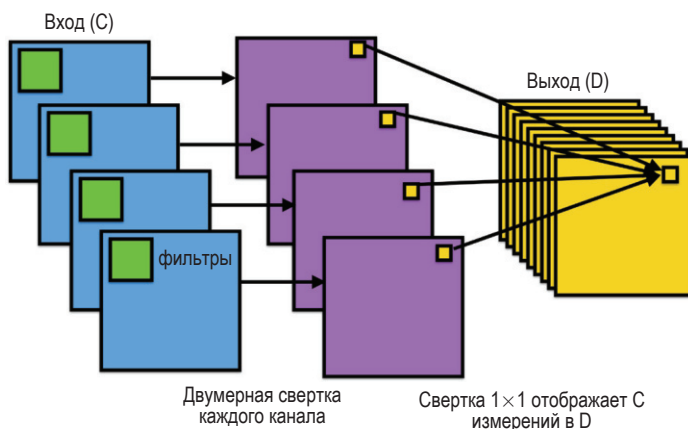


Рис. 14.26 ❖ Пространственные раздельные свертки: каждый из C входных каналов подвергается двумерной свертке и порождает C выходных каналов, которые объединяются поточечно (посредством свертки 1×1); в результате порождается D выходных каналов. Взято из статьи <https://bit.ly/2L9fm2o>. Печатается с разрешения Эджению Кулурчелло

Чтобы понять, в чем тут преимущество, рассмотрим простой числовой пример¹. Регулярная свертка входа $12 \times 12 \times 3$ с фильтром $5 \times 5 \times 3 \times 256$ дает выход размера $8 \times 8 \times 256$ (в предположении, что выполняется корректная свертка: $12 - 5 + 1 = 8$), как показано на рис. 14.13. В случае раздельной свертки мы начинаем с входа $12 \times 12 \times 3$, сворачиваем его с фильтром $5 \times 5 \times 1 \times 1$ (по пространству, но не по каналам), получая при этом выход $8 \times 8 \times 3$, а затем сворачиваем поточечно (по каналам, но не по пространству) с фильтром $1 \times 1 \times 3 \times 256$ и получаем выход $8 \times 8 \times 256$. Таким образом, размер выхода такой же, как раньше, но для определения слоя мы использовали гораздо меньше параметров и вычислений было также намного меньше. По этой причине раздельная свертка часто применяется в облегченных моделях СНС, например **MobileNet** [How+17; San+18a] и других, развертываемых на **оконечных устройствах**.

14.5. РЕШЕНИЕ ДРУГИХ ДИСКРИМИНАНТНЫХ ЗАДАЧ КОМПЬЮТЕРНОГО ЗРЕНИЯ С ПОМОЩЬЮ СНС*

В этом разделе мы кратко обсудим, как решить другие задачи компьютерного зрения с помощью СНС. При описании каждой задачи будет рассмотрено

¹ Этот пример взят из статьи Чи-Фен Вана по адресу <https://bit.ly/2Uj64Vo>.

какое-то архитектурное добавление в библиотеку основных строительных блоков. Дополнительные сведения о применении СНС в компьютерном зрении можно найти в работе [Bro19].

14.5.1. Аннотирование изображений

В задаче классификации изображений одна метка ассоциируется со всем изображением, т. е. предполагается, что выходы взаимно исключающие. Во многих задачах в изображении может присутствовать несколько объектов, и мы хотим пометить каждый из них. Это называется **аннотированием изображений** (image tagging) и является примером многометочного предсказания. В этом случае пространство выходов $\mathcal{Y} = \{0, 1\}^C$, где C – число типов. Поскольку выходные биты независимы (при условии заданного изображения), мы должны заменить конечный слой softmax набором C логистических блоков.

Пользователи социальных сетей типа **Инстаграм** часто создают хештеги для своих изображений, в результате мы «бесплатно» получаем способ создания больших наборов данных с учителем. Конечно, многие теги используются лишь эпизодически, а их визуальный смысл не слишком хорошо определен. (Например, кто-то мог сфотографировать себя после сдачи теста на ковид и аннотировать изображение хештегом «#covid», хотя визуально оно ничем не отличается от других изображений человека.) Поэтому такие созданные пользователями метки обычно считаются сильно зашумленными. Однако их можно использовать для «предобучения», как описано в работе [Mah+18].

Наконец, стоит отметить, что аннотирование изображений зачастую более разумная цель, чем их классификация, потому что на многих изображениях присутствует несколько объектов и трудно решить, какой именно брать в качестве метки. Андрей Карпатый, создавший «эталонный тест качества работы человека» на наборе данных ImageNet, заметил¹:

И СНС, и люди испытывают сложности при обработке изображений, содержащих несколько классов ImageNet (обычно более пяти) без какого-либо указания на то, какой объект занимает центральное место. Эта ошибка свойственна только задаче классификации, потому что для каждого изображения можно задать ровно одну метку. В целом 16 % человеческих ошибок попадает в эту категорию.

14.5.2. Определение объектов

Иногда мы хотим породить переменное число выходов в соответствии с переменным числом представляющих интерес объектов в изображении. (Это пример проблемы **открытого мира** с неизвестным числом объектов.)

Канонический пример такой задачи – **определение объектов**, когда требуется вернуть множество **ограничивающих прямоугольников**, которые

¹ Источник: <https://bit.ly/3cFbAlk>.

описывают местоположение интересных объектов, вместе с метками классов. Частным случаем является **определение лиц**, когда имеется только один класс, представляющий интерес (см. иллюстрацию на рис. 14.27а)¹.

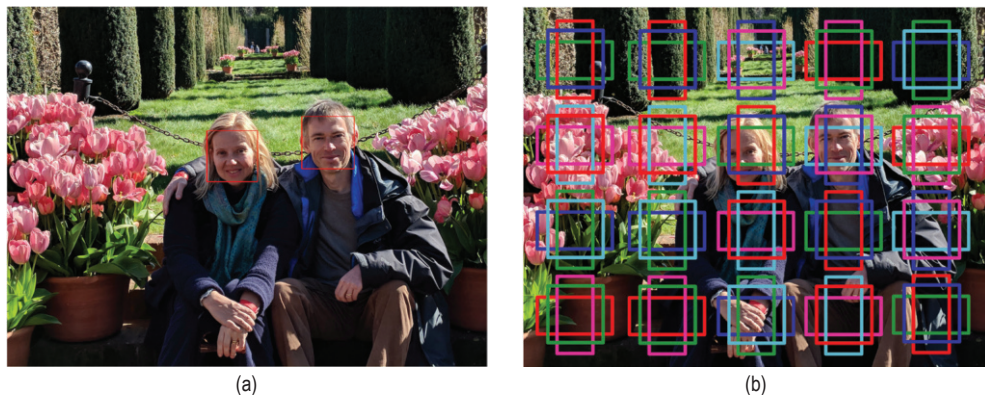


Рис. 14.27 ❖ (а) Определение лиц (фотография автора и его жены Маргарет, сделанная в садах Филоли в Калифорнии в феврале 2018 года. Изображение обработано Джонатаном Хуангом с помощью модели лиц SSD). (б) Якорные рамки. На основе [Zha+20, раздел 12.5]

Простейший подход к таким задачам определения состоит в том, чтобы преобразовать их в задачу замкнутого мира, когда имеется конечное множество возможных позиций (и ориентаций) объектов. Такие потенциальные позиции называются **якорными рамками** (anchor box). Мы можем создать рамки в нескольких позициях, с разными масштабами и отношениями сторон, как показано на рис. 14.27b. Для каждой рамки мы обучаем систему предсказывать категорию находящегося в ней объекта (если таковой имеется); можно также выполнить регрессию, чтобы предсказать смещение объекта от центра рамки. (Эти остаточные члены регрессии открывают возможность для пространственной локализации на подсетке.)

В абстрактной постановке мы обучаем функцию вида

$$f_{\theta} : \mathbb{R}^{H \times W \times K} \rightarrow [0, 1]^{A \times A} \times \{1, \dots, C\}^{A \times A} \times (\mathbb{R}^4)^{A \times A}, \quad (14.27)$$

где K – число входных каналов, A – число якорных рамок в каждом направлении, а C – число типов объектов (меток классов). Для каждой позиции рамки (i, j) мы предсказываем три выхода: вероятность присутствия объекта, $p_{ij} \in [0, 1]$, категорию объекта, $y_{ij} \in \{1, \dots, C\}$, и два двумерных вектора смещения, $\delta_{ij} \in \mathbb{R}^4$, которые можно прибавить к центру рамки, чтобы получить левый верхний и правый нижний угол.

¹ Отметим, что определение лиц отличается от **распознавания лиц**; последнее заключается в предсказании личности человека, входящего в «галерею» возможных кандидатов. Задача распознавания лиц обычно решается путем применения классификатора ко всем патчам, содержащим лица.

Было предложено несколько моделей такого типа, в том числе модель **однопроходного детектора** (single shot detector – SSD) [Liu+16] и модель **YOLO** (you only look once) [Red+16]. За прошедшие годы появилось много методов определения объектов с разными компромиссами между быстродействием, верностью, простотой и т. д. Их эмпирическое сравнение см. в работе [Hua+17b], а относительно недавний обзор – в работе [Zha+18].

14.5.3. Сегментация экземпляров

В задаче определения объектов мы предсказываем метку и ограничивающий прямоугольник для каждого объекта. В задаче **сегментации экземпляров** цель – предсказать метку и двумерную маску формы каждого экземпляра объекта в изображении, как показано на рис. 14.28. Это можно сделать, применив семантическую сегментацию к каждому обнаруженному прямоугольнику, задача которой – пометить каждый пиксель как принадлежащий переднему или заднему плану (подробнее о семантической сегментации см. раздел 14.5.4).

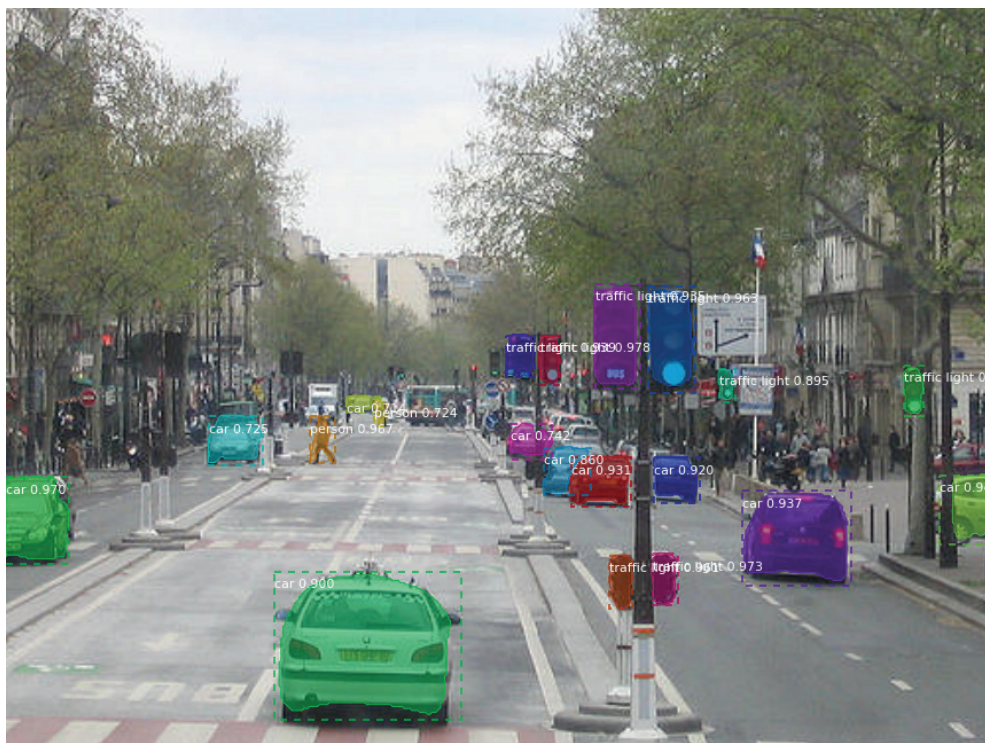


Рис. 14.28 ❖ Определение объектов и сегментация экземпляров с применением R-CNN Mask. Взято из https://github.com/matterport/Mask_RCNN. Печатается с разрешения Валида Абдуллы

14.5.4. Семантическая сегментация

Задача **семантической сегментации** состоит в том, чтобы предсказать метку класса $y_i \in \{1, \dots, C\}$ для каждого пикселя, где классы могут представлять такие объекты, как небо, дорога, автомобиль и т. д. В отличие от сегментации экземпляров, рассмотренной в разделе 14.5.3, все пиксели автомобилей получают одну и ту же метку, поэтому семантическая сегментация не различает объекты. Можно объединить семантическую сегментацию «окружения» (например, неба или дороги) с сегментацией экземпляров «предметов» (например, автомобилей или людей) – в результате получится единый каркас, получивший название «**всеобъемлющая сегментация**» (panoptic segmentation) [Kir+19].

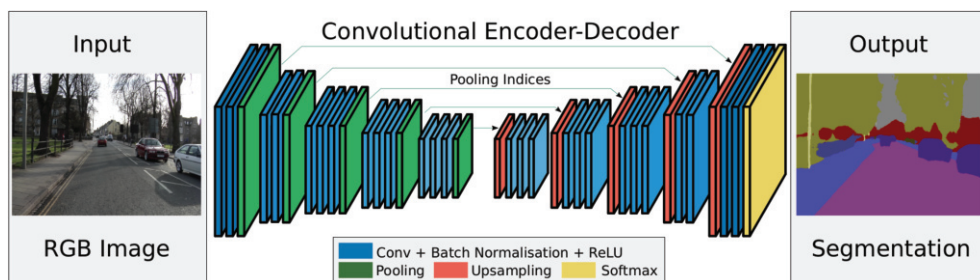


Рис. 14.29 ❖ СНС-кодировщик-декодер (или U-сеть) для семантической сегментации. Кодировщик использует свертку (понижающую передискретизацию), а декодер – транспонированную свертку (повышающую передискретизацию). На основе рис. 1 из работы [BKC17]. Печатается с разрешения Алекса Кендалла

Для семантической сегментации часто применяют архитектуру **кодировщик-декодер**, показанную на рис. 14.29. В кодировщике используется стандартная свертка для отображения входа в меньшее двумерное изображение, улавливающее общие свойства при низком пространственном разрешении. (Обычно для этой цели применяется описанная в разделе 14.4.1 техника дырявой свертки, увеличивающая рецептивное поле, т. е. предоставляющая больше контекста.) Декодер отображает уменьшенное изображение обратно в полноразмерное выходное изображение, применяя технику транспонированной свертки, описанную в разделе 14.4.2. Так как при уменьшении масштаба теряется информация, мы также можем добавить прямые связи от входных слоев к выходным. Эту модель можно изобразить, как показано на рис. 14.30. Поскольку общая структура напоминает букву U, модель часто называют **U-сетью** [RFB15].

Похожую архитектуру **кодировщик-декодер** можно использовать и в других задачах **плотного предсказания** или **преобразования изображения в изображение**, например **предсказания глубины** (расстояния от камеры до каждого пикселя $i - z_i \in \mathbb{R}$), **предсказания нормали к поверхности** (ориентации поверхности, $z_i \in \mathbb{R}^3$, для каждого патча изображения) и т. д.

Конечно, можно обучить модель решать все эти задачи одновременно, реализовав несколько выходов, как показано на рис. 14.31. (Детали см., например, в работе [Kok17].)

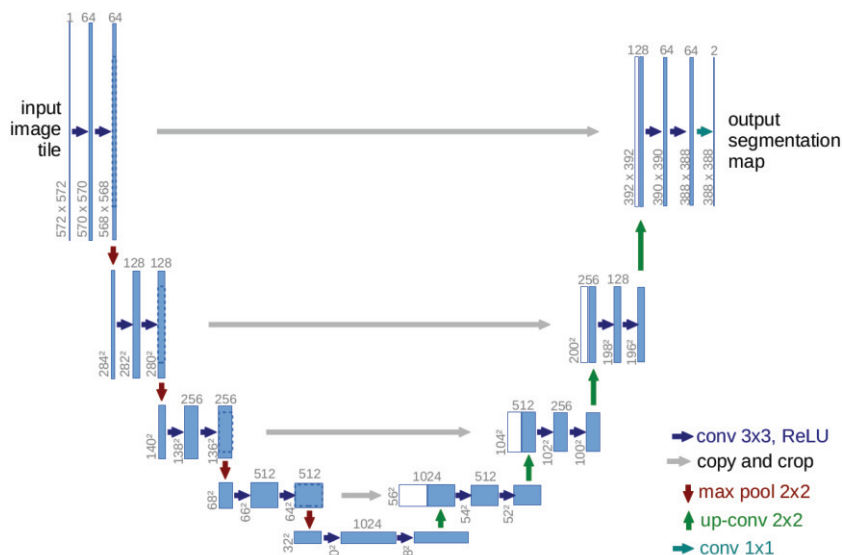


Рис. 14.30 ❖ Иллюстрация модели U-Net для семантической сегментации. Голубые прямоугольники соответствуют многоканальным картам признаков. Количество каналов показано сверху от прямоугольника, а высота и ширина – слева внизу. Белые прямоугольники обозначают скопированные карты признаков. Разноцветные стрелки соответствуют различным операциям. На основе рис. 1 из работы [RFB15]. Печатается с разрешения Олафа Розенберга

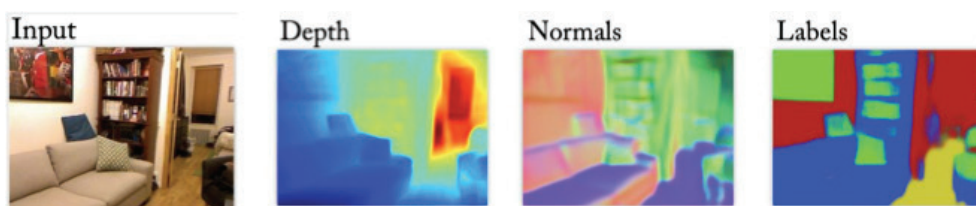


Рис. 14.31 ❖ Задача многофункционального плотного предсказания. На основе рис. 1 из работы [EF15]. Печатается с разрешения Роба Фергюса

14.5.5. Оценивание позы человека

Мы можем обучить детектор объектов определять людей и предсказывать их форму, представленную маской. А еще можно обучить модель предсказывать местоположение фиксированного множества особых точек скелета, например головы или кистей рук. Это называется **оценивание позы человека**

(см. пример на рис. 14.32). Для решения этой задачи существует несколько методов, например **PersonLab** [Pap+18] и **OpenPose** [Cao+18]. Недавний обзор см. в [Bab19].



Рис. 14.32 ❖ Определение особых точек: туловища, кистей рук и лица с помощью системы OpenPose. На основе рис. 8 из работы [Cao+18]. Печатается с разрешения Ясера Шейха

Мы также можем предсказывать трехмерные свойства найденных объектов. Основное ограничение – получение достаточного объема помеченных обучающих данных, поскольку человеку трудно создавать аннотации трехмерных изображений. Однако можно воспользоваться системами компьютерной графики и создавать синтезированные изображения с трехмерными аннотациями в любом количестве (см., например, [GNK18]).

14.6. ГЕНЕРИРОВАНИЕ ИЗОБРАЖЕНИЙ ПОСРЕДСТВОМ ИНВЕРТИРОВАНИЯ СНС*

СНС, обученная классификации изображений, – это дискриминантная модель вида $p(y|x)$, которая принимает изображение и возвращает распределение вероятностей меток классов C . В этом разделе мы поговорим о том, как «инвертировать» эту модель, преобразовав ее в (условную) **порождающую изображения модель** вида $p(x|y)$. Это позволит нам генерировать изображения, принадлежащие определенному классу. (Научно обоснованные подходы

к созданию порождающих моделей для изображений мы обсудим во втором томе этой книги, [Mur22].)

14.6.1. Преобразование обученного классификатора в порождающую модель

Мы можем определить совместное распределение изображений и меток $p(\mathbf{x}, y) = p(\mathbf{x})p(y|\mathbf{x})$, где $p(y|\mathbf{x})$ – классифицирующая СНС, а $p(\mathbf{x})$ – некоторое априорное распределение изображений. Если затем «притянуть» метку классов к конкретному значению, то можно создать условную порождающую модель $p(\mathbf{x}|y) \propto p(\mathbf{x})p(y|\mathbf{x})$. Отметим, что дискриминантный классификатор $p(y|\mathbf{x})$ был обучен «отбрасывать» информацию, поэтому $p(y|\mathbf{x})$ – необратимая функция. Таким образом, априорный член $p(\mathbf{x})$ будет играть важную роль в регуляризации этого процесса, как мы увидим в разделе 14.6.2.

Один из способов произвести выборку из этой модели – воспользоваться алгоритмом Метрополиса–Гастингса (раздел 4.6.8.4), рассматривая $\Psi_c(\mathbf{x}) = \log p(y = c|\mathbf{x}) + \log p(\mathbf{x})$ как функцию энергии. Поскольку информация о градиенте доступна, мы можем использовать вспомогательное распределение вида $\pi(\mathbf{x}'|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}(\mathbf{x}), \varepsilon \mathbf{I})$, где $\boldsymbol{\mu}(\mathbf{x}) = \mathbf{x} + (\varepsilon/2)\nabla \log \Psi_c(\mathbf{x})$. Это называется **алгоритмом Ланжевена с поправкой Метрополиса** (Metropolis-adjusted Langevin algorithm – MALA). В качестве аппроксимации можно игнорировать шаг отвержения и принимать любое предложение. Этот **алгоритм Ланжевена без поправки** использовался в работе [Ngu+17] для условного генерирования изображений. Кроме того, мы можем независимо масштабировать градиент логарифмического априорного распределения и логарифмического правдоподобия. Таким образом, мы получаем обновление в пространстве изображений, которое похоже на зашумленную версию ГС, только производные берутся по входным пикселям (по формуле (13.50)), а не по параметрам:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \varepsilon_1 \frac{\partial \log p(\mathbf{x}_t)}{\partial \mathbf{x}_t} + \varepsilon_2 \frac{\partial \log p(y = c|\mathbf{x}_t)}{\partial \mathbf{x}_t} + \mathcal{N}(\mathbf{0}, \varepsilon_3^2 \mathbf{I}). \quad (14.28)$$

Члены этой формулы интерпретируются следующим образом: ε_1 гарантирует, что изображение похоже на правду с точки зрения априорного распределения, ε_2 гарантирует, что изображение похоже на правду с точки зрения правдоподобия, а ε_3 – шум, необходимый для генерирования разнообразных примеров. Если положить $\varepsilon_3 = 0$, то алгоритм становится детерминированным и генерирует (приближенно) «самое вероятное изображение» для данного класса.

14.6.2. Априорные распределения изображений

В этом разделе мы обсудим различные априорные распределения изображений, которые можно использовать для регуляризации некорректной задачи обращения классификатора. Эти априорные распределения в совокупности

с изображением, с которого мы начинаем оптимизацию, будут определять виды генерируемых изображений.

14.6.2.1. Гауссово априорное распределения

Одного лишь задания метки класса недостаточно для определения желаемого вида изображений. Необходимо также априорное распределение $p(\mathbf{x})$ «похожих на правду» изображений. Ниже мы покажем, что это априорное распределение может сильно повлиять на качество результирующего изображения.

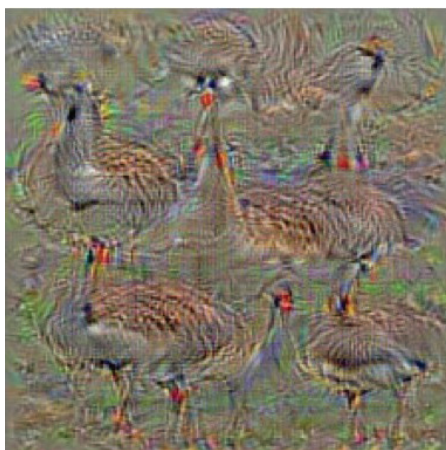
Пожалуй, самым простым априорным распределением является $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{0}, \mathbf{I})$, как предложено в работе [SVZ14]. (При этом предполагается, что пиксели изображения уже центрированы.) Оно может предотвратить появление экстремальных значений пикселей. В этом случае член обновления, связанный с априорным распределением, принимает вид:

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}_t) = \nabla_{\mathbf{x}} \left[-\frac{1}{2} \|\mathbf{x}_t - \mathbf{0}\|_2^2 \right] = -\mathbf{x}_t. \quad (14.29)$$

Таким образом, обновление в целом (в предположении, что $\varepsilon_2 = 1$ и $\varepsilon_3 = 0$) имеет вид:

$$\mathbf{x}_{t+1} = (1 - \varepsilon_1)\mathbf{x}_t + \frac{\partial \log p(y = c|\mathbf{x}_t)}{\partial \mathbf{x}_t}. \quad (14.30)$$

Примеры изображений, сгенерированных этим методом, приведены на рис. 14.33.



goose



ostrich

Рис. 14.33 ❖ Изображения, максимизирующие вероятность классов «гусь» и «страус» для набора ImageNet при простом гауссовом априорном распределении. Взято из статьи по адресу <http://yosinski.com/deepvis>. Печатается с разрешения Джеффа Клуна

14.6.2.2. Априорное распределение на основе полной вариации

Для генерирования более реалистичных изображений можно воспользоваться дополнительными регуляризаторами. В работах [MV15; MV16] предложено вычислять **полную вариацию**, или норму **TV** изображения. Она равна интегралу попиксельных градиентов, который можно аппроксимировать следующим образом:

$$\text{TV}(\mathbf{x}) = \sum_{ijk} (x_{ijk} - x_{i+1,j,k})^2 + (x_{ijk} - x_{i,j+1,k})^2, \quad (14.31)$$

где x_{ijk} – значение пикселя в строке i , столбце j и канале k (для RGB-изображений). Это выражение можно переписать в терминах **детектора** горизонтальных и вертикальных **границ Собеля**, примененного к каждому каналу:

$$\text{TV}(\mathbf{x}) = \sum_k \|\mathbf{H}(\mathbf{x}_{:, :, k})\|_F^2 + \|\mathbf{V}(\mathbf{x}_{:, :, k})\|_F^2. \quad (14.32)$$

Смотрите иллюстрацию детекторов границ на рис. 14.34. Использование распределения $p(\mathbf{x}) \propto \exp(-\text{TV}(\mathbf{x}))$ предотвращает появление высокочастотных артефактов в изображениях. В работе [Yos+15] используется гауссово размытие вместо нормы TV с похожим эффектом.

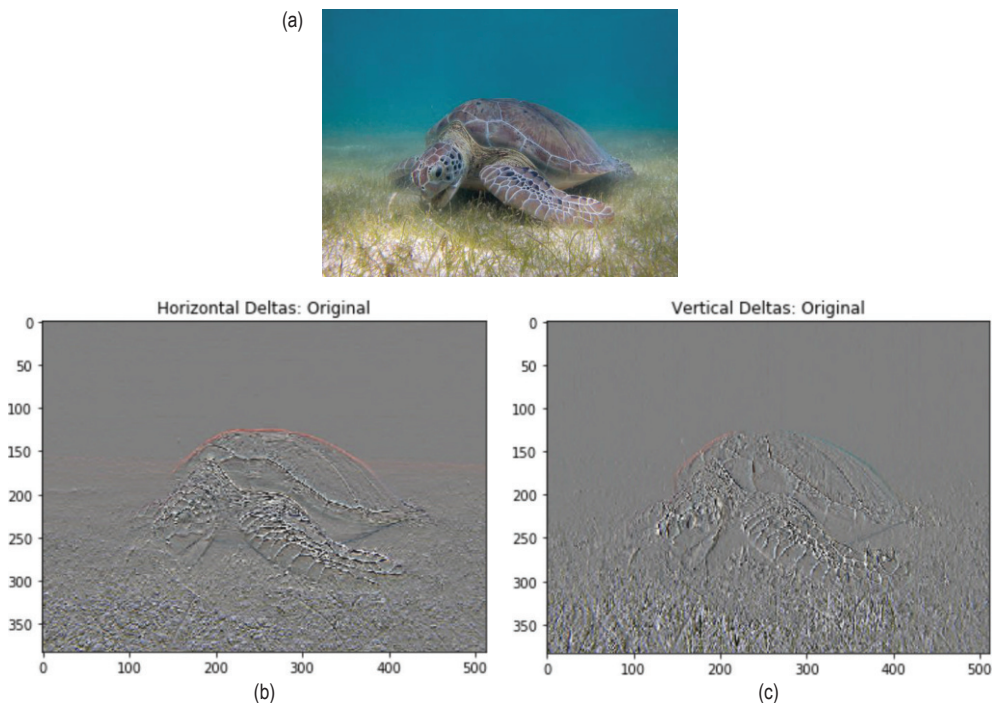


Рис. 14.34 ❖ Норма полной вариации. (а) Входное изображение: зеленая морская черепаха (печатается с разрешения автора викимедии Р. Линдгрена). (б) Горизонтальные дельты. (с) Вертикальные дельты. На основе статьи по адресу https://www.tensorflow.org/tutorials/generative/style_transfer

На рис. 14.35 показаны результаты оптимизации $\log p(y = c, \mathbf{x})$, начиная со случайного шума, с использованием априорного распределения на основе TV и СНС-правдоподобия для разных меток классов c .

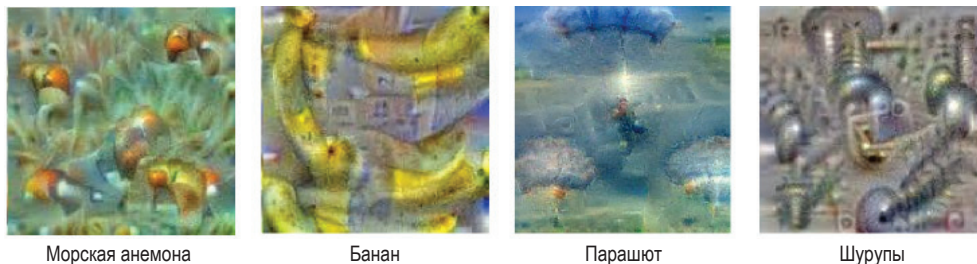


Рис. 14.35 ❖ Изображения, максимизирующие вероятность определенных классов ImageNet при априорном распределении на основе TV. Из статьи по адресу <https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>. Печатается с разрешения Александра Мордвинцева

14.6.3. Визуализация признаков, обученных с помощью СНС

Интересный вопрос – чему обучаются «нейроны» СНС. Для ответа на него можно начать со случайного изображения, а затем оптимизировать входные пиксели, стремясь доставить максимум средней активации конкретного нейрона. Это называется **максимизацией активации** (activation maximization – AM). Используется та же техника, что в разделе 14.6.1, но фиксируется значение внутреннего узла, а не метки выходного класса.

На рис. 14.36 показан результат применения этого метода (с априорным распределением на основе TV) к СНС AlexNet, обученной на классификации набора ImageNet. Мы видим, что по мере увеличения глубины нейроны обучаются распознавать границы и пятна, затем текстурные паттерны, затем части объектов и, наконец, объекты целиком. Считается, что это отдаленно напоминает иерархическое строение зрительной коры головного мозга (см., например, [Kan+12]).

Альтернативой оптимизации в пространстве пикселей является поиск в обучающем наборе таких изображений, которые дают максимальную активацию заданного нейрона. Это показано на рис. 14.36 для слоя Conv5.

Дополнительные сведения о визуализации признаков см., например, в работе [OMS17].

14.6.4. Deep Dream

До сих пор нас интересовало генерирование изображений, которые максимизируют метку класса или активацию какого-то интересного нейрона. В этом разделе мы рассмотрим приложение артистического толка, в котором

требуется сгенерировать версии входного изображения, подчеркивающие некоторые признаки.

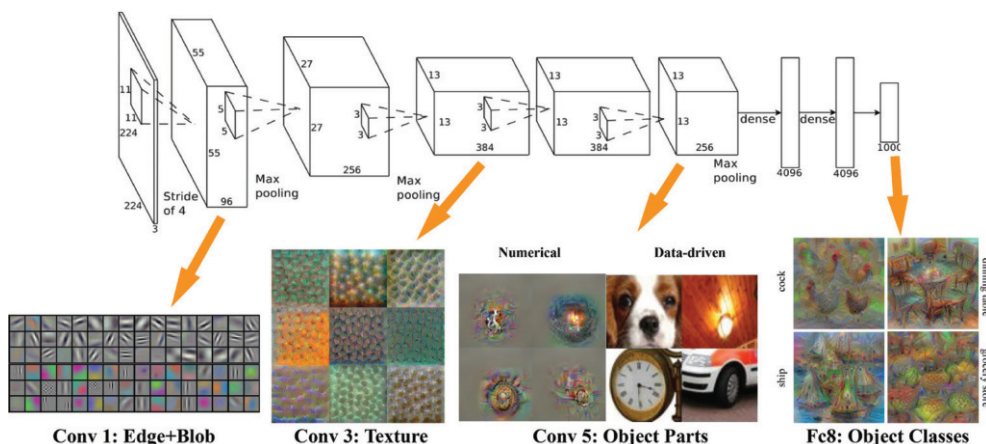


Рис. 14.36 ❖ Визуализированы «оптимальные стимулы» нейронов в слоях Conv 1, 3, 5 и fc8 в архитектуре AlexNet, обученной на наборе данных ImageNet. Для слоя Conv5 показаны также реальные изображения (в столбце «Data-driven»), порождающие похожие активации. На основе метода, описанного в работе [MV16]. Печатается с разрешения Донглай Вея

Для этого будем рассматривать предварительно обученный классификатор изображений как экстрактор признаков. Из результатов раздела 14.6.3 мы знаем, что активность нейронов в разных слоях соответствует различным видам признаков в изображении. Допустим, что мы хотим «усилить» признаки из слоя $l \in \mathcal{L}$. Это можно сделать, определив функцию энергии или потерь вида $\mathcal{L}(\mathbf{x}) = \sum_{l \in \mathcal{L}} \bar{\phi}_l(\mathbf{x})$, где $\bar{\phi}_l = (1/HWC) \sum_{hwc} \phi_{lhw}(\mathbf{x})$ – вектор признаков для слоя l . Теперь можно использовать метод градиентного спуска для оптимизации этой энергии. Получающийся процесс называется **DeepDream**¹ [MOT15], поскольку модель усиливает признаки, которые были только намечены в оригинальном изображении, а затем создает изображения, содержащие все больше и больше таких признаков².

На рис. 14.37 приведен пример. Мы начали с изображения медузы, которое передаем СНС, обученной для классификации изображений из набора ImageNet. После нескольких итераций генерируется изображение, являющееся гибридом входа и «галлюцинаций» типа тех, что мы видели на рис. 14.33; эти галлюцинации включают части собак, потому что среди меток ImageNet

¹ Глубокий сон. – Прим. перев.

² Первоначально метод был назван **Inceptionism**, потому что в нем используется СНС Inception (раздел 14.3.3).

много видов собак. Детали см. в работе [Tho16], а забавную демонстрацию на сайте <https://deepdreamgenerator.com>.

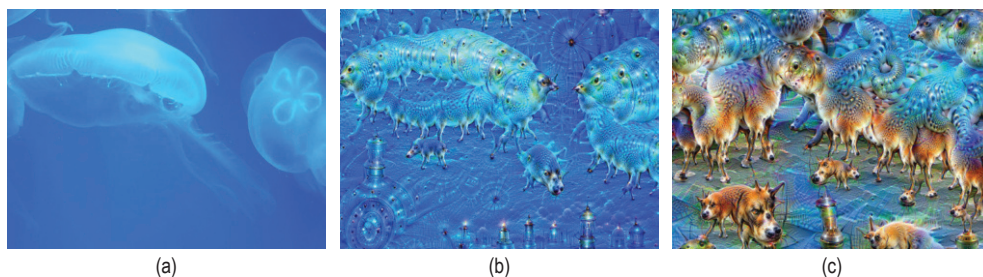


Рис. 14.37 ❖ DeepDream. В роли СНС выступает классификатор Inception, обученный на наборе данных ImageNet. (a) Начальное изображение Aurelia aurita (ушастой медузы). (b) Изображение, сгенерированное после 10 итераций. (c) Изображение, сгенерированное после 50 итераций. Взято из статьи по адресу <https://en.wikipedia.org/wiki/DeepDream>. Печатается с разрешения автора «Википедии» Мартина Тома

14.6.5. Нейронный перенос стиля

Система DeepDream на рис. 14.37 – это лишь один способ использования СНС для создания «художественных произведений». Но довольно страшненький. В этом разделе мы рассмотрим родственный подход, который дает пользователю больше контроля над процессом. В частности, пользователь задает «образец стиля» x_s и «оригинальное изображение» x_c . Затем система пытается сгенерировать новое изображение x , «перерисовывающее» x_c в стиле x_s . Эта техника **нейронного переноса стиля** иллюстрируется на рис. 14.38 и 14.39. Впервые она была предложена в работе [GEB16], и с тех пор появилось много работ на эту тему; см. недавний обзор [Jin+17].



Рис. 14.38 ❖ Пример результата работы системы нейронного переноса стиля. (a) Оригинальное изображение: зеленая морская черепаха (печатается с разрешения автора википедии П. Линдгрена). (b) Образец стиля: картина Василия Кандинского «Композиция VII». (c) Результат переноса стиля. На основе статьи https://www.tensorflow.org/tutorials/generative/style_transfer



Рис. 14.39 ❖ Применение нейронного переноса стиля к «производственной бригаде», помогавшей создавать код и демонстрации для этой книги и ее продолжения. Сверху вниз: Кэвин Мэрфи (автор), Махмуд Солиман, Алейна Кара, Срикара Джилугу, Дришти Патель, Мин Лиан Ан, Джерардо Дюран-Мартен, Кoko (общий пес). Каждый оригинал перерисован в своем стиле. На основе статьи по адресу https://www.tensorflow.org/tutorials/generative/style_transfer

14.6.5.1. Как это работает

В основе переноса стиля лежит оптимизация следующей функции энергии:

$$\mathcal{L}(\mathbf{x}|\mathbf{x}_s, \mathbf{x}_c) = \lambda_{TV}\mathcal{L}_{TV}(\mathbf{x}) + \lambda_c\mathcal{L}_{\text{content}}(\mathbf{x}, \mathbf{x}_c) + \lambda_s\mathcal{L}_{\text{style}}(\mathbf{x}, \mathbf{x}_s). \quad (14.33)$$

Смотрите общую схему на рис. 14.40.

Первый член в формуле (14.33) – априорное распределение на основе полной вариации, обсуждавшееся в разделе 14.6.2.2. Второй член измеряет степень схожести \mathbf{x} и \mathbf{x}_c путем сравнения карт признаков предобученной СНС $\Phi(\mathbf{x})$ в релевантном «слое оригинала» l :

$$\mathcal{L}_{\text{content}}(\mathbf{x}, \mathbf{x}_c) = \frac{1}{C_\ell H_\ell W_\ell} \|\Phi_\ell(\mathbf{x}) - \Phi_\ell(\mathbf{x}_c)\|_2^2. \quad (14.34)$$

Наконец, мы должны определить член стиля. Визуальный стиль можно интерпретировать как статистическое распределение некоторых видов признаков изображения. Местоположение этих признаков внутри изображения может не играть роли, но совместная встречаемость важна. Это показано на рис. 14.41. Ясно (человеку), что изображение 1 стилистически больше похоже на изображение 2, чем на изображение 3. Интуитивно кажется, что причина – наличие в изображениях 1 и 2 зеленых заостренных форм, тогда как в изображении 3 заостренные формы есть, но не зеленые.

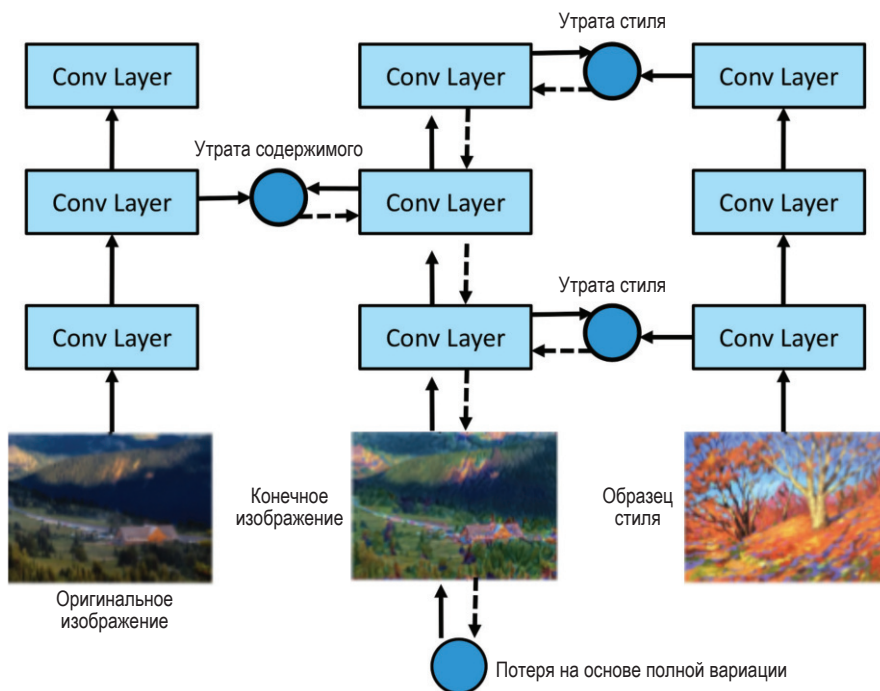


Рис. 14.40 ❖ Иллюстрация работы нейронного переноса стиля.
На основе рис. 12.12.2 из работы [Zha+20]

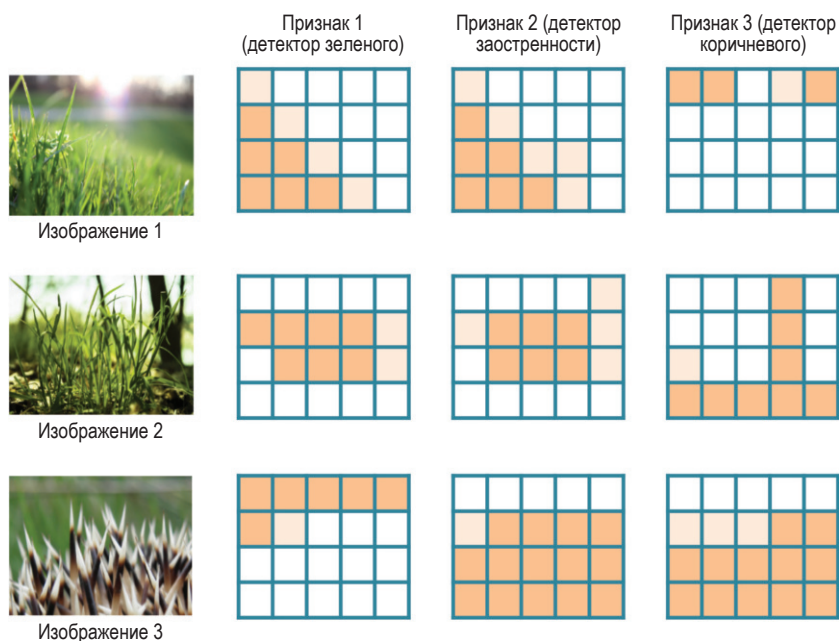


Рис. 14.41 ❖ Схематическое представление трех видов карт признаков для трех разных входных изображений. На основе рис. 5.16 из работы [Fos19]

Чтобы отразить статистику совместной встречаемости, мы вычислим матрицу Грама изображения, воспользовавшись картами признаков из конкретного слоя l :

$$G_\ell(\mathbf{x})_{c,d} = \frac{1}{H_\ell W_\ell} \sum_{h=1}^{H_\ell} \sum_{w=1}^{W_\ell} \phi_\ell(\mathbf{x})_{h,w,c} \phi_\ell(\mathbf{x})_{h,w,d}. \quad (14.35)$$

Матрица Грама размера $C_l \times C_l$ пропорциональна нецентрированной ковариационной матрице C_l -мерных векторов признаков, выбранных из каждой из $H_l W C_l$ позиций.

Имея это в виду, мы определяем стилистическую потерю в слое l следующим образом:

$$\mathcal{L}_{\text{style}}^\ell(\mathbf{x}, \mathbf{x}_c) = \|\mathbf{G}_\ell(\mathbf{x}) - \mathbf{G}_\ell(\mathbf{x}_c)\|_F^2. \quad (14.36)$$

И наконец, определяем общую стилистическую потерю как сумму потерь по множеству слоев \mathcal{S} :

$$\mathcal{L}_{\text{style}}^\ell(\mathbf{x}, \mathbf{x}_c) = \sum_{\ell \in \mathcal{S}} \mathcal{L}_{\text{style}}^\ell(\mathbf{x}, \mathbf{x}_c). \quad (14.37)$$

Например, на рис. 14.40 мы вычисляем стилистическую потерю в слоях 1 и 3. (Нижние слои улавливают визуальную текстуру, а верхние – расположение объектов.)

14.6.5.2. Ускорение метода

В работе [GEB16] использован метод L-BFGS (раздел 8.3.2) для оптимизации функции (14.33), начиная с белого шума. Получить результаты можно быстрее, если вместо BFGS воспользоваться оптимизатором типа Adam и начать с оригинального изображения, а не с белого шума. Тем не менее прогон оптимизатора для каждого нового стиля и оригинального изображения занимает много времени. В нескольких работах (см., например, [JAFF16; Uly+16; UVL16; LW16]) предложено обучать нейронную сеть прямому предсказанию результата этой оптимизации, а не решать уравнение для каждой новой пары изображений. (Это можно рассматривать как разновидность амортизированной оптимизации.) Именно, для каждого образца стиля \mathbf{x}_s мы обучаем модель f_s такую, что $f_s(\mathbf{x}_c) = \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}|\mathbf{x}_s, \mathbf{x}_c)$. Затем эта модель применяется к новым оригинальным изображениям без необходимости повторной оптимизации.

Позже в работе [DSK16] было показано, как обучить одну сеть, которая принимает на входе оригинальное изображение и дискретное представление s стиля и порождает на выходе $f(\mathbf{x}_c, s) = \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}|s, \mathbf{x}_c)$. Это позволяет обойтись без отдельной сети для каждого образца стиля. Ключевая идея – стандартизировать признаки в заданном слое, используя параметры масштаба и сдвига, специфичные для стиля. Конкретно используется следующее преобразование **условной индивидуальной нормировки**:

$$\text{CIN}(\phi(\mathbf{x}_c), s) = \gamma_s \left(\frac{\phi(\mathbf{x}_c) - \mu(\phi(\mathbf{x}_c))}{\sigma(\phi(\mathbf{x}_c))} \right) + \beta_s, \quad (14.38)$$

где $\mu(\phi(\mathbf{x}_c))$ – среднее признаков в данном слое, $\sigma(\phi(\mathbf{x}_c))$ – их стандартное отклонение, а β_s и γ_s – параметры стиля s . (Подробнее об индивидуальной нормировке см. раздел 14.2.4.2.) Удивительно, но этого простого трюка достаточно для улавливания особенностей многих стилей.

Недостаток описанного метода в том, что он работает только для фиксированного числа дискретных стилей. В работе [НВ17] показано, как обобщить метод, заменив константы β_s и γ_s выходом еще одной СНС, которая принимает произвольный образец стиля \mathbf{x}_s на входе. То есть в формуле (14.38) мы полагаем $\beta_s = f_\beta(\phi(\mathbf{x}_s))$, $\gamma_s = f_\gamma(\phi(\mathbf{x}_s))$ и обучаем параметры β и γ вместе со всеми остальными. Модель принимает вид:

$$\text{AIN}(\phi(\mathbf{x}_c), \phi(\mathbf{x}_s)) = f_\gamma(\phi(\mathbf{x}_s)) \left(\frac{\phi(\mathbf{x}_c) - \mu(\phi(\mathbf{x}_c))}{\sigma(\phi(\mathbf{x}_c))} \right) + f_\beta(\phi(\mathbf{x}_s)). \quad (14.39)$$

Авторы назвали этот метод **адаптивной индивидуальной нормировкой**.

Глава 15

Нейронные сети для последовательностей

15.1. ВВЕДЕНИЕ

В этой главе мы обсудим различные виды нейронных сетей для последовательностей. Мы рассмотрим случаи, когда вход, выход или то и другое являются последовательностями. У таких моделей много приложений, в том числе машинный перевод, распознавание речи, классификация текста, автоматическое описание изображений и т. д. Наше изложение частично следует книге [Zha+20], которую мы рекомендуем для ознакомления с деталями.

15.2. РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ (РНС)

Рекуррентной нейронной сетью (РНС, англ. RNN) называется нейронная сеть, которая отображает пространство входных последовательностей в пространство выходных последовательностей, сохраняя информацию о состоянии. То есть предсказание выхода y_t зависит не только от входа x_t , но и от скрытого состояния системы h_t , которое обновляется со временем по мере обработки последовательности. Такие модели можно использовать для генерирования, классификации и перевода последовательностей, как будет объяснено ниже¹.

15.2.1. Vec2Seq (генерирование последовательностей)

В этом разделе мы обсудим, как обучать функции вида $f_\theta : \mathbb{R}_D \rightarrow \mathbb{R}^{N_\infty C}$, где D – размер входного вектора, а выходом является последовательность произвольной длины, состоящая из векторов размера C . (Отметим, что слова – это

¹ Более подробное введение см. в статье <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>.

дискретные токены, но их можно преобразовать в вещественные векторы, как описано в разделе 1.5.4.) Такие модели носят название **vec2seq**, поскольку отображают вектор в последовательность.

Выходная последовательность $y_{1:T}$ генерируется поэлементно. На каждом шаге мы производим выборку \tilde{y}_t из скрытого состояния модели h_t , а затем подаем ее обратно на вход модели, чтобы получить новое состояние h_{t+1} (зависящее также от входа x) (см. иллюстрацию на рис. 15.1). Таким образом, мы определили условную порождающую модель вида $p(y_{1:T}|x)$, которая улавливает зависимости между выходными токенами. Более подробно мы объясним это ниже.

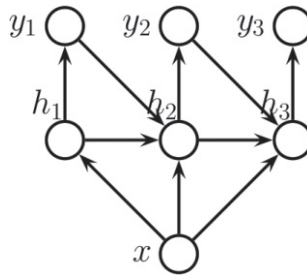


Рис. 15.1 ❖ Рекуррентная нейронная сеть (РНС) для генерирования выходной последовательности переменной длины $y_{1:T}$ по необязательному входному вектору x фиксированной длины

15.2.1.1. Модели

Для простоты обозначим T длину выхода (понимая, что на самом деле она выбирается динамически). Тогда РНС соответствует следующая условная порождающая модель:

$$p(y_{1:T}|x) = \sum_{h_{1:T}} p(y_{1:T}, h_{1:T}|x) = \sum_{h_{1:T}} \prod_{t=1}^T p(y_t|h_t) p(h_t|h_{t-1}, y_{t-1}, x), \quad (15.1)$$

где h_t – скрытое состояние, а $p(h_1|h_0, y_0, x) = p(h_1|x)$ – начальное скрытое распределение состояний (часто детерминированное).

Выходное распределение обычно имеет вид:

$$p(y_t|h_t) = \text{Cat}(y_t|\mathcal{S}(\mathbf{W}_{hy}h_t + \mathbf{b}_y)), \quad (15.2)$$

где \mathbf{W}_{hy} – веса связей между скрытыми и выходными блоками, а \mathbf{b}_y – смещение. Однако если выход вещественный, то можно использовать распределение:

$$p(y_t|h_t) = \mathcal{N}(y_t|\mathbf{W}_{hy}h_t + \mathbf{b}_y, \sigma^2\mathbf{I}). \quad (15.3)$$

Предполагается, что скрытое состояние вычисляется по формуле:

$$p(h_t|h_{t-1}, y_{t-1}, x) = \mathbb{I}(h_t = f(h_{t-1}, y_{t-1}, x)), \quad (15.4)$$

где f – некоторая детерминированная функция обновления вида

$$\mathbf{h}_t = \varphi(\mathbf{W}_{xh}[\mathbf{x}; \mathbf{y}_{t-1}] + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h), \quad (15.5)$$

где \mathbf{W}_{hh} – веса связей, ведущих из скрытых блоков в скрытые, \mathbf{W}_{xh} – веса связей, ведущих из входных блоков в скрытые, а \mathbf{b}_h – смещения (см. иллюстрацию на рис. 15.1 и демонстрационный код по адресу code.probl.ai/book1/rnn_torch).

Заметим, что \mathbf{y}_t зависит от \mathbf{h}_t , которое зависит от \mathbf{y}_{t-1} , которое зависит от \mathbf{h}_{t-1} , и т. д. Таким образом, \mathbf{y}_t неявно зависит от всех прошлых наблюдений (а также от факультативного фиксированного входа \mathbf{x}). Стало быть, РНС выходят за рамки стандартных марковских моделей, поскольку могут иметь неограниченную память. Благодаря этому РНС теоретически обладают такой же мощностью, как **машина Тьюринга** [SS95; PMB19]. Но на практике объем памяти ограничен размером скрытого состояния и силой параметров; дальнейшее обсуждение этого вопроса см. в разделе 15.2.7.

В процессе генерирования выхода РНС мы производим выборку из распределения $\tilde{\mathbf{y}}_t \sim p(\mathbf{y}_t | \mathbf{h}_t)$ и подаем выбранное значение слою скрытого состояния, чтобы детерминированно вычислить $\mathbf{h}_{t+1} = f(\mathbf{h}_t, \tilde{\mathbf{y}}_t, \mathbf{x})$, затем выбрать $\tilde{\mathbf{y}}_{t+1} \sim p(\mathbf{y}_{t+1} | \mathbf{h}_{t+1})$ и т. д. Таким образом, стохастичность, присутствующая в системе, проистекает только из шума в модели наблюдений (выходе), который подается обратно на вход на каждом шаге. (Однако имеется вариант, называемый **вариационной РНС** [Chu+15], при котором стохастичность добавляется в динамику \mathbf{h}_t независимо от шума наблюдений.)

15.2.1.2. Приложения

РНС можно использовать для безусловного генерирования последовательностей (положив $\mathbf{x} = \emptyset$) или условного, зависящего от \mathbf{x} . Безусловное генерирование последовательностей часто называют **языковым моделированием**; это означает обучение совместному распределению вероятностей последовательностей дискретных токенов, т. е. моделей вида $p(y_1, \dots, y_T)$ (см. также раздел 3.6.1.2, где обсуждалось применение марковских цепей для языкового моделирования).

На рис. 15.2 показана последовательность, сгенерированная простой РНС, обученной на романе «Машина времени» Герберта Уэллса. (Это небольшой текст, содержащий всего 32 000 слов и 170 000 знаков.) Мы видим, что сгенерированная последовательность выглядит вполне правдоподобно, хотя и не слишком осмыслена. Воспользовавшись более сложными моделями (например, обсуждаемыми в разделах 15.2.7.1 и 15.2.7.2) и взяв больше обучающих данных, мы можем создать РНС, демонстрирующую качество самого современного уровня на задаче языкового моделирования [CNB17]. (В области языкового моделирования качество обычно измеряют перплексией, равной экспоненте отрицательного логарифмического правдоподобия, усредненного по токенам; дополнительную информацию см. в разделе 6.1.5.)

the githa some thong the time traveller held in his hand was a glitteringmetallic framework scarcely larger than a small clock and verydelicately made there was ivory in it and the latter than s bettyre tat howhong s ie time thave ler simk you a dimensions le ghat dionthat shall travel indifferently in any direction of space and timeas the driver determinesilby contented himself with laughterbut i have experimental verification said the time travellerit would be remarkably convenient for the histo

Рис. 15.2 ❖ Пример выхода длины 500, сгенерированного РНС уровня символов, на вход которой подано начальное слово «the». Мы используем жадное декодирование, когда на каждом шаге вычисляется наиболее вероятный символ, который затем подается обратно на вход модели. Модель обучена на романе Герберта Уэллса «Машина времени». Сгенерировано программой по адресу figures.probl.ai/book1/15.2

Мы также можем сделать так, что сгенерированная последовательность будет зависеть от входного вектора x того или иного вида. Например, рассмотрим задачу **описания изображений**: в этом случае x – некоторое погружение изображения, вычисленное СНС, как показано на рис. 15.3 (см. обзор методов описания изображений в работах [Hos+19; LXW19], а пособие с демонстрационным кодом – по адресу <https://bit.ly/2Wvs1GK>).

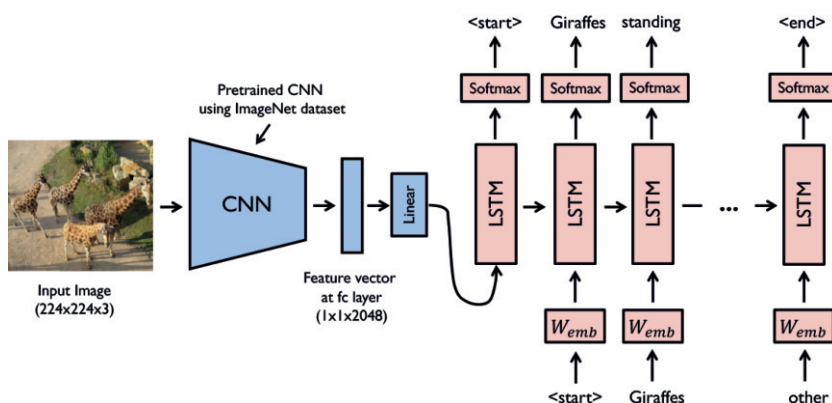


Рис. 15.3 ❖ Модель, состоящая из СНС и РНС, для описания изображений. Розовые блоки «LSTM» соответствуют специальному виду РНС, который мы обсудим в разделе 15.2.7.2. Розовые блоки «Wemb» соответствуют матрицам погружения для (выборочных) унитарных токенов, так что вход модели состоит из вещественных векторов. Взято из статьи <https://bit.ly/2FKnqHm>. Печатается с разрешения Юнджи Чоя

Можно использовать РНС и для генерирования последовательностей вещественных векторов признаков, например росчерков пера для рукописных символов [Gra13] и нарисованных от руки фигур [HE18]. Это полезно также для генерирования временных рядов, предсказывающих последовательности вещественных чисел.

15.2.2. Seq2Vec (классификация последовательностей)

В этом разделе мы предполагаем, что нужно предсказать один выходной вектор y фиксированной длины по входной последовательности переменной длины. То есть мы хотим обучить функцию вида $f_\theta : \mathbb{R}^{TD} \rightarrow \mathbb{R}^C$. Назовем это моделью **seq2vec**. Для простоты обозначений ограничимся случаем, когда на выходе получается метка класса $y \in \{1, \dots, C\}$.

Самый простой подход – использовать конечное состояние РНС как вход классификатора:

$$p(y|\mathbf{x}_{1:T}) = \text{Cat}(y|\mathcal{S}(\mathbf{W}\mathbf{h}_T)). \quad (15.6)$$

Смотрите иллюстрацию на рис. 15.4а.

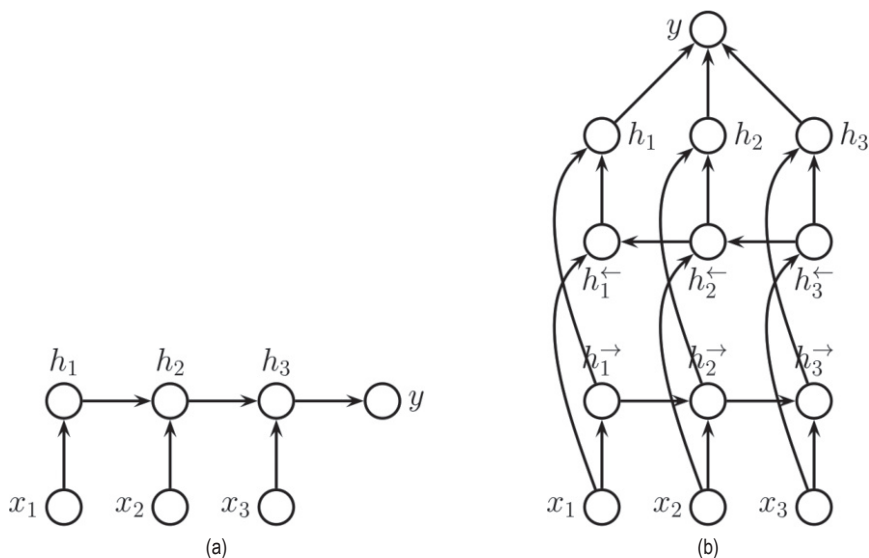


Рис. 15.4 ❖ (а) РНС для классификации последовательностей.
(б) Двухнаправленная РНС для классификации последовательностей

Зачастую можно добиться лучших результатов, если допустить зависимость скрытых состояний РНС от прошлого и будущего контекста. Для этого мы создаем две РНС, одна из которых рекурсивно вычисляет скрытые состояния в прямом направлении, а вторая – в обратном. Это называется **двухнаправленной РНС** [SP97]. Формально модель определена следующим образом:

$$\mathbf{h}_t^{\rightarrow} = \varphi(\mathbf{W}_{xh}^{\rightarrow} \mathbf{x}_t + \mathbf{W}_{hh}^{\rightarrow} \mathbf{h}_{t-1}^{\rightarrow} + \mathbf{b}_h^{\rightarrow}), \quad (15.7)$$

$$\mathbf{h}_t^{\leftarrow} = \varphi(\mathbf{W}_{xh}^{\leftarrow} \mathbf{x}_t + \mathbf{W}_{hh}^{\leftarrow} \mathbf{h}_{t+1}^{\leftarrow} + \mathbf{b}_h^{\leftarrow}). \quad (15.8)$$

Затем мы можем определить $\mathbf{h}_t = [\mathbf{h}_t^{\rightarrow}, \mathbf{h}_t^{\leftarrow}]$ как представление состояния в момент t , приняв во внимание информацию о прошлом и будущем. Наконец, мы применяем пулинг усреднением к скрытым состояниям и получаем окончательный классификатор:

$$p(y|\mathbf{x}_{1:T}) = \text{Cat}(y|\mathbf{WS}(\bar{\mathbf{h}})), \quad (15.9)$$

$$\bar{\mathbf{h}} = \frac{1}{T} \sum_{t=1}^T \mathbf{h}_t. \quad (15.10)$$

Смотрите иллюстрацию на рис. 15.4b и код по адресу code.problml.ai/book1/rnn_sentiment_torch. (Это похоже на одномерную СНС для классификации текста в разделе 15.3.1.)

15.2.3. Seq2Seq (трансляция последовательностей)

В этом разделе мы обсудим обучение функций вида $f_{\theta} : \mathbb{R}^{TD} \rightarrow \mathbb{R}^{T'C}$. Будем рассматривать два случая: $T' = T$, т. е. длины входной и выходной последовательности одинаковы (а значит, они выровнены), и $T' \neq T$, т. е. их длины различаются. Это называется задачей **seq2seq**.

15.2.3.1. Выровненный случай

Пусть входная и выходная последовательности выровнены. Эту задачу можно также интерпретировать как **плотную разметку последовательности**, так как мы предсказываем по одной метке на каждую позицию. Легко модифицировать РНС для решения этой задачи, как показано на рис. 15.5a. Это соответствует распределению

$$p(y_{1:T}|\mathbf{x}_{1:T}) = \sum_{\mathbf{h}_{1:T}} \prod_{t=1}^T p(y_t|\mathbf{h}_t) \mathbb{I}(\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)), \quad (15.11)$$

где $\mathbf{h}_1 = f(\mathbf{h}_0, \mathbf{x}_1) = f_0(\mathbf{x}_1)$ – начальное состояние.

Заметим, что \mathbf{y}_t зависит от \mathbf{h}_t , которое зависит только от прошлых входов, $\mathbf{x}_{1:t}$. Можно улучшить результаты, если позволить декодеру заглядывать в «будущее» \mathbf{x} , а не только в прошлое. Для этого нужно использовать **двунаправленную РНС**, как показано на рис. 15.5b.

Можно создать более выразительные модели, расположив несколько скрытых цепочек одну под другой, как показано на рис. 15.6. Скрытые блоки слоя 1 в момент t вычисляются по формуле:

$$\mathbf{h}_t^l = \varphi_l(\mathbf{W}_{xh}^l \mathbf{h}_t^{l-1} + \mathbf{W}_{hh}^l \mathbf{h}_{t-1}^l + \mathbf{b}_h^l). \quad (15.12)$$

Выход имеет вид:

$$\mathbf{o}_t = \mathbf{W}_{ho} \mathbf{h}_t^L + \mathbf{b}_o. \quad (15.13)$$

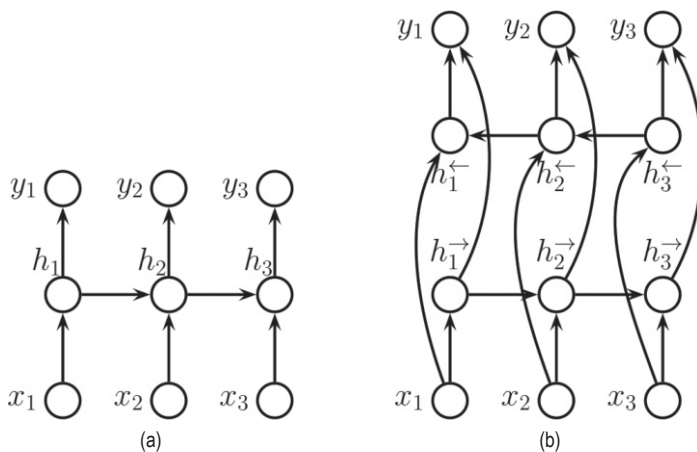


Рис. 15.5 ❖ (a) РНС для преобразования одной последовательности в другую, выровненную. (b) Двухнаправленная РНС для той же задачи

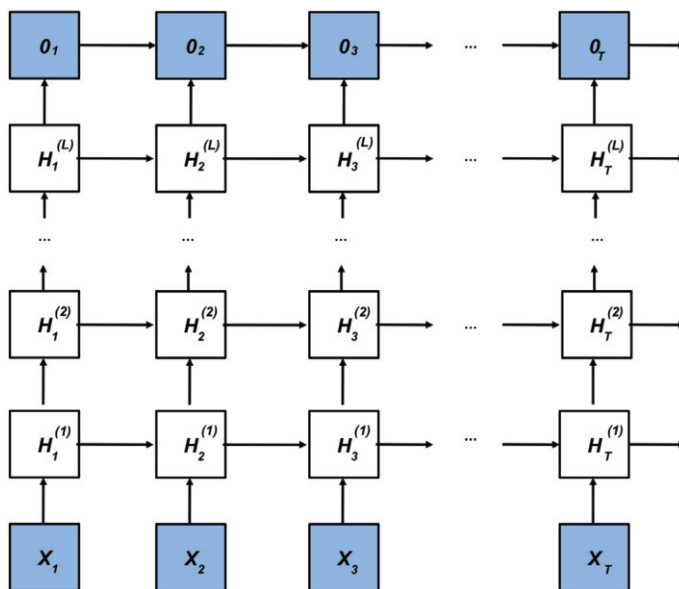


Рис. 15.6 ❖ Глубокая РНС. На основе рис. 9.3.1 из работы [Zha+20]

15.2.3.2. Невыровненный случай

В этом разделе мы обсудим, как обучить отображение одной последовательности длины T в другую последовательность длины T' . Сначала закодируем первую последовательность и получим вектор контекста $\mathbf{c} = f_e(\mathbf{x}_{1:T})$, используя последнее состояние РНС (или пулинг усреднением для двухнаправленной РНС). Затем сгенерируем выходную последовательность, применяя РНС-декодер $\mathbf{y}_{1:T'} = f_d(\mathbf{c})$. Это называется **архитектурой кодировщик-декодер** [SVL14; Cho+14b] (см. иллюстрацию на рис. 15.7).

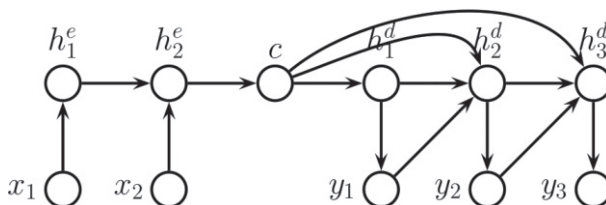


Рис. 15.7 ❖ РНС с архитектурой кодировщик-декодер для отображения последовательности $x_{1:T}$ в последовательность $y_{1:T}$

Важное приложение этой сети – **машинный перевод**. Когда эта задача решается с помощью РНС, говорят о **нейронном машинном переводе** (в отличие от более старого подхода, называемого **статистическим машинным переводом**, в котором нейронные сети не используются). Основная идея показана на рис. 15.8а, а в коде по адресу code.problml.ai/book1/nmt_torch имеются дополнительные детали. Обзор литературы по нейронному машинному переводу см. в работах [Luo16; Neu17].

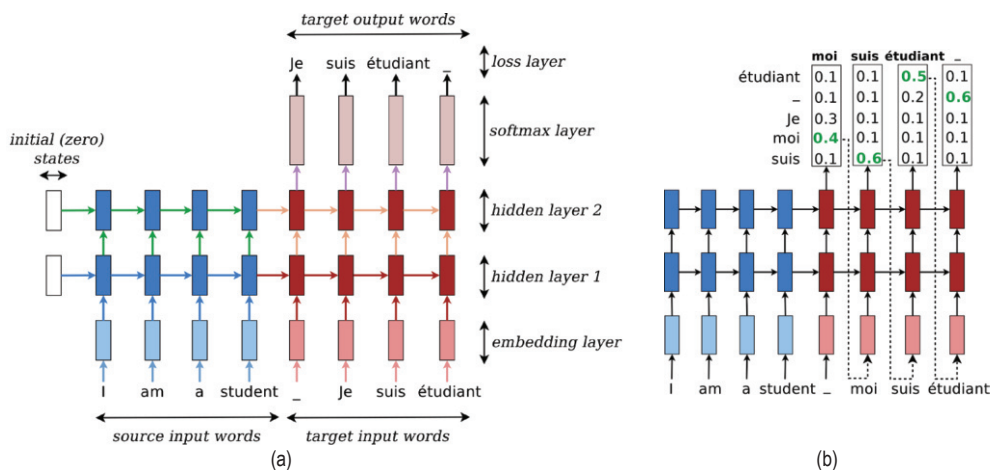


Рис. 15.8 ❖ (а) Модель seq2seq для перевода с английского на французский. Символ - обозначает конец последовательности. На основе рис. 2.4 из работы [Luo16]. Печатается с разрешения Минь-Тан Луонга. (б) Жадное декодирование. Самое вероятное французское слово на каждом шаге выделено зеленым цветом, именно оно подается на вход следующего шага декодера. На основе рис. 2.5 из работы [Luo16]. Печатается с разрешения Минь-Тан Луонга

15.2.4. Принуждение со стороны учителя

При обучении языковой модели вероятность последовательности слов w_1, w_2, \dots, w_T равна

$$p(w_{1:T}) = \prod_{t=1}^T p(w_t | w_{1:t-1}). \quad (15.14)$$

Поэтому в РНС мы полагаем вход $x_t = w_{t-1}$ и выход $y_t = w_t$. Заметим, что в качестве условия выступают *истинные* метки из прошлого, $w_{1:t-1}$, а не метки, сгенерированные моделью. Это называется **принуждением со стороны учителя**, поскольку предложенные учителем значения «насиленно» подаются на вход модели на каждом шаге (т. е. x_t полагается равным w_{t-1}).

К сожалению, принуждение со стороны учителя иногда может приводить к моделям, которые плохо ведут себя на этапе тестирования. Причина в том, что модель обучалась только на заведомо «правильных» примерах, поэтому может не знать, что делать, встретив во время тестирования сгенерированную на предыдущем шаге входную последовательность $w_{1:t-1}$, которая отличается от той, что встречалась во время обучения.

Общепринятое решение этой проблемы известно под названием **плановая выборка** (scheduled sampling) [Ben+15a]. Начинается все с принуждения со стороны учителя, но на случайных временных шагах на вход подаются примеры из модели, причем доля таких шагов постепенно увеличивается.

Альтернативно можно использовать другие виды моделей, в которых оценка максимального правдоподобия работает лучше, например одномерные СНС (раздел 15.3) и трансформеры (раздел 15.5).

15.2.5. Обратное распространение во времени

Оценку максимального правдоподобия параметров для РНС можно вычислить, решив задачу $\theta^* = \arg\max_{\theta} p(y_{1:T} | x_{1:T}, \theta)$, где для простоты обозначений мы предположили, что обучающая последовательность всего одна. Для вычисления MLE необходимо вычислить градиенты потери по параметрам. Для этого мы можем развернуть граф вычислений, как показано на рис. 15.9, а затем применить алгоритм обратного распространения. Это называется **обратным распространением во времени (BPTT)** [Wer90]:

$$h_t = W_{hx}x_t + W_{hh}h_{t-1}; \quad (15.15)$$

$$o_t = W_{ho}h_t, \quad (15.16)$$

где o_t – выходные логиты, а члены смещения опущены, чтобы не усложнять обозначения. Мы предполагаем, что y_t – истинные целевые метки на каждом временном шаге, поэтому определим потерю следующим образом:

$$L = \frac{1}{T} \sum_{t=1}^T \ell(y_t, o_t). \quad (15.17)$$

Необходимо вычислить производные $\partial L / \partial W_{hx}$, $\partial L / \partial W_{hh}$ и $\partial L / \partial W_{ho}$. Последнюю вычислить просто, потому что она локальна на каждом временном шаге. Но первые две зависят от скрытого состояния, поэтому придется возвращаться назад во времени.

Упростим обозначения, определив

$$h_t = f(x_t, h_{t-1}, w_h); \quad (15.18)$$

$$\mathbf{o}_t = g(\mathbf{h}_t, \mathbf{w}_o), \quad (15.19)$$

где \mathbf{w}_h – развернутые в вектор и объединенные вместе \mathbf{W}_{hh} и \mathbf{W}_{hx} . Сосредоточимся на вычислении $\partial L / \partial \mathbf{w}_h$. По правилу дифференцирования сложной функции имеем

$$\frac{\partial L}{\partial \mathbf{w}_h} = \frac{1}{T} \sum_{t=1}^T \frac{\partial \ell(y_t, \mathbf{o}_t)}{\partial \mathbf{w}_h} = \frac{1}{T} \sum_{t=1}^T \frac{\partial \ell(y_t, \mathbf{o}_t)}{\partial \mathbf{o}_h} \frac{\partial g(\mathbf{h}_t, \mathbf{w}_o)}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{w}_h}. \quad (15.20)$$

Последний член можно представить в виде суммы:

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{w}_h} = \frac{\partial f(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{w}_h)}{\partial \mathbf{w}_t} + \frac{\partial f(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{w}_h)}{\partial \mathbf{h}_{t-1}} \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{w}_h}. \quad (15.21)$$

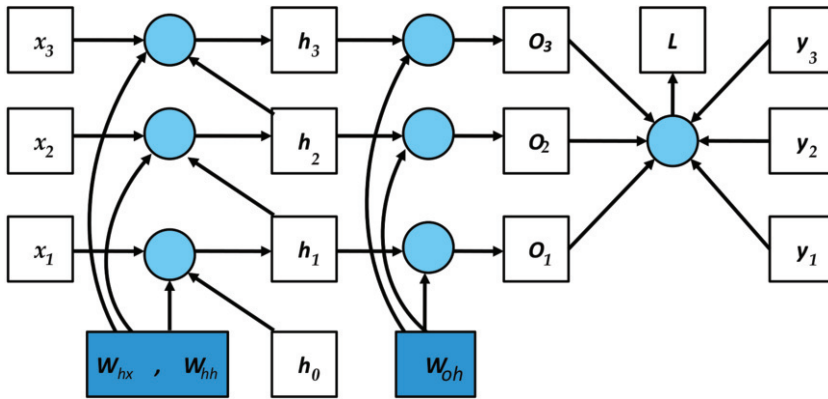


Рис. 15.9 ❖ РНС, развернутая (по вертикали) на три временных шага; целевая выходная последовательность и узел потери выделены явно. На основе рис. 8.7.2 из работы [Zha+20]. Печатается с разрешения Астона Чжана

Проделав такую замену рекурсивно, получаем следующий результат (см. вывод в [Zha+20, раздел 8.7]):

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{w}_h} = \frac{\partial f(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{w}_h)}{\partial \mathbf{w}_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^t \frac{\partial f(\mathbf{x}_j, \mathbf{h}_{j-1}, \mathbf{w}_h)}{\partial \mathbf{h}_{j-1}} \right) \frac{\partial f(\mathbf{x}_i, \mathbf{h}_{i-1}, \mathbf{w}_h)}{\partial \mathbf{w}_h}. \quad (15.22)$$

К сожалению, время вычислений составляет $O(T)$ на каждом временном шаге, так что общее время имеет порядок $O(T^2)$. Поэтому принято оставлять в сумме не более K недавних членов. Можно адаптивно выбирать подходящий параметр отсечения K [AFF19]; однако обычно он устанавливается равным длине подпоследовательности в текущем мини-пакете.

При использовании усеченного ВРТТ мы можем обучить модель на пакетах коротких последовательностей, которые обычно создаются путем извлечения непересекающихся подпоследовательностей (окон) исходной последовательности. Если предыдущая последовательность заканчивается

в момент $t - 1$, а текущая начинается в момент t , то мы можем «переносить» скрытое состояние РНС из одного пакетного обновления в другое в процессе обучения. Однако если последовательности не упорядочены, то необходимо сбрасывать скрытое состояние. Демонстрационный код, иллюстрирующий эти детали, имеется по адресу code.problml.ai/book1/rnn_torch.

15.2.6. Исчезающие и взрывные градиенты

К сожалению, активации в РНС могут стремиться к нулю или взрывообразно расти при продвижении вперед во времени, так как на каждом шаге мы умножаем на матрицу весов \mathbf{W}_{hh} . Аналогично могут вести себя градиенты, поскольку на каждом шаге мы перемножаем якобианы (детали см. в разделе 13.4.2). Простой эвристический прием – использовать отсечение градиентов (формула (13.70)). В более сложных методах производится попытка контролировать спектральный радиус λ прямого отображения, \mathbf{W}_{hh} , а также обратного отображения, описываемого якобианом \mathbf{J}_{hh} .

Простейший способ контролировать спектральный радиус – случайным образом инициализировать матрицу \mathbf{W}_{hh} , так чтобы $\lambda \approx 1$, а затем зафиксировать ее (т. е. не обучать \mathbf{W}_{hh}). В этом случае обучать нужно только выходную матрицу \mathbf{W}_{ho} , что сводится к задаче выпуклой оптимизации. Это называется **эхо-сетью** (echo state network – ESN) [JH04]. В близком подходе, известном под названием **машина неустойчивых состояний** (liquid state machine – LSM) [MNM02], используются бинарные (импульсные) нейроны вместо вещественных нейронов. Общий термин для ESN и LSM – **резервуарные вычисления** [LJ09]. Еще один подход к этой проблеме – использовать условную оптимизацию, гарантирующую, что матрица \mathbf{W}_{hh} всегда будет оставаться ортогональной [Vor+17].

Альтернатива явному контролю над спектральным радиусом – модификация самой архитектуры РНС, а именно добавление аддитивных, а не мультипликативных обновлений скрытых состояний (см. раздел 15.2.7). Это значительно повышает устойчивость обучения.

15.2.7. Вентильная и долгосрочная память

РНС с достаточным числом скрытых блоков в принципе могут запомнить входные данные из далекого прошлого. Но на практике так не поступают из-за проблемы исчезающего градиента (раздел 13.4.2). В этом разделе мы расскажем, как решить эту проблему, обновляя скрытое состояние аддитивным способом, как в остаточной сети (раздел 14.3.4).

15.2.7.1. Управляемые рекуррентные блоки (GRU)

В этом разделе мы обсудим модели, в которых используются управляемые рекуррентные блоки (GRU), как предлагается в работе [Cho+14b]. Основная идея – обучиться, когда обновлять скрытое состояние, используя управле-

мый блок. Это можно использовать для избирательного «запоминания» важных частей информации при первой встрече. Модель также можно обучить тому, когда сбрасывать скрытое состояние и, следовательно, забывать вещи, когда они больше не нужны.

Чтобы объяснить модель более детально, мы разобьем ее на два шага, следуя изложению в работе [Zha+20, раздел 8.8]. Предположим, что \mathbf{X}_t – матрица $N \times D$, где N – размер пакета, а D – размер словаря. Аналогично \mathbf{H}_t – матрица $N \times H$, где H – число скрытых блоков в момент t .

Вентиль сброса $\mathbf{R}_t \in \mathbb{R}^{N \times H}$ и **вентиль обновления** $\mathbf{Z}_t \in \mathbb{R}^{N \times H}$ вычисляются по формулам:

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r); \quad (15.23)$$

$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z), \quad (15.24)$$

где \mathbf{W} – матрица весов. Заметим, что все элементы \mathbf{R}_t и \mathbf{Z}_t принадлежат отрезку $[0, 1]$, поскольку σ – сигмоидная функция.

Зная это, определим следующий кандидат на роль вектора состояния:

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h). \quad (15.25)$$

Эта формула объединяет еще не сброшенную память о прошлом (член $\mathbf{R}_t \odot \mathbf{H}_{t-1}$) с новыми входами \mathbf{X}_t . Результирующую линейную комбинацию мы передаем функции \tanh , чтобы скрытые блоки оставались в интервале $(-1, 1)$. Если элементы вентиля сброса \mathbf{R}_t близки к 1, то мы возвращаемся к стандартному правилу обновления РНС. Если же они близки к 0, то модель больше напоминает МСП, примененный к \mathbf{X}_t . Таким образом, вентиль сброса способен уловить новую краткосрочную информацию.

Вычислив кандидата на роль нового состояния, модель вычисляет фактическое новое состояние, используя те измерения состояния-кандидата $\tilde{\mathbf{H}}_t$, которые были отобраны вентилем обновления, $1 - \mathbf{Z}_t$, а для всех остальных оставляя старые значения из \mathbf{H}_{t-1} :

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t. \quad (15.26)$$

Когда $Z_{td} = 1$, мы передаем $H_{t-1,d}$ без изменения и игнорируем \mathbf{X}_t . Таким образом, вентиль обновления может уловить долгосрочные зависимости.

Смотрите общую архитектуру на рис. 15.10, а демонстрационный код по адресу code.problm.ai/book1/gru_torch.

15.2.7.2. Долгая краткосрочная память (LSTM)

В этом разделе мы обсудим модель **долгой краткосрочной памяти (LSTM)** из работы [HS97b] – более сложный вариант GRU (который, к тому же, появился почти на 20 лет раньше). Более обстоятельное введение см. в статье по адресу <https://colah.github.io/posts/2015-08-Understanding-LSTMs>.

Основная идея состоит в том, чтобы дополнить скрытое состояние \mathbf{h}_t **ячейкой памяти** \mathbf{c}_t . Для управления этой ячейкой нам нужно три вентиля: **выходной вентиль** \mathbf{O}_t определяет, что читать из ячейки; **входной вентиль** \mathbf{I}_t

614 ❖ Нейронные сети для последовательностей

определяет, что помещать в ячейку; **вентиль забывания** определяет, когда сбрасывать ячейку. Эти три вентиля вычисляются следующим образом:

$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o); \quad (15.27)$$

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i); \quad (15.28)$$

$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f). \quad (15.29)$$

Затем мы вычисляем потенциальное состояние ячейки:

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c). \quad (15.30)$$

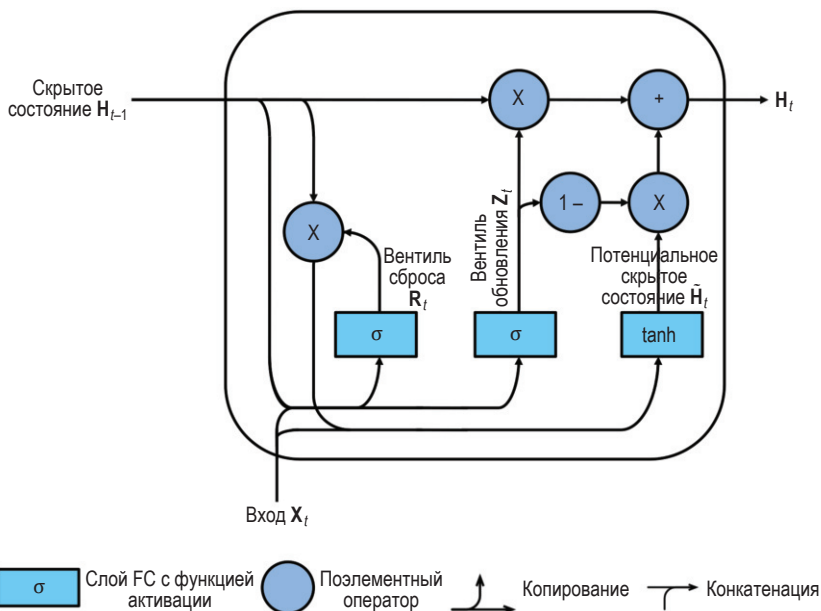


Рис. 15.10 ❖ Иллюстрация GRU. На основе рис. 9.1.3 из работы [Zha+20]

Фактическое новое состояние ячейки совпадает либо с потенциальным (если входной вентиль открыт), либо со старым (если вентиль забывания закрыт):

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t. \quad (15.31)$$

Если $\mathbf{F}_t = 1$ и $\mathbf{I}_t = 0$, то это напоминает долгосрочную память¹.

Наконец, мы можем вычислить скрытое состояние как результате преобразования ячейки при условии, что выходной вентиль открыт:

¹ В работе [JZS15] отмечена одна важная деталь: необходимо инициализировать смещение, чтобы вентиль забывания \mathbf{b}_f был велик, т. е. сигмоида была близка к 1. Это гарантирует, что информация сможет легко пройти по цепочке \mathbf{C} со временем. Если этого не сделать, то зачастую качество модели резко ухудшается.

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t). \quad (15.32)$$

Заметим, что \mathbf{H}_t является одновременно выходом блока и скрытым состоянием на следующем временном шаге. Это позволяет модели запоминать, что она только что вывела (краткосрочная память), тогда как ячейка \mathbf{C}_t работает как долгосрочная память. На рис. 15.11 показано, как выглядит модель в целом, а по адресу code.problml.ai/book1/lstm_torch имеется демонстрационный код.

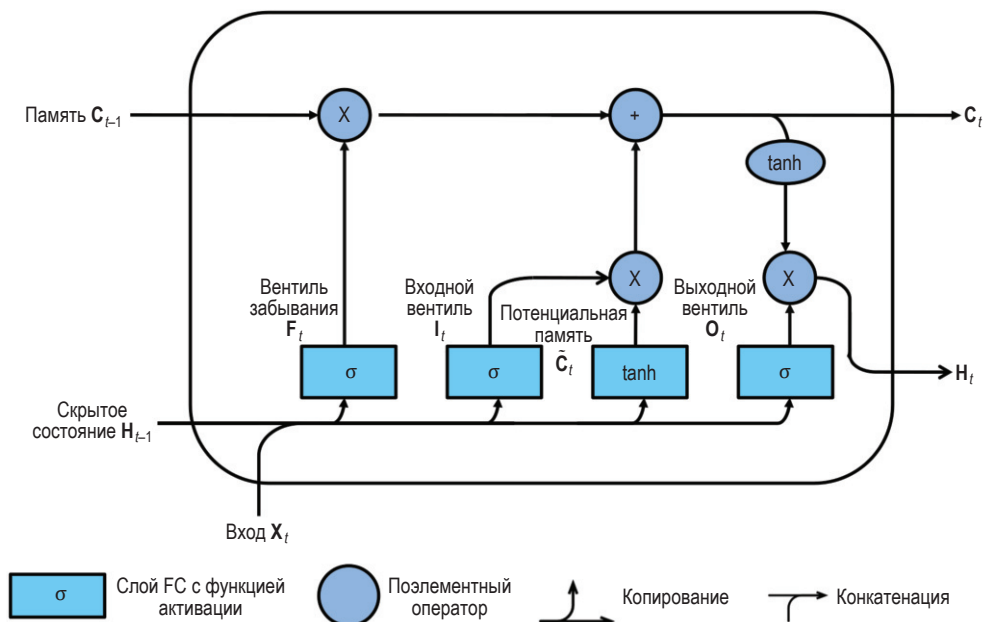


Рис. 15.11 ❖ Иллюстрация LSTM. На основе рис. 9.2.4 из работы [Zha+20]

Иногда добавляются **связи-глазки**, позволяющие передать состояние ячейки вентилям в качестве дополнительного входа. Было предложено и много других вариантов. В работе [JZS15] даже применен генетический алгоритм для тестирования свыше 10 000 различных архитектур. Некоторые из них работали лучше, чем LSTM или GRU, но в общем LSTM неизменно показывали хорошее качество на большинстве задач. К похожим выводам пришли авторы работы [Gre+17]. В сравнительно недавней работе [ZL17] контроллер на основе РНС использовался для генерирования строк, описывающих архитектуры РНС, а затем этот контроллер обучался методами обучения с подкреплением. Результатом стала новая структура ячейки, превосходящая LSTM. Однако она довольно сложна, и сообщество ее не приняло.

15.2.8. Лучевой поиск

Самый простой способ генерирования чего-то с помощью РНС – использовать **жадное декодирование**, когда на каждом шаге вычисляется $\hat{y}_t =$

$\operatorname{argmax}_y p(y_t = y | \hat{y}_{1:t}, \mathbf{x})$. Этот процесс повторяется, пока не будет сгенерирован токен конца предложения. На рис. 15.8b показано применение этого метода к нейронному машинному переводу.

К сожалению, жадное декодирование не порождает MAP-последовательности, которая определяется как $\mathbf{y}_{1:T}^* = \operatorname{argmax}_{\mathbf{y}_{1:T}} p(\mathbf{y}_{1:T} | \mathbf{x})$. Причина в том, что локально оптимальный символ на шаге t может не лежать на глобально оптимальном пути.

В качестве примера рассмотрим рис. 15.12a. Мы жадно выбираем символ с максимальной апостериорной вероятностью MAP на шаге 1, им является А. Предположим, что при таком условии $p(y_2 | y_1 = A) = [0.1, 0.4, 0.3, 0.3]$, как и показано на рисунке. Символом с максимальной MAP тогда будет В. Предположим, что при таком условии $p(y_3 | y_1 = A, y_2 = B) = [0.2, 0.2, 0.4, 0.2]$, как и показано. Тогда символом с максимальной MAP будет С. Предположим, что при таком условии $p(y_4 | y_1 = A, y_2 = B, y_3 = C) = [0.0, 0.2, 0.2, 0.6]$, как и показано. Тогда символом с максимальной MAP будет eos (конец предложения), поэтому генерирование прекращается. Общая вероятность сгенерированной последовательности равна $0.5 \times 0.4 \times 0.4 \times 0.6 = 0.048$.

Временной шаг	1	2	3	4	Временной шаг	1	2	3	4
A	0.5	0.1	0.2	0.0	A	0.5	0.1	0.1	0.1
B	0.2	0.4	0.2	0.2	B	0.2	0.4	0.6	0.2
C	0.2	0.3	0.4	0.2	C	0.2	0.3	0.2	0.1
<eos>	0.1	0.2	0.2	0.6	<eos>	0.1	0.2	0.1	0.6
(a)					(b)				

Рис. 15.12 ❖ Условные вероятности генерирования токенов на каждом шаге для двух разных последовательностей. На основе рис. 9.8.1–9.8.2 из работы [Zha+20]. Печатается с разрешения Астона Чжана

Теперь рассмотрим рис. 15.12b. Предположим, что на шаге 2 выбран токен со второй по величине вероятностью, а именно С. Предположим, что при таком условии $p(y_3 | y_1 = A, y_2 = C) = [0.1, 0.6, 0.2, 0.1]$, как и показано. Тогда символом с максимальной MAP будет В. Предположим, что при таком условии $p(y_4 | y_1 = A, y_2 = C, y_3 = B) = [0.0, 0.2, 0.2, 0.6]$, как и показано. Тогда символом с максимальной MAP будет eos (конец предложения), поэтому генерирование прекращается. Общая вероятность сгенерированной последовательности равна $0.5 \times 0.3 \times 0.6 \times 0.6 = 0.054$. Стало быть, умерив жадность, мы нашли последовательность с большим общим правдоподобием.

Для скрытых марковских моделей можно использовать алгоритм **декодирования Витерби** (пример **динамического программирования**) для вычисления глобально оптимальной последовательности за время $O(TV^2)$, где V – число слов в словаре. (Детали см. в [Mur22].) Но для РНС вычисление глобального оптимума занимает время $O(V^T)$, поскольку скрытое состояние не является достаточной статистикой для данных.

Лучевой поиск – гораздо более быстрый эвристический метод. В этом случае мы вычисляем первые K потенциальных выходов на каждом шаге. Затем каждый расходится по всем V возможным путям, в результате чего генерируется VK кандидатов, из которых мы снова отбираем первые K . Этот процесс показан на рис. 15.13.

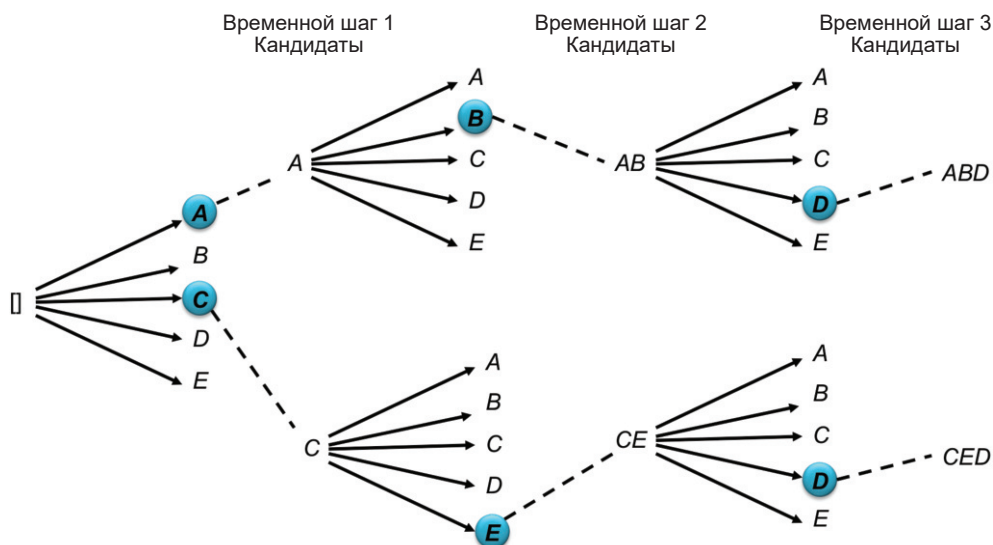


Рис. 15.13 ❖ Лучевой поиск с лучом размера $K = 2$. Словарь $\mathcal{U} = [A, B, C, D, E]$ имеет размер $V = 5$. Предполагается, что лучшие два символа на шаге 1 – A и C. На шаге 2 мы вычисляем $p(y_1 = A, y_2 = y)$ и $p(y_1 = C, y_2 = y)$ для каждого $y \in \mathcal{U}$. Это занимает время $O(KV)$. Затем выбираем два лучших частичных пути: $(y_1 = A, y_2 = B)$ и $(y_1 = C, y_2 = E)$ и продолжаем очевидным образом. На основе рис. 9.8.3 из работы [Zha+20]

Можно также модифицировать алгоритм, так что он будет выбирать лучшие K последовательностей без возвращения (т. е. выбрать лучшую, перенормировать, затем выбрать новую лучшую и т. д.), применив метод **стохастического лучевого поиска**. При этом частичные вероятности модели на каждом шаге подвергаются возмущению **шумом Гумбеля**. Детали см. в работе [KNW19], а последовательную альтернативу – в работе [SBS20]. Эти выборочные методы могут повысить разнообразие выходов (см. также детерминированный метод **лучевого поиска с повышенным разнообразием** (diverse beam search), предложенный в работе [Vij+18]).

15.3. Одномерные СНС

Сверточные нейронные сети (глава 14) вычисляют функцию в некоторой локальной окрестности каждого входа, используя связанные веса, и возвращают выход. Обычно входы двумерные, но СНС можно применять и в одно-

мерном случае, как описано ниже. Это интересная альтернатива РНС, потому что их гораздо проще обучать, так как не нужно хранить долгосрочное скрытое состояние.

15.3.1. Применение одномерных СНС для классификации последовательностей

В этом разделе мы обсудим использование одномерных СНС для обучения отображению последовательностей переменной длины в выход фиксированной длины, т. е. функции вида $f_\theta : \mathbb{R}^{DT} \rightarrow \mathbb{R}^C$, где T – длина входа, D – количество признаков в одном входном примере, а C – размер выходного вектора (например, содержащего логиты классов).

Базовая операция одномерной свертки, применяемой к одномерной последовательности, показана на рис. 14.4. Обычно входная последовательность имеет $D > 1$ входных каналов (измерений признаков). В таком случае мы можем сворачивать каждый канал отдельно и складывать результаты, применяя для каналов разные одномерные фильтры (ядра). В результате получаем $z_i = \sum_d \mathbf{x}_{i-k:i+k,d}^\top \mathbf{w}_d$, где k – размер одномерного рецептивного поля, а \mathbf{w}_d – фильтр для входного канала d . Тем самым порождается вектор $\mathbf{z} \in \mathbb{R}^T$, кодирующий вход (краевые эффекты игнорируем). Мы можем создать векторное представление для каждой позиции, используя разные векторы весов для каждого выходного канала c , и получить $z_{ic} = \sum_d \mathbf{x}_{i-k:i+k,d}^\top \mathbf{w}_{d,c}$. Таким образом, мы построили отображение TD в TC . Чтобы свести его к вектору фиксированного размера, $\mathbf{z} \in \mathbb{R}^C$, можно применить тах-пулинг по времени и получить $z_c = \max_i z_{ic}$. Затем этот вектор можно передать слою softmax.

В работе [Kim14] эта модель применена к классификации последовательностей. Идея заключается в том, чтобы погрузить каждое слово, применив слой погружения, а затем вычислить различные признаки с использованием одномерных ядер разной ширины, стремясь уловить паттерны в разном масштабе длин. Затем применяется тах-пулинг по времени, результаты конкатенируются и передаются полносвязному слою (см. иллюстрацию на рис. 15.14 и демонстрационный код по адресу code.problm.ai/book1/cnn1d_sentiment_torch).

15.3.2. Применение каузальных одномерных СНС для генерирования последовательностей

Чтобы использовать одномерную СНС в порождающих сетях, следует преобразовать ее в **каузальную СНС**, в которой каждая выходная переменная зависит только от ранее сгенерированных переменных (это еще называется **сверточной марковской моделью**). Точнее, определим модель следующим образом:

$$p(\mathbf{y}) = \prod_{t=1}^T p(y_t | \mathbf{y}_{1:t-1}) = \prod_{t=1}^T \text{Cat} \left(y_t | \mathcal{S} \left(\varphi \left(\sum_{\tau=1}^{t-k} \mathbf{w}^\top \mathbf{y}_{\tau:\tau+k} \right) \right) \right), \quad (15.33)$$

где \mathbf{w} – сверточный фильтр размера k , и для простоты обозначений предполагается единственная нелинейность φ и категориальный выход. Это похоже на обычную одномерную свертку, в которой будущие выходы «замаскированы», так что y_t зависит только от прошлых значений, а не от прошлых и будущих вместе. Это называется **каузальной сверткой**. Конечно, можно использовать более глубокие модели и обуславливать входными признаками \mathbf{x} .

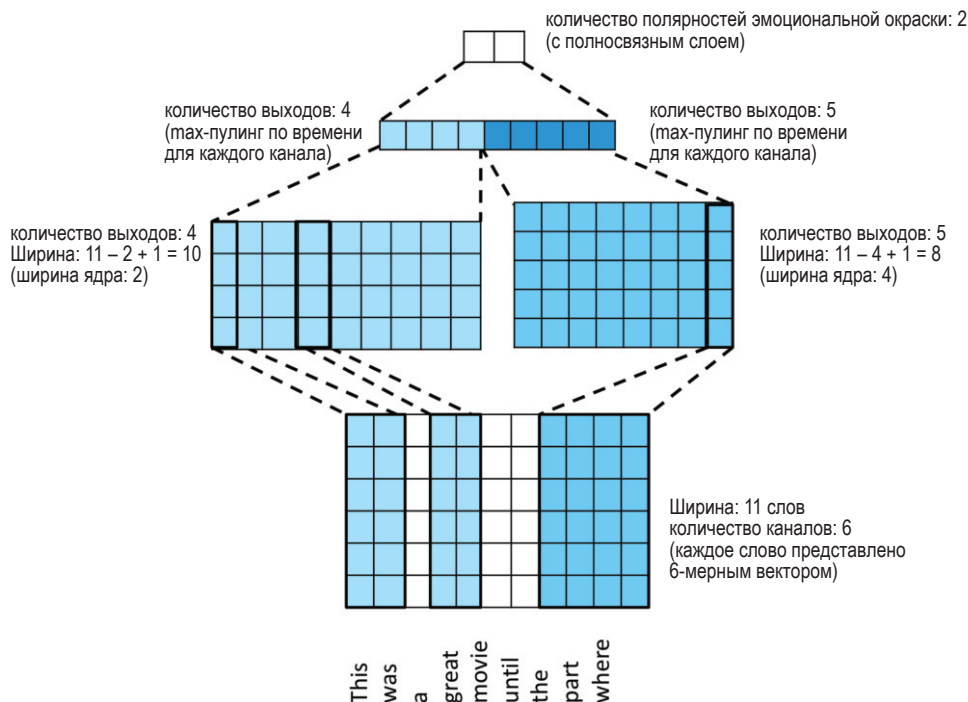


Рис. 15.14 ❖ Модель TextCNN

для бинарной классификации эмоциональной окраски.

На основе рис. 15.3.5 из работы [Zha+20]

Чтобы уловить дальние зависимости, можно воспользоваться дырявой сверткой (раздел 14.4.1), как показано на рис. 15.15. Эта модель успешно использовалась для создания современной системы **речевого синтеза** (text to speech – **TTS**) под названием **wavenet** [oog+16]. Именно, авторы последовательно соединили 10 каузальных одномерных сверточных слоев с коэффициентами дырявости 1, 2, 4, ..., 256, 512 и получили сверточный блок с эффективным рецептивным полем размера 1024. (Перед каждым слоем они дополняли входные последовательности слева нулями в количестве, равном коэффициенту дырявости, чтобы все слои имели одинаковую длину.) Затем они повторили этот блок трижды, чтобы вычислить более глубокие признаки.

В wavenet обуславливающей информацией \mathbf{x} является множество лингвистических признаков, построенное по входной последовательности слов; затем модель генерирует первичный аудиоряд. Можно также реализовать

полное сквозное решение, которое начинается со слов, а не лингвистических признаков (см. [Wan+17]).

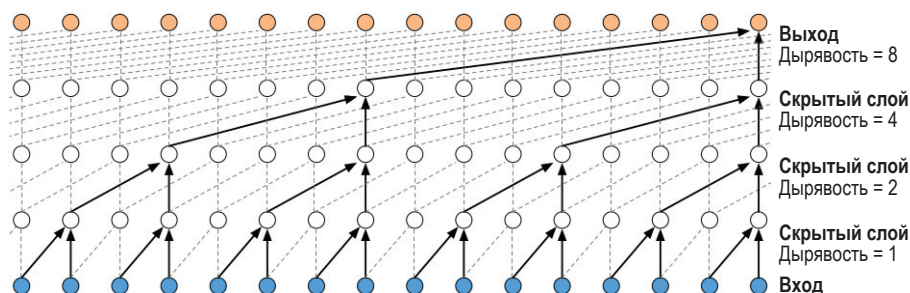


Рис. 15.15 ❖ Иллюстрация модели wavenet, в которой используются дырявые свертки с коэффициентами дырявости 1, 2, 4 и 8. На основе рис. 3 из работы [oog+16]. Печатается с разрешения Аарона ван ден Оорда

Хотя wavenet порождает высококачественную речь, она работает слишком медленно для использования в производственных системах. Однако из нее можно «выжать» параллельную порождающую модель [Oog+18]. Такого рода параллельные порождающие модели мы будем обсуждать во втором томе этой книги, [Mur22].

15.4. Модель внимания

Во всех рассмотренных до сих пор нейронных сетях скрытые активации представляли собой линейную комбинацию входных активаций, сопровождаемую нелинейностью: $\mathbf{z} = \varphi(\mathbf{W}\mathbf{v})$, где $\mathbf{v} \in \mathbb{R}^v$ – вектор скрытых признаков, а $\mathbf{W} \in \mathbb{R}^{v' \times v}$ – фиксированное множество весов, обученное на обучающем наборе.

Однако можно вообразить и более гибкую модель, в которой имеется множество m векторов признаков или **значений** $\mathbf{V} \in \mathbb{R}^{m \times v}$, а модель динамически решает (зависящим от входа способом), какой вектор использовать, исходя из того, насколько входной вектор **запроса** $\mathbf{q} \in \mathbb{R}^q$ похож на элементы множества m **ключей** $\mathbf{K} \in \mathbb{R}^{m \times k}$. Если q больше всего похож на ключ i , то используется значение \mathbf{v}_i . Это базовая идея, лежащая в основе **механизмов внимания**. Первоначально она была предложена для моделей последовательностей, так что именно в этом контексте мы ее и будем объяснять. Однако ее применимость этим не ограничивается. Наше изложение в последующих разделах следует книге [Zha+20, глава 10].

15.4.1. Механизм внимания как мягкий поиск в словаре

Мы можем рассматривать механизм внимания как поиск в словаре, когда запрос \mathbf{q} сравнивается с каждым ключом \mathbf{k}_i , а затем выбирается соответству-

ющее значение v_i . Чтобы сделать эту операцию поиска дифференцируемой, мы будем не выбирать единственное значение v_i , а вычислять выпуклую комбинацию значений следующим образом:

$$\text{Attn}(\mathbf{q}, (\mathbf{k}_1, \mathbf{v}_1), \dots, (\mathbf{k}_m, \mathbf{v}_m)) = \text{Attn}(\mathbf{q}, (\mathbf{k}_{1:m}, \mathbf{v}_{1:m})) = \sum_{i=1}^m \alpha_i(\mathbf{q}, \mathbf{k}_{1:m}) \mathbf{v}_i \in \mathbb{R}^v, \quad (15.34)$$

где $\alpha_i(\mathbf{q}, \mathbf{k}_{1:m})$ – i -й **вес внимания**; эти веса удовлетворяют условиям $0 \leq \alpha_i(\mathbf{q}, \mathbf{k}_{1:m}) \leq 1$ для всех i и $\sum_i \alpha_i(\mathbf{q}, \mathbf{k}_{1:m}) = 1$.

Веса внимания можно вычислять с помощью функции **оценки внимания** $a(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}$, которая возвращает степень сходства запроса \mathbf{q} с ключом \mathbf{k}_i . Ниже мы обсудим несколько таких функций.

Зная оценки, можно вычислить веса внимания, воспользовавшись функцией softmax:

$$\alpha_i(\mathbf{q}, \mathbf{k}_{1:m}) = \mathcal{S}_i([a(\mathbf{q}, \mathbf{k}_1), \dots, a(\mathbf{q}, \mathbf{k}_m)]) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^m \exp(a(\mathbf{q}, \mathbf{k}_j))}. \quad (15.35)$$

Смотрите иллюстрацию на рис. 15.16.

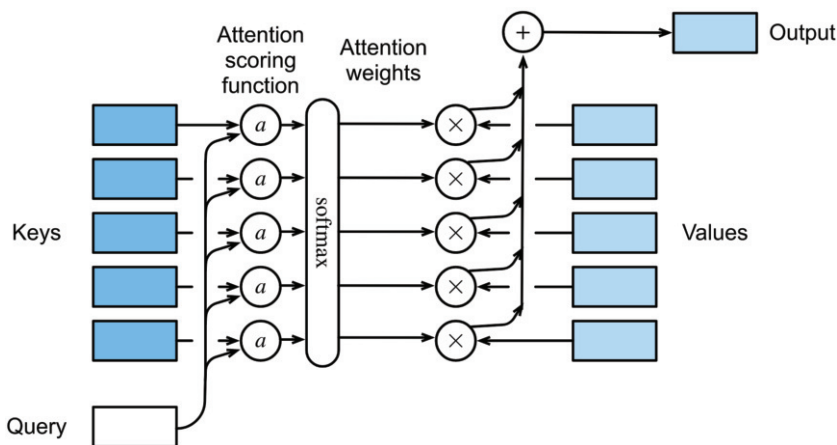


Рис. 15.16 ♦ Модель внимания вычисляет взвешенное среднее множества значений, а веса выводятся путем сравнения вектора запроса с множеством ключей. На основе рис. 10.3.1 из работы [Zha+20]. Печатается с разрешения Астана Чжана

В некоторых случаях мы хотим ограничить внимание подмножеством словаря, соответствующим допустимым элементам. Например, мы могли бы дополнять последовательности до фиксированной длины (для эффективного обучения на мини-пакетах), и тогда следовало бы «замаскировать» дополненные позиции. Это называется **замаскированным вниманием**. Такую функциональность можно эффективно реализовать, задав для замаскированных элементов оценку внимания, равную большому отрицательному чис-

лу, например -10^6 , тогда соответствующие softmax-веса будут равны 0. (Это аналог каузальной свертки, рассмотренной в разделе 15.3.2.)

15.4.2. Ядерная регрессия как непараметрическое внимание

В разделе 16.3.5 мы обсудим ядерную регрессию, т. е. непараметрическую модель вида

$$f(x) = \sum_{i=1}^n \alpha_i(x, x_{1:n}) y_i, \quad (15.36)$$

где $\alpha_i(x, x_{1:n}) \geq 0$ измеряет нормированное сходство тестового входа x с обучающим примером x_i . Для вычисления этой меры сходства оценка внимания обычно определяется в терминах ядра плотности, например гауссова:

$$\mathcal{K}_\sigma(u) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}u^2}, \quad (15.37)$$

где σ называется полосой пропускания. Затем мы определяем $a(x, x_i) = \mathcal{K}_\sigma(x - x_i)$.

Поскольку оценки нормированы, мы можем опустить член $1/\sqrt{2\pi\sigma^2}$. Кроме того, для согласованности с обозначениями из [Zha+20, глава 10] мы перепишем экспоненту в виде:

$$\mathcal{K}(x; w) = \exp\left(-\frac{w^2}{2}u^2\right). \quad (15.38)$$

Подставляя это в формулу (15.36), получаем:

$$f(x) = \sum_{i=1}^n \alpha_i(x, x_{1:n}) y_i \quad (15.39)$$

$$= \sum_{i=1}^n \frac{\exp\left[-\frac{1}{2}((x - x_i)w)^2\right]}{\sum_{j=1}^n \exp\left[-\frac{1}{2}((x - x_j)w)^2\right]} y_i \quad (15.40)$$

$$= \sum_{i=1}^n \mathcal{S}_i\left[-\frac{1}{2}((x - x_i)w)^2, \dots, -\frac{1}{2}((x - x_n)w)^2\right] y_i. \quad (15.41)$$

Это можно интерпретировать как форму непараметрического внимания, где запросами являются тестовые точки x , ключами – обучающие примеры x_i , а значениями – обучающие метки y_i .

Если положить $w = 1$, то получится матрица внимания с элементами $A_{ji} = \alpha_i(x_j, x_{1:n})$ для тестового входа j , показанная на рис. 15.17a. Результирующая предсказанная кривая показана на рис. 15.17b.

Ширина диагональной полосы на рис. 15.17а, а значит, разреженность механизма внимания зависит от параметра w . Если увеличить w , что соответствует уменьшению полосы пропускания ядра, то полоса станет уже, но модель будет склонна к переобучению.

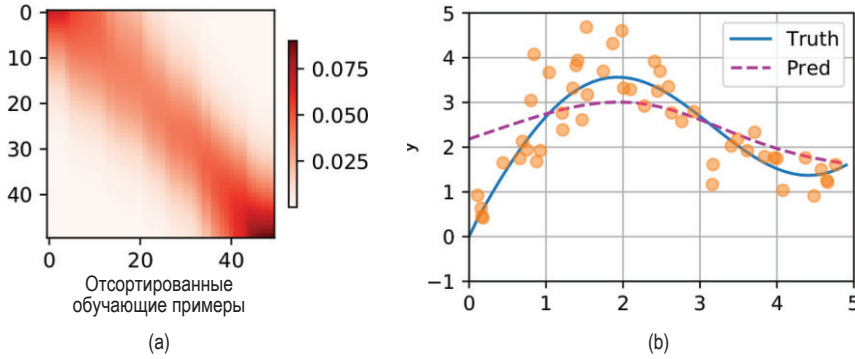


Рис. 15.17 ❖ Ядерная регрессия в одномерном случае. (а) Ядерная матрица весов. (б) Результирующие предсказания на плотной сетке тестовых точек. Построено программой по адресу figures.problm.ai/book1/15.17

15.4.3. Параметрическое внимание

В разделе 15.4.2 мы определили оценку внимания в терминах гауссова ядра, участвующего в сравнении скалярного запроса (тестовой точки) с каждым из скалярных значений в обучающем наборе. Такой подход плохо масштабируется на большие обучающие наборы, т. е. вход высокой размерности. Поэтому обратимся к параметрическим моделям, когда имеется фиксированное множество ключей и значений, а запросы и ключи сравниваются в обученном пространстве погружения.

Это можно сделать несколькими способами. В общем случае запрос $\mathbf{q} \in \mathbb{R}^q$ и ключ $\mathbf{k} \in \mathbb{R}^k$ могут быть разного размера. Сравнить их можно, отображив в общее пространство погружения размера h , для чего нужно вычислить $\mathbf{W}_q \mathbf{q}$ и $\mathbf{W}_k \mathbf{k}$, где $\mathbf{W}_q \in \mathbb{R}^{h \times q}$ и $\mathbf{W}_k \in \mathbb{R}^{h \times k}$. Затем можно передать их МСП, получив такую аддитивную функцию оценки внимания:

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{w}_v^\top \tanh(\mathbf{W}_q \mathbf{q} + \mathbf{W}_k \mathbf{k}) \in \mathbb{R}. \quad (15.42)$$

Вычислительно более эффективно предполагать, что запросы и ключи имеют одинаковую длину d , так что $\mathbf{q}^\top \mathbf{k}$ можно вычислить непосредственно. Если считать, что это независимые случайные величины с нулевым средним и единичной дисперсией, то среднее их скалярного произведения равно 0, а дисперсия равна d (это следует из формул (2.34) и (2.39)). Чтобы дисперсия скалярного произведения оставалась равной 1 независимо от размера входов, принято делить на \sqrt{d} . Таким образом, мы получаем **внимание в форме масштабированного скалярного произведения**:

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\top \mathbf{k} / \sqrt{d} \in \mathbb{R}. \quad (15.43)$$

На практике мы обычно имеем дело с мини-пакетами, содержащими n векторов. Обозначим соответствующие матрицы запросов, ключей и значений $\mathbf{Q} \in \mathbb{R}^{n \times d}$, $\mathbf{K} \in \mathbb{R}^{m \times d}$, $\mathbf{V} \in \mathbb{R}^{m \times v}$. Тогда можно вычислить взвешенные вниманием выходы следующим образом:

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathcal{S}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V} \in \mathbb{R}^{n \times v}, \quad (15.44)$$

где функция softmax \mathcal{S} применяется построчно (см. демонстрационный код по адресу code.problml.ai/book1/attention_torch).

15.4.4. Модель Seq2Seq с вниманием

Вспомним модель seq2seq из раздела 15.2.3. В ней используется РНС-декодер вида $\mathbf{h}_t^d = f_d(\mathbf{h}_{t-1}^d, \mathbf{y}_{t-1}, \mathbf{c})$, где \mathbf{c} – контекстный вектор фиксированной длины, представляющий закодированный вход $\mathbf{x}_{1:T}$. Обычно полагают \mathbf{c} равным \mathbf{h}_T^e , конечному состоянию РНС-кодировщика (или используют двунаправленную РНС с пулингом усреднением). Однако в таких задачах, как машинный перевод, это может стать причиной плохого качества, потому что выходной слой не имеет доступа к самым входным словам. Этой неприятности можно избежать, разрешив выходным словам напрямую «смотреть» на входные слова. Но на какие именно? Ведь порядок слов в разных языках может различаться (например, в немецком глаголы часто помещаются в конец предложения), поэтому нужно как-то сопоставить исходный и конечный текст.

Мы можем решить эту проблему (дифференцируемым образом), воспользовавшись (мягким) **вниманием**, впервые предложенным в работах [BCB15; LPM15]. Именно, фиксированный контекстный вектор \mathbf{c} в декодере можно заменить динамическим контекстным вектором \mathbf{c}_t , который вычисляется следующим образом:

$$\mathbf{c}_t = \sum_{i=1}^T \alpha_i(\mathbf{h}_{t-1}^d, \mathbf{h}_{1:T}^e) \mathbf{h}_i^e. \quad (15.45)$$

Здесь внимание привлекается там, где запрос является скрытым состоянием декодера на предыдущем шаге, \mathbf{h}_{t-1}^d , все ключи являются скрытыми состояниями кодировщика, а значения также являются скрытыми состояниями кодировщика. (Если РНС имеет несколько скрытых слоев, то обычно ключи и значения берутся из верхнего слоя кодировщика, а запрос – из верхнего слоя декодера.) Этот контекстный вектор конкатенируется с входным вектором декодера, \mathbf{y}_{t-1} и подается декодеру вместе с предыдущим скрытым состоянием \mathbf{h}_{t-1}^d , в результате чего создается состояние \mathbf{h}_t^d (см. схему полной модели на рис. 15.18).

Эту модель можно обучить обычным способом на парах предложений, а затем использовать для машинного перевода (см. демонстрационный код по адресу code.problml.ai/book1/nmt_attention_torch). Можно также визуализировать веса внимания, вычисленные на каждом шаге декодирования, чтобы получить представление о том, какие части входа кажутся модели наиболее

релевантными для порождения соответствующего выхода. На рис. 15.19 приведено несколько примеров.

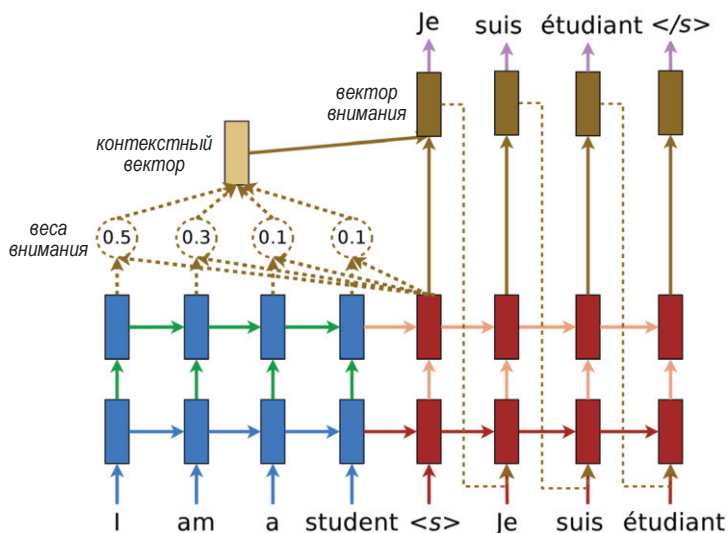


Рис. 15.18 ❖ Модель seq2seq с вниманием для перевода с английского на французский. Печатается с разрешения Минь-Тан Луонга

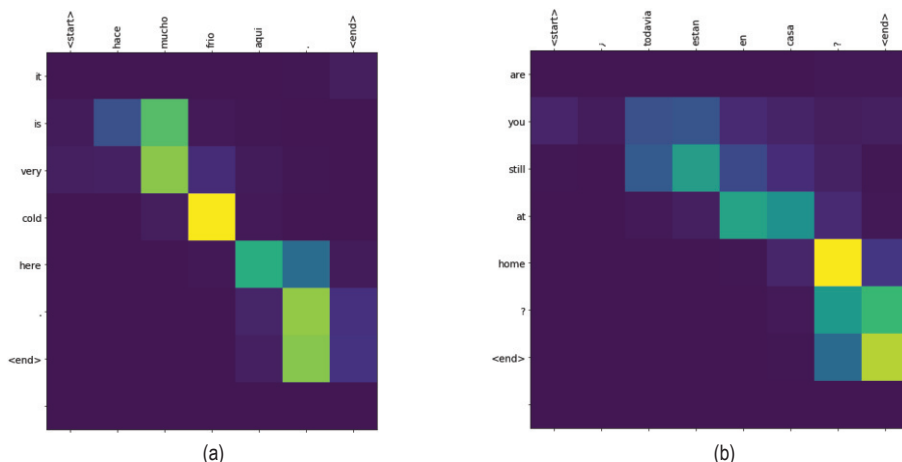


Рис. 15.19 ❖ Тепловые карты внимания, построенные при переводе двух предложений с испанского на английский. (a) На входе фраза «hace mucho frio aqui», на выходе «it is very cold here»¹. (b) На входе фраза «¿todavía están en casa?», на выходе «are you still at home?»². Заметим, что при порождении выходного токена «home» модель должна обратить внимание на входной токен «casa», но похоже, что на самом деле обращает внимание на токен «?». На основе статьи по адресу https://www.tensorflow.org/tutorials/text/nmt_with_attention

¹ Здесь очень холодно. – Прим. перев.

² Ты еще дома? – Прим. перев.

15.4.5. Модель Seq2вес с вниманием (классификация текста)

Мы также можем использовать механизм внимания в классификаторах последовательностей. Так, в работе [Raj+18] классификатор на основе РНС применяется к задаче предсказания смерти пациента. На вход подаются **записи из электронной медицинской карты**, которая представляет собой временной ряд структурированных данных, а также неструктурированного текста (клинических комментариев). Механизм внимания полезен для идентификации «релевантных» частей входа (см. рис. 15.20).

Patient Timeline

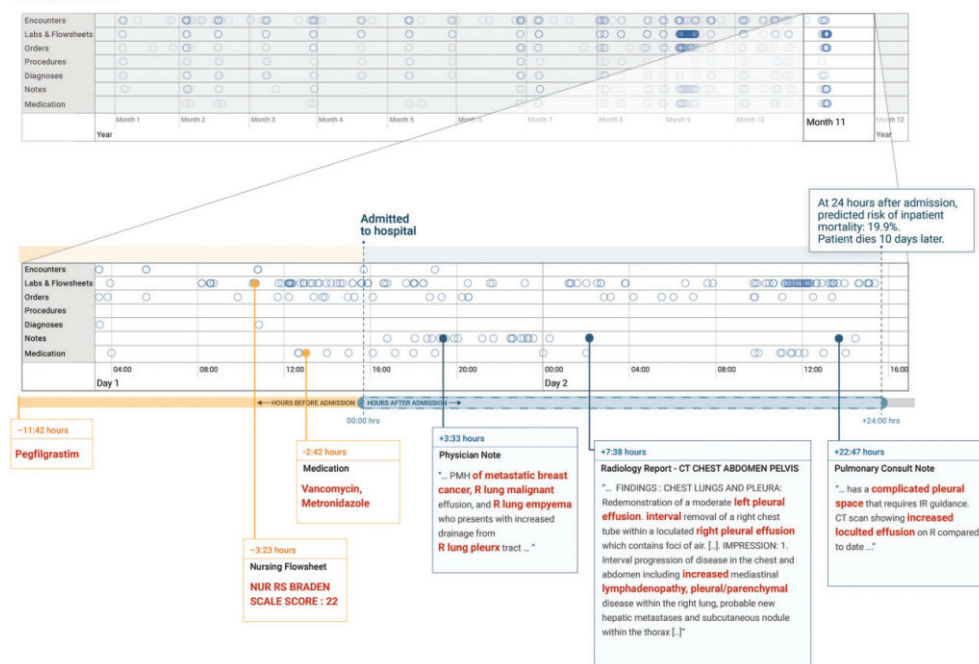


Рис. 15.20 ❖ Пример записи из электронной медицинской карты. В этом примере классификатор на основе РНС через 24 часа после госпитализации предсказывает риск смерти 19.9 %; в действительности пациент умер через 10 дней после госпитализации. «Релевантные» ключевые слова из входных клинических комментариев выделены красным цветом, именно их идентифицировал механизм внимания. На основе рис. 3 из работы [Raj+18]. Печатается с разрешения Алвина Ракомара

15.4.6. Модель Seq+Seq2Vec с вниманием (классификация пар предложений)

Допустим, что видим предложение «A person on a horse jumps over a log»¹ (назовем его **посылкой**), а далее читаем «A person is outdoors on a horse»² (назовем это предложение **гипотезой**). Можно высказать разумное предположение, что из посылки **следует** гипотеза, т. е. гипотеза более вероятна при условии истинности посылки³. Теперь предположим, что в роли гипотезы выступает предложение «A person is at a diner ordering an omelette»⁴. В этом случае мы скажем, что посылка **противоречит** гипотезе, потому что гипотеза менее вероятна при условии истинности посылки. Наконец, предположим, что гипотезой является предложение «A person is training his horse for a competition»⁵. В этом случае связь между посылкой и гипотезой **нейтральная**, так как гипотеза может как следовать, так и не следовать из посылки. Задача отнесения пары предложений к одной из этих трех категорий называется **извлечением имплицитных знаний из текста**, или «**естественно-языковым выводом**». Стандартным эталоном в этой области является корпус текстов **Stanford Natural Language Inference (SNLI)** [Bow+15]. Он содержит 550 000 помеченных пар предложений.

Интересное решение этой задачи классификации было предложено в работе [Par+16a]; в свое время она являлась самым передовым исследованием на наборе данных SNLI. Принятый в ней подход схематически представлен на рис. 15.21. Пусть $\mathbf{A} = (\mathbf{a}_1, \dots, \mathbf{a}_m)$ – посылка, а $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ – гипотеза, где $\mathbf{a}_i, \mathbf{b}_j \in \mathbb{R}^E$ – векторы погружений для слов. Модель включает три шага. Сначала каждое слово посылки \mathbf{a}_i сопоставляется с каждым словом гипотезы \mathbf{b}_j и вычисляются веса внимания:

$$e_{ij} = f(\mathbf{a}_i)^T f(\mathbf{b}_j), \quad (15.46)$$

где $f: \mathbb{R}^E \rightarrow \mathbb{R}^D$ – МСП; затем мы вычисляем взвешенное среднее соответствующих слов в гипотезе:

$$\beta_i = \frac{\sum_{j=1}^n \exp(e_{ij})}{\sum_{k=1}^n \exp(e_{ik})} \mathbf{b}_j. \quad (15.47)$$

¹ Человек на лошади прыгает через бревно. – Прим. перев

² Человек на улице на лошади. – Прим. перев

³ Заметим, что гипотеза логически не вытекает из посылки, потому что человек мог бы находиться на спине лошади и в помещении, но чаще все-таки на лошадях прогуливаются на открытом воздухе. Кроме того, мы предполагаем, что слово «человек» в обоих предложениях относится к одному и тому же человеку.

⁴ Человек в столовой заказывает омлет. – Прим. перев.

⁵ Человек тренирует свою лошадь к соревнованиям. – Прим. перев.

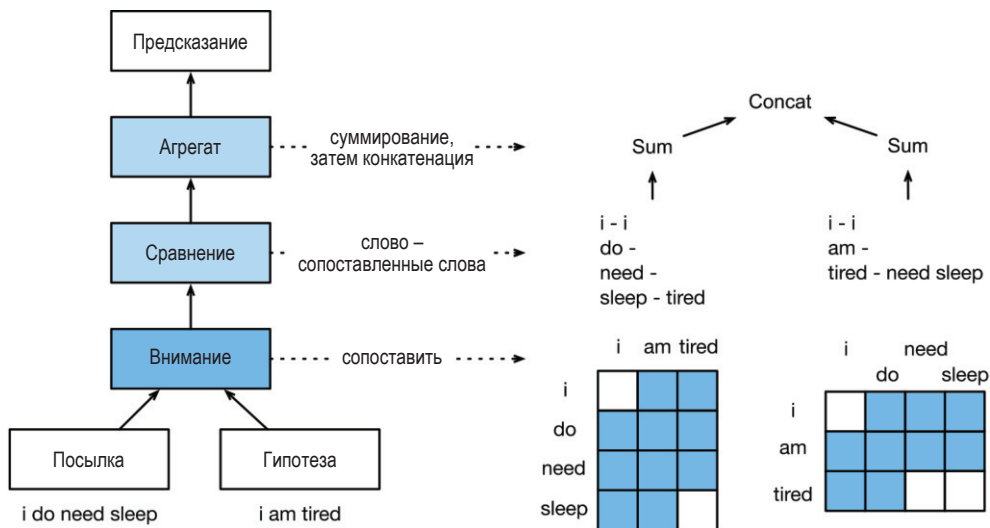


Рис. 15.21 ❖ ИмPLICITная классификация пары предложений с применением МСП с механизмом внимания для сопоставления посылки («I do need sleep»¹) с гипотезой («I am tired»²). Белыми квадратиками обозначены активные веса активации, синими – неактивные. (Для простоты предполагается жесткое внимание 0/1.) На основе рис. 15.5.2 из работы [Zha+20]. Печатается с разрешения Астона Чжана

Далее сравниваем \mathbf{a}_i с β_i , для чего отображаем их конкатенацию в скрытое пространство, используя МСП $g: \mathbb{R}^{2E} \rightarrow \mathbb{R}^H$:

$$\mathbf{v}_{A,i} = g([\mathbf{a}_i, \beta_i]), i = 1, \dots, m. \quad (15.48)$$

Наконец, агрегируем по сравнениям, чтобы получить общее сходство посылки с гипотезой:

$$\mathbf{v}_A = \sum_{i=1}^m \mathbf{v}_{A,i}. \quad (15.49)$$

Аналогично можно сравнить гипотезу с посылкой, используя

$$\alpha_j = \sum_{i=1}^m \frac{\exp(e_{ij})}{\sum_{k=1}^m \exp(e_{kj})} \mathbf{a}_i; \quad (15.50)$$

$$\mathbf{v}_{B,j} = g([\mathbf{b}_j, \alpha_j]), j = 1, \dots, n; \quad (15.51)$$

$$\mathbf{v}_B = \sum_{j=1}^n \mathbf{v}_{B,j}. \quad (15.52)$$

¹ Мне и вправду нужно поспать. – Прим. перев.

² Я устал. – Прим. перев.

И в конце классифицируем выход с помощью еще одного МСП $h : \mathbb{R}^{2H} \rightarrow \mathbb{R}^3$:

$$\hat{y} = h([\mathbf{v}_A, \mathbf{v}_B]). \quad (15.53)$$

Демонстрационный код имеется по адресу `code.probl.ai/book1/entailment_attention_mlp_torch`.

Эту модель можно модифицировать, так чтобы она обучалась другим видам отображения пар предложений в выходные метки. Например, в задаче **семантического сходства текстов** цель состоит в том, чтобы предсказать силу семантической связи двух предложений. В стандартном эталонном наборе данных для этой задачи, **STS Benchmark** [Cer+17], сила связи изменяется от 0 (вообще не связаны) до 5 (максимально связаны).

15.4.7. Мягкое и жесткое внимание

Если принудительно сделать тепловую карту внимания разреженной, так чтобы каждый выход мог оказывать внимание только одной входной позиции, а не взвешенной комбинации всех, то мы получим метод **жесткого внимания**. На рис. 15.22 проведено сравнение этих двух подходов на примере задачи описания изображений. К сожалению, жесткое внимание приводит к недифференцируемой целевой функции обучения и для аппроксимации данных моделью приходится привлекать такие методы, как обучение с подкреплением. Детали см. в работе [Xu+15].

Из приведенных примеров складывается впечатление, что тепловые карты внимания могут «объяснить», почему модель порождает данный выход. Однако интерпретируемость внимания противоречива (см., например, обсуждение в работах [JW19; WP19; SS19; Bru+19]).

15.5. ТРАНСФОРМЕРЫ

Модель **трансформера** [Vas+17] – это модель seq2seq, в которой механизм внимания используется как в кодировщике, так и в декодере, благодаря чему отпадает необходимость в РНС, как будет объяснено ниже. Трансформеры применялись для решения многих задач условного генерирования последовательностей, например: машинный перевод [Vas+17], синтаксический анализ на основе грамматики составляющих [Vas+17], генерирование музыки [Hua+18], генерирование последовательностей белков [Mad+20; Cho+20b], реферирование текста [Zha+19a], генерирование изображений [Par+18] (изображение рассматривается как растринированная одномерная последовательность) и т. д.

Трансформер – довольно сложная модель, в которой используется несколько новых видов строительных блоков, или слоев. Ниже мы опишем эти блоки, а затем обсудим, как свести все вместе¹.

¹ Более обстоятельное введение см. по адресу <https://huggingface.co/course/chapter1>.

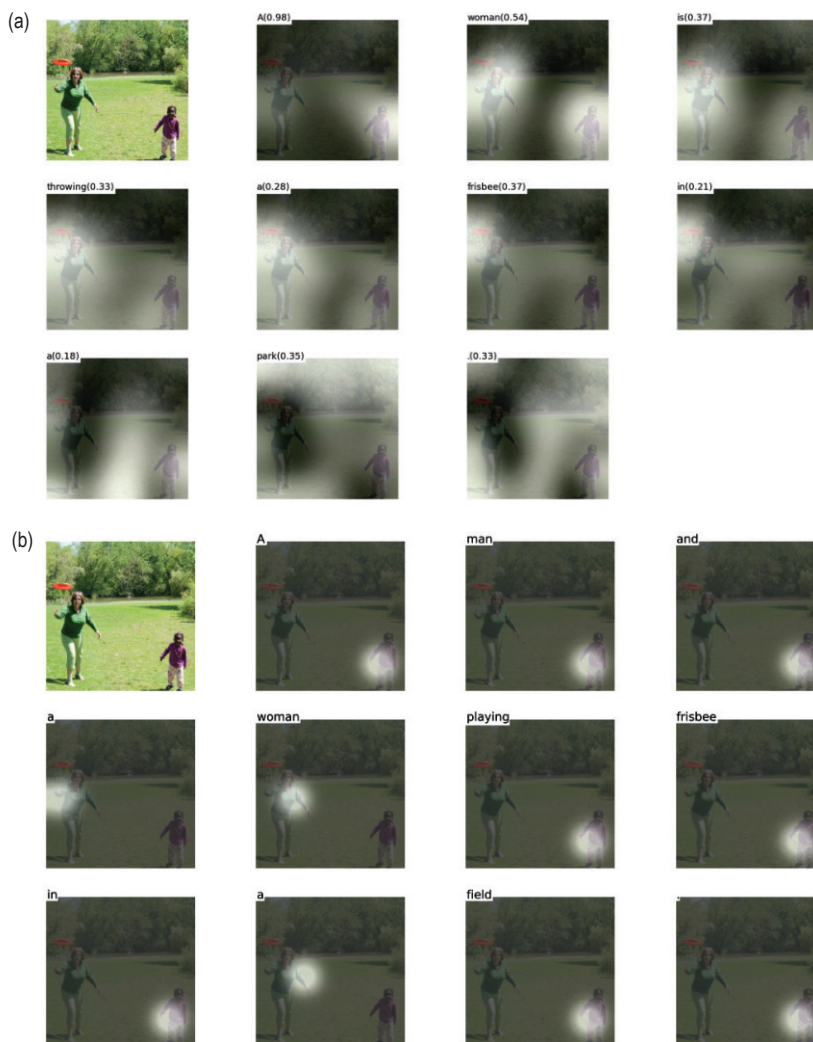


Рис. 15.22 ❖ Описание изображения с применением механизма внимания. (a) Мягкое внимание. Генерируется предложение «a woman is throwing a frisbee in a park»¹. (b) Жесткое внимание. Генерируется предложение «a man and a woman playing frisbee in a field»². На основе рис. 6 из работы [Ху+15]. Печатается с разрешения Келвина Су

15.5.1. Самовнимание

В разделе 15.4.4 мы показали, как декодер РНС может применить механизм внимания к входной последовательности, чтобы уловить контекстуальные погружения каждого входа. Однако, вместо того чтобы привлекать внимание

¹ Женщина метает фрисби в парке. – Прим. перев.

² Мужчина и женщина играют в фрисби на поле. – Прим. перев.

декодера к кодировщику, мы можем модифицировать модель, так чтобы кодировщик обращал внимание на самого себя. Это называется **самовниманием** [CDL16; Par+16b].

Точнее, пусть дана последовательность входных токенов $\mathbf{x}_1, \dots, \mathbf{x}_n$, где $\mathbf{x}_i \in \mathbb{R}^d$. Тогда механизм самовнимания генерирует последовательность выходов того же размера по правилу:

$$\mathbf{y}_i = \text{Attn}(\mathbf{x}_i, (\mathbf{x}_1, \mathbf{x}_1), \dots, (\mathbf{x}_n, \mathbf{x}_n)), \quad (15.54)$$

где запросом является \mathbf{x}_i , а ключами и значениями – все (допустимые) входы $\mathbf{x}_1, \dots, \mathbf{x}_n$.

Чтобы использовать его в декодере, мы можем положить $\mathbf{x}_i = \mathbf{y}_{i-1}$ и $n = i - 1$, чтобы все ранее сгенерированные выходы были доступны. На этапе обучения все выходы уже известны, поэтому функцию можно вычислять параллельно, обойдя тем самым последовательное узкое место, характерное для РНС.

Помимо увеличения скорости, самовнимание может порождать улучшенные представления контекста. В качестве примера рассмотрим перевод английских предложений «The animal didn't cross the street because it was too tired»¹ и «The animal didn't cross the street because it was too wide»² на французский язык. Чтобы сгенерировать французское местоимение правильного рода, мы должны знать, к чему относится «it» (это называется **разрешением кореференции**). В первом случае слово «it» относится к животному, а во втором – к улице.

На рис. 15.23 показано, как применение механизма самовнимания к английскому предложению позволяет разрешить неоднозначность. В первом предложении представление «it» зависит от предшествующих представлений «animal», а во втором – от предшествующих представлений «street».

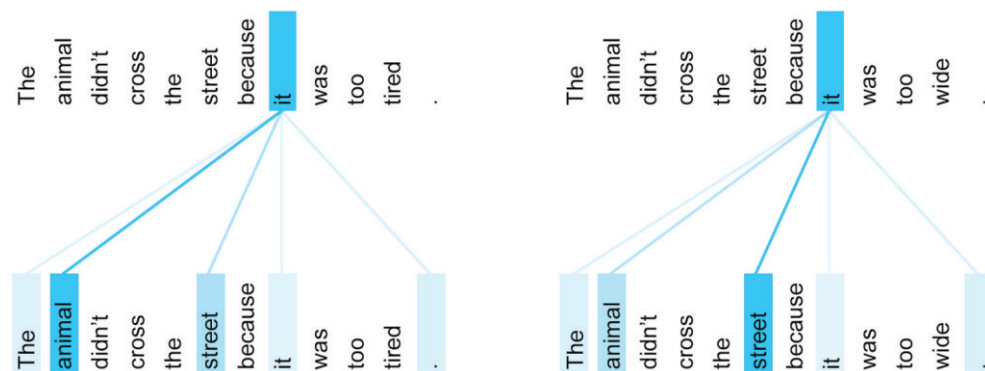


Рис. 15.23 ❖ Как самовнимание к слову «it» в кодировщике различается в зависимости от входного контекста. Из статьи <https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>. Печатается с разрешения Якоба Узкорейта

¹ Животное не переходило улицу, потому что оно слишком устало. – Прим. перев.

² Животное не переходило улицу, потому что она была слишком широкой. – Прим. перев.

15.5.2. Многопутевое внимание

Если рассматривать матрицу внимания как ядерную матрицу (см. раздел 15.4.2), то естественно возникает желание использовать несколько матриц внимания, улавливающих различные виды сходства. Эта идея лежит в основе механизма **многопутевого внимания** (multi-headed attention – MHA). Именно, пусть дан запрос $\mathbf{q} \in \mathbb{R}^{d_q}$, ключи $\mathbf{k}_j \in \mathbb{R}^{d_k}$ и значения $\mathbf{v}_j \in \mathbb{R}^{d_v}$. Определим i -й путь внимания как

$$\mathbf{h}_i = \text{Attn}(\mathbf{W}_i^{(q)}\mathbf{q}, \{\mathbf{W}_i^{(k)}\mathbf{k}_j, \mathbf{W}_i^{(v)}\mathbf{v}_j\}) \in \mathbb{R}^{p_v}, \quad (15.55)$$

где $\mathbf{W}_i^{(q)} \in \mathbb{R}^{p_q \times d_q}$, $\mathbf{W}_i^{(k)} \in \mathbb{R}^{p_k \times d_k}$ и $\mathbf{W}_i^{(v)} \in \mathbb{R}^{p_v \times d_v}$ – матрицы проекций. Затем мы объединяем h путей вместе и проецируем на \mathbb{R}^{p_o} по правилу:

$$\mathbf{h} = \text{MHA}(\mathbf{q}, \{\mathbf{k}_j, \mathbf{v}_j\}) = \mathbf{W}_0 \begin{pmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_h \end{pmatrix} \in \mathbb{R}^{p_o}, \quad (15.56)$$

где \mathbf{h}_i определено в формуле (15.55), а $\mathbf{W}_0 \in \mathbb{R}^{p_o \times hp_v}$. Если положить $p_q h = p_k h = p_v h = p_o$, то можно будет вычислять все выходные пути параллельно. Демонстрационный код см. по адресу code.problml.ai/book1/multi_head_attention.

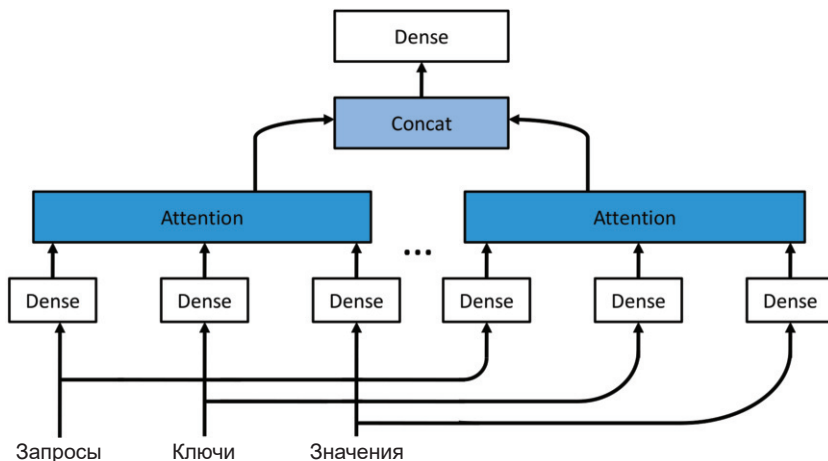


Рис. 15.24 ❖ Многопутевое внимание. На основе рис. 9.3.3 из работы [Zha+20]

15.5.3. Позиционное кодирование

Качество не оптимизированного механизма самовнимания может быть низким, поскольку внимание инвариантно относительно перестановок и, следовательно, игнорирует порядок входных слов. Чтобы решить эту проблему, мы можем конкатенировать погружения слов с **позиционным погружением**, тогда модель будет знать, в каком порядке встречаются слова.

Один из способов сделать это – представить каждую позицию целым числом. Однако просто так нейронные сети не умеют обрабатывать целые числа. Чтобы преодолеть эту трудность, мы можем закодировать целое в двоичной форме. Например, если предположить, что длина последовательности $n = 3$, то получим следующую последовательность $d = 3$ -мерных битовых векторов для каждой позиции: 000, 001, 010, 011, 100, 101, 110, 111. Как видим, быстрее всего изменяется самый правый элемент (имеет наибольшую частоту), тогда как самый левый (старший бит) изменяется медленнее всего. (Конечно, кодировку можно изменить, сделав так, чтобы самый левый бит изменялся быстрее всего.) Это можно представить позиционной матрицей $\mathbf{P} \in \mathbb{R}^{n \times d}$.

Описанное выше представление можно интерпретировать как разложение по множеству базисных функций (соответствующих степеням 2) с коэффициентами 0 и 1. Более компактный код можно получить, взяв другой набор базисных функций и вещественные веса. В работе [Vas+17] предложено использовать синусоидальный базис:

$$p_{i,2j} = \sin\left(\frac{i}{C^{2j/d}}\right), \quad p_{i,2j+1} = \cos\left(\frac{i}{C^{2j/d}}\right), \quad (15.57)$$

где $C = 10\,000$ – максимальная длина последовательности. Например, если $d = 4$, то i -я строка имеет вид:

$$\mathbf{p}_i = \left[\sin\left(\frac{i}{C^{0/4}}\right), \cos\left(\frac{i}{C^{0/4}}\right), \sin\left(\frac{i}{C^{2/4}}\right), \cos\left(\frac{i}{C^{2/4}}\right) \right]. \quad (15.58)$$

На рис. 15.25a показана позиционная матрица для $n = 60$ и $d = 32$. В данном случае быстрее всего изменяются левые столбцы. Мы видим, что у каждой строки имеется вещественный «цифровой отпечаток», представляющий ее позицию в последовательности. На рис. 15.25b показаны базисные функции (векторы-столбцы) для измерений от 6 до 9.

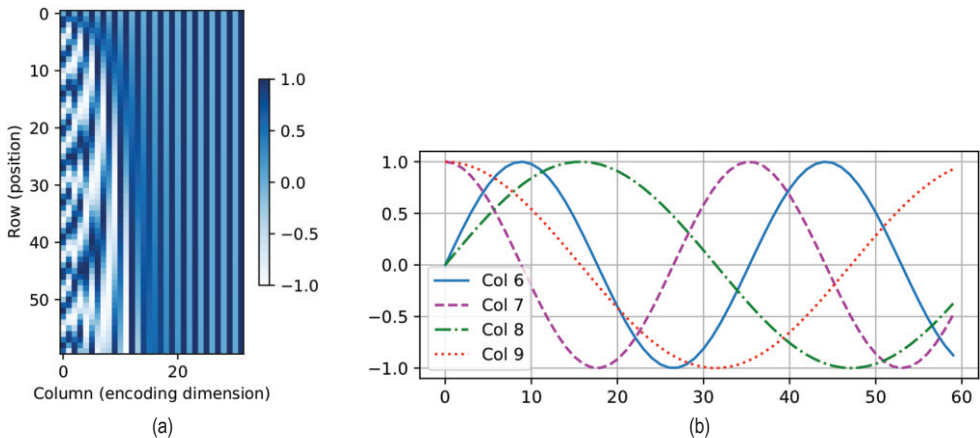


Рис. 15.25 ❖ (a) Матрица позиционного кодирования для последовательности длины $n = 60$ и размерности погружения $d = 32$. (b) Базисные функции для столбцов от 6 до 9. Построено программой по адресу figures.problml.ai/book1/15.25

У такого представления двойное преимущество. Во-первых, его можно вычислить для входной последовательности произвольной длины (не превышающей C), в отличие от обученного отображения целых чисел в векторы. Во-вторых, представление одной позиции линейно предсказуемо по любой другой, если известно их относительное расстояние. Именно, имеем $\mathbf{p}_{t+\phi} = f(\mathbf{p}_t)$, где f – линейное преобразование. Чтобы убедиться в этом, заметим, что

$$\begin{pmatrix} \sin(\omega_k(t + \phi)) \\ \cos(\omega_k(t + \phi)) \end{pmatrix} = \begin{pmatrix} \sin(\omega_k t) \cos(\omega_k \phi) + \cos(\omega_k t) \sin(\omega_k \phi) \\ \cos(\omega_k t) \cos(\omega_k \phi) - \sin(\omega_k t) \sin(\omega_k \phi) \end{pmatrix} \quad (15.59)$$

$$= \begin{pmatrix} \cos(\omega_k \phi) & \sin(\omega_k \phi) \\ -\sin(\omega_k \phi) & \cos(\omega_k \phi) \end{pmatrix} \begin{pmatrix} \sin(\omega_k t) \\ \cos(\omega_k t) \end{pmatrix}. \quad (15.60)$$

Вычислив позиционные погружения \mathbf{P} , мы должны объединить их с погружениями исходных слов \mathbf{X} по правилу¹:

$$\text{POS}(\text{Embed}(\mathbf{X})) = \mathbf{X} + \mathbf{P}. \quad (15.61)$$

15.5.4. Соберем все вместе

Трансформер – это модель seq2seq, в которой в качестве кодировщика и декодера применяется механизм самовнимания, а не РНС. Кодировщик использует серию блоков кодирования, в каждом из которых имеется многопутевое внимание (раздел 15.5.2), остаточные связи (раздел 13.4.4) и нормировка слоев (раздел 14.2.4.2).

Точнее, блок кодирования можно определить следующим образом:

```
def EncoderBlock(X):
    Z = LayerNorm(MultiHeadAttn(Q=X, K=X, V=X) + X)
    E = LayerNorm(FeedForward(Z) + Z)
    return E
```

Кодировщик в целом определяется путем применения позиционного кодирования к погружению входной последовательности, после чего следует N копий блока кодирования, где N управляет глубиной блока:

```
def Encoder(X, N):
    E = POS(Embed(X))
    for n in range(N):
        E = EncoderBlock(E)
    return E
```

Смотрите иллюстрацию в левой части рис. 15.26.

¹ Напрашивается другая схема – конкатенировать \mathbf{X} и \mathbf{P} , но при сложении потребляется меньше памяти. Кроме того, поскольку погружения \mathbf{X} получены в результате обучения, модель могла бы имитировать конкатенацию, обнулив первые K измерений \mathbf{X} и последние $D - K$ измерений \mathbf{P} , где K неявно определено паттерном разреженности. Более полное обсуждение см. по адресу <https://bit.ly/3rMG1at>.

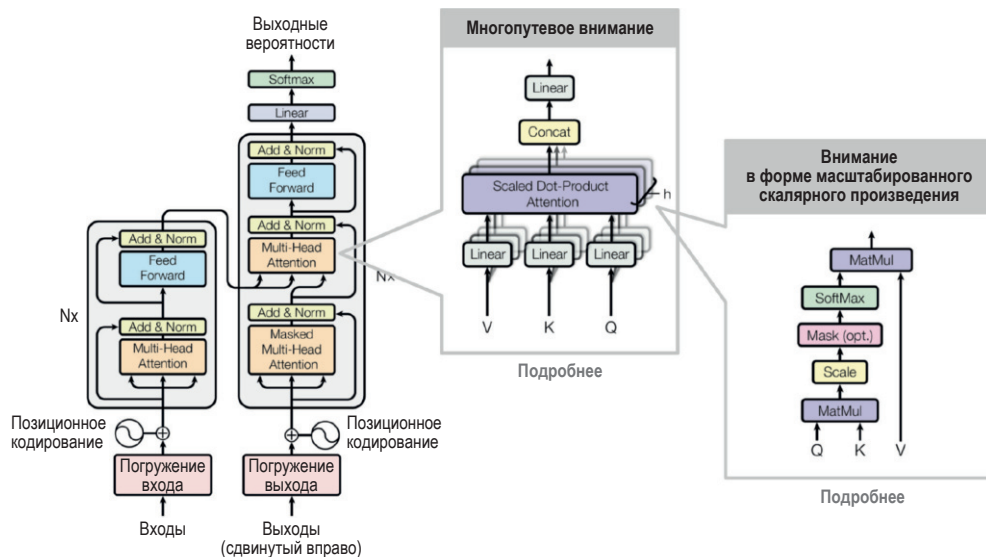


Рис. 15.26 ❖ Трансформер. Из работы [Wen18].

Печатается с разрешения Лилиан Вен. На основе рис. 1–2 из работы [Vas+17]

Структура декодера несколько сложнее. Ему предоставлен доступ к кодировщику посредством еще одного блока многопутевого внимания. Но также он имеет доступ к ранее сгенерированным выходам: они сдвинуты, а затем объединены с позиционным погружением и поданы на вход замаскированной (каузальной) модели многопутевого внимания. Наконец, параллельно вычисляется выходное распределение токенов в каждой позиции.

Приведем развернутое определение блока декодирования:

```
def DecoderBlock(X, E):
    Z = LayerNorm(MultiHeadAttn(Q=X, K=X, V=X) + X)
    Z' = LayerNorm(MultiHeadAttn(Q=Z, K=E, V=E) + Z)
    D = LayerNorm(FeedForward(Z') + Z')
    return D
```

В полном декодере имеется N копий блока декодирования:

```
def Decoder(X, E, N):
    D = POS(Embed(X))
    for n in range(N):
        D = DecoderBlock(D, E)
    return D
```

Смотрите иллюстрацию в правой части рис. 15.26.

На этапе обучения все входы X декодера уже известны, так как они получены из погружений целевой выходной последовательности с запаздыванием. На этапе вывода (тестирования) мы должны декодировать последовательно и использовать замаскированное внимание, когда сгенерированный выход подается на вход слоя погружения, а затем добавляется в множество ключей

и значений, на которые следует обращать внимание (оно инициализируется токеном начала последовательности) (см. демонстрационный код по адресу code.problml.ai/book1/transformers_torch и подробное пособие по этой модели в работах [Rus18; Ala18]).

15.5.5. Сравнение трансформеров, СНС и НУС

На рис. 15.27 сравниваются три разных архитектуры отображения последовательности $x_{1:n}$ в другую последовательность $y_{1:n}$: одномерная СНС, РНС и модель внимания. В каждой модели приняты разные компромиссы в плане быстродействия и выразительности, причем последнюю можно количественно выразить в терминах длины максимального пути между любыми двумя входами. Сводка результатов сравнения приведена в табл. 15.1.

Таблица 15.1. Сравнение трансформера с другими нейронными последовательными порождающими моделями. Здесь n – длина последовательности, d – размерность входных признаков, а k – размер сверточного ядра. На основе табл. 1 из работы [Vas+17]

Тип слоя	Сложность	Число послед. операций	Длина макс. пути
Самовнимание	$O(n^2d)$	$O(1)$	$O(1)$
Рекуррентный	$O(nd^2)$	$O(n)$	$O(n)$
Сверточный	$O(knd^2)$	$O(1)$	$O(\log_k n)$

Для одномерной СНС с ядром размера k и d каналами признаков время вычисления выхода (которое можно организовать параллельно) равно $O(knd^2)$. Нам нужен стек, в котором число слоев равно n/k или $\log_k(n)$, если используется дырявая свертка, чтобы все пары могли взаимодействовать. Например, на рис. 15.27 мы видим, что x_1 и x_5 первоначально отстоят друг от друга на пять позиций, в слое 1 уже на три позиции, а в слое 2 связаны напрямую.

Вычислительная сложность РНС составляет $O(nd^2)$, где d – размер скрытого состояния, потому что на каждом шаге необходимо выполнять умножение матрицы на вектор. Это принципиально последовательная операция. Длина максимального пути составляет $O(n)$.

Наконец, для моделей самовнимания каждый выход напрямую связан с каждым входом, поэтому длина максимального пути равна $O(1)$. Однако вычислительная сложность составляет $O(n^2d)$. Для коротких последовательностей обычно $n \ll d$, так что это нормально. А для более длинных последовательностей мы обсудим быстрые варианты механизма внимания в разделе 15.6.

15.5.6. Применение трансформеров для изображений*

Для обработки изображений чаще всего применяются модели СНС, поскольку в них встроено полезное индуктивное смещение, например локальность

(вследствие небольших ядер), эквивариантность (вследствие связывания весов) и инвариантность (вследствие пулинга). Как ни удивительно, обнаружилось, что трансформеры тоже неплохо справляются с классификацией изображений, по крайней мере, если обучены на достаточном объеме данных. (Им необходимо очень много данных, чтобы компенсировать отсутствие индуктивного смещения.)

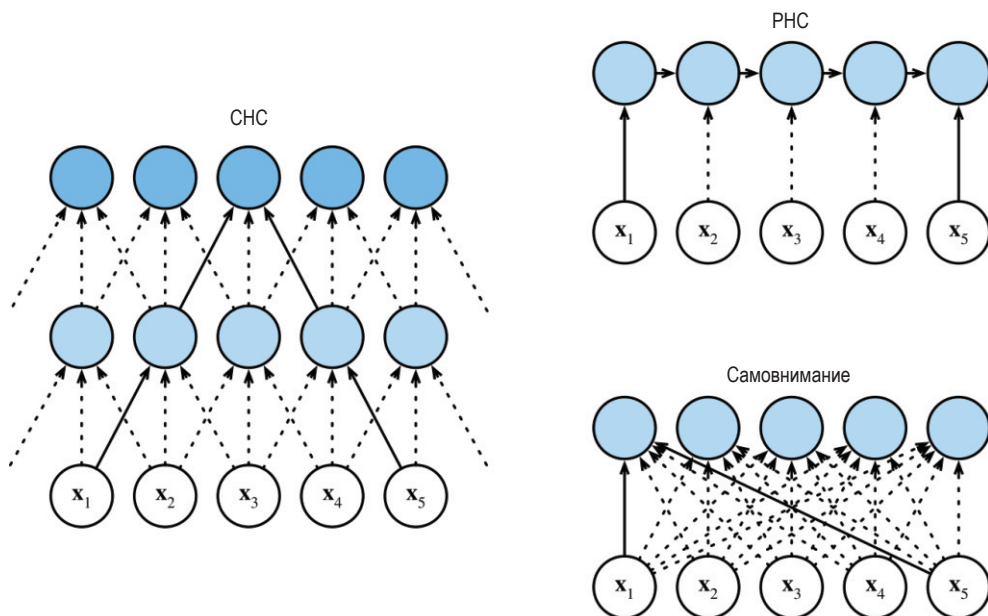


Рис. 15.27 ❖ Сравнение моделей одномерной СНС, РНС и самовнимания.
На основе рис. 10.6.1 из работы [Zha+20]. Печатается с разрешения Астона Чжана

В частности в работе [Dos+21] представлена модель **ViT** (vision transformer), которая разбивает вход на 16×16 патчей, проецирует каждый патч в пространство погружения, а затем передает этот набор погружений $x_{1:T}$ трансформеру – по аналогии с тем, как передаются трансформеру погружения слов. В начало входа добавляется специальное погружение [CLASS], x_0 . На выходе трансформера получается набор кодированных представлений $e_{0:T}$; модель отображает e_0 в целевую метку класса y и обучается с учителем (см. иллюстрацию на рис. 15.28).

После предобучения с учителем модель подвергается тонкой настройке на различных задачах классификации; этот подход называется переносом обучения (дополнительные сведения см. в разделе 19.2). При обучении на «небольших» наборах данных типа ImageNet (который содержит 1000 классов и 1.3 млн изображений) авторы обнаружили, что не могут превзойти предобученную модель на базе СНС ResNet (Section 14.3.4), известную под названием **BiT** (big transfer) [Kol+20]. Однако при обучении на более крупных наборах данных, например ImageNet-21k (21 000 классов и 14 млн изображений) или внутреннем наборе Google JFT (18 000 классов и 303 млн изо-

бражений) выяснилось, что ViT показывает лучшие результаты, чем BiT, при переносе обучения. Кроме того, ее обучение на таком масштабе данных обходится дешевле, чем ResNet. (Но все равно дорого: обучение модели ViT на наборе данных ImageNet-21k занимает 30 дней на машине Google Cloud TPUv3 с 8 ядрами!)

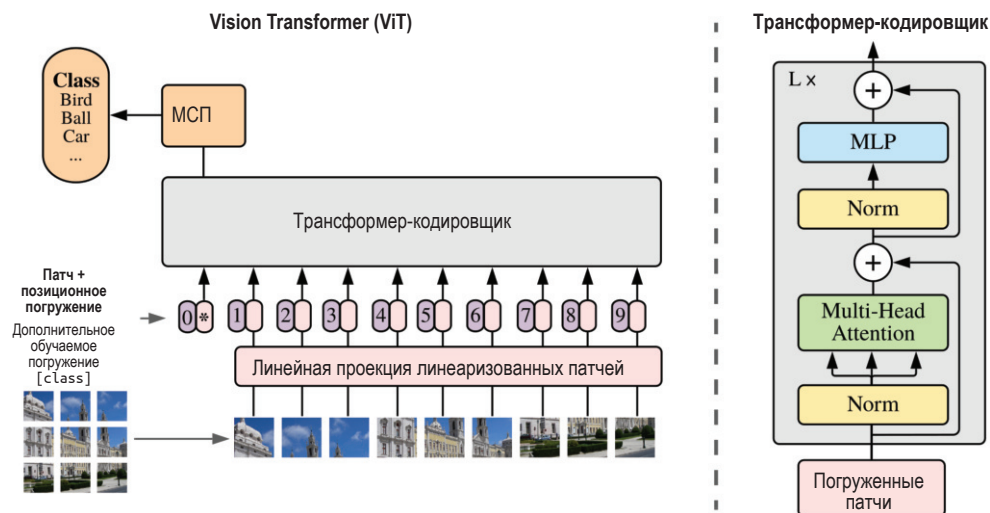


Рис. 15.28 ❖ Модель Vision Transformer (ViT). Изображение рассматривается как множество входных патчей. К входу добавляется специальный вектор погружения CLASS (обозначен *) в позиции 0. Метка класса для изображения выводится путем применения softmax к заключительной выходной кодировке в позиции 0. На основе рис. 1 из работы [Dos+21]. Печатается с разрешения Алексея Досовицкого

15.5.7. Другие варианты трансформеров*

В последние годы было опубликовано немало работ по обобщению трансформеров. Например, в статье по Gshard [Ler+21] показано, как масштабировать трансформеры на еще большее число параметров, заменив некоторые плотные слои прямого распространения модулем регрессии на основе смеси экспертов (раздел 13.6.2). Это открывает возможность для разреженных условных вычислений, когда для обработки любого заданного входа используются только часть емкости модели (выбираемая сетью-привратником).

Еще один пример – в работе [Gul+20] описана архитектура **конформера**, в которой к трансформеру добавлены сверточные слои. Показано, что она полезна в различных задачах распознавания речи.

15.6. ЭФФЕКТИВНЫЕ ТРАНСФОРМЕРЫ*

Этот раздел написан Кишиитофом Хоромански.

Для обычных трансформеров временная и пространственная сложность имеют порядок $O(N^2)$, где N – длина последовательности. Из-за этого они практически неприменимы к длинным последовательностям. За последние несколько лет были предложены различные эффективные варианты трансформеров, позволяющие обойти эту трудность. В этом разделе мы дадим краткий обзор таких методов (см. их сводку на рис. 15.29). Дополнительные сведения можно найти, например, в работах [Tay+20a; Tay+20b; Lin+21].



Рис. 15.29 ❖ На этой диаграмме Венна представлена таксономия эффективных архитектур трансформеров. Из работы [Tay+20a]. Печатается с разрешения И Тая

15.6.1. Фиксированные необучаемые локализованные паттерны внимания

Простейшая модификация механизма внимания состоит в том, чтобы ограничить его фиксированным необучаемым локализованным окном, иначе говоря, позволить каждому токenu проявлять внимание только к предварительно выбранному множеству других токенов. Если, например, каждая последовательность разбита на K блоков длины N/K , а внимание ограничено блоком, то временная и пространственная сложность уменьшаются с $O(N^2)$ до N^2/K . При $K \gg 1$ вычислительный выигрыш становится заметен. Такой под-

ход применен, в частности, в работах [Qiu+19a; Par+18]. Паттерны внимания необязательно должны принимать форму блоков. Из других подходов упомянем окна с шагом или пропусками, а также гибридные паттерны, когда комбинируется несколько фиксированных паттернов внимания Chi+19b; BPC20].

15.6.2. Обучаемые паттерны разреженного внимания

Описанный выше подход естественно обобщить, разрешив обучать компактные паттерны. Внимание по-прежнему ограничено парами токенов в одном разделе некоторого разбиения множества всех токенов, но теперь эти разбиения можно обучать. В этом классе методов можно выделить два основных подхода: на основе хеширования и на основе кластеризации. В первом случае все токены хешируются и, следовательно, разным разбиениям соответствуют разные ячейки хеша. Так устроена, например, архитектура **Reformer** [KKL20], в которой применяется локально-чувствительное хеширование (locality sensitive hashing – LSH). При этом временная сложность модуля внимания имеет порядок $O(NM^2 \log(M))$, где M – размерность погружений токенов.

В подходах на основе хеширования множество запросов должно совпадать с множеством ключей. Кроме того, количество ячеек хеша, необходимое для точного разбиения (которое рассматривается как константа), может быть велико. В подходе на основе кластеризации токены объединяются в кластеры с помощью стандартных алгоритмов кластеризации, например K средних (раздел 21.3); это называется «кластеризующим трансформером» [Roy+20]. Как и в случае блоков, если используется K кластеров одинакового размера, то пространственная сложность модуля внимания уменьшается до $O(N^2/K)$. На практике K часто выбирают порядка $K = \Theta(\sqrt{N})$, но добиться, чтобы кластеры были примерно одинакового размера, трудно.

15.6.3. Методы с добавлением памяти и рекуррентные методы

В некоторых подходах добавляется сторонний модуль памяти, которые может обращаться к нескольким токенам одновременно. Часто такие методы принимают форму алгоритма *глобальной памяти*, как, например, в работах [Lee+19; Zah+20].

Еще один подход – связать различные локальные блоки рекуррентным соотношением. Главный пример такого подхода – класс методов Transformer-XL [Dai+19].

15.6.4. Низкоранговые и ядерные методы

В этом разделе мы обсудим методы, которые аппроксимируют внимание с помощью низкоранговых матриц. В работе [She+18; Kat+20] матрица внимания A аппроксимируется непосредственно низкоранговой матрицей, так что

$$A_{ij} = \boldsymbol{\phi}(\mathbf{q}_i)^T \boldsymbol{\phi}(\mathbf{k}_j), \quad (15.62)$$

где $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^M$ – некоторый конечномерный вектор размерности $M < D$. Эту структуру можно использовать для вычисления $\mathbf{A}\mathbf{V}$ за время $O(N)$. К сожалению, в случае softmax-внимания матрица \mathbf{A} не является низкоранговой.

В архитектуре **Linformer** [Wan+20a] ключи и значения трансформируются с помощью случайных гауссовых проекций. Затем применяется лемма Джонсона–Линденштрауса [AL13] для аппроксимации softmax-внимания в этом пространстве меньшей размерности.

В архитектуре **Performer** [Cho+20a; Cho+20b] показано, что матрицу внимания можно вычислить с помощью (положительно определенной) ядерной функции. Ядерные функции мы определим в разделе 17.1, но основная идея заключается в том, что $\mathcal{K}(\mathbf{q}, \mathbf{k}) \geq 0$ – некоторая мера сходства между $\mathbf{q} \in \mathbb{R}^D$ и $\mathbf{k} \in \mathbb{R}^D$. Например, гауссово ядро, называемое также радиально-базисной функцией, имеет вид:

$$\mathcal{K}_{\text{gauss}}(\mathbf{q}, \mathbf{k}) = \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{q} - \mathbf{k}\|_2^2\right). \quad (15.63)$$

Чтобы понять, как его можно использовать для вычисления матрицы внимания, заметим, что в работе [Cho+20a] доказан следующий факт:

$$A_{i,j} = \exp\left(\frac{\mathbf{q}_i^T \mathbf{k}_j}{\sqrt{D}}\right) = \exp\left(\frac{-\|\mathbf{q}_i - \mathbf{k}_j\|_2^2}{2\sqrt{D}}\right) \times \exp\left(\frac{\|\mathbf{q}_i\|_2^2}{2\sqrt{D}}\right) \times \exp\left(\frac{\|\mathbf{k}_j\|_2^2}{2\sqrt{D}}\right). \quad (15.64)$$

Первый член в этом выражении равен $\mathcal{K}_{\text{gauss}}(\mathbf{q}_i D^{-1/4}, \mathbf{k}_j D^{-1/4})$ с $\sigma = 1$, а другие два члена – просто независимые масштабные коэффициенты.

До сих пор мы ничего не выиграли с вычислительной точки зрения. Однако в разделе 17.2.9.3 будет показано, что гауссово ядро можно записать в виде математического ожидания множества случайных признаков:

$$\mathcal{K}_{\text{gauss}}(\mathbf{x}, \mathbf{y}) = \mathbb{E}[\boldsymbol{\eta}(\mathbf{x})^T \boldsymbol{\eta}(\mathbf{y})], \quad (15.65)$$

где $\boldsymbol{\eta}(\mathbf{x}) \in \mathbb{R}^M$ – случайный вектор признаков, построенный на основе \mathbf{x} с помощью тригонометрических функций (формула (17.60)) либо с помощью экспоненциальных функций (формула (17.61)). (Во втором случае есть то преимущество, что все признаки положительны, и это дает гораздо лучшие результаты [Cho+20b].) Таким образом, для обычного softmax-внимания $A_{i,j}$ можно переписать в виде

$$A_{i,j} = \mathbb{E}[\boldsymbol{\phi}(\mathbf{q}_i)^T \boldsymbol{\phi}(\mathbf{k}_j)], \quad (15.66)$$

где $\boldsymbol{\phi}$ определено следующим образом:

$$\boldsymbol{\phi}(\mathbf{x}) \triangleq \exp\left(\frac{\|\mathbf{x}\|_2^2}{2\sqrt{D}}\right) \boldsymbol{\eta}\left(\frac{\mathbf{x}}{D^{1/4}}\right). \quad (15.67)$$

Мы можем записать полную матрицу внимания в виде

$$\mathbf{A} = \mathbb{E}[\mathbf{Q}'(\mathbf{K}')^\top], \quad (15.68)$$

где строки $\mathbf{Q}', \mathbf{K}' \in \mathbb{R}^{N \times M}$ кодируют карты признаков, соответствующие запросам и ключам. (Отметим, что качество можно улучшить, если эти случайные признаки ортогональны, детали см. в работе [Cho+20a].) Иллюстрация приведена на рис. 15.30.

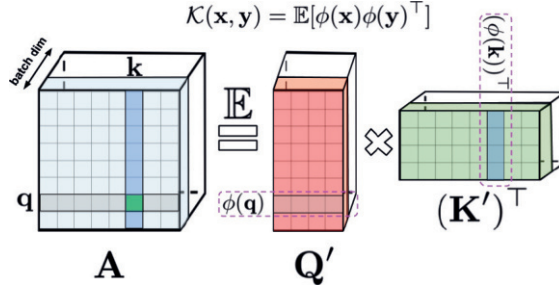


Рис. 15.30 ❖ Матрица внимания \mathbf{A} , переписанная в виде произведения двух матриц меньшего ранга \mathbf{Q}' и $(\mathbf{K}')^\top$ со случайными картами признаков $\phi(\mathbf{q}_i) \in \mathbb{R}^M$ и $\phi(\mathbf{v}_k) \in \mathbb{R}^M$ для соответствующих запросов/ключей, хранящихся в строках/столбцах. Печатается с разрешения Кшиштофа Хоромански

Мы можем создать аппроксимацию \mathbf{A} , воспользовавшись одной выборкой из случайных признаков $\phi(\mathbf{q}_i)$ и $\phi(\mathbf{k}_i)$ и взяв небольшое значение M , скажем $M = O(D \log(D))$. Затем можно аппроксимировать весь оператор внимания за время $O(N)$, воспользовавшись разложением

$$\widehat{\text{attention}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{diag}^{-1}(\mathbf{Q}'((\mathbf{K}')^\top \mathbf{1}_N))(\mathbf{Q}'((\mathbf{K}')^\top \mathbf{V})). \quad (15.69)$$

Можно показать, что это несмещенная аппроксимация точного оператора внимания softmax (рис. 15.31). (О том, как обобщить этот результат на замаскированное (каузальное) внимание, см. [Cho+20a].)

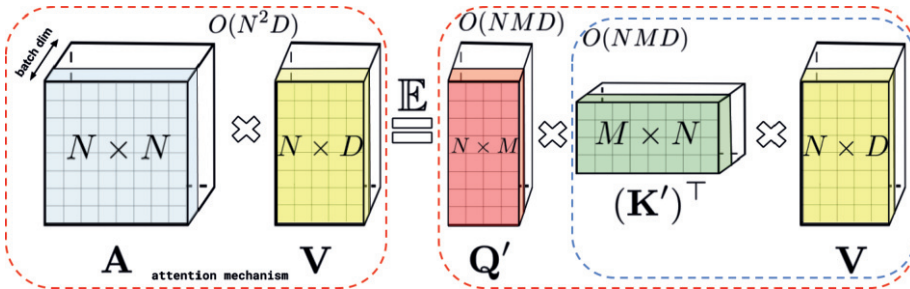


Рис. 15.31 ❖ Разложение матрицы внимания \mathbf{A} можно использовать для ускорения вычисления внимания благодаря свойству ассоциативности умножения матриц. Для вычисления \mathbf{AV} мы сначала вычисляем $\mathbf{G} = (\mathbf{k}')^\top \mathbf{V}$, а затем $\mathbf{q}' \mathbf{G}$; пространственная и временная сложность этого вычисления составляет $O(N)$. Печатается с разрешения Кшиштофа Хоромански

15.7. ЯЗЫКОВЫЕ МОДЕЛИ И ОБУЧЕНИЕ ПРЕДСТАВЛЕНИЙ БЕЗ УЧИТЕЛЯ

Мы обсудили, как РНС и авторегрессионные (включающие только декодер) трансформеры можно использовать в качестве **языковых моделей**, т. е. порождающих моделей вида $p(x_1, \dots, x_T) = \prod_{t=1}^T p(x_t | x_{1:t-1})$, где каждое x_t – дискретный токен, например слово или часть слова (см. обсуждение методов предварительной обработки текста в разделе 1.5.4). Затем латентное состояние этих моделей можно использовать в качестве непрерывного векторного представления текста. То есть вместо унитарного вектора \mathbf{x}_t или его обученного погружения (типа тех, что обсуждаются в разделе 20.5) мы используем скрытое состояние \mathbf{h}_t , зависящее от всех предыдущих слов в предложении. Эти векторы можно далее использовать как **контекстуальные погружения слов** для таких целей, как классификация текста или задачи типа seq2seq (см., например, обзор [LKB20]). Преимущество такого подхода в том, что мы можем **предобучить** языковую модель без учителя на большом корпусе текстов, а затем **дообучить** (fine-tune) ее с учителем на небольшом помеченном наборе данных, специально подобранном для данной задачи. (Этот общий подход называется **переносом обучения**, детали см. в разделе 19.2.)

Если наша основная цель – вычислить полезные представления для переноса обучения, а не генерирование текста, то можно заменить порождающую последовательности модель некаузальными моделями, которые умеют вычислять представление предложения, но не умеют генерировать его. У таких моделей есть важное преимущество: теперь скрытое состояние \mathbf{h}_t может зависеть от прошлого $\mathbf{y}_{1:t-1}$, настоящего \mathbf{y}_t и будущего $\mathbf{y}_{t+1:T}$. Иногда это дает более качественные представления, потому что учитывается больше контекста.

В следующих разделах мы кратко обсудим некоторые модели без учителя, каузальные и некаузальные, для обучения представлений текста.

15.7.1. ELMo

В работе [Pet+18] представлен метод **ELMo** (Embeddings from Language Model). Основная идея – обучить две языковые модели типа РНС слева направо и справа налево, а затем объединить представления их скрытых состояний и так получить погружения для каждого слова. В отличие от двунаправленной РНС (раздел 15.2.2), которой нужна пара вход–выход, ELMo без учителя обучается минимизировать отрицательное логарифмическое правдоподобие входного предложения $\mathbf{x}_{1:T}$:

$$\mathcal{L}(\theta) = -\sum_{t=1}^T [\log p(x_t | \mathbf{x}_{1:t-1}; \theta_e, \theta^{\rightarrow}, \theta_s) + \log p(x_t | \mathbf{x}_{t+1:T}; \theta_e, \theta^{\leftarrow}, \theta_s)], \quad (15.70)$$

где θ_e – разделяемые параметры слоя погружения, θ_s – разделяемые параметры выходного слоя softmax, а θ^{\rightarrow} и θ^{\leftarrow} – параметры обеих моделей РНС. (Ав-

торы использовали РНС типа LSTM, описанную в разделе 15.2.7.2.) Смотрите иллюстрацию на рис. 15.32.

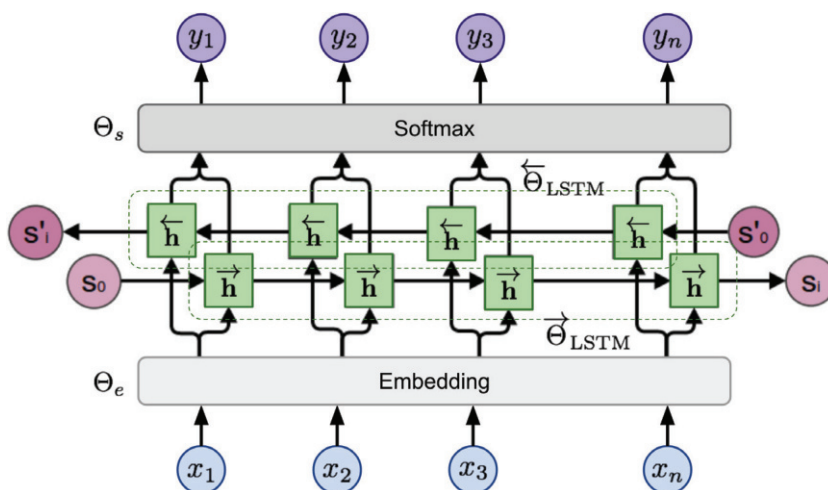


Рис. 15.32 ❖ Двухнаправленная языковая модель ELMo. Здесь $y_t = x_{t+1}$ в прямой LSTM и $y_t = x_{t-1}$ в обратной LSTM. (Для обработки граничных случаев мы добавляем специальные маркера bos и eos.) Из работы [Wen19]. Печатается с разрешения Лилян Вен

После обучения мы определяем контекстуальное представление $\mathbf{r}_t = [\mathbf{e}_t, \mathbf{h}_{t,1:L}^{\rightarrow}, \mathbf{h}_{t,1:L}^{\leftarrow}]$, где L – число слоев в LSTM. Затем мы обучаем набор зависящих от задачи линейных весов для отображения в конечное зависящее от контекста погружение каждого токена: $\mathbf{r}_t^j = \mathbf{r}_t^T \mathbf{w}^j$, где j – идентификатор задачи. Если решается задача синтаксического разбора, например **частеречная разметка** (part-of-speech – POS) (т. е. пометка каждого слова как существительного, глагола, прилагательного и т. д.), то модель обучится придавать больший вес нижним слоям. Если же решается задача семантического разбора, например **разрешение лексической неоднозначности** (word sense disambiguation – WSD), то модель обучится придавать больший вес верхним слоям. В обоих случаях необходим лишь небольшой объем зависящих от задачи размеченных данных, поскольку мы обучаем единственный вектор весов для отображения $\mathbf{r}_{1:T}$ на целевые метки $\mathbf{y}_{1:T}$.

15.7.2. BERT

В этом разделе мы опишем модель **BERT** (Bidirectional Encoder Representations from Transformers) из работы [Dev+19]. Как и ELMo, это некаузальная модель, которую можно использовать для создания представлений текста, но не для генерирования текста. В ней используется модель трансформера для отображения модифицированной версии последовательности обратно в немодифицированную форму. В позиции t модифицированного входа присутствуют

все слова, кроме t -го, а задача заключается в том, чтобы предсказать отсутствующее слово. Эта задача называется **заполнением пропусков**, или **cloze**.

15.7.2.1. Замаскированная языковая модель

Точнее, модель обучается минимизировать отрицательное логарифмическое **псевдоправдоподобие**:

$$\mathcal{L} = \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{\mathbf{m}} \sum_{i \in \mathbf{m}} -\log p(x_i | \mathbf{x}_{-\mathbf{m}}), \quad (15.71)$$

где \mathbf{m} – случайная двоичная маска. Например, если обучать модель на расшифровках кулинарных видеороликов, то обучающее предложение может иметь вид¹:

Let's make [MASK] chicken! [SEP] It [MASK] great with orange sauce.

где [SEP] – токен, разделяющий два предложения. Желаемые целевые метки для опущенных (замаскированных) слов – «some» и «tastes»². (Этот пример взят из работы [Sun+19a].)

Условная вероятность вычисляется применением softmax к скрытому вектору последнего слоя в позиции i :

$$p(x_i | \hat{\mathbf{x}}) = \frac{\exp(\mathbf{h}(\hat{\mathbf{x}})_i^\top \mathbf{e}(x_i))}{\sum_{x'} \exp(\mathbf{h}(\hat{\mathbf{x}})_i^\top \mathbf{e}(x'))}, \quad (15.72)$$

где $\hat{\mathbf{x}} = \mathbf{x}_{-\mathbf{m}}$ – замаскированная входная последовательность, а $\mathbf{e}(x)$ – погружение токена x . Это используется для вычисления потери в замаскированных позициях, поэтому часто встречается название **замаскированная языковая модель**. (Она похожа на шумоподавляющий автокодировщик из раздела 20.3.2.) Модель показана на рис. 15.33а.

15.7.2.2. Задача предсказания следующего предложения

Помимо целевой функции для замаскированной языковой модели, в оригинальной статье по BERT есть еще одна целевая функция, позволяющая обучить модель предсказывать, следует ли одно предложение за другим. Формально модели подается на вход последовательность:

$$\text{CLS } A_1 A_2; \dots A_m; \text{SEP } B_1 B_2; \dots; B_n \text{ SEP}, \quad (15.73)$$

где SEP – специальный токен-разделитель, а CLS – специальный токен-маркер класса. Если предложение B следует за предложением A в оригинальном тексте, то мы полагаем целевую метку $y = 1$, а если B – случайно выбранное предложение, то полагаем $y = 0$. Это задача **предсказания следующего предложения**. Такого рода предобучение может быть полезно для класси-

¹ Давайте приготовим [MASK] курочку! [SEP] Она отлично [MASK] с апельсиновым соусом. – Прим. перев.

² Примерно так: «вкусную» и «сочетается». – Прим. перев.

фикации пар предложений, например в задачах извлечения имплицитных знаний из текста или установления сходства текстов (см. раздел 15.4.6). (Отметим, что такое предобучение считается обучением без учителя или с частичным привлечением учителя, потому что целевые метки генерируются автоматически.)

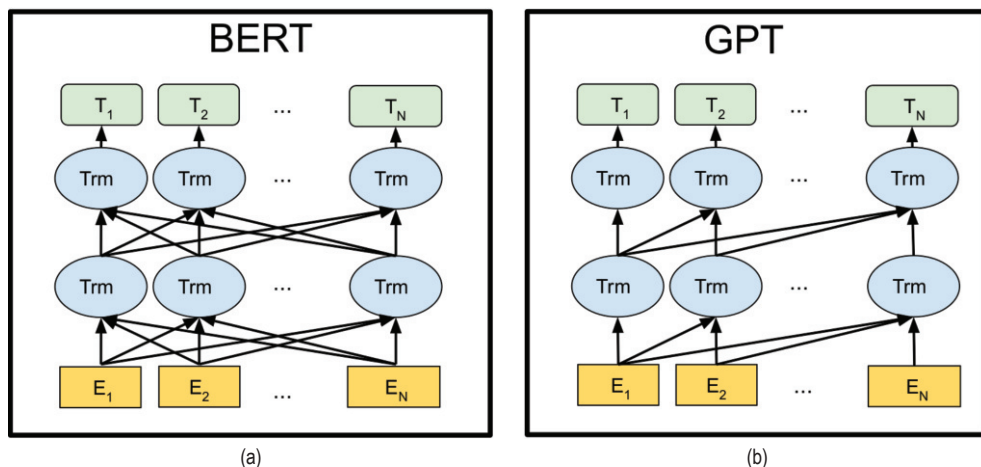


Рис. 15.33 ❖ Иллюстрация (a) BERT и (b) GPT. E_t – вектор погружения для входного токена в позиции t , а T_t – подлежащий предсказанию выход. На основе рис. 3 из работы [Dev+19]. Печатается с разрешения Мин-Вей Чана

Для предсказания следующего предложения на вход модели подается три разных погружения: токена, метки каждого сегмента (предложение A или B) и позиции (с использованием обученного позиционного погружения). Затем они складываются (см. иллюстрацию на рис. 15.34). Далее в BERT используется кодировщик трансформера для обучения отображения этой входной последовательности погружений в выходную последовательность погружений, которая затем декодируется в метки слов (для замаскированных позиций) или в метку класса (для позиции с токеном CLS).

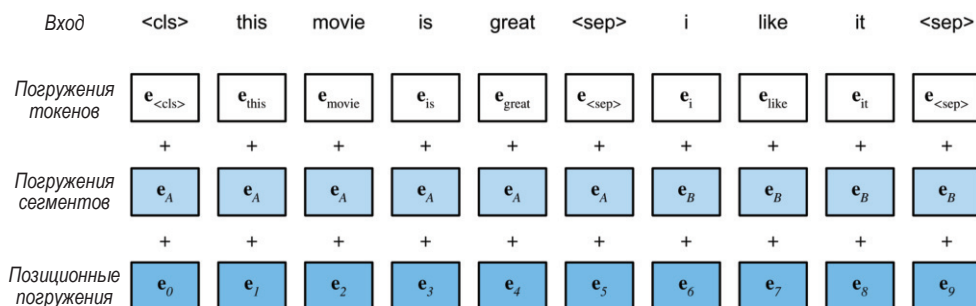


Рис. 15.34 ❖ Как пара входных предложений A и B кодируется перед подачей на вход BERT. На основе рис. 14.8.2 из работы [Zha+20]. Печатается с разрешения Астона Чжана

15.7.2.3. Дообучение BERT для приложений NLP

После предобучения BERT без учителя мы можем использовать ее для решения различных задач, выполнив дообучение с учителем. (Введение в такие методы переноса обучения см. в разделе 19.2.)

На рис. 15.35 показано, как модифицировать модель BERT для решения различных задач, просто добавив один или несколько выходных путей в последний скрытый слой (см. также демонстрационный код по адресу code.probl.ai/book1/bert_torch).

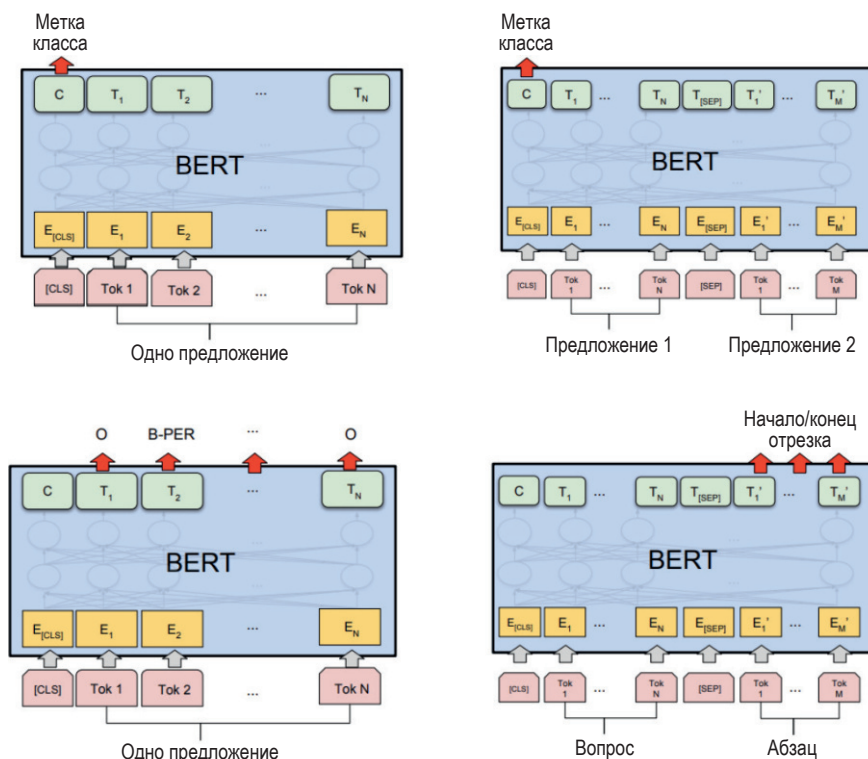


Рис. 15.35 ❖ Как использовать модель BERT для разных задач обучения с учителем в NLP. (а) Классификация одного предложения (например, анализ эмоциональной окраски). (б) Классификация пары предложений (например, извлечение имплицитной информации из текста). (с) Разметка одного предложения (например, поверхностный синтаксический анализ). (д) Ответы на вопросы. На основе рис. 4 из работы [Dev+19]. Печатается с разрешения Мин-Вей Чана

На рис. 15.35а показано, как можно классифицировать одно предложение (например, с целью анализа его эмоциональной окраски): мы просто берем вектор признаков, ассоциированный со специальным токеном CLS, и подаем его на вход МСП. Поскольку каждый выход обращает внимание на все входы, этот скрытый вектор обобщает все предложение. Затем МСП обучается отображать его в нужное пространство меток.

На рис. 15.35b показано, как можно классифицировать пару предложений (например, для имплицитного извлечения знаний из текста, рассмотренного в разделе 15.4.6): мы подаем на вход оба предложения, отформатированных, как в (15.73), а затем классифицируем токен CLS.

На рис. 15.35c показано, как разметить одно предложение, т. е. ассоциировать метку с каждым словом, а не со всем предложением. Типичное приложение – **частеречная разметка**, когда каждое слово аннотируется частью речи: существительное, глагол, прилагательное и т. д. Еще одно приложение – **выделение именных групп**, или **поверхностный синтаксический анализ**, когда требуется аннотировать все именные группы. Предложение размечается в нотации **ВЮ**, где В – начало группы, I – внутренние слова, О – слово вне любой группы. Например, рассмотрим следующее предложение:

B I O O O B I O B I I
British Airways rose after announcing its withdrawal from the UAI deal

Здесь мы видим три именных группы, «British Airways», «its withdrawal» и «the UAI deal». (Мы требуем, чтобы метки В, I и О встречались в этом порядке, так что это априорное ограничение можно включить в модель.)

Можно также ассоциировать типы с каждой именной группой, например выделить имена людей, названия мест, организаций и т. д. Тогда пространство меток принимает вид {B-Per, I-Per, B-Loc, I-Loc, B-Org, I-Org, Outside}. Это задача **распознавания именованных сущностей** (NER), она является важным шагом извлечения информации. Например, рассмотрим следующее предложение:

BP IP O O O BL IL BP O O O O
Mrs Green spoke today in New York. Green chairs the finance committee.

Мы можем сделать вывод, что в первом предложении имеются две именованные сущности, а именно: «Mrs Green» (типа Person – человек) и «New York» (типа Location – место). Во втором предложении также упоминается человек, «Green», скорее всего, тот же самый, хотя это перекрестное разрешение сущностей не является частью задачи NER.

Наконец, на рис. 15.35d показано, как можно реализовать **вопросно-ответную систему**. Здесь первое предложение – вопрос, второе – текст, а на выходе требуется указать позиции начала и конца частей текста, в которых содержится ответ на вопрос (см. табл. 1.4). Начало s и конец e вычисляются путем применения двух разных МСП к подвергнутой пулингу версии выходных кодировок текста; выходом обоих МСП является результат softmax по всем позициям. На этапе тестирования мы можем извлечь отрезок (i, j) , который максимизирует сумму оценок $s_i + e_j$ для $i \leq j$.

BERT показывает лучшее на текущий момент качество на многих задачах NLP. В работе [TDP19] показано, что BERT неявно сводится к стандартному конвейеру NLP, в котором различные слои выполняют такие задачи, как частеречная разметка (POS), синтаксический анализ, распознавание именованных сущностей (NER), назначение семантических ролей (semantic role labeling – SRL), разрешение кореференции и т. д. Дополнительные сведения о NLP можно найти в работе [JM20].

15.7.3. GPT

В работе [Rad+18] предложена модель **GPT** (Generative Pre-training Transformer). Это каузальная (порождающая) модель, в которой в качестве декодера используется замаскированный трансформер (см. иллюстрацию на рис. 15.33b).

В оригинальной статье авторы выполняют совместную оптимизацию на большом неразмеченном и на малом размеченном наборе данных. В случае применения модели для классификации потеря имеет вид $\mathcal{L} = \mathcal{L}_{cls} + \lambda \mathcal{L}_{LM}$, где $\mathcal{L}_{cls} = -\sum_{(x,y) \in \mathcal{D}_L} \log p(y|x)$ – потеря классификации на размеченных данных, а $\mathcal{L}_{LM} = -\sum_{x \in \mathcal{D}_U} \sum_t p(x_t | x_{1:t-1})$ – потеря языкового моделирования на неразмеченных данных.

В работе [Rad+19] предложена модель GPT-2, увеличенная версия GPT, обученная на большом корпусе документов из веб, получившем название **WebText**. Авторы также устранили зависящую от задачи стадию обучения, а вместо этого просто обучали сеть как языковую модель. Недавно компания OpenAI разработала модель GPT-3 [Bro+20], даже большую, чем GPT-2, но основанную на тех же принципах. Версия этой модели с открытым исходным кодом, доступная по адресу <https://huggingface.co/EleutherAI>, обучена на корпусе англоязычных текстов размером 800 ГБ, названном «The Pile» [Gao+20].

15.7.3.1. Приложения GPT

GPT умеет генерировать текст по начальной **подсказке**. Подсказка может описывать задачу; если сгенерированный текст удовлетворяет условиям задачи без модификации, то говорят, что модель выполняет **перенос обучения без примеров** (zero-shot task transfer) (детали см. в разделе 19.6).

Например, для **реферирования** (abstractive summarization) входного текста $x_{1:T}$ (в отличие от **квазиреферирования** (extractive summarization), при котором просто извлекается подмножество входных слов) мы производим выборку из распределения $p(x_{T+1:T+100} | [x_{1:T}; \text{TL;DR}])$, где **TL;DR** – специальный токен, добавленный в конец входного текста, который говорит системе, что пользователь хочет получить реферат. Аббревиатура **TL;DR** означает «too long; didn't read»¹ и часто встречается в веб-текстах, за ней обычно следует созданный человеком реферат. Добавив этот токен в состав входных данных, пользователь надеется перевести декодер трансформера в состояние, из которого он войдет в режим реферирования. (Чтобы указать модели, какую задачу она должна выполнить, есть способ лучше – обучить ее на парах вход–выход, как описано в разделе 15.7.4.)

15.7.4. T5

Многие модели обучаются без учителя, а затем дообучаются на конкретных задачах. Можно также обучить одну модель решению нескольких задач; для

¹ Слишком длинно, не стал читать. – Прим. перев.

этого нужно включить в состав входного предложения указание на то, какую задачу выполнять, а затем обучить модель по типу seq2seq, как показано на рис. 15.36. Этот подход применен в модели **T5** [Raf+19], что означает «Text-to-text Transfer Transformer». Это стандартный трансформер seq2seq, предобученный без учителя на парах (x' , x''), где x' – замаскированная версия x , а x'' – пропущенные токены, которые нужно предсказать, и затем дообученный с учителем на нескольких парах (x , y).

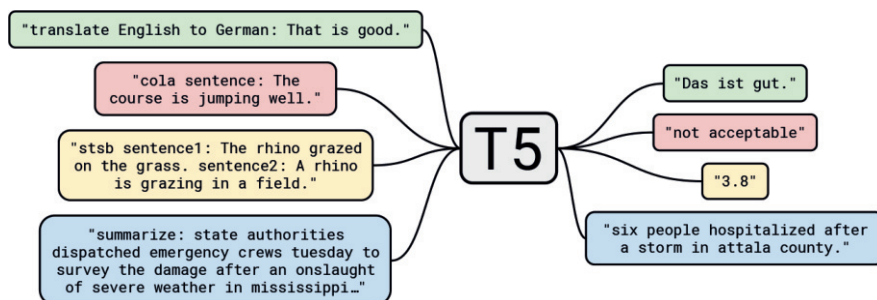


Рис. 15.36 ❖ Как можно использовать модель T5 (Text-to-text Transfer Transformer) для решения нескольких задач NLP, например: перевод с английского на немецкий; определение лингвистической корректности предложения (CoLA – Corpus of Linguistic Acceptability); определение степени семантического сходства (STSB – Semantic Textual Similarity Benchmark) и реферирование текста. На основе рис. 1 из работы [Raf+19]. Печатается с разрешения Колина Раффела

Данные, на которых производится обучение без учителя, берутся из корпуса **C4** (Colossal Clean Crawled Corpus) взятых из веб текстов суммарным объемом 750 ГБ. Они используются для предобучения с использованием шумоподавляющей целевой функции, похожей на ту, что применяется в BERT. Например, предложение x = «Thank you for inviting me to your party last week» можно преобразовать во вход x' = «Thank you <X> me to your party <Y> week» и выход (цель) x'' = «<X> for inviting <Y> last <EOS>», где <X> и <Y> – токены, уникальные для данного примера. Наборы данных для обучения с учителем создают вручную или берутся из литературы. Этот подход в настоящее время считается самым современным для многих задач NLP.

15.7.5. Обсуждение

Гигантские языковые модели типа BERT и GPT-3 в последнее время вызвали большой интерес и даже получили освещение в крупных СМИ¹. Однако есть сомнения, действительно ли такие системы «понимают» язык в каком-то разумном смысле или просто переупорядочивают фразы, которые встреча-

¹ См., например, <https://www.nytimes.com/2020/11/24/science/artificial-intelligence-ai-gpt3.html>.

лись в огромных обучающих наборах. Например, в работе [NK19] показано, что способность BERT показывать почти такие же результаты, как человек в задаче «Argument Reasoning Comprehension» (Аргументация Рассуждение Понимание), «можно полностью объяснить использованием не несущих никакого смысла статистических подсказок в наборе данных». Стоит немного изменить набор данных, как качество уменьшится до уровня случайности. Критику подобных моделей см. также в работах [BK20; Mar20].

Часть IV

НЕПАРАМЕТРИЧЕСКИЕ МОДЕЛИ

Глава 16

Методы на основе эталонов

До сих пор мы в этой книге рассматривали в основном **параметрические модели**, безусловные $p(y|\theta)$ или условные $p(y|x, \theta)$, где θ – вектор параметров фиксированной размерности. Параметры оцениваются с использованием набора данных переменного размера, $\mathcal{D} = \{(x_n, y_n) : n = 1 \dots N\}$, но после обучения модели эти данные отбрасываются.

В этом разделе мы рассмотрим различные виды **непараметрических моделей**, в которых обучающие данные сохраняются. Таким образом, эффективное число параметров модели может расти вместе с $|\mathcal{D}|$. Нас будут интересовать модели, которые можно определить в терминах **сходства** между тестовым входом x и каждым из обучающих входов x_n . Альтернативно модель можно определить в терминах **несходства** или функции расстояния $d(x, x_n)$. Поскольку модели сохраняют обучающие примеры до этапа тестирования, мы называем их **моделями на основе эталонов** (exemplar-based model). (Этот подход называют также **обучением на примерах** (instance-based learning) [AKA91] или обучением **на основе запоминания** (memory-based learning)).

16.1. Классификация методом К ближайших соседей (KNN)

В этом разделе мы обсудим один из самых простых классификаторов – метод **К ближайших соседей** (K nearest neighbor – **KNN**). Идея следующая: чтобы классифицировать новый вход x , мы находим K ближайших к x примеров в обучающем наборе – обозначим их $N_K(x, \mathcal{D})$, – а затем смотрим на их метки и строим распределение выходов в локальной окрестности x . Точнее, мы вычисляем распределение:

$$p(y = c|x, \mathcal{D}) = \frac{1}{K} \sum_{n \in N_K(x, \mathcal{D})} \mathbb{I}(y_n = c). \quad (16.1)$$

Затем мы можем вернуть это распределение или мажоритарную метку.

Два основных параметра этой модели – размер окрестности, K , и метрика $d(\mathbf{x}, \mathbf{x}')$. В качестве метрики часто принимается **расстояние Махаланобиса**:

$$d_M(\mathbf{x}, \boldsymbol{\mu}) = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \mathbf{M} (\mathbf{x} - \boldsymbol{\mu})}, \quad (16.2)$$

где \mathbf{M} – положительно определенная матрица. Если $\mathbf{M} = \mathbf{I}$, то эта метрика сводится к евклидову расстоянию. Как обучить метрику, мы обсудим в разделе 16.2.

Можно показать, что, несмотря на простоту, KNN-классификатор при $N \rightarrow \infty$ не более чем в два раза хуже байесовской ошибки (которая измеряет качество наилучшего возможного классификатора) [CH67; CD14]. (Конечно, на практике скорость сходимости к этому оптимальному качеству может быть низкой, причины мы обсудим в разделе 16.1.2.)

16.1.1. Пример

На рис. 16.1(a) показан KNN-классификатор на плоскости при $K = 5$. Тестовая точка обозначена крестиком. Три из пяти ближайших к ней соседей имеют метку 1, и два оставшихся – метку 0. Поэтому мы предсказываем $p(y = 1 | \mathbf{x}, \mathcal{D}) = 3/5 = 0.6$.

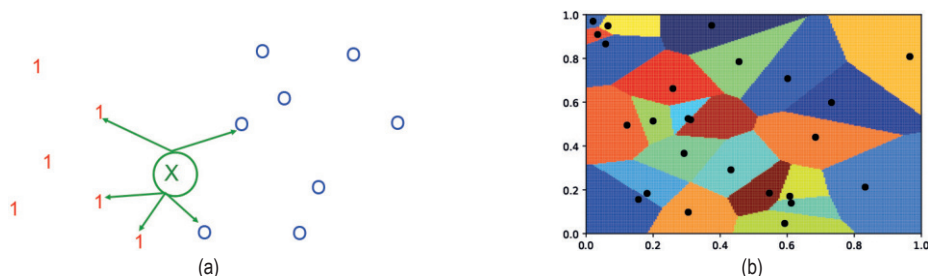


Рис. 16.1 ❖ (a) Классификатор по K ближайшим соседям на двумерной плоскости при $K = 5$. Ближайшие соседи тестовой точки \mathbf{x} имеют метки $\{1, 1, 1, 0, 0\}$, поэтому мы предсказываем $p(y = 1 | \mathbf{x}, \mathcal{D}) = 3/5$. (b) Диаграмма Вороного, индуцированная классификатором 1-NN. На основе рис. 4.13 из работы [DHS01].

Построено программой по адресу figures.problm.lai/book1/16.1

Если положить $K = 1$, то мы будем просто возвращать метку ближайшего соседа, поэтому прогнозное распределение сводится к дельта-функции. KNN-классификатор с $K = 1$ индуцирует **диаграмму Вороного** множества точек (см. рис. 16.1b). Это разбиение пространства, при котором с каждой точкой \mathbf{x}_n ассоциируется область $V(\mathbf{x}_n)$ такая, что все точки, принадлежащие $V(\mathbf{x}_n)$, расположены к \mathbf{x}_n ближе, чем к любой другой точке. Внутри каждой ячейки предсказывается метка соответствующей обучающей точки. Таким образом, при $K = 1$ ошибка на обучающем наборе равна 0. Однако такая модель обычно переобучена, как будет показано ниже.

На рис. 16.2 приведен пример применения KNN-классификатора к двумерному набору данных с тремя классами. Мы видим, что при $K = 1$ метод не допускает ни одной ошибки на обучающем наборе. При увеличении K решающие границы становятся более гладкими (поскольку мы производим усреднение по большим окрестностям), поэтому ошибка на обучающем наборе возрастает, а модель постепенно становится недообученной. Это показано на рис. 16.2d. График ошибки на тестовом наборе имеет обычную U-образную форму.

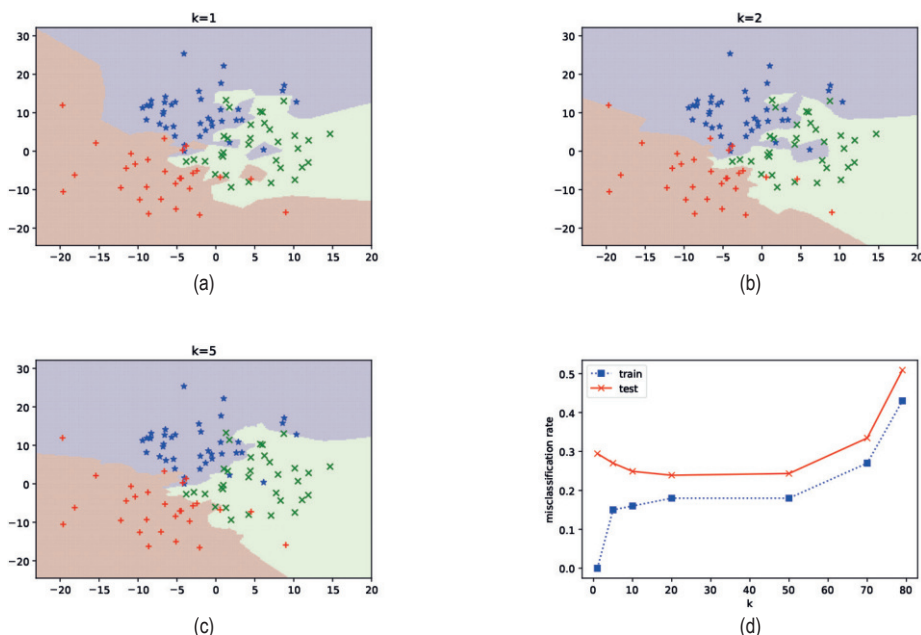


Рис. 16.2 ❖ Решающие границы, индуцированные KNN-классификатором. (a) $K = 1$. (b) $K = 2$. (c) $K = 5$. (d) Зависимость ошибки на обучающем и тестовом наборе от K . Построено программой по адресу figures.problml.ai/book1/16.2

16.1.2. Проклятие размерности

Главная статистическая проблема KNN-классификаторов – то, что они плохо работают для входных данных высокой размерности. Этот феномен называется **проклятием размерности**.

Суть проблемы состоит в том, что пространственный объем экспоненциально растет с увеличением размерности, поэтому для поиска ближайшего соседа, возможно, придется заглянуть очень далеко. Чтобы было понятнее, рассмотрим пример из книги [НТФ09, стр. 22]. Предположим, что KNN-классификатор применяется к входным данным, равномерно распределенным в D -мерном единичном кубе. Допустим, что плотность распределения меток классов в окрестности тестовой точки \mathbf{x} оценивается путем «растяжения» гиперкуба вокруг \mathbf{x} , пока он не включит заданную долю p входных точек.

Ожидаемая длина ребра этого куба будет равна $e_D(s) \triangleq p^{1/D}$; график этой функции показан на рис. 16.3b. Если $D = 10$ и мы хотим, чтобы оценка была основана на 10 % данных, то имеем $e_{10}(0.1) = 0.8$, т. е. мы должны растянуть куб на 80 % по каждому измерению. Даже если используется всего 1 % данных, все равно $e_{10}(0.01) = 0.63$. Принимая во внимание, что диапазон изменения данных в каждом измерении – от 0 до 1, этот метод оказывается не очень-то локальным, несмотря на слова «ближайшие соседи». А проблема в том, что соседи, расположенные так далеко, могут оказаться плохими предикторами поведения функции в данной точке.

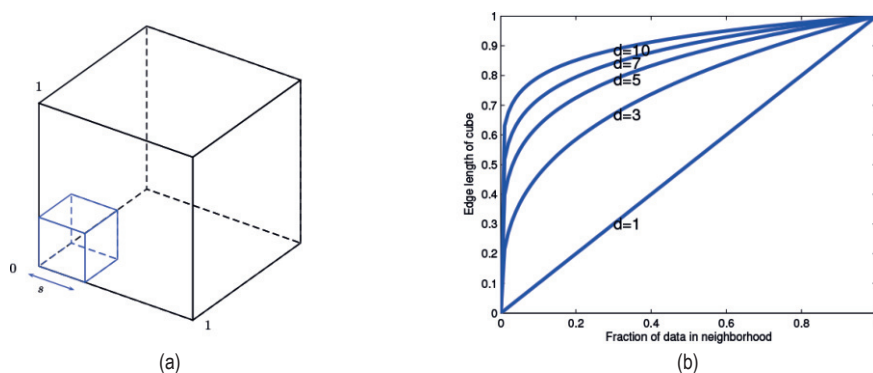


Рис. 16.3 ❖ Проклятие размерности. (а) Малый куб со стороной s вложен в больший единичный куб. (б) Длина ребра куба, содержащего заданную долю объема единичного куба, как функция от числа измерения. На основе рис. 2.6 из книги [НТФ09]. Построено программой по адресу figures.probl.ai/book1/16.3

Существует два основных способа избавиться от проклятия размерности: сделать некоторые допущения о форме функции (т. е. использовать параметрическую модель) или использовать метрику, которая принимает во внимание только часть измерений (см. раздел 16.2).

16.1.3. Снижение требований к скорости и памяти

KNN-классификаторы хранят все обучающие данные. Очевидно, что это очень расточительно. Были предложены различные способы обрезки, т. е. удаления точек, не влияющих на решающие границы; см., например, [WM00]. В разделе 17.4 обсуждается теоретически подкрепленный подход, основанный на априорном распределении, поощряющем разреженность; получающийся метод называется разреженной ядерной машиной и позволяет хранить только подмножество, содержащее наиболее полезные примеры.

Что касается времени работы, то наша цель заключается в том, чтобы найти K ближайших соседей за время, меньшее $O(N)$, где N – размер обучающего набора. Задача нахождения точных ближайших соседей вычислительно неразрешима, когда размерность превышает 10, поэтому большинство методов довольствуется нахождением приближенных. Есть два основных класса

таких методов: основанные на разбиении пространства на области и на использовании хеширования.

Что касается методов разбиения, то можно воспользоваться какой-нибудь разновидностью **k-d деревьев**, которые делят пространство на ортогональные области, или каким-нибудь методом кластеризации, в котором применяются якорные точки. Из методов на основе хеширования самым распространенным является **локально-чувствительное хеширование** (locality sensitive hashing – **LSH**) [GIM99], хотя существуют и более современные методы обучения функции хеширования на данных (см., например, [Wan+15]). Хорошее введение в методы хеширования имеется в книге [LRU14].

По адресу <https://github.com/facebookresearch/faiss> можно скачать библиотеку с открытым исходным кодом **FAISS**, реализующую эффективный точный и приближенный поиск ближайшего соседа (и кластеризацию методом К средних) для плотных векторов. Эта библиотека описана в работе [JDJ17].

16.1.4. Распознавание открытого множества

Не спрашивай, как это называется. Спрашивай, на что это похоже.

— Моше Бар [Bar09]

Во всех задачах классификации, которые мы рассматривали до сих пор, предполагалось, что множество классов \mathcal{C} фиксировано. (Это пример **допущения замкнутости мира**, заключающегося в том, что существует фиксированное число (типов) сущностей.) Однако во многих реальных задачах могут появляться тестовые примеры, принадлежащие новым категориям. Это так называемое **распознавание открытого множества** мы и обсудим ниже.

16.1.4.1. Онлайновое обучение, обнаружение посторонних и распознавание открытого множества

Предположим, что мы обучаем систему распознавания лиц предсказывать личность человека из фиксированного множества, или **галереи** изображений лиц. Обозначим $\mathcal{D}_t = \{(\mathbf{x}_n, \mathbf{y}_n) : \mathbf{x}_n \in \mathcal{X}, \mathbf{y}_n \in \mathcal{C}_t, n = 1 \dots N_t\}$ размеченный набор данных в момент t , где \mathcal{X} – множество изображений (лиц), а $\mathcal{C}_t = \{1, \dots, C_t\}$ – множество людей, известных системе в момент t (где $C_t \leq t$). На этапе тестирования система может встретить нового человека, которого не видела раньше. Пусть \mathbf{x}_{t+1} – это новое изображение, а $\mathbf{y}_{t+1} = C_{t+1}$ – его новая метка. Система должна понять, что вход принадлежит новой категории и не пытаться по ошибке сопоставить ему метку из \mathcal{C}_t . Это называется **обнаружением новизны**. В этом случае вход генерируется из распределения $p(\mathbf{x}|\mathbf{y} = C_{t+1})$, где $C_{t+1} \notin \mathcal{C}_t$ – новая «метка класса». Определить, что \mathbf{x}_{t+1} принадлежит новому классу, может быть трудно, если внешне новое изображение похоже на одно из уже присутствующих в \mathcal{D}_t .

Обнаружив, что \mathbf{x}_{t+1} принадлежит новой категории, система может запросить идентификатор этого экземпляра, назовем его C_{t+1} . Затем она может добавить помеченную пару $(\mathbf{x}_{t+1}, C_{t+1})$ в набор данных, который назовем \mathcal{D}_{t+1} ,

и расширить множество классов, добавив C_{t+1} в C_t (см. [JK13]). Это называется **инкрементным обучением, онлайнным обучением, обучением на протяжении всей жизни** или **непрерывным обучением**. В будущем система может встретить изображение, выбранное из распределения $p(\mathbf{x}|y = c)$, где c – существующий или новый класс, или изображение, выбранное из совершенно нового распределения $p'(\mathbf{x})$, никак не связанного с лицами (например, кто-то загрузил фото своей собаки). В последнем случае говорят об **обнаружении посторонних** (out-of-distribution, или **OOD**).

В такой онлайнной постановке мы часто имеем всего несколько (иногда только один) примеров из каждого класса. В этом случае предсказание называется **классификацией по малому числу примеров** и подробнее обсуждается в разделе 19.6. KNN-классификаторы хорошо подходят для такой задачи. Например, мы можем просто сохранить все примеры из каждого класса в галерее, как описано выше. В момент $t + 1$, получив вход \mathbf{x}_{t+1} , мы не предсказываем его метку, сравнивая \mathbf{x}_{t+1} с какой-то параметрической моделью для каждого класса, а просто находим в галерее пример, ближайший (или самый похожий) к \mathbf{x}_{t+1} ; назовем его \mathbf{x}' . Затем нужно определить, достаточно ли \mathbf{x}' и \mathbf{x}_{t+1} похожи, чтобы считаться одним и тем же. (В контексте классификации лиц это называется **повторной идентификацией человека**, или **верификацией лица**, см., например, [WSH16]).) Если совпадения не найдено, можно объявить пример новым, или посторонним.

Ключевым компонентом всех описанных выше задач является метрика сходства (или различия) входов. Как обучить ее, мы обсудим в разделе 16.2.

16.1.4.2. Другие задачи открытого мира

Проблемы распознавания открытого множества и инкрементного обучения – это лишь два примера задач, требующих **допущения открытого мира**, см. [Rus15]. Есть много других задач такого рода.

Например, рассмотрим задачу **разрешения сущностей**, или **связывания сущностей**. В ней требуется решить, относятся ли разные строки (например, «John Smith» и «Jon Smith») к одной сущности или к разным. Детали см. в работе [SHF15].

Еще одно важное приложение – **многоцелевое сопровождение**. Например, когда радар обнаруживает новую отметку, он должен определить, относится ли она к уже сопровождаемой цели или это новый объект, появившийся в воздушном пространстве. Элегантный математический аппарат для решения таких задач – теория **случайных конечных множеств** – описан в работах [Mah07; Mah13; Vo+15].

16.2. ОБУЧЕНИЕ МЕТРИК

Умение вычислять «семантическое расстояние» между парой точек, $d(\mathbf{x}, \mathbf{x}') \in \mathbb{R}^+$ для $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, или, эквивалентно, их сходство $s(\mathbf{x}, \mathbf{x}') \in \mathbb{R}^+$, имеет огромное значение для таких задач, как классификация по ближайшим соседям (раз-

дел 16.1), самообучение (раздел 19.2.4.4), кластеризация по сходству (раздел 21.5), поиск по содержимому, визуальное сопровождение и т. д.

Если пространство входов $\mathcal{X} = \mathbb{R}^D$, то в качестве метрики чаще всего выбирается расстояние Махаланобиса:

$$d_{\mathbf{M}}(\mathbf{x}, \mathbf{x}') = \sqrt{(\mathbf{x} - \mathbf{x}')^{\top} \mathbf{M} (\mathbf{x} - \mathbf{x}')}. \quad (16.3)$$

В разделе 16.2.1 мы обсудим некоторые методы обучения матрицы \mathbf{M} . Если вход имеет высокую размерность или структурирован, то лучше сначала обучить погружение $\mathbf{e} = f(\mathbf{x})$, а затем вычислять расстояния в пространстве погружения. Если f – глубокая нейронная сеть, то говорят о **глубоком обучении метрики**, эту тему мы обсудим в разделе 16.2.2.

16.2.1. Линейные и выпуклые методы

В этом разделе мы обсудим несколько методов обучения матрицы \mathbf{M} в расстоянии Махаланобиса – как прямых (когда задача рассматривается как выпуклая), так и косвенных, с помощью линейного проецирования. О других подходах к обучению метрики см., например, работы [Kul13; Kim19].

16.2.1.1. Метод ближайших соседей с большим зазором

В работе [WS09] предложено обучать матрицу Махаланобиса \mathbf{M} , так чтобы результирующая метрика хорошо работала совместно с классификатором по ближайшим соседям. Этот метод называется методом **ближайших соседей с большим зазором** (large margin nearest neighbor – LMNN).

Работает он следующим образом. Для каждого примера i обозначим N_i множество целевых соседей; обычно в этой роли выступает множество K точек с одинаковой меткой класса, ближайших в смысле евклидовой метрики. Затем \mathbf{M} оптимизируется, так чтобы минимизировать суммарное расстояние между каждой точкой i и всеми ее целевыми соседями $j \in N_i$:

$$\mathcal{L}_{\text{pull}}(\mathbf{M}) = \sum_{i=1}^N \sum_{j \in N_i} d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j)^2. \quad (16.4)$$

Мы также хотим убедиться, что примеры с неподходящими метками расположены далеко друг от друга. Для этого мы стремимся сделать так, чтобы пример i был ближе (с некоторым зазором $m \geq 0$) к своим целевым соседям j , чем к точкам l с другими метками (они называются **самозванцами**). Этого можно добиться посредством минимизации функции

$$\mathcal{L}_{\text{push}}(\mathbf{M}) = \sum_{i=1}^N \sum_{j \in N_i} \sum_{l=1}^N \mathbb{I}(y_i \neq y_l) [m + d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j)^2 - d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_l)^2]_+, \quad (16.5)$$

где $[z]_+ = \max(z, 0)$ – функция кусочно-линейной потери (раздел 4.3.2). Полная целевая функция $\mathcal{L}(\mathbf{M}) = (1 - \lambda)\mathcal{L}_{\text{pull}}(\mathbf{M}) + \lambda\mathcal{L}_{\text{push}}(\mathbf{M})$, где $0 < \lambda < 1$. Это

выпуклая функция, определенная на выпуклом множестве, и для ее минимизации можно применить методы **полуопределенного программирования**. Альтернативно можно параметризовать задачу, положив $\mathbf{M} = \mathbf{W}^T \mathbf{W}$, а затем минимизировать относительно \mathbf{W} методами безусловной градиентной оптимизации. Эта задача уже не выпуклая, но допускает отображение \mathbf{W} низкой размерности.

Для больших наборов данных нужно учитывать, что вычислительная сложность решения задачи (16.5) составляет $O(N^3)$. Мы обсудим некоторые приемы ускорения в разделе 16.2.5.

16.2.1.2. Анализ компонент соседства

Еще один способ обучить линейное отображение \mathbf{W} такое, что $\mathbf{M} = \mathbf{W}^T \mathbf{W}$, называется **анализом компонент соседства** (neighborhood components analysis — **NCA** [Gol+05]. Его идея заключается в нахождении вероятности того, что пример \mathbf{x}_i имеет \mathbf{x}_j в качестве ближайшего соседа, с применением линейной функции softmax:

$$p_{ij}^{\mathbf{W}} = \frac{\exp(-\|\mathbf{W}\mathbf{x}_i - \mathbf{W}\mathbf{x}_j\|_2^2)}{\sum_{l \neq i} \exp(-\|\mathbf{W}\mathbf{x}_i - \mathbf{W}\mathbf{x}_l\|_2^2)}. \quad (16.6)$$

(Это версия стохастических погружений соседства, обсуждаемых в разделе 20.4.10.1, только обучаемая с учителем.) Математическое ожидание числа примеров, правильно классифицированных 1NN-классификатором с метрикой \mathbf{W} , равно $J(\mathbf{W}) = \sum_{i=1}^N \sum_{j=i, y_j=y_i} p_{ij}^{\mathbf{W}}$. Обозначим $\mathcal{L}(\mathbf{W}) = 1 - J(\mathbf{W})/N$ ошибку при исключении по одному. Для минимизации \mathcal{L} относительно \mathbf{W} можно применить градиентные методы.

16.2.1.3. Анализ латентных совпадений

Еще один способ обучить линейное отображение \mathbf{W} такое, что $\mathbf{M} = \mathbf{W}^T \mathbf{W}$, называется **анализом латентных совпадений** (latent coincidence analysis — **LCA**) [DS12]. Определяется условная модель латентных переменных для отображения пары входов, \mathbf{x} и \mathbf{x}' , в метку $y \in \{0, 1\}$, которое описывает, являются ли входы похожими (например, имеют одну и ту же метку классу) или непохожими. Каждый вход $\mathbf{x} \in \mathbb{R}^D$ отображается в латентную точку $\mathbf{z} \in \mathbb{R}^L$ в пространстве низкой размерности с помощью стохастического отображения $p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mathbf{W}\mathbf{x}, \sigma^2 \mathbf{I})$, и $p(\mathbf{z}'|\mathbf{x}') = \mathcal{N}(\mathbf{z}'|\mathbf{W}\mathbf{x}', \sigma^2 \mathbf{I})$. (Сравните с факторным анализом, обсуждаемым в разделе 20.2.) Затем определяется вероятность того, что оба входа похожи: $p(y = 1|\mathbf{z}, \mathbf{z}') = \exp((-1/2\kappa^2)\|\mathbf{z} - \mathbf{z}'\|)$, см. иллюстрацию предположений модели на рис. 16.4.

Мы можем максимизировать логарифмическое маргинальное правдоподобие $\ell(\mathbf{W}, \sigma^2, \kappa^2) = \sum_n \log p(y_n|\mathbf{x}_n, \mathbf{x}'_n)$, применив ЕМ-алгоритм (раздел 8.7.2). (Без ограничения общности можно положить $\kappa = 1$, потому что это значение влияет только на масштаб \mathbf{W} .) Точнее, на Е-шаге мы вычисляем апостериорное распределение $p(\mathbf{z}, \mathbf{z}'|\mathbf{x}, \mathbf{x}', y)$ (что можно сделать в замкнутой форме), а на М-шаге решаем задачу оптимизации методом взвешенных наименьших

квадратов (см. раздел 13.6.2). ЕМ-алгоритм гарантирует монотонный рост целевой функции и не нуждается в корректировке размера шага, в отличие от градиентных методов, применяемых в анализе компонентов соседства (раздел 16.2.1.2). (Для обучения модели можно использовать также вариационный байесовский вывод (раздел 4.6.8.3), равно как и различные его разреженные и нелинейные обобщения, обсуждаемые в работе [ZMY19].)

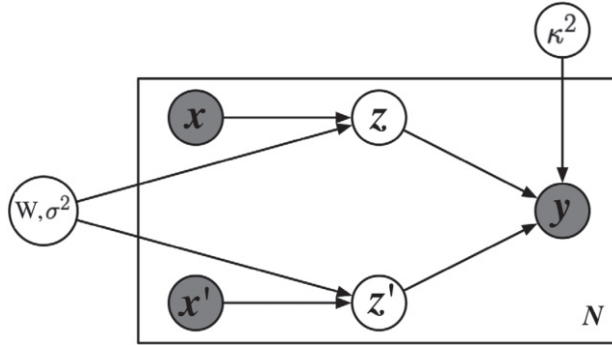


Рис. 16.4 ❖ Иллюстрация анализа латентных совпадений (LCA) в виде ориентированной графовой модели. Входы $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^D$ отображаются в латентные гауссовы переменные $\mathbf{z}, \mathbf{z}' \in \mathbb{R}^L$ с помощью линейного отображения \mathbf{W} . Если обе латентных точки совпадают (с точностью до масштаба длины κ), то мы полагаем метку сходства $y = 1$, в противном случае полагаем $y = 0$. На основе рис. 1 из работы [DS12]. Печатается с разрешения Лоуренса Сола

16.2.2. Глубокое обучение метрики

При измерении расстояния между многомерными или структурированными входами очень полезно сначала обучить погружение в «семантическое» пространство меньшей размерности, в котором расстояние более осмыслено и в меньшей степени подвержено проклятию размерности (раздел 16.1.2). Пусть $\mathbf{e} = f(\mathbf{x}; \boldsymbol{\theta}) \in \mathbb{R}^L$ – погружение входа, сохраняющее «релевантные» семантические аспекты входа, а $\hat{\mathbf{e}} = \mathbf{e} / \|\mathbf{e}\|_2$ – результат его ℓ_2 -нормировки. После нормировки все точки будут лежать на поверхности гиперболы. Тогда мы можем измерить расстояние между двумя точками, пользуясь евклидовой метрикой:

$$d(\mathbf{x}_i, \mathbf{x}_j; \boldsymbol{\theta}) = \|\hat{\mathbf{e}}_i - \hat{\mathbf{e}}_j\|_2^2, \quad (16.7)$$

при которой точки тем более схожи, чем меньше расстояние между ними. Или же можно воспользоваться коэффициентом Отиаи:

$$d(\mathbf{x}_i, \mathbf{x}_j; \boldsymbol{\theta}) = \hat{\mathbf{e}}_i^\top \hat{\mathbf{e}}_j, \quad (16.8)$$

чем он больше, тем более схожи точки. (Коэффициент Отиаи измеряет угол между двумя векторами, как показано на рис. 20.43.) Эти величины связаны соотношением:

$$\|\hat{\mathbf{e}}_i - \hat{\mathbf{e}}_j\|_2^2 = (\hat{\mathbf{e}}_i - \hat{\mathbf{e}}_j)^T (\hat{\mathbf{e}}_i - \hat{\mathbf{e}}_j) = 2 - 2\hat{\mathbf{e}}_i^T \hat{\mathbf{e}}_j. \quad (16.9)$$

Этот общий подход называется **глубоким обучением метрики** (deep metric learning – **DML**).

Основная идея DML – обучить функцию погружения такую, что похожие примеры ближе друг к другу, чем непохожие. Формально предположим, что имеется размеченный набор данных $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1 \dots N\}$, из которого можно выбрать похожие пары $\mathcal{S} = \{(i, j) : \mathbf{y}_i = \mathbf{y}_j\}$. Если $(i, j) \in \mathcal{S}$, но $(i, k) \notin \mathcal{S}$, то мы предполагаем, что \mathbf{x}_i и \mathbf{x}_j должны быть ближе в пространстве погружения, тогда как \mathbf{x}_i и \mathbf{x}_k должны быть дальше. Ниже мы обсудим разные способы гарантировать выполнение этого свойства. Отметим, что эти методы работают и тогда, когда меток классов нет, при условии, что существует какой-то другой способ определить похожие пары. Например, в разделе 19.2.4.3 мы обсудим подходы к обучению представлений на основе самообучения, при которых семантически похожие пары создаются автоматически, и обучим погружения, при которых такие пары гарантированно оказываются ближе, чем семантически не связанные.

Прежде чем переходить к детальному обсуждению DML, отметим, что многие недавние подходы к DML не так хороши, как было заявлено; см. работы [MBL20; Rot+20]. (Утверждения, содержащиеся в некоторых из упомянутых там статей, часто неверны из-за некорректного экспериментального сравнения, этот недостаток вообще свойствен современным исследованиям по МО, как обсуждается, например, в работах [BLV19; LS19b].) Поэтому мы сосредоточимся на немного более старых и простых методах, которые, однако, надежнее.

16.2.3. Потери классификации

Предположим, что имеется размеченный набор данных с C классами. Тогда можно обучить модель классификации за время $O(NC)$, а затем использовать скрытые признаки как функцию погружения. (Часто используют предпоследний слой, потому что он лучше обобщается на новые классы, чем последний.) Этот подход прост и хорошо масштабируется. Однако так можно обучиться только погружению примеров по правильную сторону от решающей границы, поэтому вовсе необязательно, что похожие примеры будут находиться близко друг к другу, а непохожие далеко. Кроме того, этот метод нельзя использовать, если отсутствуют помеченные обучающие данные.

16.2.4. Потери ранжирования

В этом разделе мы рассмотрим минимизацию **потери ранжирования** с целью гарантировать, что похожие примеры ближе друг к другу, чем непохожие. По большей части эти методы не нуждаются в метках классов (хотя иногда мы предполагаем, что метки существуют, потому что так проще ввести нотацию для определения сходства).

16.2.4.1. Попарная (сопоставительная) потеря и сиамские сети

Один из самых ранних подходов к обучению представлений по похожим и непохожим парам был основан на следующей **сопоставительной потере** (contrastive loss) [CHL05]:

$$\mathcal{L}(\theta; \mathbf{x}_i, \mathbf{x}_j) = \mathbb{I}(y_i = y_j) d(\mathbf{x}_i, \mathbf{x}_j)^2 + \mathbb{I}(y_i \neq y_j) [m - d(\mathbf{x}_i, \mathbf{x}_j)]_+^2, \quad (16.10)$$

где $[z]_+ = \max(0, z)$ – кусочно-линейная потеря, а $m > 0$ – параметр, определяющий зазор. Интуитивно понятно, что мы хотим, чтобы положительные пары (с одинаковыми метками) были близки, а отрицательные (с разными метками) находились друг от друга на расстоянии, большем минимального зазора. Эта функция потерь минимизируется на множестве всех пар примеров. Наивный алгоритм занимает время $O(N^2)$, но в разделе 16.2.5 описано, как его можно ускорить.

Отметим, что мы используем один и тот же экстрактор признаков $f(\cdot, \theta)$ для обоих входов \mathbf{x}_i и \mathbf{x}_j при вычислении расстояния, как показано на рис. 16.5а. Поэтому получающаяся сеть называется **сиамской сетью** (отсылка к сиамским близнецам).

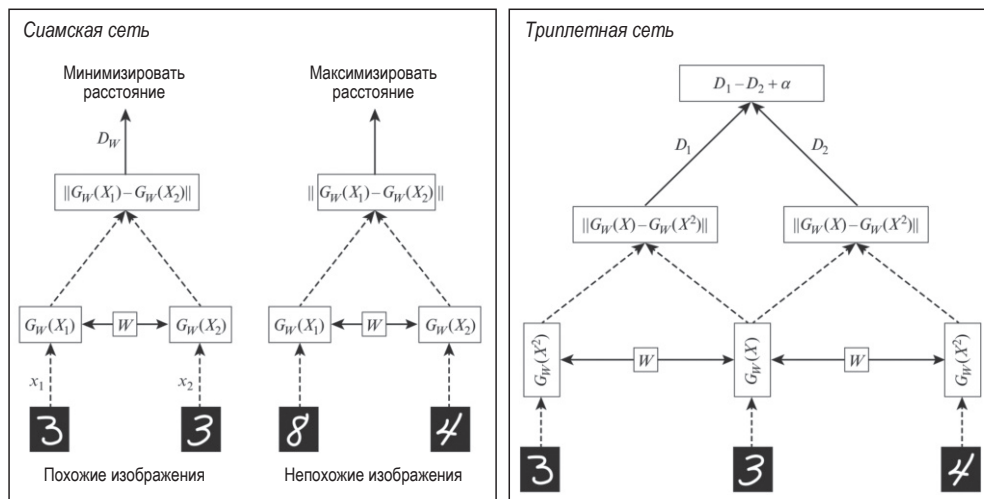


Рис. 16.5 ❖ Сети для глубокого обучения метрики.

(а) Сиамская сеть. (б) Триплетная сеть. На основе рис. 5 из работы [KB19]

16.2.4.2. Триплетная потеря

Недостатком всех попарных потерь является то, что оптимизация положительных и отрицательных пар производится независимо, из-за чего их абсолютные величины могут оказаться несравнимыми. Решение – использовать **триплетную потерю** [SKP15], определенную следующим образом. Для каждого примера i (называемого **якорем**) находим похожий (положительный)

пример \mathbf{x}_i^+ и непохожий (отрицательный) пример \mathbf{x}_i^- . Затем минимизируем следующую потерю, усредненную по всем тройкам:

$$\mathcal{L}(\theta; \mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-) = [d_\theta(\mathbf{x}_i, \mathbf{x}_i^+)^2 - d_\theta(\mathbf{x}_i, \mathbf{x}_i^-)^2 + m]_+. \quad (16.11)$$

Интуитивно это означает, что мы хотим, чтобы расстояние от якоря до положительного примера было меньше (на величину некоторого безопасного зазора m), чем до отрицательного. Триpletную потерю можно вычислить с помощью tripletной сети, как показано на рис. 16.5b.

Наивная минимизация tripletной потери занимает время $O(N^3)$. На практике мы вычисляем потерю на мини-пакете (выбранном так, чтобы для якорной точки, которая часто является первым элементом мини-пакета, существовал по крайней мере один похожий и один непохожий пример). Тем не менее метод может работать медленно. Некоторые способы ускорения мы обсудим в разделе 16.2.5.

16.2.4.3. N -парная потеря

У tripletной потери есть одна проблема – каждый якорь сравнивается только с одним отрицательным примером за раз. Из-за этого обучающий сигнал может оказаться недостаточно сильным. Одно из решений – создать задачу многоклассовой классификации, в которой для каждого якоря имеется набор из $N - 1$ отрицательных и одного положительного примера. Это называется **N -парной потерей** [Soh16]. Формально для каждого набора определяется следующая потеря:

$$\mathcal{L}(\theta; \mathbf{x}_i, \mathbf{x}_j^+, \{\mathbf{x}_k^-\}_{k=1}^{N-1}) = -\log \left(1 + \sum_{k=1}^{N-1} \exp(\hat{\mathbf{e}}_\theta(\mathbf{x}_i)^\top \hat{\mathbf{e}}_\theta(\mathbf{x}_k^-) - \hat{\mathbf{e}}_\theta(\mathbf{x}_i)^\top \hat{\mathbf{e}}_\theta(\mathbf{x}_j^+)) \right) \quad (16.12)$$

$$= -\log \frac{\exp(\hat{\mathbf{e}}_\theta(\mathbf{x}_i)^\top \hat{\mathbf{e}}_\theta(\mathbf{x}_k^-))}{\exp(\hat{\mathbf{e}}_\theta(\mathbf{x}_i)^\top \hat{\mathbf{e}}_\theta(\mathbf{x}_j^+)) + \sum_{k=1}^{N-1} \exp(\hat{\mathbf{e}}_\theta(\mathbf{x}_i)^\top \hat{\mathbf{e}}_\theta(\mathbf{x}_k^-))}. \quad (16.13)$$

Отметим, что N -парная потеря – то же самое, что потеря **InfoNCE** в статье о сопоставительном предиктивном кодировании (CPC) [OLV18]. В работе [Che+20a] предложена версия, в которой степень сходства масштабируется с помощью члена температуры; авторы назвали ее потерей **NT-Xent** (normalized temperature-scaled cross-entropy). Температурный параметр можно рассматривать как масштабирование радиуса гиперболы, на поверхности которого расположены данные.

При $N = 2$ эта потеря сводится к логистической:

$$\mathcal{L}(\theta; \mathbf{x}, \mathbf{x}^+, \mathbf{x}_i) = -\log(1 + \exp(\hat{\mathbf{e}}_\theta(\mathbf{x})^\top \hat{\mathbf{e}}_\theta(\mathbf{x}_i) - \hat{\mathbf{e}}_\theta(\mathbf{x})^\top \hat{\mathbf{e}}_\theta(\mathbf{x}^+))). \quad (16.14)$$

Сравните с потерей с зазором, используемой в tripletном обучении (при $m = 1$):

$$\mathcal{L}(\theta; \mathbf{x}, \mathbf{x}^+, \mathbf{x}^-) = -\max(0, \hat{\mathbf{e}}(\mathbf{x})^\top \hat{\mathbf{e}}(\mathbf{x}^-) - \hat{\mathbf{e}}(\mathbf{x})^\top \hat{\mathbf{e}}(\mathbf{x}^+) + 1). \quad (16.15)$$

Сравнение обеих функций представлено на рис. 4.2.

16.2.5. Ускорение оптимизации потери ранжирования

Главный недостаток потери ранжирования – стоимость вычисления функции потерь, $O(N^2)$ или $O(N^3)$, из-за необходимости сравнивать все пары или тройки примеров. В этом разделе мы обсудим различные способы ускорения работы.

16.2.5.1. Добычные методы

Важнейшее наблюдение заключается в том, что не нужно рассматривать все отрицательные примеры для каждого якоря, потому что большинство из них неинформативно (т. е. дают нулевую потерю). Вместо этого можно сосредоточиться на тех отрицательных примерах, которые ближе к якорю, чем ближайший положительный пример. Они называются **твердо отрицательными** и особенно полезны для ускорения вычисления триплетной потери.

Формально если a – якорная точка, а p – ближайший к ней положительный пример, то говорят, что пример n твердо отрицателен (для a), если $d(\mathbf{x}_a, \mathbf{x}_n) < d(\mathbf{x}_a, \mathbf{x}_p)$ и $y_n \neq y_a$. Иногда у якоря может не быть ни одного твердо отрицательного примера. Поэтому пул кандидатов можно расширить, включив в него **полутвердо отрицательные примеры**, для которых

$$d(\mathbf{x}_a, \mathbf{x}_p) < d(\mathbf{x}_a, \mathbf{x}_n) < d(\mathbf{x}_a, \mathbf{x}_p) + m, \quad (16.16)$$

где $m > 0$ – параметр зазора, см. иллюстрацию на рис. 16.6а. Эта техника используется в модели Google **FaceNet** [SKP15], которая обучает функцию погружения для лиц таким образом, что похожие лица группируются в кластер, которому пользователь может дать имя.

На практике твердо отрицательные примеры обычно выбираются внутри мини-пакета. Поэтому необходимо увеличить размер пакета, чтобы обеспечить достаточное разнообразие. Или же можно организовать отдельный процесс, который непрерывно обновляет множество потенциальных твердо отрицательных примеров по мере того, как в ходе обучения изменяется метрика.

16.2.5.2. Методы на основе представителей

Минимизация триплетной потери обходится дорого даже в случае добычи отрицательных примеров (раздел 16.2.5.1). В идеале хотелось бы найти метод, работающий за время $O(N)$ как функция потерь при классификации.

Один такой метод, предложенный в работе [MA+17], измеряет расстояние между каждым якорем и множеством P **представителей** каждого класса, а не напрямую расстояние от якоря до примеров. Представителей нужно обновлять в онлайн-режиме по мере эволюции метрики в процессе обучения. Вся процедура занимает время $O(NP^2)$, где $P \sim C$.

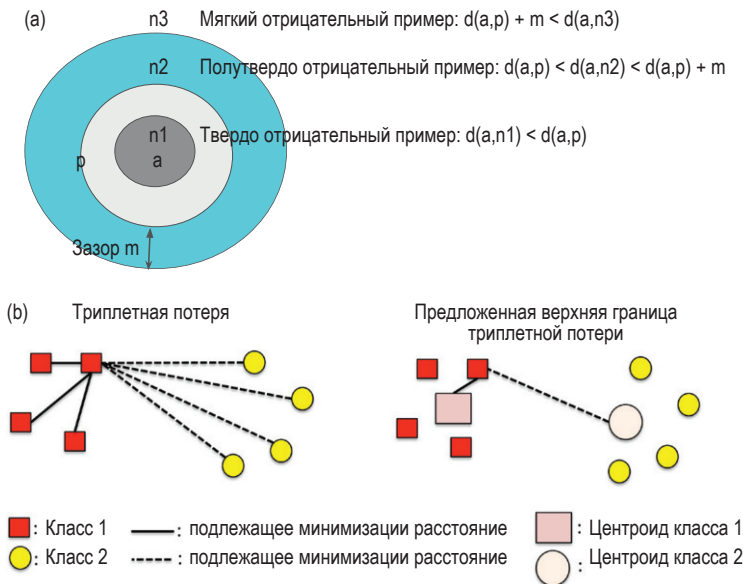


Рис. 16.6 ❖ Ускорение минимизации триплетной потери. (a) Твердые и мягкие отрицательные примеры. Здесь a – якорная точка, p – положительная точка и имеется n_i отрицательных точек. На основе рис. 4 из работы [KB19]. (b) Вычисление стандартной триплетной потери потребовало бы $8 \times 3 \times 4 = 96$ операций, тогда как для вычисления потери на основе представителей (с одним представителем на класс) нужно всего $8 \times 2 = 16$ вычислений. На основе рис. 1 из работы [Do+19]. Печатается с разрешения Густаво Чернейро

Недавно в работе [Qia+19] было предложено представлять каждый класс несколькими прототипами. При этом используемая **мягкая триплетная потеря** все равно обеспечивает линейную временную сложность.

16.2.5.3. Оптимизация верхней границы

В работе [Do+19] предложен простой и быстрый метод оптимизации триплетной потери. Основная идея – определить по одному *фиксированному* представителю или центроиду в каждом классе, а затем использовать расстояние до представителя как верхнюю границу триплетной потери.

Формально рассмотрим упрощенную форму триплетной потери, без зазора:

$$\ell_t(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) = \|\hat{\mathbf{e}}_i - \hat{\mathbf{e}}_j\| - \|\hat{\mathbf{e}}_i - \hat{\mathbf{e}}_k\|, \quad (16.17)$$

где $\hat{\mathbf{e}}_i = \hat{\mathbf{e}}_{\theta}(\mathbf{x}_i)$ и т. д. Согласно неравенству треугольника

$$\|\hat{\mathbf{e}}_i - \hat{\mathbf{e}}_j\| \leq \|\hat{\mathbf{e}}_i - \mathbf{c}_{y_i}\| + \|\hat{\mathbf{e}}_j - \mathbf{c}_{y_i}\|; \quad (16.18)$$

$$\|\hat{\mathbf{e}}_i - \hat{\mathbf{e}}_k\| \geq \|\hat{\mathbf{e}}_i - \mathbf{c}_{y_k}\| - \|\hat{\mathbf{e}}_k - \mathbf{c}_{y_k}\|. \quad (16.19)$$

Отсюда

$$\ell_t(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \leq \ell_u(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \triangleq \|\hat{\mathbf{e}}_i - \mathbf{c}_{y_i}\| - \|\hat{\mathbf{e}}_i - \mathbf{c}_{y_k}\| + \|\hat{\mathbf{e}}_j - \mathbf{c}_{y_i}\| + \|\hat{\mathbf{e}}_k - \mathbf{c}_{y_k}\|. \quad (16.20)$$

Мы можем использовать этот факт для вывода практически вычислимой верхней границы триплетной потери следующим образом:

$$\mathcal{L}_t(\mathcal{D}, \mathcal{S}) = \sum_{\substack{(i,j) \in \mathcal{S}, (i,k) \notin \mathcal{S}, \\ i, j, k \in \{1, \dots, N\}}} \ell_t(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \leq \sum_{\substack{(i,j) \in \mathcal{S}, (i,k) \notin \mathcal{S}, \\ i, j, k \in \{1, \dots, N\}}} \ell_u(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \quad (16.21)$$

$$= C' \sum_{i=1}^N \left(\|\mathbf{x}_i - \mathbf{c}_{y_i}\| - \frac{1}{3(C-1)} \sum_{m=1, m \neq y_i}^C \|\mathbf{x}_i - \mathbf{c}_m\| \right) = \mathcal{L}_u(\mathcal{D}, \mathcal{S}), \quad (16.22)$$

где $C' = 3(C-1) \left(\frac{N}{C} - 1 \right) \frac{N}{C}$ – постоянная. Ясно, что \mathcal{L}_u можно вычислить за время $O(NC)$, см. иллюстрацию на рис. 16.6b.

В работе [Do+19] показано, что $0 \leq \mathcal{L}_t - \mathcal{L}_u \leq \frac{N^3}{C^2} K$, где K – постоянная, зависящая от разброса центроидов. Чтобы граница была точной, центроиды должны находиться как можно дальше друг от друга, а расстояния между ними должны быть максимально похожи. Простой способ обеспечить выполнение этих условий – определить унитарные векторы \mathbf{c}_m , по одному для каждого класса. Эти векторы уже имеют единичную норму и взаимно ортогональны. Расстояние между каждой парой центроидов равно $\sqrt{2}$, так что верхняя оценка довольно точная.

Недостаток этого подхода в том, что размерность слоя погружения предполагается равной $L = C$. Решить эту проблему можно двумя способами. Во-первых, после обучения можно добавить слой линейного проецирования, который отобразит C в $L \neq C$, или же можно взять предпоследний слой сети погружения. Второй подход – выбрать много точек на L -мерной единичной гиперсфере (для чего достаточно произвести выборку из стандартного нормального распределения, а затем нормировать ее [Mar72]), после чего выполнить кластеризацию методом K средних (раздел 21.3) с $K = C$. В экспериментах, описанных в работе [Do+19], оба подхода дали схожие результаты.

Интересно, что в работе [Rot+20] показано, что увеличение $\pi_{\text{intra}}/\pi_{\text{inter}}$ приводит к повышению качества решения различных последующих задач поиска. Здесь

$$\pi_{\text{intra}} = \frac{1}{Z_{\text{intra}}} \sum_{c=1}^C \sum_{i \neq l: y_i = y_l = c} d(\mathbf{x}_i, \mathbf{x}_l) \quad (16.23)$$

– среднее внутриклассовое расстояние, а

$$\pi_{\text{inter}} = \frac{1}{Z_{\text{inter}}} \sum_{c=1}^C \sum_{c'=1}^C d(\boldsymbol{\mu}_c, \boldsymbol{\mu}_{c'}) \quad (16.24)$$

– среднее межклассовое расстояние, где $\mu_c = (1/Z_c) \sum_{i: y_i=c} \hat{e}_i$ – среднее погружение для примеров из класса c . Это наводит на мысль, что нужно не только разносить центроиды подальше друг от друга (чтобы максимизировать числитель), но и не позволять примерам находиться слишком близко к своим центроидам (чтобы минимизировать знаменатель); этот последний член в методе из работы [Do+19] не отражается.

16.2.6. Другие приемы глубокого обучения метрики

Помимо методов ускорения, описанных в разделе 16.2.5, есть много других деталей, о которых нужно позаботиться, чтобы обеспечить приемлемое качество DML. Многие из них обсуждаются в работах [MBL20; Rot+20]. Здесь мы лишь кратко упомянем о некоторых.

Важный вопрос – как создаются мини-пакеты. В задачах классификации (по крайней мере, со сбалансированными классами) случайной выборки примеров из обучающего набора как правило достаточно. Но в случае DML нужно гарантировать, что для каждого примера в мини-пакете существуют другие похожие и непохожие на него примеры. Один из подходов такого рода – добыча твердых примеров (раздел 16.2.5.1). Другая идея – применить методы ядерных множеств (coreset) к ранее обученным погружениям и таким образом выбирать разнообразный пакет на каждом шаге [Sin+20]. Однако в работе [Rot+20] показано, что следующая простая стратегия также хорошо работает при создании каждого пакета: выбрать B/n классов, а затем случайным образом выбрать N_c примеров из каждого класса, где B – размер пакета, а $N_c = 2$ – настроенный параметр.

Еще одна важная тема – как избежать переобучения. Поскольку большинство наборов данных, упоминаемых в литературе по DML, малы, стало стандартной практикой использовать классификатор изображений, например GoogLeNet (раздел 14.3.3) или ResNet (раздел 14.3.4), предобученные на наборе данных ImageNet, а затем дообучить модели с применением DML-потери. (Подробнее о таком способе переноса обучения см. раздел 19.2.) Кроме того, принято использовать приращение данных (см. раздел 19.1). (На самом деле в некоторых методах самообучения приращение данных – единственный способ создать похожие пары.)

В работе [ZLZ20] предложено использовать **сферическое ограничение на погружение** (spherical embedding constraint – SEC); это дополнительный регуляризационный член на уровне пакета, который поощряет одинаковую норму всех примеров. То есть регуляризатор – просто эмпирическая дисперсия норм (ненормированных) погружений в данном пакете, см. иллюстрацию на рис. 16.7. Этот регуляризатор можно прибавить к любой из существующих DML-потерь, рассчитывая на умеренное улучшение скорости и устойчивости обучения, а равно окончательного качества – по аналогии с использованием пакетной нормировки (раздел 14.2.4.1).

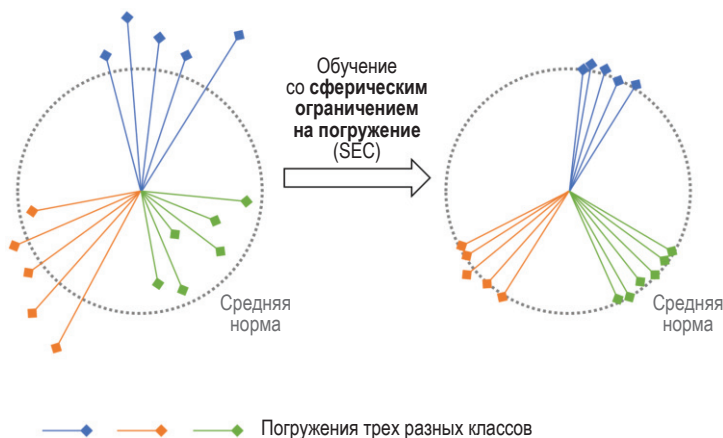


Рис. 16.7 ❖ Добавление сферического ограничения на погружение в метод глубокого обучения метрики. Печатается с разрешения Динай Чжана

16.3. ЯДЕРНЫЕ ОЦЕНКИ ПЛОТНОСТИ

В этом разделе мы рассмотрим одну из форм непараметрического оценивания плотности – **ядерную оценку плотности** (kernel density estimation – **KDE**). Это порождающая модель, потому что определяет распределение вероятностей $p(x)$, которое можно вычислять поточечно и из которого можно производить выборку для порождения новых данных.

16.3.1. Ядра плотности

Прежде чем переходить к объяснению KDE, нужно определить, что такое «ядро». В машинном обучении и статистике у этого термина есть несколько значений¹. В этом разделе мы будем иметь дело с так называемым **ядром плотности**. Это функция $\mathcal{K}: \mathbb{R} \rightarrow \mathbb{R}_+$ такая, что $\int \mathcal{K}(x) dx = 1$ и $\mathcal{K}(-x) = \mathcal{K}(x)$. Из последнего свойства симметрии следует, что

$$\int x \mathcal{K}(x - x_n) dx = x_n. \quad (16.25)$$

Простой пример дает **прямоугольное ядро** (boxcar kernel), представляющее собой равномерное распределение в единичном интервале с центром в начале координат:

$$\mathcal{K}(x) \triangleq 0.5 \mathbb{I}(|x| \leq 1). \quad (16.26)$$

¹ На эту тему есть хорошая статья в блоге по адресу <https://francisbach.com/cursed-kernels/>.

Другой пример – **гауссово ядро**:

$$\mathcal{K}(x) = \frac{1}{(2\pi)^{\frac{1}{2}}} e^{-x^2/2}. \quad (16.27)$$

Ширину ядра можно регулировать с помощью параметра **полосы пропускания** h :

$$\mathcal{K}_h(x) \triangleq \frac{1}{h} \mathcal{K}\left(\frac{x}{h}\right). \quad (16.28)$$

Для обобщения на векторные входные данные можно определить ядро в виде **радиально-базисной функции** (radial basis function – **RBF**):

$$\mathcal{K}_h(\mathbf{x})/\mathcal{K}_h(\|\mathbf{x}\|). \quad (16.29)$$

В случае гауссового ядра оно принимает вид:

$$\mathcal{K}_h(x) = \frac{1}{h^D (2\pi)^{D/2}} \prod_{d=1}^D \exp\left(-\frac{1}{2h^2} x_d^2\right). \quad (16.30)$$

Гауссовы ядра популярны, но у них неограниченный носитель. Некоторые альтернативные ядра с компактным носителем (вычислительно более быстрые) перечислены в табл. 16.1. Графики этих ядерных функций показаны на рис. 16.8.

Таблица 16.1. Перечень некоторых популярных одномерных нормированных ядер. Значение 1 в столбце «Компактное» означает, что функция отлична от нуля только в конечной области. Значение 1 в столбце «Гладкое» означает, что функция дифференцируема на всем своем носителе. Значение 1 в столбце «Границы» означает, что функция дифференцируема также на границах носителя

Название	Определение	Компактное	Гладкое	Границы
Гауссово	$\mathcal{K}(x) = (2\pi)^{-\frac{1}{2}} e^{-x^2/2}$	0	1	1
Прямоугольное	$\mathcal{K}(x) = \frac{1}{2} \mathbb{I}(x \leq 1)$	1	0	0
Епанечникова	$\mathcal{K}(x) = \frac{3}{4} (1 - x^2) \mathbb{I}(x \leq 1)$	1	1	0
Трехкубовое	$\mathcal{K}(x) = \frac{70}{81} (1 - x ^2)^3 \mathbb{I}(x \leq 1)$	1	1	1

16.3.2. Оконная оценка плотности Парцена

Чтобы объяснить, как ядра используются для определения непараметрической оценки плотности, вспомним модель гауссовой смеси из раздела 3.5.1. В предположении фиксированной сферической гауссовой ковариации и равномерной смеси весов имеем

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{K} \sum_{k=1}^K \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \sigma^2 \mathbf{I}). \quad (16.31)$$

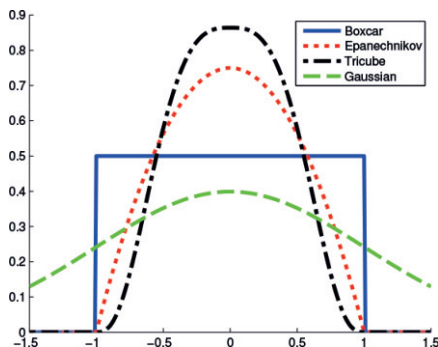


Рис. 16.8 ❖ Сравнение некоторых популярных нормированных ядер.
Построено программой по адресу probm1.ai/book1/16.8

Проблема заключается в том, что эта модель требует задания количества кластеров K , а также их местоположений $\boldsymbol{\mu}_k$. Вместо того чтобы оценивать эти параметры, мы можем сделать центром кластера каждую точку данных. В таком случае модель принимает вид:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \mathcal{N}(\mathbf{x}|\mathbf{x}_n, \sigma^2 \mathbf{I}). \quad (16.32)$$

Формулу (16.32) можно обобщить, написав

$$p(\mathbf{x}|\mathcal{D}) = \frac{1}{N} \sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n), \quad (16.33)$$

где \mathcal{K}_h – ядро плотности. Это называется **оконной оценкой плотности Парцена**, или **ядерной оценкой плотности (KDE)**.

Преимущество перед параметрической моделью в том, что не требуется никакого обучения (кроме выбора h , обсуждаемого в разделе 16.3.3) и нет нужды выбирать число центров кластеров. А недостаток в том, что модель потребляет очень много памяти (необходимо хранить все данные) и много времени для обчета.

На рис. 16.9 показана KDE в одномерном случае для двух разных ядер. В верхнем ряду мы видим прямоугольное ядро; результирующая модель просто подсчитывает, сколько точек оказалось в интервале длины h вокруг каждой точки \mathbf{x}_n , так что получается кусочно-постоянная плотность. В нижнем ряду показано гауссово ядро, при котором плотность получается более гладкой.

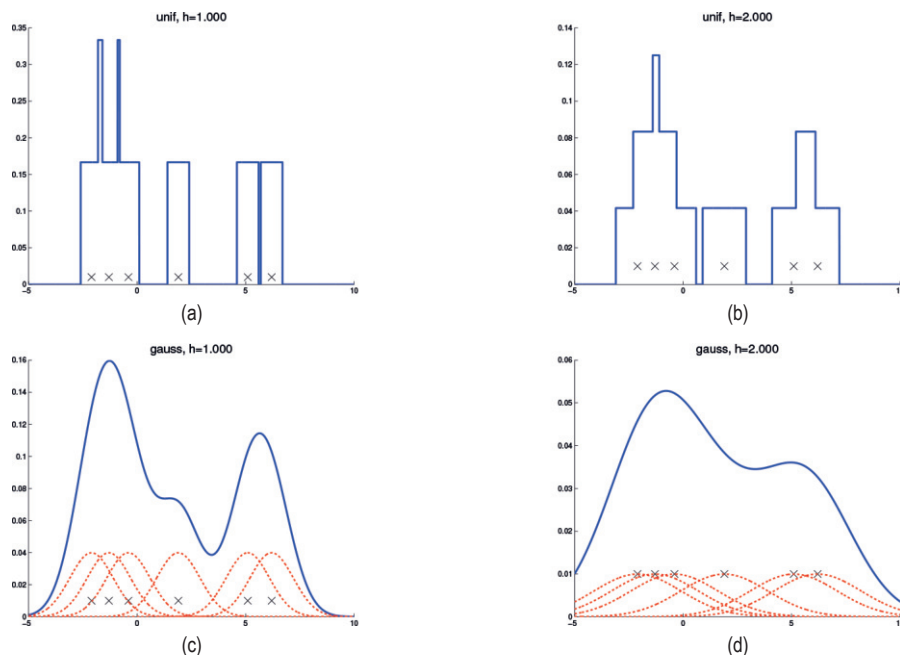


Рис. 16.9 ❖ Непараметрическая одномерная оценка плотности (Парцена) по 6 точкам, обозначенным крестиками. Верхний ряд: равномерное ядро. Нижний ряд: гауссово ядро. Левый столбец: полоса пропускания $h = 1$. Правый столбец: полоса пропускания $h = 2$. На основе статьи http://en.wikipedia.org/wiki/Kernel_density_estimation. Построено программой по адресу figures.probml.ai/book1/16.9

16.3.3. Как выбирать полосу пропускания

На рис. 16.9 мы видим, что полоса пропускания h оказывает большое влияние на обученное распределение. Можно считать, что это способ управления сложностью модели.

В случае одномерных данных, когда «истинные» данные, порождающие распределение, предполагаются гауссовыми, можно показать [BA97a], что оптимальная полоса пропускания для гауссова ядра (с точки зрения минимизации частотного риска) равна $h = \sigma(4/3N)^{1/5}$. Робастную аппроксимацию стандартного отклонения можно найти, вычислив сначала **медианное абсолютное отклонение** (MAD), $\text{median}(|x - \text{median}(x)|)$, а затем воспользовавшись тем, что $\hat{\sigma} = 1.4826 \text{ MAD}$. В D -мерном случае можно оценить h_d отдельно для каждого измерения, а затем положить $h = (\prod_{d=1}^D h_d)^{1/D}$.

16.3.4. От KDE к KNN-классификации

В разделе 16.1 мы обсуждали классификатор по K ближайшим соседям как эвристический подход к классификации. Интересно, что его можно вывести как порождающий классификатор, в котором условные плотности классов

$p(\mathbf{x}|y = c)$ моделируются с применением KDE. Вместо того чтобы использовать фиксированную полосу пропускания и подсчитывать, сколько точек данных попало в гиперкуб с центром в некоторой точке данных, мы разрешим разные полосы пропускания, или объемы для каждой точки. Именно, будем «надувать» куб вокруг точки \mathbf{x} , пока в него не попадет K точек данных неважно каких классов. Это называется **баллонной оценкой ядерной плотности** [TS92]. Пусть результирующий куб имеет объем $V(\mathbf{x})$ (раньше он был равен h^D) и пусть в нем оказалось $N_c(\mathbf{x})$ примеров из класса c . Тогда условную плотность класса можно оценить следующим образом:

$$p(\mathbf{x}|y = c, \mathcal{D}) = \frac{N_c(\mathbf{x})}{N_c V(\mathbf{x})}, \quad (16.34)$$

где N_c – общее число примеров из класса c во всем наборе данных. Если в качестве априорного взять распределение классов $p(y = c) = N_c/N$, то апостериорное распределение будет иметь вид:

$$\begin{aligned} p(y = c|\mathbf{x}, \mathcal{D}) &= \frac{\frac{N_c(\mathbf{x})}{N_c V(\mathbf{x})} \frac{N_c}{N}}{\sum_{c'} \frac{N_{c'}(\mathbf{x})}{N_{c'} V(\mathbf{x})} \frac{N_{c'}}{N}} = \frac{N_c(\mathbf{x})}{\sum_{c'} N_{c'}(\mathbf{x})} = \frac{N_c(\mathbf{x})}{K} \\ &= \frac{1}{K} \sum_{n \in N_K(\mathbf{x}, \mathcal{D})} \mathbb{I}(y_n = c), \end{aligned} \quad (16.35)$$

где мы воспользовались тем фактом, что $\sum_c N_c(\mathbf{x}) = K$, поскольку мы выбирали куб так, чтобы вокруг каждой точки было K точек (без учета класса). Это то же самое, что формула (16.1).

16.3.5. Ядерная регрессия

KDE можно использовать не только для порождающих классификаторов (см. раздел 16.1), но и для порождающих моделей регрессии.

16.3.5.1. Оценка среднего Надарая–Ватсона

В случае регрессии наша цель – вычислить условное математическое ожидание:

$$\mathbb{E}[y|\mathbf{x}, \mathcal{D}] = \int y p(y|\mathbf{x}, \mathcal{D}) dy = \frac{\int y p(\mathbf{x}, y|\mathcal{D}) dy}{\int p(\mathbf{x}, y|\mathcal{D}) dy}. \quad (16.36)$$

Если в качестве $p(y, \mathbf{x}|\mathcal{D})$ взять многомерное нормальное распределение, то получится результат, эквивалентный линейной регрессии, как было показано в разделе 11.2.3.5. Однако предположение о гауссовости $p(y, \mathbf{x}|\mathcal{D})$ несколько ограничительно. Мы можем следующим образом воспользоваться KDE, чтобы более точно аппроксимировать совместную плотность $p(y, \mathbf{x}|\mathcal{D})$:

$$p(y, \mathbf{x}|\mathcal{D}) \approx \frac{1}{N} \sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n) \mathcal{K}_h(y - y_n). \quad (16.37)$$

Отсюда

$$\mathbb{E}[y|\mathbf{x}, \mathcal{D}] = \frac{\frac{1}{N} \sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n) \int y \mathcal{K}_h(y - y_n) dy}{\frac{1}{N} \sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n) \int \mathcal{K}_h(y - y_n) dy}. \quad (16.38)$$

Числитель можно упростить, воспользовавшись тем, что $\int y \mathcal{K}_h(y - y_n) dy = y_n$ (это следует из формулы (16.25)). Чтобы упростить знаменатель, заметим, что интеграл ядерной плотности равен 1, т. е.:

$$\mathbb{E}[y|\mathbf{x}, \mathcal{D}] = \frac{\sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n) y_n}{\sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n)} = \sum_{n=1}^N y_n w_n(\mathbf{x}); \quad (16.39)$$

$$w_n(\mathbf{x}) \triangleq \frac{\mathcal{K}_h(\mathbf{x} - \mathbf{x}_n)}{\sum_{n'=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_{n'})}. \quad (16.40)$$

Как видим, предсказания – это просто взвешенная сумма выходов в обучающих точках, а веса зависят от степени похожести \mathbf{x} на сохраненные обучающие точки. Этот метод называется **ядерной регрессией**, **ядерным сглаживанием** или моделью **Надарая–Ватсона** (Н-В). На рис. 16.10 приведен пример в случае использования гауссова ядра.

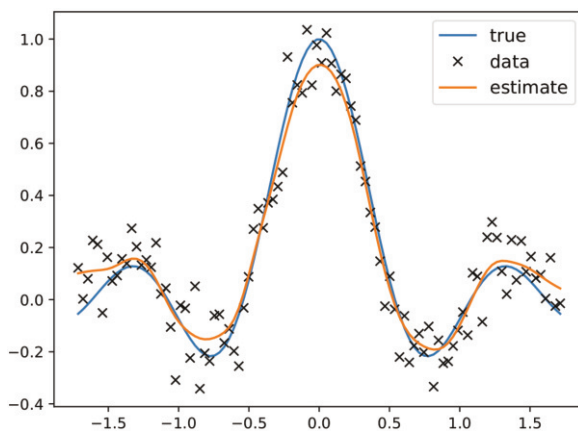


Рис. 16.10 ❖ Пример одномерной ядерной регрессии с использованием гауссова ядра.

Построено программой по адресу figures.probml.ai/book1/16.10

В разделе 17.2.3 мы обсудим связь между ядерной регрессией и регрессией на основе гауссовых процессов.

16.3.5.2. Оценка дисперсии

Иногда полезно вычислить прогнозную дисперсию, а не только прогнозное среднее. Это можно сделать, заметим, что

$$\mathbb{V}[y|\mathbf{x}, \mathcal{D}] = \mathbb{E}[y^2|\mathbf{x}, \mathcal{D}] - \mu(\mathbf{x})^2, \quad (16.41)$$

где $\mu(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}, \mathcal{D}]$ – оценка Н-В. Взяв гауссово ядро с дисперсией σ^2 , мы сможем вычислить $\mathbb{E}[y^2|\mathbf{x}, \mathcal{D}]$ следующим образом:

$$\mathbb{E}[y^2|\mathbf{x}, \mathcal{D}] = \frac{\sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n) \int y^2 \mathcal{K}_h(y - y_n) dy}{\sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n) \int \mathcal{K}_h(y - y_n) dy} \quad (16.42)$$

$$= \frac{\sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n) (\sigma^2 + y_n^2)}{\sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n)}, \quad (16.43)$$

где мы воспользовались тем фактом, что

$$\int y^2 \mathcal{N}(y|y_n, \sigma^2) dy = \sigma^2 + y_n^2. \quad (16.44)$$

Объединив (16.43) с (16.41), получаем

$$\mathbb{V}[y|\mathbf{x}, \mathcal{D}] = \sigma^2 + \sum_{n=1}^N w_n(\mathbf{x}) y_n^2 - \mu(\mathbf{x})^2. \quad (16.45)$$

Это совпадает с формулой 8 из работы [BA10] (с точностью до первого члена σ^2).

16.3.5.3. Локально взвешенная регрессия

Опустив нормировочный член в формуле (16.39), получим

$$\mu(\mathbf{x}) = \sum_{n=1}^N y_n \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n). \quad (16.46)$$

Это просто взвешенная сумма наблюдаемых откликов, причем веса зависят от степени похожести тестового входа \mathbf{x} на обучающие точки \mathbf{x}_n .

Вместо того чтобы просто интерполировать хранимые отклики y_n , мы можем обучить локально линейную модель в окрестности каждой обучающей точки:

$$\mu(\mathbf{x}) = \min_{\boldsymbol{\beta}} \sum_{n=1}^N [y_n - \boldsymbol{\beta}^\top \boldsymbol{\phi}(\mathbf{x}_n)]^2 \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n), \quad (16.47)$$

где $\boldsymbol{\phi}(\mathbf{x}) = [1, \mathbf{x}]$. Это называется **локально линейной регрессией** (locally linear regression – LRR) или **сглаживанием локально-взвешенной диаграммы рассеяния** (locally-weighted scatterplot smoothing), употребляются также акронимы **LOWESS** или **LOESS** [CD88]. Она часто используется для аннотирования диаграмм рассеяния линиями локальных трендов.

Глава 17

Ядерные методы*

В этой главе мы рассмотрим непараметрические методы регрессии и классификации. Они не предполагают фиксированной параметрической формы прогнозной функции, а пытаются оценить самую функцию (а не ее параметры) непосредственно по данным. Основная идея заключается в том, что если мы наблюдаем функцию в фиксированном множестве N точек, а именно $y_n = f(\mathbf{x}_n)$, $n = 1 \dots N$, где f – неизвестная функция, то для предсказания значения функции в новой точке \mathbf{x}_* нужно просто посмотреть, насколько \mathbf{x}_* похожа на каждую из N обучающих точек, $\{\mathbf{x}_n\}$, и тогда можно будет предсказать $f(\mathbf{x}_*)$ в виде взвешенной линейной комбинации значений $\{f(\mathbf{x}_n)\}$. Таким образом, нам, возможно, придется «запомнить» весь обучающий набор $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}$, чтобы делать предсказания на этапе тестирования – мы не сможем «сжать» \mathcal{D} в вектор параметров фиксированного размера.

Веса, используемые для предсказания, определяются сходством между \mathbf{x}_* и каждым \mathbf{x}_n , а это сходство вычисляется с помощью специальной ядерной функции $\mathcal{K}(\mathbf{x}_n, \mathbf{x}_*) \geq 0$, о которой мы поговорим в разделе 17.1. Этот подход напоминает RBF-сети (раздел 13.6.1), только в качестве «якорей» используются сами точки \mathbf{x}_n , а не обученные центроиды μ_n .

В разделе 17.2 мы обсудим подход на основе гауссовых процессов, позволяющий использовать ядро для определения *априорного распределения функций*, которое можно обновлять с учетом имеющихся данных с целью получения *апостериорного распределения*. Альтернативно можно использовать одно и то же ядро в методе опорных векторов, который вычисляет оценку MAP функции (см. раздел 17.3).

17.1. Ядра МЕРСЕРА

Ключ к непараметрическим методам – способ кодирования априорных знаний о сходстве двух входных векторов. Если мы знаем, что \mathbf{x}_i похож на \mathbf{x}_j , то можем поощрить модель к тому, чтобы предсказанные выходы $f(\mathbf{x}_i)$ и $f(\mathbf{x}_j)$ тоже были похожи.

Для определения сходства введем понятие **ядерной функции**. Слово «ядро» имеет много значений в математике, например: ядра плотности (раз-

дел 16.3.1), переходные ядра марковской цепи (раздел 3.6.1.2) и сверточные ядра (раздел 14.1). Здесь мы рассматриваем **ядро Мерсера**, называемое также **положительно определенным ядром**. Это произвольная симметричная функция $\mathcal{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$ такая, что

$$\sum_{i=1}^N \sum_{j=1}^N \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) c_i c_j \geq 0 \quad (17.1)$$

для любого множества N (различных) точек $\mathbf{x}_i \in \mathcal{X}$ и любого набора чисел $c_i \in \mathbb{R}$. (Предполагается, что $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) > 0$, так что равенство в неравенстве выше достигается только тогда, когда $c_i = 0$ для всех i .)

Есть еще один способ интерпретировать это условие. Для заданного множества N точек определим **матрицу Грама** размера $N \times N$ и будем рассматривать ее как матрицу сходства:

$$\mathbf{K} = \begin{pmatrix} \mathcal{K}(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \mathcal{K}(\mathbf{x}_1, \mathbf{x}_N) \\ & \ddots & \\ \mathcal{K}(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \mathcal{K}(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}. \quad (17.2)$$

\mathcal{K} является ядром Мерсера тогда и только тогда, когда матрица Грама положительно определена для любого множества (различных) входов $\{\mathbf{x}_i\}_{i=1}^N$.

Для вещественных входов чаще всего используется **квадратично-экспоненциальное ядро** (SE-ядро), называемое также **экспоненциально-квадратичным**, **гауссовым** или **RBF-ядром**. Оно определяется следующим образом:

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right). \quad (17.3)$$

Здесь ℓ – линейный масштаб ядра, т. е. расстояние, на котором, как мы ожидаем, различия играют роль. Он также называется **полосой пропускания**. RBF-ядро измеряет сходство двух векторов в \mathbb{R}^D с помощью (масштабированного) евклидова расстояния. В разделе 17.1.2 мы обсудим еще несколько ядер.

В разделе 17.2 мы покажем, как использовать ядра для определения априорных и апостериорных распределений функций. Основная идея такова: если $\mathcal{K}(\mathbf{x}, \mathbf{x}')$ велико, т. е. входы похожи, то мы ожидаем, что будут похожи и выходы функции, т. е. $f(\mathbf{x}) \approx f(\mathbf{x}')$. Точнее, информация, полученная о $f(\mathbf{x})$ в процессе обучения, помогает предсказать $f(\mathbf{x}')$ для всех \mathbf{x}' , коррелирующих с \mathbf{x} , т. е. для тех, для которых $\mathcal{K}(\mathbf{x}, \mathbf{x}')$ велико.

В разделе 17.3 будет показано, как использовать ядра для обобщения евклидова расстояния, чтобы можно было использовать геометрические методы, например линейный дискриминантный анализ в пространстве неявных признаков, а не в пространстве входов.

17.1.1. Теорема Мерсера

Напомним (см. раздел 7.4), что любую положительно определенную матрицу \mathbf{K} можно представить в виде спектрального разложения $\mathbf{K} = \mathbf{U}^T \mathbf{\Lambda} \mathbf{U}$, где $\mathbf{\Lambda}$ – диагональная матрица, содержащая собственные значения $\lambda_i > 0$, и \mathbf{U} – матрица, составленная из собственных векторов. Теперь рассмотрим элемент (i, j) матрицы \mathbf{K} :

$$k_{ij} = (\mathbf{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:,i})^T (\mathbf{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:,j}), \quad (17.4)$$

где $\mathbf{U}_{:,i}$ – i -й столбец \mathbf{U} . Положив $\boldsymbol{\phi}(\mathbf{x}_i) = \mathbf{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:,i}$, мы сможем записать:

$$k_{ij} = \boldsymbol{\phi}(\mathbf{x}_i)^T \boldsymbol{\phi}(\mathbf{x}_j) = \sum_m \phi_m(\mathbf{x}_i) \phi_m(\mathbf{x}_j). \quad (17.5)$$

Таким образом, элементы ядерной матрицы можно найти, вычислив скалярное произведение некоторых векторов признаков, неявно определяемых собственными векторами ядерной матрицы. Эту идею можно обобщить, применив к ядерным функциям, а не только к ядерным матрицам; соответствующий результат называется **теоремой Мерсера**.

Например, рассмотрим **квадратичное ядро** $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle^2$. В двумерном случае имеем

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = (x_1 x'_1 + x_2 x'_2)^2 = x_1^2 (x'_1)^2 + 2x_1 x_2 x'_1 x'_2 + x_2^2 (x'_2)^2. \quad (17.6)$$

Это можно записать в виде $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}')$, если определить $\boldsymbol{\phi}(\mathbf{x}_1, \mathbf{x}_2) = [x_1^2, \sqrt{2}x_1 x_2, x_2^2] \in \mathbb{R}^3$. Таким образом, мы погружаем двумерные входы \mathbf{x} в трехмерное пространство признаков $\boldsymbol{\phi}(\mathbf{x})$.

Теперь рассмотрим RBF-ядро. В этом случае соответствующее представление признаков бесконечномерно (детали см. в разделе 17.2.9.3). Однако ядерные функции позволяют избежать работы с бесконечномерными векторами.

17.1.2. Некоторые популярные ядра Мерсера

В следующих разделах описаны некоторые популярные ядра Мерсера. Дополнительные сведения можно найти в работе [Wil14] и по адресу <https://www.cs.toronto.edu/~duvenaud/cookbook/>.

17.1.2.1. Стационарные ядра для вещественных векторов

Для вещественных входов, $\mathcal{X} = \mathbb{R}^D$, принято использовать **стационарные ядра**, т. е. функции вида $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathcal{K}(\|\mathbf{x} - \mathbf{x}'\|)$; это значит, что значение зависит только от разности входов. RBF-ядро стационарное. Другие примеры приводятся ниже.

Ядро ARD

RBF-ядро можно обобщить, заменив евклидово расстояние расстоянием Махаланобиса:

$$\mathcal{K}(\mathbf{r}) = \sigma^2 \exp\left(-\frac{1}{2} \mathbf{r}^\top \mathbf{\Sigma}^{-1} \mathbf{r}\right), \quad (17.7)$$

где $\mathbf{r} = \mathbf{x} - \mathbf{x}'$. Если матрица $\mathbf{\Sigma}$ диагональная, то эту формулу можно переписать в виде

$$\mathcal{K}(\mathbf{r}; \boldsymbol{\ell}, \sigma^2) = \sigma^2 \exp\left(-\frac{1}{2} \sum_{d=1}^D \frac{1}{\ell_d^2} r_d^2\right) = \prod_{d=1}^D \mathcal{K}(r_d; \ell_d, \sigma^{2/d}), \quad (17.8)$$

где

$$\mathcal{K}(r; \ell, \tau^2) = \tau^2 \exp\left(-\frac{1}{2} \frac{1}{\ell^2} r^2\right). \quad (17.9)$$

Величину σ^2 можно интерпретировать как полную дисперсию, а ℓ_d – как характеристический линейный масштаб в направлении измерения d . Если d – нерелевантное входное измерение, то можно положить $\ell_d = \infty$, тогда оно будет проигнорировано. Это называется **автоматическим определением релевантности** (automatic relevancy determination – ARD) (раздел 11.7.7). Поэтому соответствующее ядро называется **ядром ARD** (см. рис. 17.1, где показаны две двумерные функции, выбранные из гауссова процесса с применением такого априорного распределения).

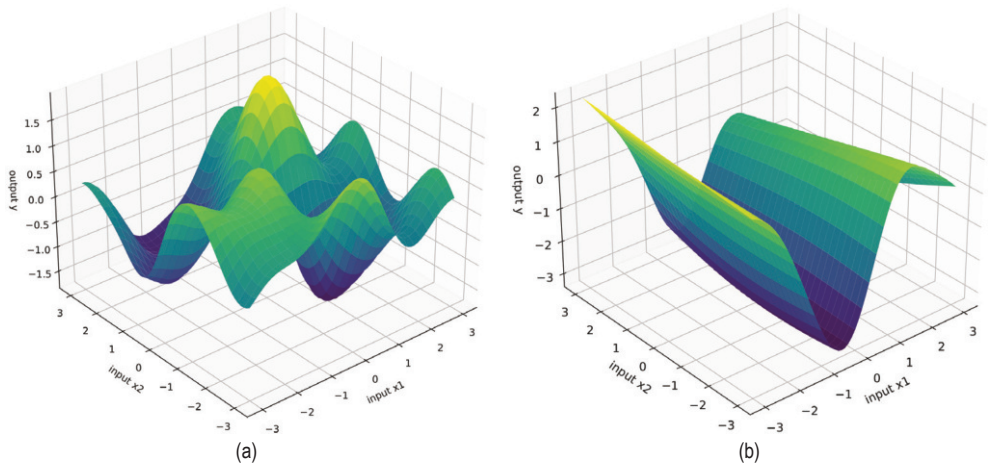


Рис. 17.1 ❖ Функции, выбранные из гауссова процесса, с применением ядра ARD. (a) $\ell_1 = \ell_2 = 1$. Оба измерения дают вклад в отклик. (b) $\ell_1 = 1, \ell_2 = 5$. Второе измерение практически игнорируется. На основе рис. 5.1 из работы [RW06]. Построено программой по адресу figures.probml.ai/book1/17.1

Ядра Матерна

Ядро SE дает бесконечно дифференцируемые, а значит, очень гладкие функции. Во многих приложениях предпочтительнее использовать ядро Матерна,

которое дает функции «погрубее», способные моделировать локальные «извивы», не прибегая к слишком мелкому линейному масштабу.

Ядро Матерна имеет вид:

$$\mathcal{K}(r; \nu, \ell) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}r}{\ell} \right)^\nu \mathcal{K}_\nu \left(\frac{\sqrt{2\nu}r}{\ell} \right), \quad (17.10)$$

где K_ν – модифицированная функция Бесселя, а ℓ – линейный масштаб. Функции, выбранные из этого гауссова процесса, являются k раз дифференцируемыми тогда и только тогда, когда $\nu > k$. При $\nu \rightarrow \infty$ это ядро стремится к ядру SE.

Для $\nu \in \{1/2, 3/2, 5/2\}$ функция упрощается следующим образом:

$$\mathcal{K}\left(r; \frac{1}{2}, \ell\right) = \exp\left(-\frac{r}{\ell}\right); \quad (17.11)$$

$$\mathcal{K}\left(r; \frac{3}{2}, \ell\right) = \left(1 + \frac{\sqrt{3}r}{\ell}\right) \exp\left(-\frac{\sqrt{3}r}{\ell}\right); \quad (17.12)$$

$$\mathcal{K}\left(r; \frac{5}{2}, \ell\right) = \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}r}{\ell}\right). \quad (17.13)$$

Значение $\nu = 1/2$ соответствует **процессу Орнштейна–Уленбека**, который описывает скорость частицы, совершающей броуновское движение. Соответствующая функция непрерывна, но не дифференцируема, а потому сильно «изломана» (см. иллюстрацию на рис. 17.2b).

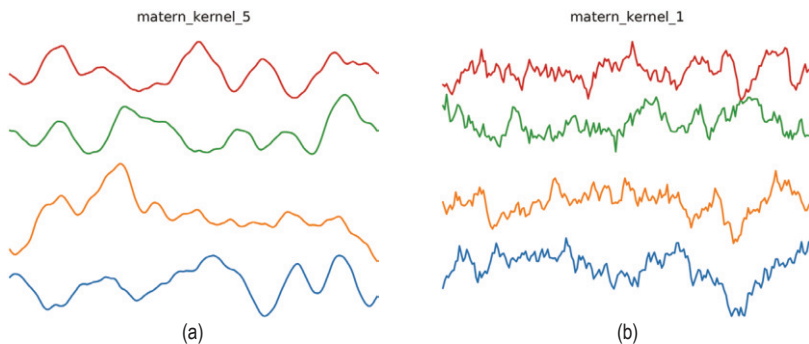


Рис. 17.2 ❖ Функции, выбранные из гауссова процесса, с применением ядра Матерна. (a) $\nu = 5/2$. (b) $\nu = 1/2$. Построено программой по адресу figures.probml.ai/book1/17.2

Периодические ядра

Периодическое ядро улавливает повторяющуюся структуру и имеет вид:

$$\mathcal{K}_{\text{per}}(r; \ell, p) = \exp\left(-\frac{2}{\ell^2} \sin^2\left(\pi \frac{r}{p}\right)\right), \quad (17.14)$$

где p – период (см. иллюстрацию на рис. 17.3a).

С ним тесно связано **косинусное ядро**:

$$\mathcal{K}(r; p) = \cos\left(2\pi \frac{r}{p}\right) \quad (17.15)$$

(см. иллюстрацию на рис. 17.3b).

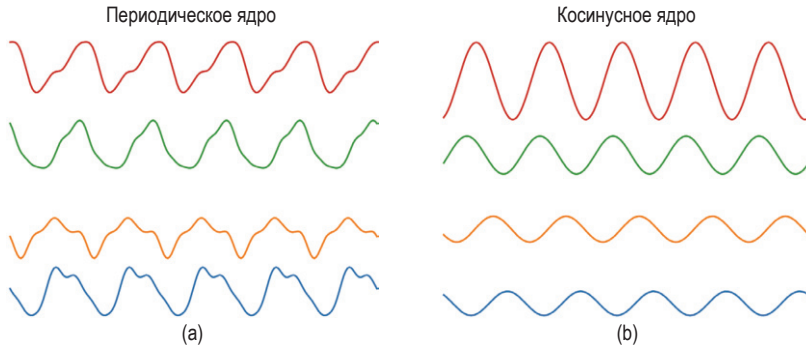


Рис. 17.3 ❖ Функции, выбранные из гауссова процесса, с применением стационарных периодических ядер. Построено программой по адресу figures.probl.ai/book1/17.3

17.1.2.2. Создание новых ядер из существующих

Если имеются два ядра $\mathcal{K}_1(\mathbf{x}, \mathbf{x}')$ и $\mathcal{K}_2(\mathbf{x}, \mathbf{x}')$, то можно создать новое ядро любым из следующих способов:

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = c\mathcal{K}_1(\mathbf{x}, \mathbf{x}') \text{ для любой постоянной } c > 0, \quad (17.16)$$

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})\mathcal{K}_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \text{ для любой функции } f, \quad (17.17)$$

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = q(\mathcal{K}_1(\mathbf{x}, \mathbf{x}')) \text{ для любого полинома } q \text{ с неотр. коэффициентами}, \quad (17.18)$$

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \exp(\mathcal{K}_1(\mathbf{x}, \mathbf{x}')), \quad (17.19)$$

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{A} \mathbf{x}' \text{ для любой положительно полуопределенной матрицы } \mathbf{A}. \quad (17.20)$$

Например, предположим, что имеется линейное ядро $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathbf{x}\mathbf{x}'$. Мы знаем, что это ядро Мерсера, потому что соответствующая матрица Грама не что иное, как масштабированная ковариационная матрица данных. Согласно приведенным выше правилам, полиномиальное ядро $\mathcal{K}(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{A} \mathbf{x}')^M$ – тоже ядро Мерсера. Оно содержит все одночлены порядка M . Например, если $M = 2$ и входные данные двумерные, то имеем

$$(\mathbf{x}^T \mathbf{x}')^2 = (x_1 x'_1 + x_2 x'_2)^2 = (x_1 x'_1)^2 + (x_2 x'_2)^2 + (x_1 x'_1)(x_2 x'_2). \quad (17.21)$$

Это утверждение можно обобщить на полином, содержащий все члены степени не выше M , взяв ядро $\mathcal{K}(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{A} \mathbf{x}' + c)^M$. Например, если $M = 2$ и входные данные двумерные, то имеем

$$\begin{aligned} (\mathbf{x}^T \mathbf{x}' + 1)^2 &= (x_1 x'_1)^2 + (x_1 x'_1)(x_2 x'_2) + (x_1 x'_1) \\ &\quad + (x_2 x'_2)(x_1 x'_1) + (x_2 x'_2)^2 + (x_2 x'_2) \\ &\quad + (x_1 x'_1) + (x_2 x'_2) + 1. \end{aligned} \quad (17.22)$$

Применив эти правила, можно также убедиться, что гауссово ядро является ядром Мерсера. Действительно, заметим, что

$$\|\mathbf{x} - \mathbf{x}'\|^2 = \mathbf{x}^T \mathbf{x} + (\mathbf{x}')^T \mathbf{x}' - 2\mathbf{x}^T \mathbf{x}', \quad (17.23)$$

и, следовательно,

$$\begin{aligned} \mathcal{K}(\mathbf{x}, \mathbf{x}') &= \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2) \\ &= \exp(-\mathbf{x}^T \mathbf{x} / 2\sigma^2) \exp(\mathbf{x}^T \mathbf{x}' / \sigma^2) \exp(-(\mathbf{x}')^T \mathbf{x}' / 2\sigma^2) \end{aligned} \quad (17.24)$$

является допустимым ядром.

17.1.2.3. Комбинирование ядер с помощью сложения и умножения

Ядра можно также складывать и умножать:

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathcal{K}_1(\mathbf{x}, \mathbf{x}') + \mathcal{K}_2(\mathbf{x}, \mathbf{x}'), \quad (17.25)$$

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathcal{K}_1(\mathbf{x}, \mathbf{x}') \times \mathcal{K}_2(\mathbf{x}, \mathbf{x}'). \quad (17.26)$$

Перемножение двух положительно определенных ядер всегда дает положительно определенное ядро. Таким образом, можно получить конъюнкцию индивидуальных свойств ядер, как показано на рис. 17.4.

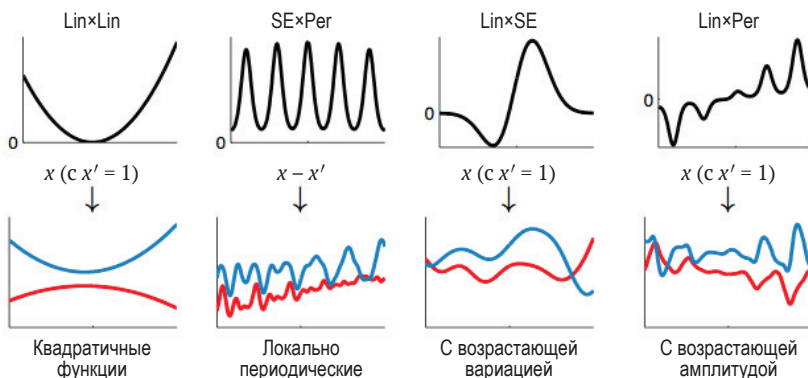


Рис. 17.4 ❖ Примеры одномерных структур, получаемых перемножением элементарных ядер. В верхнем ряду показаны ядра вида $\mathcal{K}(\mathbf{x}, \mathbf{x}') = 1$. В нижнем ряду показаны некоторые функции, выбранные из гауссова процесса $(f|0, \mathcal{K})$. На основе рис. 2.2 из работы [Duv14]. Печатается с разрешения Дэвида Дювенода

Сложение двух положительно определенных ядер тоже дает положительно определенное ядро. Таким образом, можно получить дизъюнкцию индивидуальных свойств ядер, как показано на рис. 17.5.

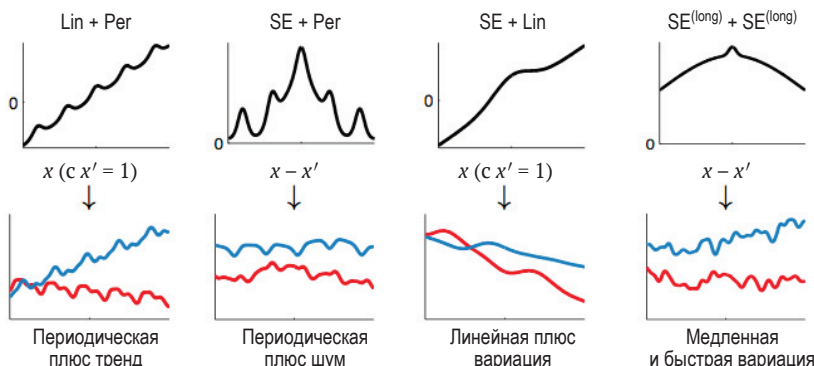


Рис. 17.5 ❖ Примеры одномерных структур, получаемых сложением элементарных ядер. Здесь $SE^{(short)}$ и $SE^{(long)}$ – два SE-ядра с разным линейным масштабом. На основе рис. 2.4 из работы [Duv14]. Печатается с разрешения Давида Дювенота

17.1.2.4. Ядра для структурированных входов

Ядра особенно полезны, когда входами являются структурированные объекты, например строки и графы, так как зачастую выделить признаки из входов переменного размера трудно. Например, можно определить **строковое ядро**, которое сравнивает строки в терминах количества общих n -грамм [Lod+02; BC17].

Можно также определить ядра на графах [KJM19]. Например, **ядро случайного блуждания** концептуально выполняет случайные блуждания одновременно на двух графах, а затем подсчитывает количество путей, порожденных в обоих блужданиях. Его можно вычислить эффективно, как показано в работе [Vis+10]. Дополнительные сведения о ядрах на графах см. в работе [KJM19].

Обзор ядер на структурированных объектах см., например, в работе [Gär03].

17.2. ГАУССОВЫ ПРОЦЕССЫ

В этом разделе мы обсудим **гауссовы процессы** (ГП), т. е. способ определить распределения функций вида $f : \mathcal{X} \rightarrow \mathbb{R}$, где \mathcal{X} – произвольное множество. Основное предположение заключается в том, что значения функции на множестве $M > 0$ входов, $\mathbf{f} = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_M)]$, имеют совместное гауссово распределение со средним ($\boldsymbol{\mu} = m(\mathbf{x}_1), \dots, m(\mathbf{x}_M)$) и ковариацией $\Sigma_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$, где m – средняя функция, а \mathcal{K} – положительно определенное ядро (ядро Мерсера). Поскольку мы предполагаем, что это справедливо для любого $M > 0$, значит, и для $M = N + 1$, когда множество содержит N обучающих точек \mathbf{x}_n и одну тестовую точку \mathbf{x}_* . Таким образом, мы можем вывести $f(\mathbf{x}_*)$ из $f(\mathbf{x}_1), \dots, f(\mathbf{x}_N)$, манипулируя совместным гауссовым распределением $p(f(\mathbf{x}_1), \dots, f(\mathbf{x}_N))$.

$f(\mathbf{x}_*)$), как будет объяснено ниже. Это можно также обобщить на случай, когда наблюдаются зашумленные функции $f(\mathbf{x}_n)$, как бывает, например, в задачах классификации или регрессии.

17.2.1. Незашумленные наблюдения

Предположим, что наблюдается обучающее множество $\mathcal{D} = \{(\mathbf{x}_n, y_n) : n = 1 \dots N\}$, где $y_n = f(\mathbf{x}_n)$ – незашумленное наблюдение функции, вычисленной в точке \mathbf{x}_n . Если мы просим ГП предсказать $f(\mathbf{x})$ для значения \mathbf{x} , которое он уже видел, то хотим, чтобы был возвращен ответ $f(\mathbf{x})$ без всякой неопределенности. Иными словами, ГП должен работать как **интерполятор** обучающих данных.

Теперь рассмотрим случай предсказания выходов для новых значений, отсутствующих в \mathcal{D} . Формально дано тестовое множество \mathbf{X}_* размера $N_* \times D$, и мы хотим предсказать выходы функции $\mathbf{f}_* = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_{N_*})]$. По определению ГП совместное распределение $p(\mathbf{f}_X, \mathbf{f}_* | \mathbf{X}, \mathbf{X}_*)$ имеет вид:

$$\begin{pmatrix} \mathbf{f}_X \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N} \left(\begin{pmatrix} \boldsymbol{\mu}_X \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K}_{X,X} & \mathbf{K}_{X,*} \\ \mathbf{K}_{X,*}^\top & \mathbf{K}_{*,*} \end{pmatrix} \right), \quad (17.27)$$

где $\boldsymbol{\mu}_X = [m(\mathbf{x}_1), \dots, m(\mathbf{x}_N)]$, $\boldsymbol{\mu}_* = [m(\mathbf{x}_1^*), \dots, m(\mathbf{x}_{N_*}^*)]$, $\mathbf{K}_{X,X} = \mathcal{K}(\mathbf{X}, \mathbf{X})$ – матрица размера $N \times N$, $\mathbf{K}_{X,*} = \mathcal{K}(\mathbf{X}, \mathbf{X}_*)$ – матрица размера $N \times N_*$, а $\mathbf{K}_{*,*} = \mathcal{K}(\mathbf{X}_*, \mathbf{X}_*)$ – матрица размера $N_* \times N_*$. См. иллюстрацию на рис. 17.6. По стандартным правилам вычисления условных гауссовых распределений (раздел 3.2.3) апостериорное распределение имеет вид:

$$p(\mathbf{f}_* | \mathbf{X}_*, \mathcal{D}) = \mathcal{N}(\mathbf{f}_* | \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*); \quad (17.28)$$

$$\boldsymbol{\mu}_* = m(\mathbf{X}_*) + \mathbf{K}_{X,*}^\top \mathbf{K}_{X,X}^{-1} (\mathbf{f}_X - m(\mathbf{X})); \quad (17.29)$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{*,*} - \mathbf{K}_{X,*}^\top \mathbf{K}_{X,X}^{-1} \mathbf{K}_{X,*}. \quad (17.30)$$

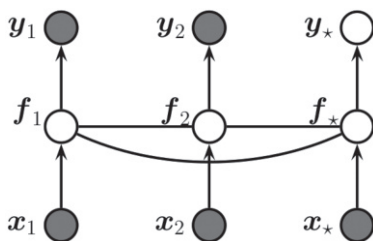


Рис. 17.6 ❖ Гауссов процесс для двух обучающих точек, \mathbf{x}_1 и \mathbf{x}_2 , и одной тестовой точки, \mathbf{x}_* , описываемый графовой моделью, представляющей распределение вероятностей $p(\mathbf{y}, \mathbf{f}_X | \mathbf{X}) = \mathcal{N}(\mathbf{f}_X | m(\mathbf{X}), \mathcal{K}(\mathbf{X})) \prod_i p(y_i | f_i)$. Скрытые вершины $f_i = f(\mathbf{x}_i)$ представляют значение функции в каждой точке данных. Эти скрытые вершины соединены между собой неориентированными ребрами, образующими гауссову графовую модель; сила ребер представляет члены ковариации $\Sigma_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$. Если тестовая точка \mathbf{x}_* похожа на обучающие точки \mathbf{x}_1 и \mathbf{x}_2 , то значение скрытой функции f_* будет похоже на f_1 и f_2 , а значит, предсказанный выход y_* будет похож на обучающие значения y_1 и y_2

Этот процесс показан на рис. 17.7. Слева мы видим выборку из априорного распределения $p(f)$, где используется RBF-ядро (раздел 17.1) и нулевая средняя функция. Справа же приведена выборка из апостериорного распределения $p(f|\mathcal{D})$. Как видим, модель идеально интерполирует обучающие данные, а неопределенность предсказания увеличивается по мере удаления от наблюдаемых данных.

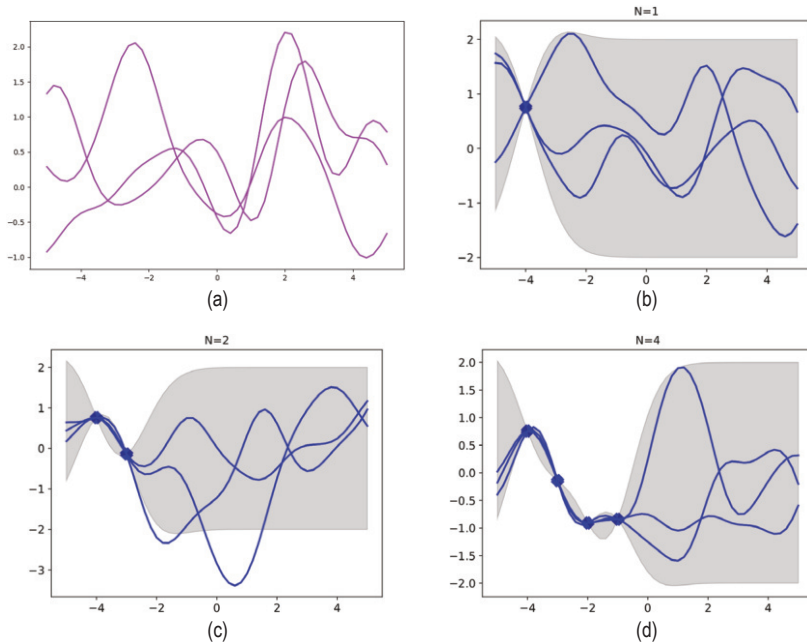


Рис. 17.7 ❖ (а) Функции, выбранные из априорного распределения ГП с квадратично-экспоненциальным ядром. (b–d). Функции, выбранные из апостериорного распределения ГП после обусловливания 1, 2 и 4 незашумленными наблюдениями. Закрашенная область представляет $\mathbb{E}[f(\mathbf{x})] \pm 2\text{std}[f(\mathbf{x})]$. На основе рис. 2.2 из работы [RW06]. Построено программой по адресу figures.probl.ai/book1/17.7

17.2.2. Зашумленные наблюдения

Теперь рассмотрим случай, когда наблюдается зашумленная версия истинной функции, $y_n = f(\mathbf{x}_n) + \varepsilon_n$, где $\varepsilon_n \sim \mathcal{N}(0, \sigma_y^2)$. В этом случае не требуется, чтобы модель интерполировала данные, но она должна давать результаты, «близкие» к наблюдаемым данным. Ковариация наблюдаемых зашумленных ответов равна

$$\text{Cov}[y_i, y_j] = \text{Cov}[f_i, f_j] + \text{Cov}[\varepsilon_i, \varepsilon_j] = K(\mathbf{x}_i, \mathbf{x}_j) + \sigma_y^2 \delta_{ij}, \quad (17.31)$$

где $\delta_{ij} = \mathbb{I}(i = j)$. Иными словами,

$$\text{Cov}[\mathbf{y}|\mathbf{X}] = \mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I}_N \triangleq \hat{\mathbf{K}}_{X,X}. \quad (17.32)$$

Совместная плотность наблюдаемых данных и латентной незашумленной функции на тестовых точках имеет вид:

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N} \left(\begin{pmatrix} \boldsymbol{\mu}_X \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \hat{\mathbf{K}}_{X,X} & \mathbf{K}_{X,*} \\ \mathbf{K}_{X,*}^\top & \mathbf{K}_{*,*} \end{pmatrix} \right). \quad (17.33)$$

Отсюда апостериорная прогнозная плотность на множестве тестовых точек \mathbf{X}_* равна

$$p(\mathbf{f}_* | \mathcal{D}, \mathbf{X}_*) = \mathcal{N}(\mathbf{f}_* | \boldsymbol{\mu}_{*|X}, \boldsymbol{\Sigma}_{*|X}), \quad (17.34)$$

$$\boldsymbol{\mu}_{*|X} = \boldsymbol{\mu}_* + \mathbf{K}_{X,*}^\top \hat{\mathbf{K}}_{X,X}^{-1} (\mathbf{y} - \boldsymbol{\mu}_X), \quad (17.35)$$

$$\boldsymbol{\Sigma}_{*|X} = \mathbf{K}_{*,*} - \mathbf{K}_{X,*}^\top \hat{\mathbf{K}}_{X,X}^{-1} \mathbf{K}_{X,*}. \quad (17.36)$$

Если имеется только одна тестовая точка, то это выражение упрощается:

$$p(f_* | \mathcal{D}, \mathbf{x}_*) = \mathcal{N}(f_* | m_* + \mathbf{k}_*^\top \hat{\mathbf{K}}_{X,X}^{-1} (\mathbf{y} - \boldsymbol{\mu}_X), k_{**} - \mathbf{k}_*^\top \hat{\mathbf{K}}_{X,X}^{-1} \mathbf{k}_*), \quad (17.37)$$

где $\mathbf{k}_* = [\mathcal{K}(\mathbf{x}_*, \mathbf{x}_1), \dots, \mathcal{K}(\mathbf{x}_*, \mathbf{x}_N)]$ и $k_{**} = \mathcal{K}(\mathbf{x}_*, \mathbf{x}_*)$. Если средняя функция равна нулю, то апостериорное среднее можно переписать в виде

$$\boldsymbol{\mu}_{*|X} = \mathbf{k}_*^\top (\hat{\mathbf{K}}_{X,X}^{-1} \mathbf{y}) = \mathbf{k}_*^\top \mathbf{a} = \sum_{n=1}^N \mathcal{K}(\mathbf{x}_*, \mathbf{x}_n) \alpha_n. \quad (17.38)$$

Это совпадает с предсказаниями ядерной гребневой регрессии (17.108).

17.2.3. Сравнение с ядерной регрессией

В разделе 16.3.5 мы обсуждали ядерную регрессию – порождающий подход к регрессии, при котором мы аппроксимируем $p(y, \mathbf{x})$ с помощью ядерной оценки плотности. В частности, формула (16.39) дает

$$\mathbb{E}[y | \mathbf{x}, \mathcal{D}] = \frac{\sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n) y_n}{\sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n)} = \sum_{n=1}^N y_n w_n(\mathbf{x}); \quad (17.39)$$

$$w_n(\mathbf{x}) \triangleq \frac{\mathcal{K}_h(\mathbf{x} - \mathbf{x}_n)}{\sum_{n'=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_{n'})}. \quad (17.40)$$

Это очень похоже на формулу (17.38). Однако существует несколько важных отличий. Во-первых, в ГП используется положительное определенное ядро (Мерсера), а не ядро плотности; ядра Мерсера можно определить на структурированных объектах, например строках и графах, а с ядрами плотности такое проделать труднее. Во-вторых, ГП – это интерполятор (по крайней мере, при $\sigma^2 = 0$), поэтому $\mathbb{E}[y | \mathbf{x}_n, \mathcal{D}] = y_n$. Напротив, ядерная регрессия не является интерполятором (хотя и может быть сделана таковым путем

итеративной аппроксимации невязок, как в работе [K]16)). В-третьих, ГП – это байесовский метод, предполагающий, что для оценки гиперпараметров (ядра) максимизируется маргинальное правдоподобие, тогда как в случае ядерной регрессии мы должны использовать перекрестную проверку для оценки параметров ядра, например полосы пропускания. В-четвертых, вычисление весов w_n для ядерной регрессии занимает время $O(N)$, где $N = |\mathcal{D}|$, а вычисление весов α_n для ГП-регрессии – время $O(N^3)$ (хотя существуют приближенные методы, позволяющие сократить его до $O(NM^2)$, и мы обсудим их в разделе 17.2.9).

17.2.4. Пространство весов и пространство функций

В этом разделе мы покажем, что байесовская линейная регрессия является частным случаем ГП.

Рассмотрим модель линейной регрессии $y = f(\mathbf{x}) + \varepsilon$, где $f(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$ и $\varepsilon \sim \mathcal{N}(0, \sigma_y^2)$. Если взять гауссово априорное распределение $\mathcal{N}(\mathbf{w}|\mathbf{0}, \boldsymbol{\Sigma}_w)$, то апостериорное распределение имеет вид (см. вывод в разделе 11.7.2):

$$p(\mathbf{w}|\mathcal{D}) = \mathcal{N}(\mathbf{w} | \frac{1}{\sigma_y^2} \mathbf{A}^{-1} \boldsymbol{\Phi}^\top \mathbf{y}, \mathbf{A}^{-1}), \quad (17.41)$$

где $\boldsymbol{\Phi}$ – матрица плана размера $N \times D$ и

$$\mathbf{A} = \sigma_y^{-2} \boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \boldsymbol{\Sigma}_w^{-1}. \quad (17.42)$$

Поэтому апостериорное прогнозное распределение для $f_* = f(\mathbf{x}_*)$ имеет вид:

$$p(f_*|\mathcal{D}, \mathbf{x}_*) = \mathcal{N}(f_* | \frac{1}{\sigma_y^2} \boldsymbol{\phi}_*^\top \mathbf{A}^{-1} \boldsymbol{\Phi}^\top \mathbf{y}, \boldsymbol{\phi}_*^\top \mathbf{A}^{-1} \boldsymbol{\phi}_*), \quad (17.43)$$

где $\boldsymbol{\phi}_* = \boldsymbol{\phi}(\mathbf{x}_*)$. Это представление проблемы вывода и предсказания в **пространстве весов**.

Теперь мы покажем, что это эквивалентно предсказаниям, сделанным ГП с использованием ядра вида $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\Sigma}_w \boldsymbol{\phi}(\mathbf{x}')$. Чтобы убедиться в этом, положим $\mathbf{K} = \boldsymbol{\Phi}^\top \boldsymbol{\Sigma}_w \boldsymbol{\Phi}$, $\mathbf{k}_* = \boldsymbol{\Phi}^\top \boldsymbol{\Sigma}_w \boldsymbol{\phi}_*$ и $k_{**} = \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma}_w \boldsymbol{\phi}_*$. Пользуясь этими обозначениями и леммой об обращении матрицы, мы можем переписать (17.43) следующим образом:

$$p(f_*|\mathcal{D}, \mathbf{x}_*) = \mathcal{N}(f_* | \boldsymbol{\mu}_{*|X}, \boldsymbol{\Sigma}_{*|X}), \quad (17.44)$$

$$\boldsymbol{\mu}_{*|X} = \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma}_w \boldsymbol{\Phi}^\top (\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{y} = \mathbf{k}_*^\top \hat{\mathbf{K}}_{X,X}^{-1} \mathbf{y}, \quad (17.45)$$

$$\boldsymbol{\Sigma}_{*|X} = \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma}_w \boldsymbol{\phi}_* - \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma}_w \boldsymbol{\Phi}^\top (\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \boldsymbol{\Phi} \boldsymbol{\Sigma}_w \boldsymbol{\phi}_* = k_{**} - \mathbf{k}_*^\top \hat{\mathbf{K}}_{X,X}^{-1} \mathbf{k}_*, \quad (17.46)$$

что совпадает с формулой (17.37), если положить $m(\mathbf{x}) = 0$. (Отразить ненулевое среднее можно, прибавив к $\boldsymbol{\phi}(\mathbf{x})$ постоянный признак, равный 1.)

Таким образом, мы можем вывести ГП из байесовской линейной регрессии. Заметим, однако, что линейная регрессия предполагает, что $\phi(\mathbf{x})$ – вектор конечномерный, тогда как ГП позволяет работать непосредственно в терминах ядер, которые могут соответствовать бесконечномерным векторам признаков (см. раздел 17.1.1). То есть ГП работает в **пространстве функций**.

17.2.5. Численные проблемы

В этом разделе мы обсудим вычислительные и численные проблемы, возникающие при реализации описанных выше формул. Для простоты предположим, что априорное среднее равно нулю, т. е. $m(\mathbf{x}) = 0$.

Апостериорное прогнозное среднее равно $\mu_* = \mathbf{k}_*^T \hat{\mathbf{K}}_{X,X}^{-1} \mathbf{y}$. По причинам, связанным с численной устойчивостью, обращаться $\hat{\mathbf{K}}_{X,X}$ непосредственно нежелательно. Лучше вычислить разложение Холецки $\mathbf{K}_{X,X} = \mathbf{L}\mathbf{L}^T$, что занимает время $O(N^3)$. Затем мы вычисляем $\alpha = \mathbf{L}^T \setminus (\mathbf{L} \setminus \mathbf{y})$, где оператор \setminus обозначает обратный ход (раздел 7.7.1). При таком подходе вычисление апостериорного среднего для каждого тестового примера займет время $O(N)$, если использовать формулу:

$$\mu_* = \mathbf{k}_*^T \hat{\mathbf{K}}_{X,X}^{-1} \mathbf{y} = \mathbf{k}_*^T \mathbf{L}^{-T} (\mathbf{L}^{-1} \mathbf{y}) = \mathbf{k}_*^T \alpha. \quad (17.47)$$

Дисперсию можно вычислить за время $O(N^2)$ для каждого тестового примера по формуле

$$\sigma_*^2 = k_{**} - \mathbf{k}_*^T \mathbf{L}^{-T} \mathbf{L}^{-1} \mathbf{k}_* = k_{**} - \mathbf{v}^T \mathbf{v}, \quad (17.48)$$

где $\mathbf{v} = \mathbf{L} \setminus \mathbf{k}_*$.

Наконец, логарифмическое маргинальное правдоподобие (необходимое для обучения ядра, раздел 17.2.6) можно вычислить по формуле:

$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2} \mathbf{y}^T \alpha - \sum_{n=1}^N \log L_{nn} - \frac{N}{2} \log(2\pi). \quad (17.49)$$

17.2.6. Оценивание параметров ядра

У большинства ядер имеются свободные параметры, которые могут сильно влиять на предсказания модели. Например, предположим, что выполняется одномерная регрессия с помощью ГП с RBF-ядром вида

$$\mathcal{K}(x_p, x_q) = \sigma_f^2 \exp\left(-\frac{1}{2\ell^2} (x_p - x_q)^2\right). \quad (17.50)$$

Здесь ℓ – горизонтальный масштаб, на котором функция заметно изменяется, а σ_f^2 управляет вертикальным масштабом функции. Предположим, что шум наблюдений имеет дисперсию σ_y^2 .

Мы выбрали 20 наблюдений из многомерного нормального распределения с ковариацией $\Sigma = \mathcal{K}(x_i, x_j)$ для сетки, состоящей из точек $\{x_i\}$, и прибавили

шум со значением σ_y . Затем мы аппроксимировали эти данные гауссовыми процессами с одним и тем же ядром, но разными гиперпараметрами. На рис. 17.8 показано, как изменяется результат при варьировании этих параметров. На рис. 17.8a $(\ell, \sigma_f, \sigma_y) = (1, 1, 0.1)$ и аппроксимация хороша. На рис. 17.8b мы уменьшили линейный масштаб до $\ell = 0.3$ (другие параметры были оптимизированы с помощью техники максимального маргинального правдоподобия, обсуждаемой ниже); теперь функция стала более «извилистой». Кроме того, неопределенность растет быстрее, поскольку эффективное расстояние до обучающих точек увеличивается с большей скоростью. На рис. 17.8c линейный масштаб увеличен до $\ell = 3$, функция выглядит более гладкой.

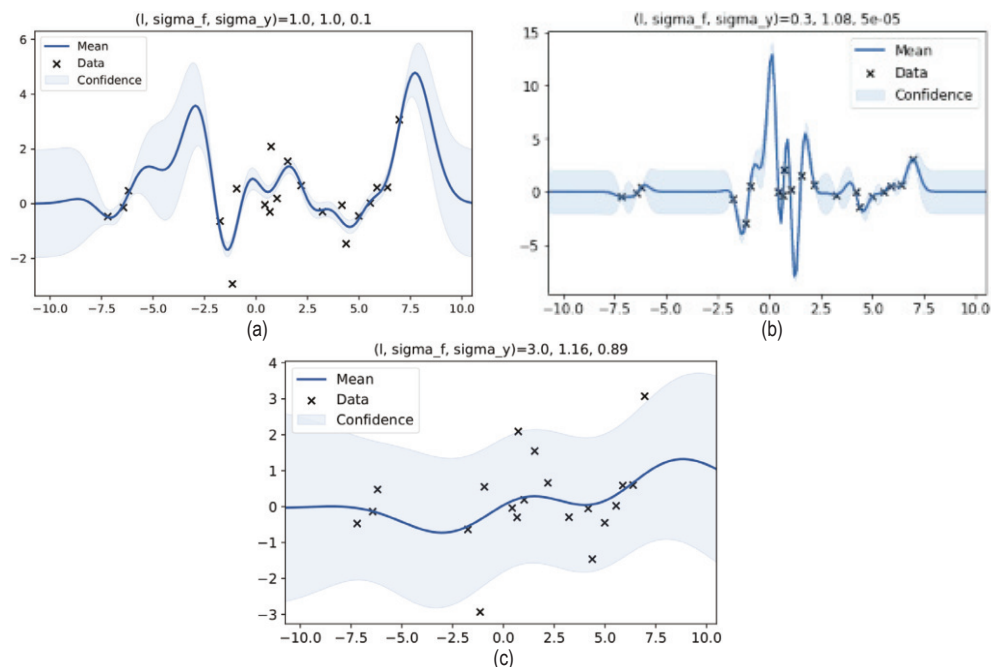


Рис. 17.8 ❖ Одномерные гауссовы процессы с ядрами SE, но разными гиперпараметрами, обученные на 20 зашумленных наблюдениях. Гиперпараметры $(\ell, \sigma_f, \sigma_y)$ следующие: (a) $(1, 1, 0.1)$; (b) $(0.3, 1.08, 5e-05)$; (c) $(3.0, 1.16, 0.89)$. На основе рис. 2.5 из работы [RW06]. Построено программой по адресу figures.probml.ai/book1/17.8

17.2.6.1. Эмпирическая байесовская оценка

Для оценивания параметров ядра θ (иногда называемых гиперпараметрами) можно было бы использовать исчерпывающий поиск на дискретной сетке значений с потерей на контрольном наборе в качестве целевой функции, но это может работать очень медленно. (Это подход, применяемый в невероятных методах, например SVM (раздел 17.3), для настройки ядер.) Здесь мы рассмотрим эмпирический байесовский подход к оцениванию, который

позволит воспользоваться гораздо более быстрыми градиентными методами оптимизации. Именно, мы будем максимизировать маргинальное правдоподобие:

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{X})p(\mathbf{f}|\mathbf{X}, \boldsymbol{\theta})d\mathbf{f}. \quad (17.51)$$

(Причина, по которой эта величина названа маргинальным, а не просто правдоподобием, связана с тем, что мы интегрируем по латентному гауссовому вектору \mathbf{f} , тем самым исключая его.)

Для простоты предположим, что средняя функция равна 0. Так как $p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$ и $p(\mathbf{y}|\mathbf{f}) = \prod_{n=1}^N \mathcal{N}(y_n|f_n, \sigma_y^2)$, маргинальное правдоподобие равно

$$\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \log \mathcal{N}(\mathbf{y}|\mathbf{0}, \hat{\mathbf{K}}_{X,X}) = -\frac{1}{2}\mathbf{y}^\top \hat{\mathbf{K}}_{X,X}^{-1} \mathbf{y} - \frac{1}{2} \log |\hat{\mathbf{K}}_{X,X}| - \frac{N}{2} \log(2\pi), \quad (17.52)$$

где зависимость $\hat{\mathbf{K}}_{X,X} = \mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I}_N$ от $\boldsymbol{\theta}$ неявная. Первое слагаемое – член аппроксимации данных, второе – член сложности модели, а третье – просто постоянная. Чтобы понять компромисс между первыми двумя членами, рассмотрим ядро SE в одномерном случае, когда линейный масштаб ℓ изменяется, а σ_y^2 фиксировано. Для малых масштабов аппроксимация будет хорошей, поэтому величина $\mathbf{y}^\top \hat{\mathbf{K}}_{X,X}^{-1} \mathbf{y}$ будет мала. Однако сложность модели будет высокой: \mathbf{K} будет почти диагональной (как на правом верхнем рис. 13.22), поскольку будет считаться, что большинство точек расположено «близко» к любой другой, так что член $\log |\hat{\mathbf{K}}_{X,X}|$ будет велик. Для большого линейного масштаба аппроксимация будет плохой, но сложность модели низкой: \mathbf{K} будет почти полностью состоять из единиц (как на правом нижнем рис. 13.22), так что член $\log |\hat{\mathbf{K}}_{X,X}|$ будет мал.

Теперь обсудим, как максимизировать маргинальное правдоподобие. Можно показать, что

$$\frac{\partial}{\partial \theta_j} \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \frac{1}{2} \mathbf{y}^\top \hat{\mathbf{K}}_{X,X}^{-1} \frac{\partial \hat{\mathbf{K}}_{X,X}}{\partial \theta_j} \hat{\mathbf{K}}_{X,X}^{-1} \mathbf{y} - \frac{1}{2} \text{tr} \left(\hat{\mathbf{K}}_{X,X}^{-1} \frac{\partial \hat{\mathbf{K}}_{X,X}}{\partial \theta_j} \right) \quad (17.53)$$

$$= \frac{1}{2} \text{tr} \left((\boldsymbol{\alpha} \boldsymbol{\alpha}^\top - \hat{\mathbf{K}}_{X,X}^{-1} \frac{\partial \hat{\mathbf{K}}_{X,X}}{\partial \theta_j}) \right), \quad (17.54)$$

где $\boldsymbol{\alpha} = \hat{\mathbf{K}}_{X,X}^{-1} \mathbf{y}$. Для вычисления $\hat{\mathbf{K}}_{X,X}^{-1}$ требуется время $O(N^3)$, а затем еще $O(N^2)$ на каждый гиперпараметр для вычисления градиента.

Форма частной производной $\partial \hat{\mathbf{K}}_{X,X} / \partial \theta_j$ зависит от формы ядра и параметра, по которому берется производная. Часто на гиперпараметры накладываются ограничения, например $\sigma_y^2 \geq 0$. В этом конкретном случае можно определить $\theta = \log(\sigma_y^2)$, а затем воспользоваться правилом дифференцирования сложной функции.

Имея выражения для логарифмического маргинального правдоподобия и его производной, мы можем оценить параметры ядра с помощью любого стандартного градиентного оптимизатора. Но, поскольку целевая функция невыпуклая, проблемой могут стать локальные минимумы, как будет показано ниже, поэтому, возможно, придется выполнить несколько перезапусков.

В качестве примера рассмотрим RBF-ядро в формуле (17.50) с $\sigma_f^2 = 1$. На рис. 17.9а показан график $\log p(\mathbf{y}|\mathbf{X}, \ell, \sigma_y^2)$ (где \mathbf{X} и \mathbf{y} – семь точек данных, показанных на рисунках (b) и (c), соответствующих изменению ℓ и σ_y^2). Два локальных оптимума обозначены знаками +. Левый нижний оптимум соответствует решению с низким шумом и малым линейным масштабом (показано на рисунке b). Правый верхний оптимум соответствует решению с высоким шумом и большим линейным масштабом (показано на рисунке c). При наличии всего семи точек данных недостаточно, чтобы уверенно решить, какое решение разумнее, хотя у более сложной модели (рисунок b) маргинальное правдоподобие примерно на 60 % выше, чем у более простой (рисунок c). При увеличении объема данных более сложная модель, вероятно, станет еще более предпочтительной.

На рис. 17.9 показаны другие интересные (и типичные) особенности. Область, в которой $\sigma_y^2 \approx 1$ (верхняя часть на рисунке а), соответствует случаю, когда шум очень высокий; в этом режиме маргинальное правдоподобие нечувствительно к линейному масштабу (о чем свидетельствуют почти горизонтальные линии уровня), поскольку все данные объясняются как шум. Область, в которой $\ell \approx 0.5$ (левая часть на рисунке а), соответствует случаю, когда линейный масштаб очень мал; в этом режиме маргинальное правдоподобие нечувствительно к шуму (о чем свидетельствуют почти вертикальные линии уровня), так как данные идеально интерполируются. Хороший оптимизатор не должен выбирать ни ту, ни другую область.

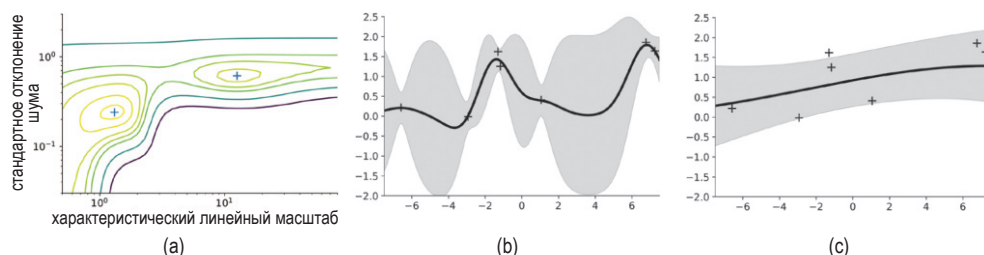


Рис. 17.9 ❖ Локальные минимумы на поверхности маргинального правдоподобия. (а) График зависимости логарифмического маргинального правдоподобия от линейного масштаба ядра ℓ и шума наблюдений σ_y для фиксированного уровня сигнала $\sigma_f = 1$, построенный по семи точкам, показанным на рис. b и c. (b) Функция, соответствующая левому нижнему локальному минимуму (ℓ, σ_y) $\approx (1, 0.2)$. Она очень «извилистая», а шум низкий. (c) Функция, соответствующая правому верхнему локальному минимуму (ℓ, σ_y) $\approx (10, 0.8)$. Очень «гладкая», шум высокий. Данные были сгенерированы при $(\ell, \sigma_f, \sigma_y) = (1, 1, 0.1)$. На основе рис. 5.5 из работы [RW06]. Построено программой по адресу figures.problm.ai/book1/17.9

17.2.6.2. Байесовский вывод

Если число точек невелико (например, когда гауссовы процессы используются для байесовской оптимизации), то использование точечной оценки параметров ядра может давать очень плохие результаты [Bul11; WF14]. В таких

случаях может быть желательно аппроксимировать апостериорное распределение параметров ядра. Можно использовать несколько методов. Например, в работе [MA10] показано, как использовать выборку по уровням, в работе [Hen+15] – как использовать гамильтонов метод Монте-Карло, а в работе [BBV11] – как использовать последовательный метод Монте-Карло.

17.2.7. Применение гауссовых процессов для классификации

До сих пор мы рассматривали применение ГП для регрессии с помощью гауссова правдоподобия. В этом случае апостериорное распределение – тоже гауссов процесс, и все вычисления можно выполнить аналитически. Но если правдоподобие не гауссово, а, например, правдоподобие Бернулли в случае бинарной классификации, то апостериорное распределение уже нельзя вычислить точно.

Существуют различные аппроксимации, некоторые из них обсуждаются во втором томе этой книги, [Mur22]. В этом разделе мы будем использовать гамильтонов метод Монте-Карло (раздел 4.6.8.4) для оценки как латентной гауссовой функции f , так и гиперпараметров ядра θ . Основная идея – задать отрицательное логарифмическое совместное распределение:

$$\begin{aligned} -\Psi(f, \theta) &= \log p(f, \theta | \mathbf{X}, \mathbf{y}) \\ &= \log \mathcal{N}(f | \mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X})) + \sum_{n=1}^N \log \text{Ber}(y_n | f_n(\mathbf{x}_n)) + \log p(\theta). \end{aligned} \quad (17.55)$$

Затем мы используем autograd, чтобы вычислить $\nabla_f \Psi(f, \theta)$ и $\nabla_\theta \Psi(f, \theta)$ и использовать эти градиенты как входы для вычисления гауссова вспомогательного распределения.

Рассмотрим одномерный пример из работы [Mar18]. Он похож на пример байесовской логистической регрессии на рис. 4.20, в котором цель – классифицировать виды ирисов Setosa и Versicolor, $y_n \in \{0, 1\}$, зная длину чашелистика x_n . Будем использовать ядро SE с линейным масштабом ℓ и считать, что ℓ имеет априорное распределение $\text{Ga}(2, 0.5)$.

На рис. 17.10а показаны результаты использования ядра SE. Они похожи на результаты линейной логистической регрессии (см. рис. 4.20) с тем отличием, что на краях (далеко от данных) кривая вероятности изгибается в сторону 0.5. Это объясняется тем, что априорная средняя функция $m(x) = 0$ и $\sigma(0) = 0.5$. Этот артефакт можно исключить, взяв более гибкую модель, в которой закодирована априорная информация: мы ожидаем, что выход будет монотонно возрастающей или монотонно убывающей функцией входа. Это можно сделать, воспользовавшись **линейным ядром**:

$$\mathcal{K}(x, x') = (x - c)(x' - c). \quad (17.56)$$

Мы можем масштабировать это выражение и прибавить к ядру SE, в результате чего получим

$$K(x, x') = \tau(x - c)(x' - c) + \exp\left[-\frac{(x - x')^2}{2\ell^2}\right]. \quad (17.57)$$

Результат показан на рис. 17.10b и выглядит более разумно.

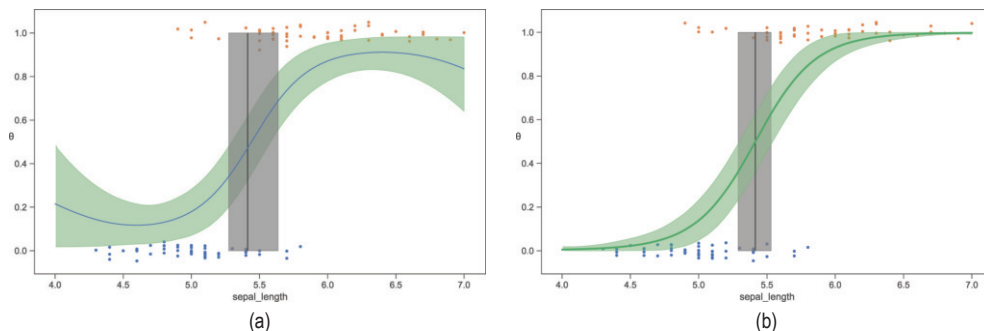


Рис. 17.10 ❖ ГП-классификатор для задачи бинарной классификации набора данных об ирисах (Setosa или Versicolor) по одному входному признаку (длина чашелистика). Толстая вертикальная область – байесовский доверительный интервал для решающей границы. (а) Ядро SE. (б) Ядро SE плюс линейное ядро. На основе рис. 7.11–7.12 из работы [Mar18]. Построено программой по адресу figures.problml.ai/book1/17.10

Возникает вопрос, зачем «заморачиваться» с ГП, если результат ничем не лучше, чем при использовании простой модели линейной логистической регрессии. Причина в том, что ГП – гораздо более гибкий подход, не требующий почти никаких априорных предположений, кроме гладкости. Например, предположим, что данные выглядят как на рис. 17.11а. В этом случае модель линейной логистической регрессии не смогла бы аппроксимировать данные. В принципе, можно было бы использовать нейронную сеть, но вряд ли она будет прилично работать, когда в обучающем наборе имеется всего-то 60 точек. Однако ГП хорошо приспособлены как раз для таких небольших наборов данных. На рис. 17.11b показаны результаты аппроксимации этих данных гауссовым процессом с ядром SE. Выглядит вполне разумно.

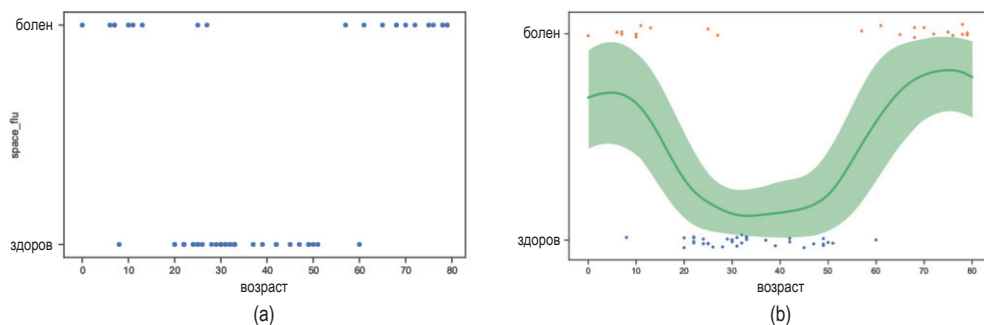


Рис. 17.11 ❖ (а) Фиктивная задача бинарной классификации «космической лихорадки». (б) Аппроксимация гауссовым процессом с ядром SE. На основе рис. 7.13–7.14 из работы [Mar18]. Построено программой по адресу figures.problml.ai/book1/17.11

17.2.8. Связи с глубоким обучением

Оказывается, что между ГП и глубокими нейронными сетями есть много интересных связей и сходных черт. Например, можно показать, что RBF-сеть с одним скрытым слоем, бесконечно широкая, эквивалентна ГП с RBF-ядром. (Это следует из того, что RBF-ядро можно выразить в виде скалярного произведения бесконечного числа признаков.) Вообще, многие виды ГНС можно интерпретировать как ГП с ядром, зависящим от архитектуры модели. Детали см. во втором томе этой книги, [Mur22].

17.2.9. Масштабирование ГП на большие наборы данных

Главный недостаток ГП (и других ядерных методов, в частности SVM, которые мы обсудим в разделе 17.3) заключается в том, что обращение ядерной матрицы размера $N \times N$ занимает время $O(N^3)$, что делает метод слишком медленным для больших наборов данных. Для ускорения ГП было предложено немало приближенных схем (см., например, обзор [Liu+18a]). В этом разделе мы кратко упомянем некоторые из них. Дополнительные сведения см. во втором томе этой книги, [Mur22].

17.2.9.1. Разреженные аппроксимации

Простой подход к ускорению вывода на основе ГП – использовать меньше данных. Более предпочтительное решение – попытаться «обобщить» N обучающих точек \mathbf{X} , построив $M \ll N$ **вспомогательных точек** (inducing point), или **псевдовходов** \mathbf{Z} . Это позволит заменить $p(f|\mathbf{f}_\mathbf{X})$ на $p(f|\mathbf{f}_\mathbf{Z})$, где $\mathbf{f}_\mathbf{X} = \{f(\mathbf{x}) : \mathbf{x} \in \mathbf{X}\}$ – вектор наблюдаемых значений функции в обучающих точках, а $\mathbf{f}_\mathbf{Z} = \{f(\mathbf{z}) : \mathbf{z} \in \mathbf{Z}\}$ – вектор оцененных значений функции во вспомогательных точках. Оптимизировав $(\mathbf{Z}, \mathbf{f}_\mathbf{Z})$, мы сможем обучиться «сжимать» обучающие данные $(\mathbf{X}, \mathbf{f}_\mathbf{X})$ в «дайджест» $(\mathbf{Z}, \mathbf{f}_\mathbf{Z})$ и тем самым сократить время вычислений с $O(N^3)$ до $O(M^3)$. Это называется **разреженным ГП**. Процедуре можно сделать более строгой, применив аппарат вариационного вывода. Дополнительные сведения см. во втором томе этой книги, [Mur22].

17.2.9.2. Распараллеливание с использованием структуры ядерной матрицы

Для вычисления разложения Холецки $\mathbf{K}_{\mathbf{X},\mathbf{X}}$ требуется время $O(N^3)$, необходимое для решения линейной системы $\hat{\mathbf{K}}_{\mathbf{X},\mathbf{X}} \alpha = \mathbf{y}$ и вычисления $|\mathbf{K}_{\mathbf{X},\mathbf{X}}|$, где $\hat{\mathbf{K}}_{\mathbf{X},\mathbf{X}} = \mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma^2 \mathbf{I}_N$. Альтернатива разложению Холецки – использовать методы линейной алгебры, часто называемые **методами крыловского типа** (или подпространства Крылова), основанные только на умножении матрицы на вектор. Эти подходы гораздо быстрее, поскольку могут воспользоваться естественной структурой ядерной матрицы. Но, даже если ядерная матрица не об-

ладает специальной структурой, операция умножения матриц тривиально распараллеливается, поэтому ее можно значительно ускорить, применив GPU, – в отличие от методов на основе разложения Холески, в большинстве своем последовательных. Эти соображения легли в основу популярного пакета **GPYtorch** [Gar+18]. Дополнительные сведения см. во втором томе этой книги, [Mur22].

17.2.9.3. Аппроксимация случайными признаками

Хотя эффективность ядер связана с их способностью избегать работы с представлениями входных данных признаками, ядерные методы требуют времени $O(N^3)$ на обращение матрицы Грама \mathbf{K} . Это может стать препятствием для их применения к большим наборам данных. По счастью, для многих (инвариантных относительно сдвигов) ядер карту признаков можно аппроксимировать, взяв случайно выбранное конечное множество M базисных функций и тем самым уменьшив временные затраты до $O(NM + M^3)$. Ниже мы кратко обсудим эту идею. Дополнительные сведения см., например, в работе [Liu+20].

Случайные признаки для RBF-ядра

Ограничимся случаем гауссова RBF-ядра. Можно показать, что

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') \approx \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}'), \quad (17.58)$$

где (вещественный) вектор признаков имеет вид:

$$\boldsymbol{\phi}(\mathbf{x}) \triangleq \frac{1}{\sqrt{T}} [(\sin(\boldsymbol{\omega}_1^\top \mathbf{x}), \dots, \sin(\boldsymbol{\omega}_T^\top \mathbf{x}), \cos(\boldsymbol{\omega}_1^\top \mathbf{x}), \dots, \cos(\boldsymbol{\omega}_T^\top \mathbf{x}))] \quad (17.59)$$

$$= \frac{1}{\sqrt{T}} [\sin(\boldsymbol{\Omega} \mathbf{x}), \cos(\boldsymbol{\Omega} \mathbf{x})], \quad (17.60)$$

где $T = M/2$, а $\boldsymbol{\Omega} \in \mathbb{R}^{T \times D}$ – случайная гауссова матрица с независимыми и одинаково распределенными элементами, выбранными из $\mathcal{N}(0, 1/\sigma^2)$, где σ – полосу пропускания ядра. Смещение аппроксимации уменьшается при увеличении M . На практике мы берем конечное M и вычисляем аппроксимацию математического ожидания методом Монте-Карло, выбирая одну случайную матрицу. Признаки в формуле (17.60) называются **случайными признаками Фурье** (random Fourier features – **RFF**) [RR08] или «взвешенными суммами случайной всячины» [RR09].

Можно также использовать положительные, а не тригонометрические случайные признаки, и в некоторых приложениях это предпочтительнее, например в моделях, где используется внимание (см. раздел 15.6.4). В частности, можно взять

$$\boldsymbol{\phi}(\mathbf{x}) \triangleq e^{-\|\mathbf{x}\|^2/2} \frac{1}{\sqrt{M}} [(\exp(\boldsymbol{\omega}_1^\top \mathbf{x}), \dots, \exp(\boldsymbol{\omega}_M^\top \mathbf{x}))], \quad (17.61)$$

где ω_m выбираются так же, как и раньше. Детали см. в работе [Cho+20b].

Независимо от того, используются тригонометрические или положительные признаки, можно получить более низкую оценку дисперсии, гарантировав, что строки \mathbf{Z} , хоть и случайные, но ортогональные; они называются **ортогональными случайными признаками**. Такую выборку можно эффективно произвести с помощью ортогонализации Грама-Шмидта неструктурированных гауссовых матриц [Yu+16] или нескольких еще более быстрых аппроксимаций (см. [CRW17; Cho+19]).

Фастфудная аппроксимация

К сожалению, для хранения случайной матрицы Ω требуется память объема $O(DM)$, а вычисление $\Omega\mathbf{x}$ занимает время $O(DM)$, где D — размерность входа, а M — число случайных признаков. Это может оказаться неприемлемым, если $M \gg D$, а так много признаков может понадобиться, чтобы получить какие-то выгоды по сравнению с использованием оригинального множества признаков. К счастью, можно использовать **быстрое преобразование Адамара**, позволяющее сократить объем памяти с $O(MD)$ до $O(M)$, а время — с $O(MD)$ до $O(M \log D)$. Этот подход был назван «**фастфудом**» [LSS13] — отсылка к оригинальному термину «kitchen sinks» (букв. «кухонная мойка»), обозначающему всякую всячину.

Машины экстремального обучения

Мы можем использовать аппроксимацию ядра случайными признаками, чтобы преобразовать ГП в линейную модель вида

$$f(\mathbf{x}; \theta) = \mathbf{W}\phi(\mathbf{x}) = \mathbf{W}h(\mathbf{Z}\mathbf{x}), \quad (17.62)$$

где $h(a) = \sqrt{1/M}[\sin(a), \cos(a)]$ для RBF-ядер. Это эквивалентно однослойному МСП со случайными (и фиксированными) весами связей между входным и скрытым слоем. При $M > N$ это соответствует перепараметризованной модели, которая идеально интерполирует обучающие данные.

В работе [Cur+17] этот метод применен для обучения модели логистической регрессии вида $f(\mathbf{x}, \theta) = \mathbf{W}^T h(\mathbf{Z}\mathbf{x}) + \mathbf{b}$ методом ГГС; авторы назвали свой метод **McKernel**. Мы можем также оптимизировать \mathbf{Z} , а не только \mathbf{W} (см. [Alb+17]), хотя тогда задача перестанет быть выпуклой.

Альтернативно можно использовать $M < N$, но составить вместе много таких случайных нелинейных слоев и просто оптимизировать выходные веса. Этот метод был назван **машиной экстремального обучения** (extreme learning machine – **ELM**) (см. [Hua14], хотя эта работа вызывает нарекания¹).

17.3. МЕТОД ОПОРНЫХ ВЕКТОРОВ

В этом разделе мы обсудим невероятностные предикторы для классификации и регрессии вида

¹ Нарекания вызваны тем, что автора Гуан-Бинь Хуана обвинили в отсутствии ссылок на предшествующие работы на ту же тему, в частности на эквивалентный подход, основанный на аппроксимациях ядра случайными признаками. Детали см. в статье https://en.wikipedia.org/wiki/Extreme_learning_machine#Controversy.

$$f(\mathbf{x}) = \sum_{i=1}^N \alpha_i \mathcal{K}(\mathbf{x}, \mathbf{x}_i). \quad (17.63)$$

Добавив подходящие ограничения, мы сможем гарантировать, что многие из коэффициентов α_i равны 0, так что предсказания на этапе тестирования будут зависеть только от подмножества обучающих точек. Эти имеющие значение точки называются **опорными векторами**, а сама модель – **методом опорных векторов** (support vector machine – **SVM**). Ниже приведен краткий обзор. Дополнительные сведения можно найти в работах [VGS97; SS01].

17.3.1. Классификаторы с широким зазором

Рассмотрим бинарный классификатор вида $h(\mathbf{x}) = \text{sign}(f(\mathbf{x}))$, где решающая граница определяется следующей линейной функцией:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0. \quad (17.64)$$

(В литературе по SVM принято использовать метки классов -1 и $+1$, а не 0 и 1 . Во избежание путаницы мы обозначаем такие целевые метки \tilde{y} , а не y .) Может существовать много прямых, разделяющих данные. Но интуитивно понятно, что хочется выбрать прямую с максимальным **зазором**, т. е. расстоянием до ближайшей к ней точки, поскольку такое решение будет самым робастным. Эта идея иллюстрируется на рис. 17.12: у решения слева зазор больше, чем у решения справа, и интуитивно оно лучше, так как менее чувствительно к возмущениям данных.

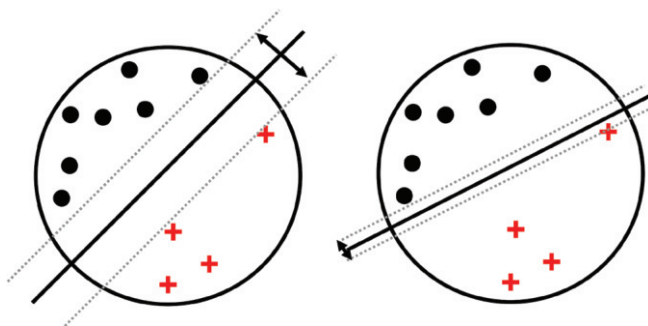


Рис. 17.12 ❖ Принцип большого зазора.

Слева: разделяющая гиперплоскость с широким зазором.

Справа: разделяющая гиперплоскость с малым зазором

Как вычислить такой **классификатор с широким зазором**? Сначала необходимо вывести выражение для расстояния от точки до решающей границы. Глядя на рис. 17.13а, мы видим, что

$$\mathbf{x} = \mathbf{x}_\perp + r \frac{\mathbf{w}}{\|\mathbf{w}\|}, \quad (17.65)$$

где r – расстояние от \mathbf{x} до решающей границы, определяемой нормальным вектором \mathbf{w} , а \mathbf{x}_\perp – ортогональная проекция \mathbf{x} на эту границу.

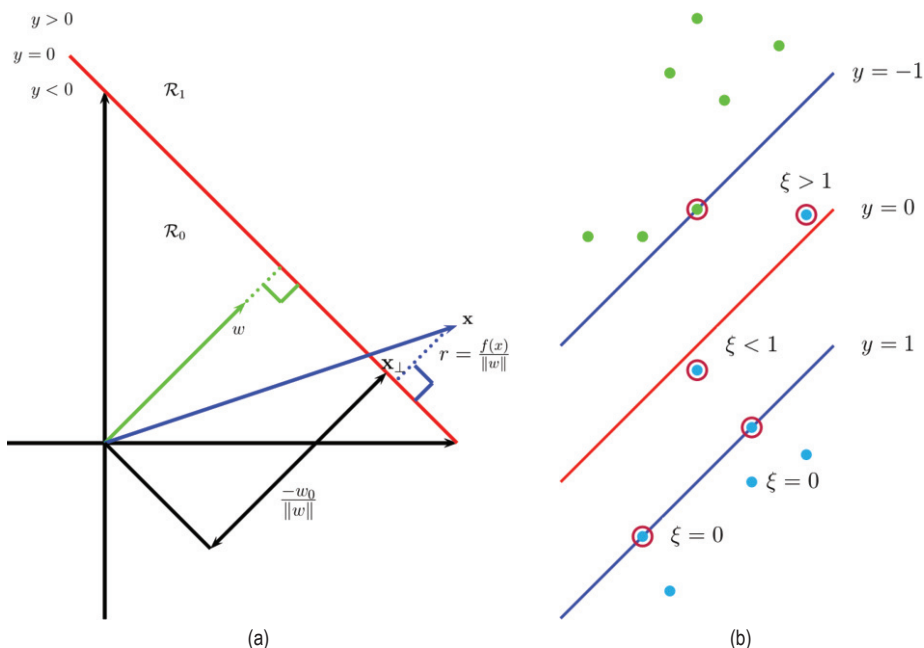


Рис. 17.13 ❖ (а) Геометрия линейной решающей границы в двумерном случае. Точка \mathbf{x} классифицируется как принадлежащая области \mathcal{R}_1 , если $f(\mathbf{x}) > 0$, и области \mathcal{R}_0 в противном случае; \mathbf{w} – вектор, перпендикулярный решающей границе. Член w_0 управляет расстоянием от решающей границы до начала координат, \mathbf{x}_\perp – ортогональная проекция \mathbf{x} на границу. Расстояние со знаком между \mathbf{x} и границей равно $f(\mathbf{x})/\|\mathbf{w}\|$. На основе рис. 4.1 из работы [Bis06]. (б) Точки, обведенные окружностями, – опорные векторы, с ними связаны двойственные переменные $\alpha_n > 0$. В случае мягкого зазора мы ассоциируем с каждым примером переменную невязки ξ_n . Если $0 < \xi_n < 1$, то точка находится внутри зазора, но по правильную сторону решающей границы. Если $\xi_n > 1$, то точка находится по неправильную сторону границы. На основе рис. 7.3 из работы [Bis06]

Мы хотели бы максимизировать величину r , поэтому должны выразить ее в виде функции от \mathbf{w} . Сначала отметим, что

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0 = (\mathbf{w}^\top \mathbf{x}_\perp + w_0) + r \frac{\mathbf{w}^\top \mathbf{w}}{\|\mathbf{w}\|} = (\mathbf{w}^\top \mathbf{x}_\perp + w_0) + r\|\mathbf{w}\|. \quad (17.66)$$

Поскольку $0 = f(\mathbf{x}_\perp) = \mathbf{w}^\top \mathbf{x}_\perp + w_0$, имеем $f(\mathbf{x}) = r\|\mathbf{w}\|$ и, следовательно, $r = f(\mathbf{x})/\|\mathbf{w}\|$.

Так как мы хотим, чтобы каждая точка находилась по правильную сторону от границы, необходимо также потребовать, чтобы $f(\mathbf{x}_n)\tilde{y}_n > 0$. Мы хотим максимизировать расстояние до ближайшей точки, поэтому окончательно целевая функция принимает вид:

$$\max_{\mathbf{w}, w_0} \frac{1}{\|\mathbf{w}\|} \min_{n=1}^N [\tilde{y}_n(\mathbf{w}^\top \mathbf{x}_n + w_0)]. \quad (17.67)$$

Отметим, что перемасштабирование параметров $\mathbf{w} \rightarrow k\mathbf{w}$ и $w_0 \rightarrow kw_0$ не изменяет расстояния от любой точки до границы, потому что множитель k сокращается при делении на $\|\mathbf{w}\|$. Поэтому определим масштабный коэффициент, так чтобы $\tilde{y}_n f_n = 1$ для точки, ближайшей к решающей границе. Таким образом, мы требуем, чтобы было $\tilde{y}_n f_n \geq 1$ для всех n . Наконец, отметим, что максимизация $1/\|\mathbf{w}\|$ эквивалентна минимизации $\|\mathbf{w}\|^2$. Поэтому мы получаем новую целевую функцию:

$$\max_{\mathbf{w}, w_0} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{при условии } \tilde{y}_n(\mathbf{w}^\top \mathbf{x}_n + w_0) \geq 1, n = 1 : N. \quad (17.68)$$

(Множитель $\frac{1}{2}$ добавлен для удобства и не влияет на оптимальные параметры.) Это ограничение означает, что мы хотим, чтобы все точки находились по правильную сторону границы с зазором не меньше 1.

Заметим, что важно масштабировать входные переменные до применения SVM, иначе зазор будет измерять расстояние от точки до границы, используя все входные измерения одинаков (см. иллюстрацию на рис. 17.14).

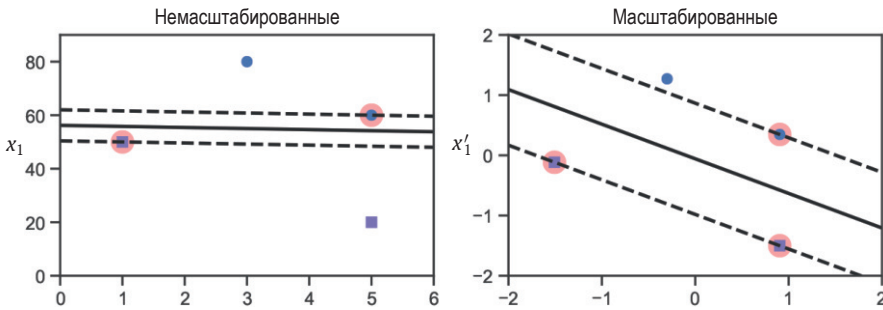


Рис. 17.14 ❖ Какие преимущества дает масштабирование входных признаков прежде вычисления классификатора с максимальным зазором. На основе рис. 5.2 из работы [Gér19]. Построено программой по адресу figures.problml.ai/book1/17.14

17.3.2. Двойственная задача

Целевая функция (17.68) – это стандартная задача квадратичного программирования (раздел 8.5.4), поскольку мы имеем квадратичную целевую функцию с линейными ограничениями. В ней $N + D + 1$ переменных и N ограничений. Это так называемая **основная задача**.

В случае выпуклой оптимизации для каждой основной задачи можно сформулировать **двойственную задачу**. Пусть $\alpha \in \mathbb{R}^N$ – двойственные переменные, соответствующие множителям Лагранжа, обеспечивающим выполнение N ограничений в виде неравенств. Обобщенный лагранжиан имеет вид (см. введение в выпуклую оптимизацию в разделе 8.5.2):

$$\mathcal{L}(\mathbf{w}, w_0, \alpha) = \frac{1}{2} \mathbf{w}^\top \mathbf{w} - \sum_{n=1}^N \alpha_n (\tilde{y}_n (\mathbf{w}^\top \mathbf{x}_n + w_0) - 1). \quad (17.69)$$

Чтобы оптимизировать его, мы должны найти стационарную точку, удовлетворяющую условиям:

$$(\hat{\mathbf{w}}, \hat{w}_0, \hat{\alpha}) = \min_{\mathbf{w}, w_0} \max_{\alpha} \mathcal{L}(\mathbf{w}, w_0, \alpha). \quad (17.70)$$

Это можно сделать, приравняв нулю частные производные по \mathbf{w} и w_0 . Имеем

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, w_0, \alpha) = \mathbf{w} - \sum_{n=1}^N \alpha_n \tilde{y}_n \mathbf{x}_n; \quad (17.71)$$

$$\frac{\partial}{\partial w_0} \mathcal{L}(\mathbf{w}, w_0, \alpha) = - \sum_{n=1}^N \alpha_n \tilde{y}_n, \quad (17.72)$$

откуда

$$\hat{\mathbf{w}} = \sum_{n=1}^N \hat{\alpha}_n \tilde{y}_n \mathbf{x}_n; \quad (17.73)$$

$$0 = \sum_{n=1}^N \hat{\alpha}_n \tilde{y}_n. \quad (17.74)$$

Подстановка этих выражений в лагранжиан дает

$$\mathcal{L}(\hat{\mathbf{w}}, \hat{w}_0, \alpha) = \frac{1}{2} \hat{\mathbf{w}}^\top \hat{\mathbf{w}} - \sum_{n=1}^N \alpha_n \tilde{y}_n \hat{\mathbf{w}}^\top \mathbf{x}_n - \sum_{n=1}^N \alpha_n \tilde{y}_n w_0 + \sum_{n=1}^N \alpha_n \quad (17.75)$$

$$= \frac{1}{2} \hat{\mathbf{w}}^\top \hat{\mathbf{w}} - \hat{\mathbf{w}}^\top \hat{\mathbf{w}} - 0 + \sum_{n=1}^N \alpha_n \quad (17.76)$$

$$= -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j \mathbf{x}_i^\top \mathbf{x}_j + \sum_{n=1}^N \alpha_n. \quad (17.77)$$

Эта форма целевой функции называется **двойственной**. Мы хотим минимизировать ее относительно α при условии $\sum_{n=1}^N \alpha_n \tilde{y}_n = 0$ и $0 \leq \alpha_n$ для $n = 1 \dots N$.

Приведенная выше целевая функция соответствует квадратичной задаче с N переменными. Стандартные алгоритмы ее решения занимают время $O(N^3)$. Однако были разработаны специальные алгоритмы именно для этой задачи, например **последовательная минимальная оптимизация** (sequential minimal optimization – **SMO**) [Pla98], занимающие время от $O(N)$ до $O(N^2)$.

Поскольку эта целевая функция выпукла, решение должно удовлетворять условиям Каруша–Куна–Таккера (раздел 8.5.2), утверждающим, что имеют место следующие свойства:

$$\alpha_n \geq 0; \quad (17.78)$$

$$\tilde{y}_n f(\mathbf{x}_n) - 1 \geq 0; \quad (17.79)$$

$$\alpha_n (\tilde{y}_n f(\mathbf{x}_n) - 1) = 0. \quad (17.80)$$

Отсюда либо $\alpha_n = 0$ (в этом случае n игнорируется при вычислении $\hat{\mathbf{w}}$), либо активно ограничение $\tilde{y}_n(\hat{\mathbf{w}}^\top \mathbf{x}_n + \hat{w}_0) = 1$. Последнее условие означает, что пример n лежит на решающей границе; эти точки, называемые **опорными векторами**, показаны на рис. 17.13b. Обозначим множество опорных векторов \mathcal{S} .

Для предсказания воспользуемся уравнением:

$$f(\mathbf{x}; \hat{\mathbf{w}}, \hat{w}_0) = \hat{\mathbf{w}}^\top \mathbf{x} + \hat{w}_0 = \sum_{n=1}^N \alpha_n \tilde{y}_n \mathbf{x}_n^\top \mathbf{x} + \hat{w}_0. \quad (17.81)$$

Чтобы решить его относительно $\hat{\mathbf{w}}_0$, заметим, что для любого опорного вектора имеет место равенство $\tilde{y}_n f(\mathbf{x}; \hat{\mathbf{w}}, \hat{w}_0) = 1$. Умножив обе части на \tilde{y}_n и воспользовавшись тем, что $\tilde{y}_n^2 = 1$, получаем $\hat{\mathbf{w}}_0 = \tilde{y}_n - \hat{\mathbf{w}}^\top \mathbf{x}_n$. На практике результаты можно улучшить, усреднив по всем опорным векторам:

$$f(\mathbf{x}; \hat{\mathbf{w}}, \hat{w}_0) = \hat{w}_0 = \frac{1}{|\mathcal{S}|} \sum_{n \in \mathcal{S}} (\tilde{y}_n - \hat{\mathbf{w}}^\top \mathbf{x}_n) = \frac{1}{|\mathcal{S}|} \sum_{n \in \mathcal{S}} \left(\tilde{y}_n - \sum_{m \in \mathcal{S}} \alpha_m \tilde{y}_m \mathbf{x}_m^\top \mathbf{x}_n \right). \quad (17.82)$$

17.3.3. Классификаторы с мягким зазором

Если данные линейно неразделимы, то не существует решения, для которого $\tilde{y}_n f_n \geq 1$ при всех n . Поэтому мы вводим **переменные невязки** $\xi_n \geq 0$ и заменяем жесткие ограничения $\tilde{y}_n f_n \geq 1$ **ограничениями мягкого зазора** $\tilde{y}_n f_n \geq 1 - \xi_n$. Новая целевая функция имеет вид:

$$\min_{\mathbf{w}, w_0, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N \xi_n \quad \text{при условиях } \xi_n \geq 0, \tilde{y}_n (\mathbf{x}_n^\top \mathbf{w} + w_0) \geq 1 - \xi_n, \quad (17.83)$$

где $C \geq 0$ – гиперпараметр, управляющий количеством точек, которым разрешено нарушить ограничение зазора. (Если $C = 1$, то мы возвращаемся к нерегуляризованному классификатору с жестким зазором.)

Лагранжиан для классификатора с мягким зазором принимает вид:

$$\begin{aligned} \mathcal{L}(\mathbf{w}, w_0, \alpha, \xi, \mu) = & \frac{1}{2} \mathbf{w}^\top \mathbf{w} + C \sum_{n=1}^N \xi_n \\ & - \sum_{n=1}^N \alpha_n (\tilde{y}_n (\mathbf{w}^\top \mathbf{x}_n + w_0) - 1 + \xi_n) - \sum_{n=1}^N \mu_n \xi_n, \end{aligned} \quad (17.84)$$

где $\alpha_n \geq 0$ и $\mu_n \geq 0$ – множители Лагранжа. Оптимизируя по \mathbf{w} , w_0 и ξ , получаем двойственную форму:

$$\mathcal{L}(\mathbf{a}) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j \mathbf{x}_i^\top \mathbf{x}_j, \quad (17.85)$$

Это совпадает со случаем жесткого зазора, однако ограничения различны. Именно, из условий Каруша–Куна–Таккера вытекает

$$0 \leq \alpha_n \leq C; \quad (17.86)$$

$$\sum_{n=1}^N \alpha_n \tilde{y}_n. \quad (17.87)$$

Если $\alpha_n = 0$, то точка игнорируется. Если $0 < \alpha_n < C$, то $\xi_n = 0$, т. е. точка лежит на границе зазора. Если $\alpha_n = C$, то точка лежит внутри зазора и может быть либо классифицирована правильно, если $\xi_n \leq 1$, или неправильно, если $\xi_n > 1$ (см. иллюстрацию на рис. 17.13b). Следовательно, $\sum_n \xi_n$ – верхняя граница количества неправильно классифицированных точек.

Как и раньше, член смещения можно вычислить по формуле:

$$\hat{w}_0 = \frac{1}{|\mathcal{M}|} \sum_{n \in \mathcal{M}} \left(\tilde{y}_n - \sum_{m \in \mathcal{S}} \alpha_m \tilde{y}_m \mathbf{x}_m^\top \mathbf{x}_n \right), \quad (17.88)$$

где \mathcal{M} – множество точек, для которых $0 < \alpha_n < C$.

Существует другая формулировка SVM с мягким зазором – так называемый **классификатор ν -SVM** [Sch+00]. В ней требуется максимизировать функцию

$$\mathcal{L}(\alpha) = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j \mathbf{x}_i^\top \mathbf{x}_j \quad (17.89)$$

при ограничениях:

$$0 \leq \alpha_n \leq 1 = N; \quad (17.90)$$

$$\sum_{n=1}^N \alpha_n \tilde{y}_n = 0; \quad (17.91)$$

$$\sum_{n=1}^M \alpha_n \geq \nu. \quad (17.92)$$

Преимущество такой формулировки в том, что параметр ν , заменивший C , можно интерпретировать как верхнюю границу доли **ошибка зазора** (точек, для которых $\xi_n > 0$) или как нижнюю границу количества опорных векторов.

17.3.4. Ядерный трюк

До сих пор мы преобразовывали задачу бинарной классификации с широким зазором в двойственную задачу с N неизвестными (α), решение которой в общем случае требует времени $O(N^3)$, а это слишком медленно. Однако принципиальное преимущество двойственной задачи заключается в том, что все операции скалярного произведения $\mathbf{x}^\top \mathbf{x}'$ можно заменить обращением к положительно определенному ядру (Мерсера), $\mathcal{K}(\mathbf{x}, \mathbf{x}')$. Это называется **ядерным трюком**.

Именно, функцию предсказания (17.81) можно переписать следующим образом:

$$f(\mathbf{x}) = \hat{\mathbf{w}}^T \mathbf{x} + \hat{w}_0 = \sum_{n \in \mathcal{S}} \alpha_n \tilde{y}_n \mathbf{x}_n^T \mathbf{x} + \hat{w}_0 = \sum_{n \in \mathcal{S}} \alpha_n \tilde{y}_n \mathcal{K}(\mathbf{x}_n, \mathbf{x}) + \hat{w}_0. \quad (17.93)$$

Необходимо также перенести в ядро член смещения. Это можно сделать, переписав формулу (17.82) в виде

$$\hat{w}_0 = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \left(\tilde{y}_i - \left(\sum_{j \in \mathcal{S}} \hat{\alpha}_j \tilde{y}_j \mathbf{x}_j \right)^T \mathbf{x}_i \right) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \left(\tilde{y}_i - \left(\sum_{j \in \mathcal{S}} \hat{\alpha}_j \tilde{y}_j \mathcal{K}(\mathbf{x}_j, \mathbf{x}_i) \right) \right). \quad (17.94)$$

Ядерный трюк позволяет, во-первых, избежать явного представления данных признаками, а во-вторых, без труда применять классификаторы к структурированным объектам, в частности строкам и графам.

17.3.5. Преобразование выходов SVM в вероятности

SVM-классификатор порождает метки $\hat{y}(\mathbf{x}) = \text{sign}(f(\mathbf{x}))$. Но часто мы хотим измерять степень доверия к предсказаниям. Один из эвристических подходов заключается в интерпретации $f(\mathbf{x})$ как логарифма отношения шансов, $\log p(y=1|\mathbf{x})/p(y=0|\mathbf{x})$. Тогда можно преобразовать выход SVM в вероятность по формуле:

$$p(y = 1|\mathbf{x}, \boldsymbol{\theta}) = \sigma(a f(\mathbf{x}) + b), \quad (17.95)$$

где a, b можно оценить с помощью максимального правдоподобия на отдельном контрольном наборе. (Применение обучающего набора для оценивания a и b приводит к сильной переобученности.) Эта техника впервые была предложена в работе [Pla00] и называется **масштабированием Платта**.

Однако получающиеся вероятности не очень хорошо откалиброваны, потому что в процедуре обучения SVM нет ничего такого, что оправдывало бы интерпретацию $f(\mathbf{x})$ как логарифм отношения шансов. Для иллюстрации этой мысли рассмотрим пример из работы [Tip01]. Предположим, что имеются одномерные данные, для которых $p(x|y = 0) = \text{Unif}(0, 1)$ и $p(x|y = 1) = \text{Unif}(0.5, 1.5)$. Так как условные распределения при условии разных классов перекрываются на отрезке $[0.5, 1]$, логарифм отношения шансов класса 1 к классу 0 должен быть равен 0 в этой области и бесконечности вне нее. Мы производим выборку 1000 точек из этой модели и затем обучаем вероятностный ядерный классификатор (RVM, описанный в разделе 17.4.1) и SVM-классификатор с гауссовым ядром ширины 0.1. Обе модели идеально улавливают решающую границу и достигают ошибки обобщаемости 25 %, что является байесовским оптимумом для этой задачи. Вероятностный выход RVM является хорошей аппроксимацией истинного логарифма отношения шансов, чего не скажешь о выходе SVM – см. рис. 17.15.

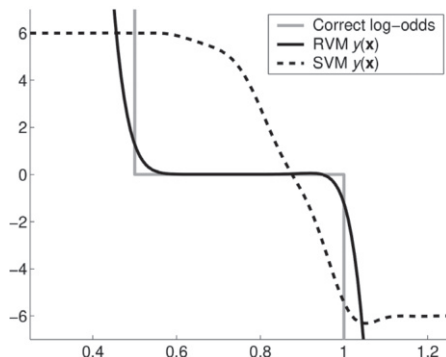


Рис. 17.15 ❖ Зависимость логарифма отношения шансов от x для трех разных методов. На основе рис. 10 из работы [Tir01]. Печатается с разрешения Майка Типпинга

17.3.6. Связь с логистической регрессией

Мы видели, что для точек, лежащих по правильную сторону от решающей границы, $\xi_n = 0$, а для остальных $\xi_n = 1 - \tilde{y}_n f(\mathbf{x}_n)$. Поэтому мы можем переписать целевую функцию (17.83) следующим образом:

$$\mathcal{L}(\mathbf{w}) = \sum_{n=1}^N \ell_{\text{hinge}}(\tilde{y}_n, f(\mathbf{x}_n)) + \lambda \|\mathbf{w}\|^2, \quad (17.96)$$

где $\lambda = (2C)^{-1}$ и $\ell_{\text{hinge}}(y, \eta)$ – **кусочно-линейная функция потерь**:

$$\ell_{\text{hinge}}(\tilde{y}, \eta) = \max(0, 1 - \tilde{y}\eta). \quad (17.97)$$

На рис. 4.2 показано, что это выпуклая, кусочно-дифференцируемая верхняя граница ступенчатой функции потерь, имеющая форму раскрытой дверной петли.

С другой стороны, логистическая регрессия (со штрафом) заключается в оптимизации функции:

$$\mathcal{L}(\mathbf{w}) = \sum_{n=1}^N \ell_{\text{ll}}(\tilde{y}_n, f(\mathbf{x}_n)) + \lambda \|\mathbf{w}\|^2, \quad (17.98)$$

где **логарифмическая потеря** равна

$$\ell_{\text{ll}}(\tilde{y}, \eta) = -\log p(y|\eta) = \log(1 + e^{-\tilde{y}\eta}). \quad (17.99)$$

Ее график также показан на рис. 4.2. Мы видим, что он похож на кусочно-линейную потерю, но есть два важных отличия. Во-первых, в силу кусочной линейности последней к ней нельзя применить регулярные методы градиентной оптимизации. (Можно, однако, вычислить субградиент при $\tilde{y}\eta = 1$.) Во-вторых, для кусочно-линейной потери существует область, в которой она строго равна 0, что приводит к разреженным оценкам.

Мы видим, что обе функции являются выпуклыми верхними границами ступенчатой потери, описываемой формулой:

$$\ell_{01}(\tilde{y}, \hat{y}) = \mathbb{I}(\tilde{y} \neq \hat{y}) = \mathbb{I}(\tilde{y}\hat{y} < 0). \quad (17.100)$$

Эти верхние границы проще оптимизировать и можно рассматривать как суррогаты ступенчатой потери. Детали см. в разделе 4.3.2.

17.3.7. Многоклассовая классификация с применением SVM

SVM, по сути своей, – бинарный классификатор. Чтобы превратить его в модель многоклассовой классификации, можно обучить S бинарных классификаторов, где данные класса s считаются положительными, а данные всех остальных классов – отрицательными. Затем окончательная метка вычисляется по правилу $\hat{y}(x) = \operatorname{argmax}_c f_c(x)$, где $f_c(x) = \log \frac{p(c=1|x)}{p(c=0|x)}$ – оценка, данная

классификатором s . Этот подход называется «**один против остальных**» (или «**один против всех**»).

К сожалению, у этого подхода есть несколько проблем. Во-первых, он может приводить к появлению областей с неоднозначными метками. Например, для зеленой области в верхней части рис. 17.16а предсказан и класс 1, и класс 2. Вторая проблема заключается в том, что оценки f_c не откалиброваны, поэтому их трудно сравнивать. Наконец, для любой бинарной подзадачи, скорее всего, будет наблюдаться несбалансированность классов (раздел 10.3.8.2). Например, пусть имеется 10 одинаково представленных классов. При обучении f_1 мы будем иметь 10 % положительных примеров и 90 % отрицательных, что плохо отразится на качестве.

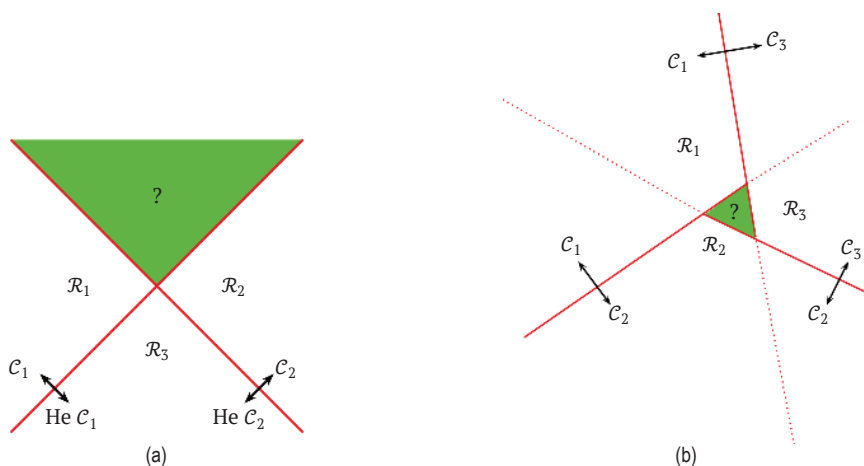


Рис. 17.16 ❖ (а) Подход «один против остальных». Для зеленой области предсказаны как класс 1, так и класс 2. (б) Подход «каждый против каждого». Метка зеленой области неоднозначна. На основе рис. 4.2 из работы [Bis06]

Другой подход называется «**каждый против каждого**» (one-versus-one – OVO), или «**на всех парах**»; в этом случае мы обучаем $C(C - 1)/2$ классификаторов различать все пары $f_{c,c'}$, а затем относим точку к классу, за который отдано наибольшее число голосов. Но и этот подход может приводить к неоднозначностям, как показано на рис. 17.16b. Кроме того, требуется обучить $O(C^2)$ моделей.

17.3.8. Как выбирать регуляризатор C

В методе SVM требуется задать ядерную функцию и параметр C . Обычно C выбирается с помощью перекрестной проверки. Заметим, однако, что между C и параметрами ядра существует сильная взаимозависимость. Например, предположим, что используется RBF-ядро с точностью $\gamma = 1/(2\sigma^2)$. Если γ велика, что соответствует узким ядрам, то может потребоваться сильная регуляризация, а значит, малое C . Если γ мала, то следует использовать большее значение C . Таким образом, γ и C тесно связаны, как показано на рис. 17.17.

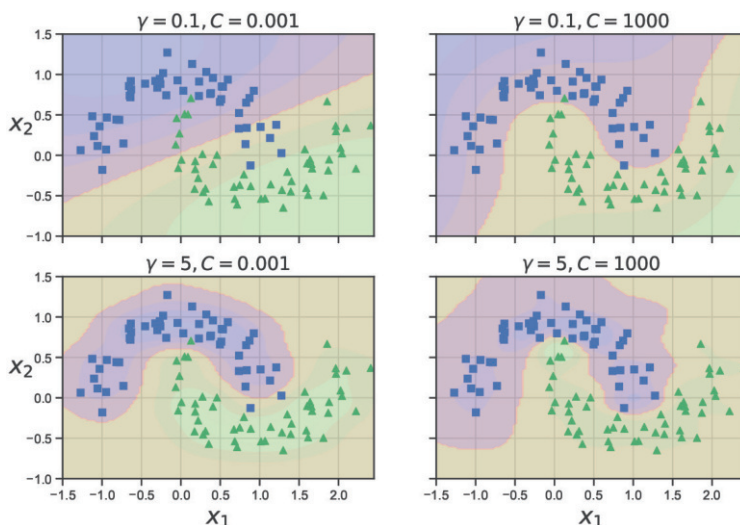


Рис. 17.17 ❖ SVM-классификатор с RBF-ядром с точностью γ и регуляризатором C применительно к данным о двух лунах. На основе рис. 5.9 из работы [Gér19]. Построено программой по адресу figures.probml.ai/book1/17.17

Авторы библиотеки `libsvm` [HCL09] рекомендуют использовать перекрестную проверку на двумерной сетке со значениями $C \in \{2^{-5}, 2^{-3}, \dots, 2^{15}\}$ и $\gamma \in \{2^{-15}, 2^{-13}, \dots, 2^3\}$. На рис. 17.18 показана зависимость оценки ошибки 0-1 при перекрестной проверке от C и γ .

Для эффективного выбора C можно разработать алгоритм прослеживания пути в духе LARS (раздел 11.4.4). Основная идея – начать с малых C , чтобы зазор был широким и, следовательно, все точки попадали внутрь него и для них имело место равенство $\alpha_i = 1$. При постепенном увеличении C небольшое

множество точек будет перемещаться изнутри зазора вовне, а их значения α_i будут изменяться от 1 до 0 по мере того, как они перестают быть опорными векторами. Когда C максимально, область внутри зазора становится пустой и никаких опорных векторов не остается. Детали см. в работе [Has+04].

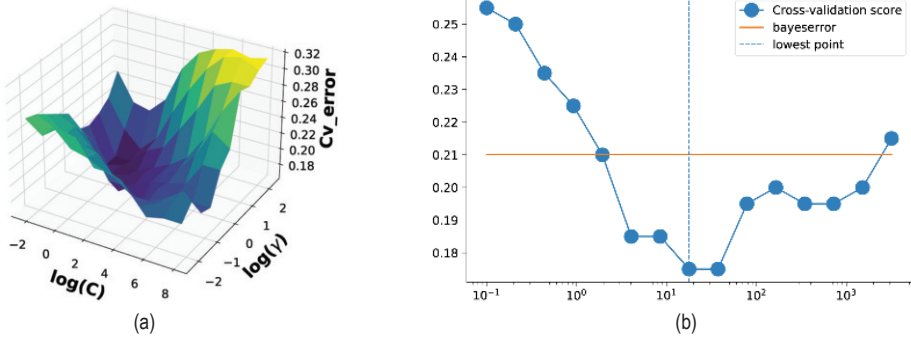


Рис. 17.18 ❖ (а) Оценка ошибки 0-1 методом перекрестной проверки для SVM-классификатора с RBF-ядром с разной точностью $\gamma = 1/(2\sigma^2)$ и разными регуляризаторами $\lambda = 1/C$ применительно к синтезированному набору данных, выбранных из смеси двух гауссовых распределений. (б) Сечение этой поверхности для $\gamma = 5$. Желтая прямая – оптимальная байесовская ошибка, вычисленная с помощью применения правила Байеса к модели, по которой генерировались данные. На основе рис. 12.6 из работы [HTF09]. Построено программой по адресу figures.probl.ai/book1/17.18

17.3.9. Ядерная гребневая регрессия

Вспомним уравнение гребневой регрессии (11.55):

$$\hat{\mathbf{w}}_{\text{map}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_D)^{-1} \mathbf{X}^T \mathbf{y} = \left(\sum_n \mathbf{x}_n \mathbf{x}_n^T + \lambda \mathbf{I}_D \right)^{-1} \left(\sum_n \tilde{y}_n \mathbf{x}_n \right). \quad (17.101)$$

Применив лемму об обращении матрицы (раздел 7.3.3), мы сможем переписать оценку гребневой регрессии в виде

$$\mathbf{w} = \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I}_N)^{-1} \mathbf{y} = \sum_n \mathbf{x}_n \left(\left(\sum_n \mathbf{x}_n^T \mathbf{x}_n + \lambda \mathbf{I}_N \right)^{-1} \mathbf{y} \right)_n. \quad (17.102)$$

Определим следующие **двойственные переменные**:

$$\boldsymbol{\alpha} = (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I}_N)^{-1} \mathbf{y} = \left(\sum_n \mathbf{x}_n^T \mathbf{x}_n + \lambda \mathbf{I}_N \right)^{-1} \mathbf{y}. \quad (17.103)$$

Тогда **основные переменные** можно записать в виде:

$$\mathbf{w} = \mathbf{X}^T \boldsymbol{\alpha} = \sum_{n=1}^N \alpha_n \mathbf{x}_n. \quad (17.104)$$

Это означает, что вектор решения – просто линейная комбинация N обучающих векторов. Подставляя это на этапе тестирования для вычисления прогнозного среднего, получаем

$$f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x} = \sum_{n=1}^N \alpha_n \mathbf{x}_n^T \mathbf{x}. \quad (17.105)$$

Теперь можно воспользоваться ядерным трюком и переписать это в виде

$$f(\mathbf{x}; \mathbf{w}) = \sum_{n=1}^N \alpha_n \mathcal{K}(\mathbf{x}_n, \mathbf{x}), \quad (17.106)$$

где

$$\alpha = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y}. \quad (17.107)$$

Иными словами,

$$f(\mathbf{x}; \mathbf{w}) = \mathbf{k}^T (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y}, \quad (17.108)$$

где $\mathbf{k} = [\mathcal{K}(\mathbf{x}, \mathbf{x}_1), \dots, \mathcal{K}(\mathbf{x}, \mathbf{x}_N)]$. Это называется **ядерной гребневой регрессией**.

Проблема заключается в том, что вектор решения α неразрезанный, поэтому предсказания на этапе тестирования занимают время $O(N)$. Как ее решить, мы обсудим в разделе 17.3.10.

17.3.10. Применение SVM для регрессии

Рассмотрим следующую ℓ_2 -регуляризованную задачу минимизации эмпирического риска:

$$J(\mathbf{w}, \lambda) = \lambda \|\mathbf{w}\|^2 + \sum_{n=1}^N \ell(\hat{y}_n, \hat{y}_n), \quad (17.109)$$

где $\hat{y}_n = \mathbf{w}^T \mathbf{x}_n + w_0$. Если использовать квадратичную потерю, $\ell(y, \hat{y}) = (y - \hat{y})^2$, где $y, \hat{y} \in \mathbb{R}$, то мы вернемся к гребневой регрессии (раздел 11.3). Если затем применить ядерный трюк, то мы вернемся к ядерной гребневой регрессии (раздел 17.3.9).

Проблема ядерной гребневой регрессии в том, что решение зависит от всех N обучающих примеров, из-за чего задача становится вычислительно неразрешимой. Однако изменив функцию потерь, мы сможем сделать оптимальное множество коэффициентов при базисных функциях, α^* , разреженным, как показано ниже.

Именно, рассмотрим следующий вариант функции потерь Хьюбера (раздел 5.1.5.3), называемый **эпсилон-нечувствительной функцией потерь**:

$$L_\epsilon(y, \hat{y}) = \begin{cases} 0, & \text{если } |y - \hat{y}| < \epsilon \\ |y - \hat{y}| - \epsilon, & \text{в противном случае} \end{cases}. \quad (17.110)$$

Это означает, что любая точка, лежащая внутри ε -трубки вокруг предсказания, не штрафуются, как показано на рис. 17.19.

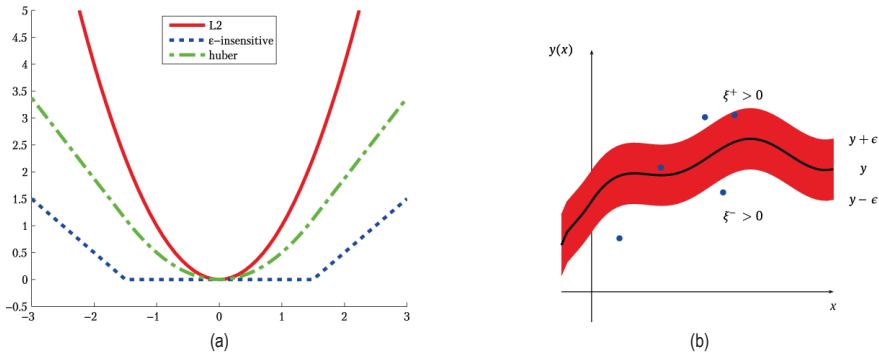


Рис. 17.19 ❖ (а) Иллюстрация функций потерь: ℓ_2 , Хьюбера и ε -нечувствительной, где $\varepsilon = 1.5$. Построено программой по адресу figures.problmlai/book1/17.19. (б) Иллюстрация ε -трубки, используемой в SVM-регрессии. Для точек выше трубки $\xi_i^+ > 0$ и $\xi_i^- = 0$. Для точек ниже трубки $\xi_i^+ = 0$ и $\xi_i^- > 0$. Для точек внутри трубки $\xi_i^+ = \xi_i^- = 0$. На основе рис. 7.7 из работы [Bis06]

Соответствующая целевая функция обычно записывается в виде

$$J = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N L_\varepsilon(\tilde{y}_n, \hat{y}_n), \quad (17.111)$$

где $\hat{y}_n = f(\mathbf{x}_n) = \mathbf{w}^\top \mathbf{x}_n + w_0$ и $C = 1/\lambda$ – регуляризационная постоянная. Эта целевая функция выпукла и ничем не ограничена, но не дифференцируема из-за функции абсолютной величины в члене потери. Как и в разделе 11.4.9, где обсуждалась задача lasso, есть несколько возможных решений. Популярный подход – переформулировать задачу как задачу условной оптимизации. Именно, введем **переменные невязки**, показывающие, степень удаленность точки от трубки:

$$\tilde{y}_n \leq f(\mathbf{x}_n) + \varepsilon + \xi_n^+, \quad (17.112)$$

$$\tilde{y}_n \geq f(\mathbf{x}_n) - \varepsilon - \xi_n^-. \quad (17.113)$$

Тогда целевую функцию можно переписать в виде

$$J = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N (\xi_n^+ + \xi_n^-). \quad (17.114)$$

Это квадратичная функция от \mathbf{w} , и ее следует минимизировать с учетом линейных ограничений (17.112)-(17.113), а также ограничений положительности $\xi_n^+ \geq 0$ и $\xi_n^- \geq 0$. Это стандартный квадратичный алгоритм с $2N + D + 1$ переменными.

Построив и оптимизировав лагранжиан, как уже было сделано выше, можно показать, что оптимальное решение имеет вид:

$$\hat{\mathbf{w}} = \sum_n \alpha_n \mathbf{x}_n, \quad (17.115)$$

где $\alpha_n \geq 0$ – двойственные переменные. (Детали см., например, в [SS02].) К счастью, вектор α разреженный, т. е. многие его элементы равны 0. Это связано с тем, что в потере не учитываются ошибки, меньшие ε . Степень разреженности контролируется параметрами C и ε .

Точки \mathbf{x}_n , для которых $\alpha_n > 0$, называются **опорными векторами**; это те точки, для которых ошибка лежит на границе или вне ε -трубки. Только эти обучающие примеры и нужно сохранять для предсказания на этапе тестирования, поскольку

$$f(\mathbf{x}) = \hat{w}_0 + \hat{\mathbf{w}}^\top \mathbf{x} = \hat{w}_0 + \sum_{n:\alpha_n>0} \alpha_n \mathbf{x}_n^\top \mathbf{x}. \quad (17.116)$$

Наконец, можно воспользоваться ядерным трюком и получить

$$f(\mathbf{x}) = \hat{w}_0 + \sum_{n:\alpha_n>0} \alpha_n \mathcal{K}(\mathbf{x}_n, \mathbf{x}). \quad (17.117)$$

Весь метод в целом, называемый **регрессией опорных векторов** или **SVM-регрессией**, впервые был предложен в работе [VGS97].

На рис. 17.20 приведен пример использования RBF-ядра с $\gamma = 1$. Если C мало, то модель сильно регуляризована, а если C велико, то модель регуляризована слабее и может аппроксимировать данные лучше. Видно также, что при малых ε трубка уже, поэтому опорных векторов больше.

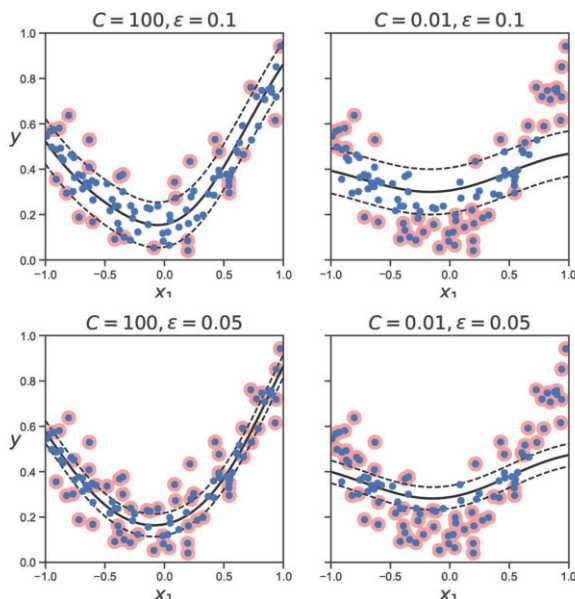


Рис. 17.20 ❖ Регрессия опорных векторов. На основе рис. 5.11 из работы [Gér19]. Построено программой по адресу figures.problml.ai/book1/17.20

17.4. МЕТОД РАЗРЕЖЕННЫХ ВЕКТОРОВ

Гауссовы процессы – очень гибкие модели, но требуют для предсказания времени $O(N)$, иногда это недопустимо много. SVM решает эту проблему, оценивая разреженный вектор весов. Однако SVM не дает откалиброванных вероятностей на выходе.

Лучшее из обоих миров можно взять, используя параметрические модели, в которых вектор признаков определен с использованием базисных функций с центрами в обучающих точках:

$$\phi(\mathbf{x}) = [\mathcal{K}(\mathbf{x}, \mathbf{x}_1), \dots, \mathcal{K}(\mathbf{x}, \mathbf{x}_N)], \quad (17.118)$$

где \mathcal{K} – ядро сходства, необязательно ядро Мерсера. Отображение (17.118) переводит $\mathbf{x} \in \mathcal{X}$ в $\phi(\mathbf{x}) \in \mathbb{R}^N$. Мы можем подставить этот новый вектор признаков в любую дискриминантную модель, например, логистической регрессии. Поскольку имеется $D = N$ параметров, необходимо использовать какую-то регуляризацию, чтобы предотвратить переобучение. Если обучить такую модель с помощью ℓ_2 -регуляризации (тогда она называется **L2VM**), то результат зачастую показывает хорошее качество предсказания, но вектор весов \mathbf{w} будет плотным и зависящим от всех N обучающих точек. Естественное решение – наложить на \mathbf{w} априорное распределение, поощряющее разреженность, так чтобы можно было хранить не все примеры. В результате получаются **методы разреженных векторов**.

17.4.1. Метод релевантных векторов

Самый простой способ гарантировать, что \mathbf{w} разрежен, – использовать ℓ_1 -регуляризацию, как описано в разделе 11.4. Этот подход называется **L1VM** или **методом векторов Лапласа** (Laplace vector machine), потому что он эквивалентен оценке MAP, когда \mathbf{w} имеет априорное распределение Лапласа.

Но иногда ℓ_1 -регуляризация не дает достаточного уровня разреженности для заданного уровня верности. Альтернативный подход основан на использовании **автоматического определения релевантности (ARD)**, когда применяется максимальное правдоподобие типа II (оно же эмпирическое байесовское) для оценки разреженного вектора весов [Mac95; Nea96]. Если применить эту идею к вектору признаков, определенных в терминах ядра, как в формуле (17.118), то получим **метод релевантных векторов** (relevance vector machine – **RVM**) [Tip01; TF03].

17.4.2. Сравнение разреженных и плотных ядерных методов

На рис. 17.21 проведено сравнение методов L2VM, L1VM, RVM и SVM с RBF-ядром на задаче бинарной классификации в двумерном случае. Мы используем перекрестную проверку для выбора $C = 1/\lambda$ для SVM (см. раздел 17.3.8),

а затем берем то же самое значение параметра регуляризации для L2VM и L1VM. Как видим, качество предсказания у всех методов примерно одинаково. Однако RVM дает самую разреженную модель, поэтому на этапе выполнения она будет работать быстрее других.

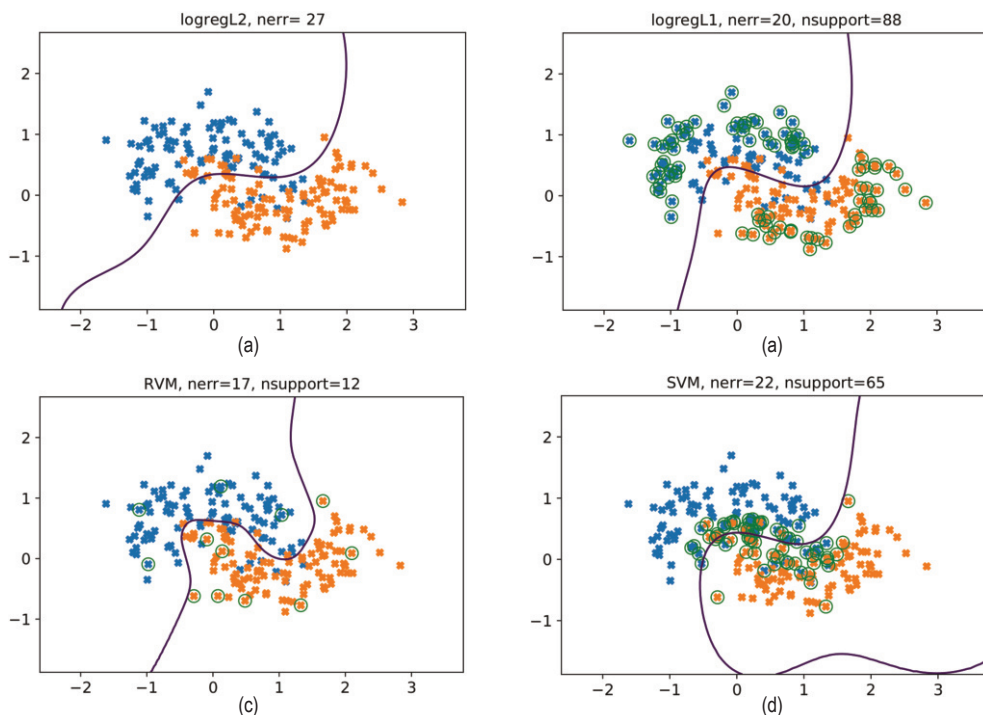


Рис. 17.21 ❖ Пример нелинейной бинарной классификации с RBF-ядром с полосой пропуска $\sigma = 0.3$. (a) L2VM. (b) L1VM. (c) RVM. (d) SVM. Черными кружочками обозначены опорные векторы. Построено программой по адресу figures.probl.ai/book1/17.21

На рис. 17.22 методы L2VM, L1VM, RVM и SVM с RBF-ядром сравниваются на задаче регрессии в одномерном случае. И снова предсказания очень похожи, но RVM дает самую разреженную модель, за ним следует L1VM, а за ним SVM. Сравнение продолжено на рис. 17.23.

Помимо этих небольших эмпирических примеров, в табл. 17.1 приведена более подробная сводка различных методов. Столбцы интерпретируются следующим образом.

- **Опт. \mathbf{w} :** ключевой вопрос – является ли целевая функция $\mathcal{L}(\mathbf{w}) = -\log p(\mathcal{D}|\mathbf{w}) - \log p(\mathbf{w})$ выпуклой или нет. В методах L2VM, L1VM и SVM целевые функции выпуклые, а в RVM нет. ГП относятся к классу байесовских методов, в которых веса исключаются посредством интегрирования.
- **Опт. ядра:** все методы требуют «настройки» параметров ядра, в частности полосы пропуска RBF-ядра, а также уровня регуляризации. Для

методов, основанных на гауссовых априорных распределениях, в том числе L2VM, RVM и ГП, можно использовать эффективные градиентные оптимизаторы для максимизации маргинального правдоподобия. Для SVM и L1VM необходимо использовать перекрестную проверку, работающую медленнее (см. раздел 17.3.8).

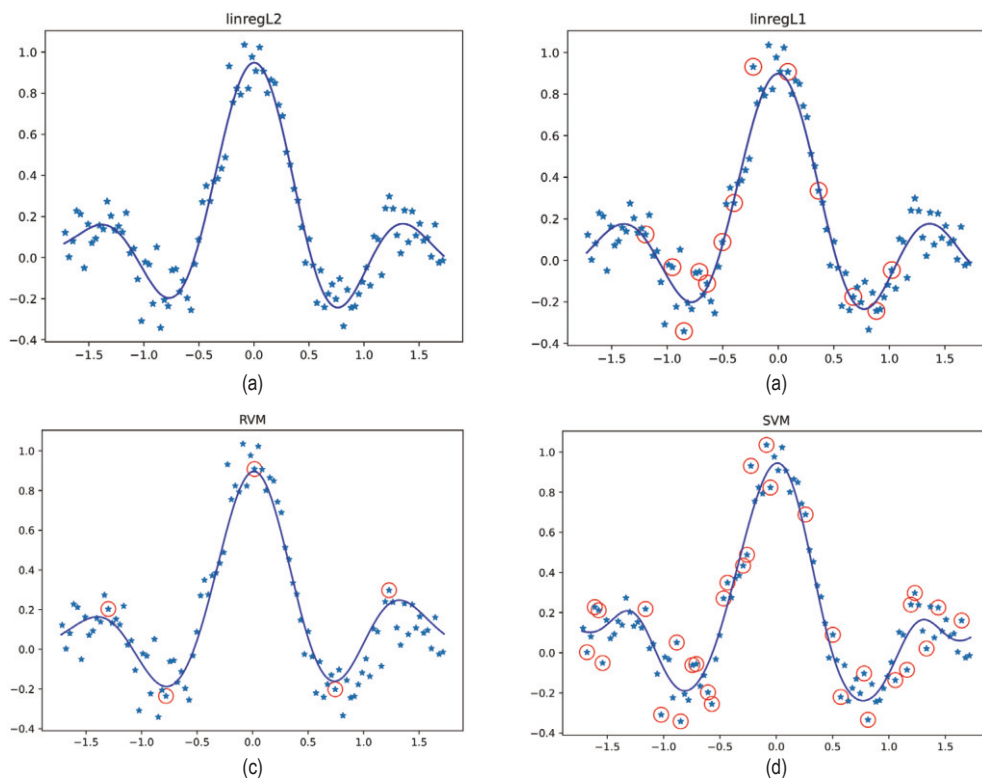


Рис. 17.22 ❖ Модель, обученная ядерной регрессии на зашумленной функции sinc с RBF-ядром с полосой пропускания $\sigma = 0.3$. (a) L2VM с $\lambda = 0.5$. (b) L1VM с $\lambda = 0.5$. (c) RVM. (d) SVM-регрессия с параметром $C = 1/\lambda$, выбранным методом перекрестной проверки. Красными кружочками обозначены сохраненные обучающие примеры. Построено программой по адресу figures.probl.ai/book1/17.22

- Разреж.: L1VM, RVM и SVM – разреженные ядерные методы в том смысле, что используется только подмножество обучающих примеров. ГП и L2VM разреженными не являются, в них используются все примеры. Принципиальное преимущество разреженности в том, что предсказание на этапе тестирования обычно быстрее. Однако это часто приводит к излишней уверенности в результатах предсказания.
- Вер.: все методы, кроме SVM, порождают на выходе распределение вероятностей вида $p(y|\mathbf{x})$. SVM возвращает уровень «доверия», который можно преобразовать в вероятность, но такие вероятности обычно очень плохо откалиброваны (см. раздел 17.3.5).

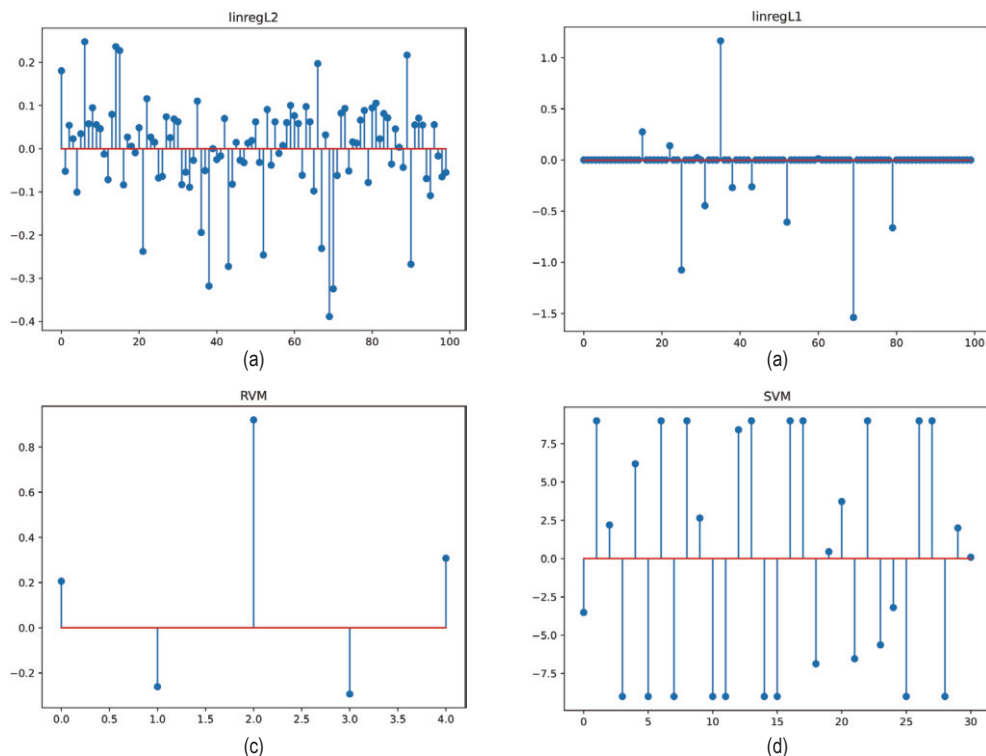


Рис. 17.23 ❖ Оценки коэффициентов для моделей на рис. 17.22.
Построено программой по адресу figures.problm.ai/book1/17.23

- Многокл.: все методы, кроме SVM, естественно работают в многоклассовой постановке, поскольку используется категориальное распределение, а не распределение Бернулли. SVM можно преобразовать в многоклассовый классификатор, но такой подход сталкивается с различными трудностями, описанными в разделе 17.3.7.
- Не Мерсера: SVM и ГП требуют, чтобы ядро было положительно определенным. В остальных методах это необязательно, потому что ядро в формуле (17.118) может быть произвольной функцией двух аргументов.

Таблица 17.1. Сравнение различных ядерных классификаторов. EB = эмпирический байесовский, CV = перекрестная проверка. Детали см. в тексте

Метод	Опт. w	Опт. ядра	Разреж.	Вер.	Многокл.	Не Мерсера	Раздел
SVM	Выпуклая	CV	Да	Нет	Косвенно	Нет	17.3
L2VM	Выпуклая	EB	Нет	Да	Да	Да	17.4.1
L1VM	Выпуклая	CV	Да	Да	Да	Да	17.4.1
RVM	Невыпуклая	EB	Да	Да	Да	Да	17.4.1
ГП	Неприменимо	EB	Да	Да	Нет	Нет	17.2.7

17.5. УПРАЖНЕНИЯ

Упражнение 17.1 [обучение SVM-классификатора вручную*].

(Источник: Яаккола.)

Рассмотрим одномерный набор данных с двумя точками: $(x_1 = 0, y_1 = -1)$ и $(x_2 = \sqrt{2}, y_2 = 1)$. Рассмотрим отображение каждой точки в трехмерное пространство с помощью вектора признаков $\phi(x) = [1, \sqrt{2}x, x^2]^T$. (Это эквивалентно использованию полиномиального ядра второй степени.) Классификатор с максимальным зазором имеет вид:

$$\min \|\mathbf{w}\|^2 \text{ при условиях} \quad (17.119)$$

$$y_1(\mathbf{w}^T \phi(\mathbf{x}_1) + w_0) \geq 1, \quad (17.120)$$

$$y_2(\mathbf{w}^T \phi(\mathbf{x}_2) + w_0) \geq 1. \quad (17.121)$$

- Найдите вектор, параллельный оптимальному вектору \mathbf{w} . *Указание:* обратитесь к рис. 7.8, где показано, что вектор \mathbf{w} перпендикулярен решающей границе между двумя точками в трехмерном пространстве признаков.
- Каково значение зазора при таком \mathbf{w} ? *Указание 1:* вспомните, что зазор – это расстояние от каждого опорного вектора до решающей границы. *Указание 2:* подумайте о геометрии двух точек и разделяющей их прямой в пространстве.
- Решите уравнения относительно \mathbf{w} , воспользовавшись тем, что зазор равен $1/\|\mathbf{w}\|$.
- Решите уравнения относительно w_0 , используя найденное значение \mathbf{w} и формулы (17.119)–(17.121). *Указание:* точки будут лежать на решающей границе, поэтому неравенства точные.
- Выпишите дискриминантную функцию $f(x) = w_0 + \mathbf{w}^T \phi(x)$ в виде явной функции от x .

Глава 18

Деревья, леса, бэггинг и бустинг

18.1. ДЕРЕВЬЯ КЛАССИФИКАЦИИ И РЕГРЕССИИ

Деревья классификации и регрессии, или CART-модели [BFO84], называемые также решающими деревьями [Qui86; Qui93], определяются рекурсивным разбиением пространства входов и определением локальной модели в каждой получающейся области. Модель в целом представлена деревом, имеющим по одному листовому узлу на каждую область.

18.1.1. Определение модели

Начнем с рассмотрения деревьев регрессии, в которых все входные данные вещественны. Дерево состоит из множества вложенных решающих правил. В каждом узле i измерение d_i признака входного вектора \mathbf{x} сравнивается с пороговым значением t_i , после чего \mathbf{x} передается ниже по левой ветви, если d_i больше порога, и по правой, если меньше. В листовых узлах модель определяет предсказанный выход для всех векторов, попавших в эту часть пространства входов.

Например, рассмотрим дерево регрессии на рис. 18.1а. В первом узле задается вопрос, верно ли, что x_1 меньше некоторого порога t_1 . Если да, то затем дерево смотрит, верно ли, что x_2 меньше другого порога t_2 . Если да, то мы переходим к листовому узлу вдоль левой ветви. Это соответствует области пространства:

$$R_1 = \{\mathbf{x} : x_1 \leq t_1, x_2 \leq t_2\}. \quad (18.1)$$

Мы можем ассоциировать с этой областью вычисление предсказанного выхода, основываясь на **осепараллельном разбиении** пространства, индуцированном ветвями дерева. В данном случае двумерное пространство входов разбивается на пять областей, как показано на рис. 18.1б¹. Теперь

¹ Если областей достаточно (т. е. дерево достаточно глубокое), то можно построить кусочно-линейную аппроксимацию решающих границ более сложной формы, но для обучения такой модели необходимо очень много данных.

можно связать с каждой из этих областей средний отклик, что приведет к кусочно-постоянной поверхности, показанной на рис. 18.1b. Например, выход для области 1 можно оценить по формуле:

$$w_1 = \frac{\sum_{n=1}^N y_n \mathbb{I}(\mathbf{x}_n \in R_1)}{\sum_{n=1}^N \mathbb{I}(\mathbf{x}_n \in R_1)}. \quad (18.2)$$

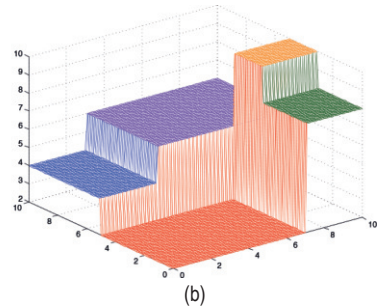
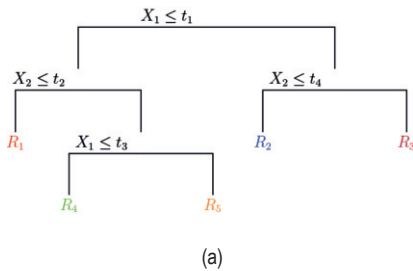


Рис. 18.1 ❖ (a) Дерево регрессии, построенное по двум входам. (b) Соответствующая кусочно-постоянная поверхность. На основе рис. 9.2 из работы [HTF09]. Построено программой по адресу figures.problm.ai/book1/18.1

Формально дерево регрессии определяется следующим образом:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \sum_{j=1}^J w_j \mathbb{I}(\mathbf{x} \in R_j), \quad (18.3)$$

где R_j – область, заданная j -м листовым узлом, w_j – предсказанный выход для этого узла, а $\boldsymbol{\theta} = \{(R_j, w_j) : j = 1 \dots J\}$, где J – число узлов. Сами области определяются размерностями признаков, использованных при каждом делении пространства, и соответствующими порогами на пути от корня к листу. Например, на рис. 18.1a имеем $R_1 = [(d_1 \leq t_1), (d_2 \leq t_2)]$, $R_2 = [(d_1 \leq t_1), (d_2 > t_2), (d_3 \leq t_3)]$ и т. д. (Для категориальных признаков можно определить разбиение, основанное на сравнении признака d_i с каждым из его возможных значений, а не с числовым порогом.) О том, как обучать решающее дерево, мы поговорим в разделе 18.1.2.

Для задач классификации листья содержат распределение меток классов, а не средний отклик, см. пример дерева классификации на рис. 18.2.

18.1.2. Обучение модели

Для обучения модели нужно минимизировать следующую потерю:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^N \ell(y_n, f(\mathbf{x}_n; \boldsymbol{\theta})) = \sum_{j=1}^J \sum_{\mathbf{x}_n \in R_j} \ell(y_n, w_j). \quad (18.4)$$

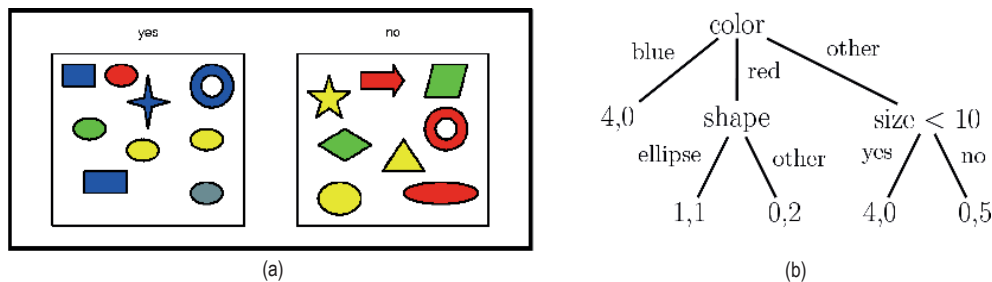


Рис. 18.2 ❖ (а) Множество геометрических фигур с соответствующими бинарными метками. Признаками являются: цвет (значения «синий», «красный», «другой»), форма (значения «эллипс», «другая») и размер (вещественное). (б) Гипотетическое дерево классификации для аппроксимации этих данных. Лист с меткой (n_1, n_0) означает, что в эту область попало n_1 положительных примеров и n_0 отрицательных

К сожалению, эта функция не дифференцируема из-за необходимости обучать дискретную структуру дерева. В действительности задача нахождения оптимального разделения данных является NP-полной [HR76]. Стандартная практика – использовать жадную процедуру, итеративно добавляя по одному узлу за раз. Этот подход используется в алгоритмах CART [BFO84], C4.5 [Qui93] и ID3 [Qui86] – трех популярных реализациях метода.

Идея следующая. Допустим, что мы находимся в узле i , и пусть $\mathcal{D}_i = \{(\mathbf{x}_n, y_n) \in N_i\}$ – множество примеров, достигающих этого узла. Обозначим $\mathcal{D}_i^L(j, t) = \{(\mathbf{x}_n, y_n) \in N_i : y_{n,j} \leq t\}$ и $\mathcal{D}_i^R(j, t) = \{(\mathbf{x}_n, y_n) \in N_i : y_{n,j} > t\}$ разделение этих примеров на левое и правое поддерево. (Для категориальных признаков полагаем $\mathcal{D}_i^L(j, t) = \{(\mathbf{x}_n, y_n) \in N_i : y_{n,j} = t\}$ и $\mathcal{D}_i^R(j, t) = \{(\mathbf{x}_n, y_n) \in N_i : y_{n,j} \neq t\}$.) Выбираем лучший признак j , по которому производить разделение, и лучшее значение этого признака, t_i , следующим образом:

$$(j, t_i) = \arg \min_{j \in \{1, \dots, D\}} \min_{t \in T_j} \frac{|\mathcal{D}_i^L(j, t)|}{|D|} c(\mathcal{D}_i^L(j, t)) + \frac{|\mathcal{D}_i^R(j, t)|}{|D|} c(\mathcal{D}_i^R(j, t)), \quad (18.5)$$

где функция стоимости $c()$ определена ниже.

Множество возможных порогов T_j для признака j можно получить, отсортировав уникальные значения $\{x_{nj}\}$. Например, если признак 1 принимает значения $\{4.5, -12, 72, -12\}$, то полагаем $T_1 = \{-12, 4.5, 72\}$.

В случае категориальных входов предположим, что признак j принимает K_j возможных значений. Тогда мы проверяем совпадение признака с каждым из этих значений. Тем самым определяется множество K_j возможных бинарных разделений. Или же можно допустить многопутевое разделение с K_k ветвями, как показано на рис. 18.2. Однако при таком подходе возможна **фрагментация**, когда в каждое поддерево попадает слишком мало данных, что ведет к переобучению. Поэтому чаще всего используют бинарное разделение.

Теперь обсудим функцию стоимости $c(\mathcal{D}_i)$, которая используется для вычисления стоимости узла i . В случае регрессии можно использовать среднеквадратическую ошибку:

$$\text{cost}(\mathcal{D}_i) = \frac{1}{|\mathcal{D}_i|} \sum_{n \in \mathcal{D}_i} (y_n - \bar{y})^2, \quad (18.6)$$

где $\bar{y} = (1/|\mathcal{D}_i|) \sum_{n \in \mathcal{D}_i} y_n$ – среднее значение откликов по всем примерам, достигающим узла i .

В случае классификации сначала вычисляем эмпирическое распределение меток классов для этого узла:

$$\hat{\pi}_{ic} = \frac{1}{|\mathcal{D}_i|} \sum_{n \in \mathcal{D}_i} \mathbb{I}(y_n = c). \quad (18.7)$$

Зная его, можно вычислить **индекс Джини**:

$$G_i = \sum_{c=1}^C \hat{\pi}_{ic}(1 - \hat{\pi}_{ic}) = \sum_c \hat{\pi}_{ic} - \sum_c \hat{\pi}_{ic}^2 = 1 - \sum_c \hat{\pi}_{ic}^2. \quad (18.8)$$

Это ожидаемая частота ошибок. Чтобы убедиться в этом, заметим, что $\hat{\pi}_{ic}$ – вероятность, что случайный пример в листовом узле действительно принадлежит классу c , а $1 - \hat{\pi}_{ic}$ – вероятность его неправильной классификации.

Альтернативно можно определить стоимость как энтропию или **девиацию** узла:

$$H_i = \mathbb{H}(\hat{\pi}_i) = - \sum_{c=1}^C \hat{\pi}_{ic} \log \hat{\pi}_{ic}. \quad (18.9)$$

Для **чистого** узла (содержащего примеры только из одного класса) энтропия будет нулевой.

Взяв одну из этих функций, мы можем воспользоваться формулой (18.5) для выбора наилучшего признака и лучшего порога в каждом узле. Затем разделяем данные и вызываем алгоритм обучения рекурсивно для каждого подмножества.

18.1.3. Регуляризация

Если позволить дереву стать слишком глубоким, то оно может достичь нулевой ошибки на обучающем наборе (в предположении незашумленности меток). Для этого нужно только разделить пространство входов на достаточно мелкие области, в которых метки одинаковы. Однако это, как правило, ведет к переобучению. Есть два основных способа предотвратить его. Первый – применить некоторую эвристику для остановки процесса роста дерева, например не допускать слишком малого числа примеров в узле или ограничить максимальную глубину. Второй – дать возможность дереву достичь максимальной глубины, когда дальнейшее разделение невозможно, а затем произвести **обрезку**, т. е. объединить поддеревья в их общий родитель (см., например, [BA97b]). Это может в какой-то мере компенсировать жадный рост дерева сверху вниз. (Например, рассмотрим применение нисходящего подхода к данным XOR на рис. 13.1: алгоритм не произвел бы ни одного

разделения, потому что никакой признак сам по себе не обладает предсказательной способностью.) Однако прямой рост в сочетании с обратной обрезкой медленнее жадного нисходящего алгоритма.

18.1.4. Обработка отсутствующих входных признаков

В общем случае дискриминантным моделям, к которым относятся и нейронные сети, трудно справиться с отсутствием некоторых входных признаков, мы обсуждали этот вопрос в разделе 1.5.5. Однако конкретно для деревьев имеются простые эвристики, которые могут дать неплохой результат.

Стандартная эвристика для обработки отсутствующих входов в решающих деревьях – поискать серию «резервных» переменных, которые могут индуцировать разделение, похожее на то, что дает выбранная переменная, в каждой точке разделения; их можно использовать, когда на этапе тестирования выбранная переменная не наблюдается. Такие разделения называют **суррогатными**. Этот метод находит сильно коррелированные признаки, так что его можно рассматривать как обучение локальной совместной модели входа. Преимущество перед порождающей моделью в том, что моделируется не все совместное распределение входов, а недостаток в том, что метод очень сильно зависит от конкретной ситуации. Более простой подход, применимый к категориальным переменным, – завести новое значение «отсутствует» и после этого считать, что данные наблюдаются в полном объеме.

18.1.5. Плюсы и минусы

Древовидные модели популярны по нескольким причинам:

- они легко поддаются интерпретации;
- они позволяют обрабатывать смесь дискретных и непрерывных входов;
- они нечувствительны к монотонным преобразованиям входов (потому что точки разделения определяются ранжированием примеров), поэтому нет необходимости стандартизировать данные;
- они автоматически производят отбор признаков;
- они относительно робастны к выбросам;
- они быстро обучаются и хорошо масштабируются на большие наборы данных;
- они способны обрабатывать отсутствие входных признаков.

Однако у древовидных моделей есть и недостатки. Главный из них – что их предсказания не очень точны по сравнению с моделями других видов. Отчасти это вызвано жадностью алгоритма построения дерева.

С этим связана еще одна проблема – **неустойчивость** деревьев: небольшие изменения входных данных оказывают большое влияние на структуру всего дерева вследствие иерархической природы процесса его роста, из-за чего ошибки на верхнем уровне распространяются на все дерево. Например, рассмотрим дерево на рис. 18.3b. Стоило убрать единственную точку из обучающего набора, как решающая поверхность кардинально изменилась

(см. рис. 18.3с) из-за того, что разделения параллельны осям. (Пропуск признаков также может вызывать неустойчивость.) В разделах 18.3 и 18.4 мы превратим эту неустойчивость в достоинство.

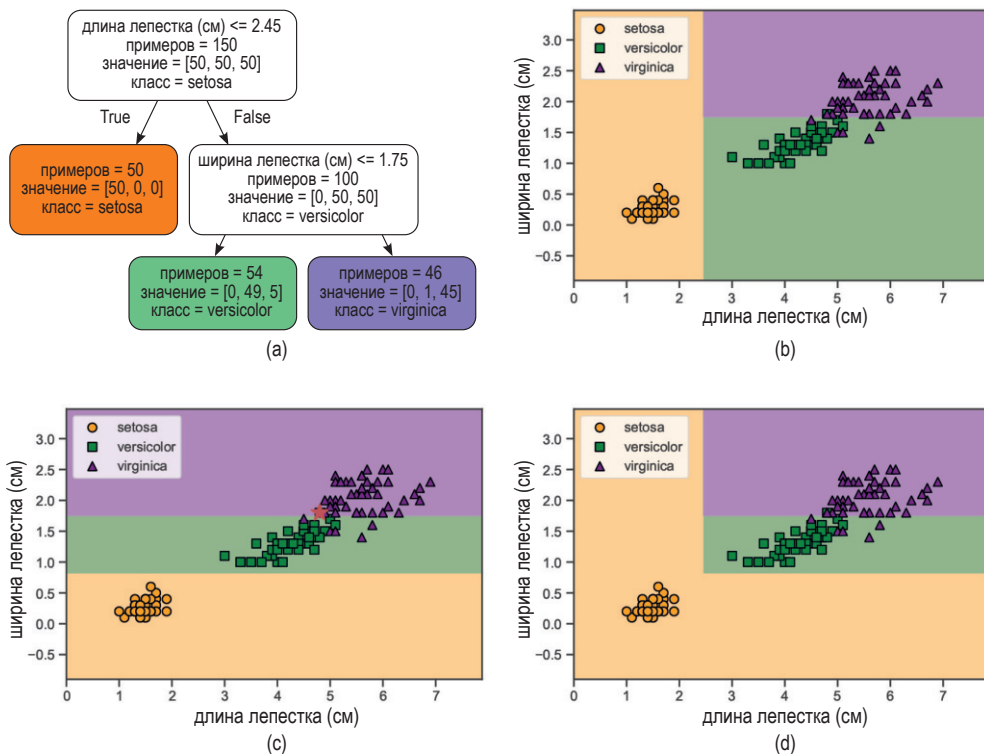


Рис. 18.3 ❖ (а) Решающее дерево глубины 2 для моделирования данных об ирисах по длине и ширине лепестка. Листовые узлы кодируются цветом в соответствии с мажоритарным классом. Количество обучающих примеров, прошедших от корня к каждому узлу, показано внутри блоков, и там же показано распределение их значений по классам. Можно произвести нормировку и получить распределение меток классов для каждого узла. (б) Решающая поверхность, индуцированная деревом (а). (в) Аппроксимация данных в случае, когда одна точка (обозначенная красной звездочкой) опущена. (г) Ансамбль обеих моделей (б) и (в). Построено программой по адресу figures.problm.lai/book1/18.3

18.2. АНСАМБЛЕВОЕ ОБУЧЕНИЕ

В разделе 18.1 мы видели, что решающие деревья могут быть весьма неустойчивыми в том смысле, что предсказания резко изменяются при малом возмущении обучающих данных. Иными словами, решающие деревья дают оценку с высокой дисперсией. Простой способ уменьшить дисперсию – производить усреднение по нескольким моделям. Это называется **ансамблевым обучением**. Результирующая модель имеет вид:

$$f(y|\mathbf{x}) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} f_m(y|\mathbf{x}), \quad (18.10)$$

где f_m – m -я базовая модель. Ансамбль будет иметь примерно такое же смещение, как базовые модели, но меньшую дисперсию, что обычно приводит к улучшенному качеству в целом (см. описание компромисса между дисперсией и смещением в разделе 4.7.6.3).

Усреднение – разумный способ объединить предсказания моделей регрессии. Для классификаторов иногда лучше брать мажоритарный выход (иногда это называется **методом комитета**). Чтобы понять, почему это помогает, предположим, что каждая базовая модель – это бинарный классификатор с верностью θ и что правильным является класс 1. Пусть $Y_m \in \{0, 1\}$ – предсказание m -й модели и $S = \sum_{m=1}^M Y_m$ – число голосов, отданных за класс 1. Определим окончательный предиктор как мажоритарный класс, т. е. класс 1, если $S > M/2$, и класс 0 в противном случае. Вероятность, что ансамбль выберет класс 1, равна

$$p = \Pr(S > M/2) = 1 - B(M/2, M, \theta), \quad (18.11)$$

где $B(x, M, \theta)$ – функция биномиального распределения с параметрами M и θ , вычисленными в точке x . Для $\theta = 0.51$ и $M = 1000$ получаем $p = 0.73$, а при $M = 10\,000$ получаем $p = 0.97$.

Качество подхода на основе голосования значительно улучшилось, потому что мы предположили, что каждый предиктор дает независимые ошибки. На практике ошибки могут коррелировать, но если объединить в ансамбль разнообразных моделей, то все равно можно добиться успеха.

18.2.1. Стековое обобщение

Альтернатива использованию невзвешенного среднего или мажоритарного голосования – обучиться комбинированию базовых моделей:

$$f(y|\mathbf{x}) = \sum_{m \in \mathcal{M}} w_m f_m(y|\mathbf{x}). \quad (18.12)$$

Это называется **стековым обобщением** [Wol92] (иногда употребляется термин «стоговоевание»). Отметим, что веса в этой формуле необходимо обучать на отдельном наборе данных, иначе вся их масса придется на базовую модель, дающую наилучшее качество.

18.2.2. Ансамблевое обучение не то же, что байесовское усреднение моделей

Стоит отметить, что ансамбль моделей не то же самое, что использование байесовского усреднения моделей (БМА) (раздел 4.6), как отмечено в работе [Min00]. Ансамбль рассматривает более широкий класс гипотез вида

$$f(y|\mathbf{x}, \mathbf{w}, \boldsymbol{\theta}) = \sum_{m \in \mathcal{M}} w_m p(y|\mathbf{x}, \boldsymbol{\theta}_m), \quad (18.13)$$

тогда как в ВМА используются модели вида

$$f(y|\mathbf{x}, \mathcal{D}) = \sum_{m \in \mathcal{M}} p(m|\mathcal{D}) p(y|\mathbf{x}, m, \mathcal{D}). \quad (18.14)$$

Ключевое различие в том, что в случае ВМА сумма весов $p(m|\mathcal{D})$ равна 1, и в пределе, когда данные бесконечны, будет выбрана только одна модель (а именно модель MAP). В ансамбле же веса w_m могут быть произвольны и не сходятся к единственной модели.

18.3. Бэггинг

В этом разделе мы обсудим **бэггинг** [Bre96] (сокращение от «bootstrap aggregating» – бутстрэп-агрегирование). Это простая форма ансамблевого обучения, когда M разных базовых моделей обучаются на разных случайных выборках данных; это повышает разнообразие предсказаний. Выборка из набора данных производится с возвращением (техника, называемая бутстрэпом, – см. раздел 4.7.3), т. е. один и тот же пример может встречаться несколько раз, пока не наберется N примеров для каждой модели (где N – число исходных точек данных).

Недостаток бутстрэпа заключается в том, что каждая базовая модель видит в среднем только 63 % уникальных входных примеров. Чтобы убедиться в этом, заметим, что вероятность того, что некоторый элемент не будет выбран из набора размера N ни в одной из N попыток, равна $(1 - 1/N)^N$. В пределе, когда N стремится к бесконечности, это выражение стремится к $e^{-1} \approx 0.37$, а значит, выбрано будет только $1 - 0.37 = 0.63$ всего множества точек.

37 % обучающих примеров, не используемых заданной базовой моделью, называются **не вошедшими в набор экземплярами** (out-of-bag – oob). Качество предсказаний базовой модели на этих экземплярах можно использовать как оценку качества на тестовом наборе. Это полезная альтернатива перекрестной проверке.

Главное преимущество бутстрэпа в том, что он не дает ансамблю слишком сильно зависеть от любого конкретного примера, что повышает робастность и обобщаемость [Gra04]. Например, сравнивая рис. 18.3b и 18.3c, мы видим, что исключение всего одного примера из обучающего набора может оказывать сильное влияние на обученное решающее дерево (хотя алгоритм роста дерева детерминирован). Усредняя предсказания обеих моделей, мы получаем более надежную модель, показанную на рис. 18.3d. Это преимущество становится еще более заметным с увеличением размера ансамбля, как показано на рис. 18.4. (Конечно, при этом потребляется больше времени и памяти.)

Бэггинг не всегда улучшает качество. В частности, он опирается на то, что базовые модели дают неустойчивую оценку, поэтому отсутствие некоторых данных значительно изменяет свойства результирующей модели. Так обсто-

ит дело для решающих деревьев, но не для моделей других видов, например классификаторов по ближайшим соседям. Для нейронных сетей ситуация еще более запутанная. Они могут быть неустойчивы относительно своего обучающего набора. С другой стороны, глубокие сети будут работать хуже, если видят только 63 % своих данных, поэтому бэггинг в применении к ним обычно не показывает выдающихся результатов [NTL20].

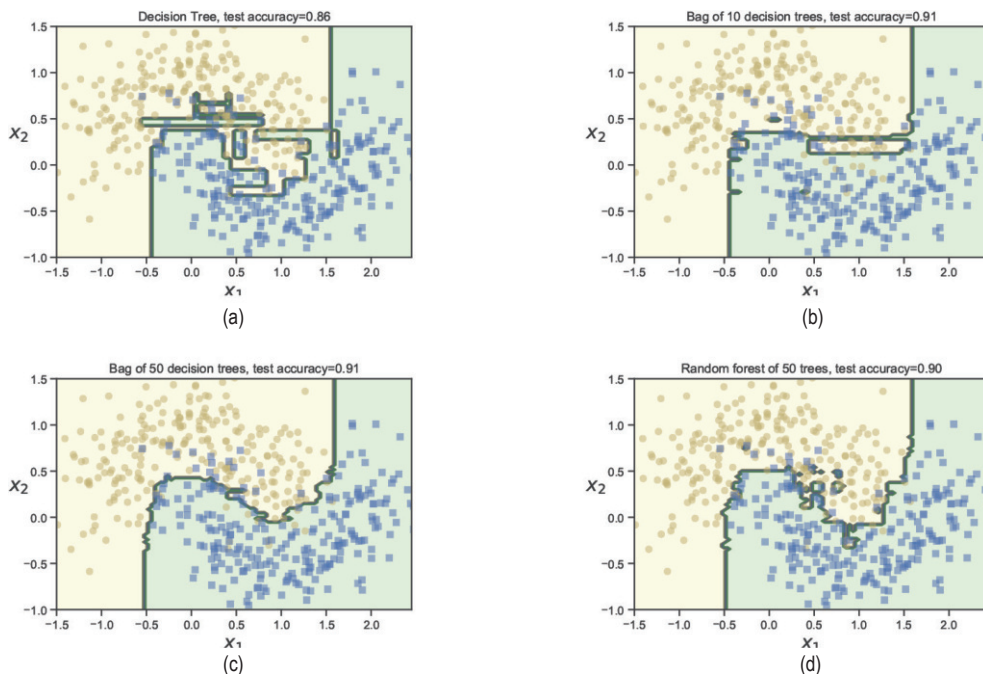


Рис. 18.4 ❖ (a) Одно решающее дерево. (b-c) Бэггинговый ансамбль из 10 и 50 деревьев. (d) Случайный лес из 50 деревьев. На основе рис. 7.5 из работы [Gér19]. Построено программой по адресу figures.probml.ai/book1/18.4

18.4. СЛУЧАЙНЫЕ ЛЕСА

Бэггинг опирается на предположение о том, что повторный прогон одного и того же алгоритма на разных подмножествах набора данных будет давать достаточно разнообразные базовые модели. Метод **случайных лесов** [Bre01] стремится еще больше декоррелировать базовых обучаемых за счет того, что обучает деревья на случайно выбранном подмножестве входных переменных (в каждом узле дерева) и на случайно выбранном подмножестве данных. Для этого формула (18.5) изменяется, так что измерение j признака, по которому производится разделение, оптимизируется на случайном подмножестве признаков, $S_i \subset \{1, \dots, D\}$. Например, рассмотрим набор данных для обучения распознаванию почтового спама [HTF09, стр. 301]. Этот набор данных со-

держит 4601 сообщений, классифицированных как спам (1) или неспам (0). Данные были выложены в открытый доступ Джорджем Форманом из Hewlett-Packard (HP) Labs.

Существует 57 количественных (вещественных) предикторов, а именно:

- 48 признаков, соответствующих проценту слов в сообщении, совпадающих с заданными словами, например «remove» или «labs»;
- шесть признаков, соответствующих проценту символов в электронной почте, совпадающих с заданным символом, а именно ; . [! \$ #;
- три признака, соответствующих средней длине, максимальной длине и сумме длин непрерывных последовательностей прописных букв (эти признаки называются CAPAVE, CAPMAX и CAPTOT).

На рис. 18.5 показано, что случайные леса работают намного лучше, чем решающие деревья, полученные бэггингом, потому что много входных признаков нерелевантны. (Мы также видим, что метод, называемый «бустингом» и обсуждаемый в разделе 18.5, работает еще лучше, однако он требует последовательного обучения деревьев, тогда как случайные леса можно обучать параллельно.)

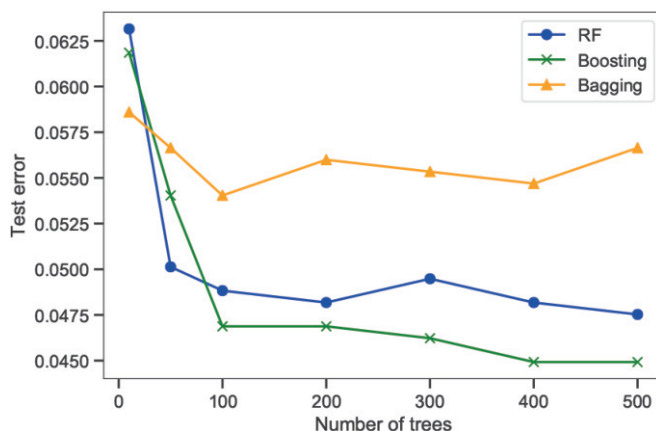


Рис. 18.5 ❖ Зависимость верности предсказаний от размера ансамбля деревьев для бэггинга, случайных лесов и градиентного бустинга с логарифмической потерей. На основе рис. 15.1 из работы [HTF09]. Построено программой по адресу figures.problml.ai/book1/18.5

18.5. БУСТИНГ

Ансамбли деревьев, все равно обученные с помощью бэггинга или алгоритма случайных лесов, соответствуют модели вида

$$f(\mathbf{x}; \boldsymbol{\theta}) = \sum_{m=1}^M \beta_m F_m(\mathbf{x}; \boldsymbol{\theta}_m), \quad (18.15)$$

где F_m – m -е дерево, а β_m – соответствующий вес, который часто полагают равным 1. Это можно обобщить, допустив в качестве F_m функции-аппроксиматоры общего вида, в частности нейронные сети, а не только деревья. Результат называется **аддитивной моделью** [НТФ09]. Можно считать, что это линейная модель с **адаптивными базисными функциями**. Цель, как обычно, – минимизировать эмпирическую потерю (с факультативным регуляризатором):

$$\mathcal{L}(f) = \sum_{i=1}^N \ell(y_i, f(\mathbf{x}_i)). \quad (18.16)$$

Бустингом (усилением) [Sch90; FS96] называется алгоритм последовательного обучения аддитивных моделей, где каждая функция F_m – бинарный классификатор, который возвращает $F_m \in \{-1, +1\}$. Именно, мы сначала обучаем F_1 на оригинальных данных, а затем назначаем образцам данных веса, равные ошибкам, допущенным F_1 , так что неправильно классифицированные примеры получают больший вес. Этот процесс повторяется, пока не будет обучено достаточное число компонентов M . (M – гиперпараметр, управляющий сложностью модели в целом, его можно выбрать, следя за качеством на контрольном наборе и применяя раннюю остановку.)

Можно показать, что, коль скоро верность каждого предиктора F_m лучше случайной (даже на взвешенном наборе данных), ансамбль классификаторов будет иметь лучшую верность, чем любой из его компонентов. То есть если F_m – **слабый обучаемый** (его верность лишь чуть выше 50 %), то его качество можно усилить, применив описанную выше процедуру, так что окончательный предиктор f станет **сильным обучаемым**. (Дополнительные сведения о подходе к бустингу на основе теории обучения см., например, в работе [SF12].)

Заметим, что бустинг уменьшает смещение сильного обучаемого благодаря тому, что обучает деревья, зависящие друг от друга, тогда как бэггинг и случайные леса уменьшают дисперсию, обучая независимые деревья. Во многих случаях бустинг работает лучше (см. пример на рис. 18.5).

Оригинальный алгоритм бутинга был ориентирован на бинарную классификацию с вполне конкретной функцией потерь, как будет объяснено в разделе 18.5.3, и был выведен из теории РАС-обучения (см. раздел 5.4.4). Далее в этом разделе мы будем говорить о статистической версии бустинга, разработанной в [FHT00; Fri01], которая применима к произвольным функциям потерь, что делает метод пригодным для регрессии, многоклассовой классификации, ранжирования и т. д. Наше изложение опирается на работы [НТФ09, глава 10] и [ВН07], в которых можно найти дополнительную информацию.

18.5.1. Прямое поэтапное аддитивное моделирование

В этом разделе мы обсудим **прямое поэтапное аддитивное моделирование**, заключающееся в последовательной оптимизации целевой функции (18.16) для произвольных (дифференцируемых) функций потерь, где f – аддитивная модель вида (18.15). Таким образом, на итерации m мы вычисляем

$$(\beta_m, \theta_m) = \operatorname{argmin}_{\beta, \theta} \sum_{i=1}^N \ell(y_i, f_{m-1}(\mathbf{x}_i) + \beta F(\mathbf{x}_i; \theta)). \quad (18.17)$$

Затем полагаем

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}_i; \theta_m) = f_{m-1}(\mathbf{x}) + \beta_m F_m(\mathbf{x}_i). \quad (18.18)$$

(Отметим, что мы не изменяем параметры ранее добавленных моделей.) Как именно выполнять этот шаг оптимизации, зависит от выбранной функции потерь и (в некоторых случаях) от формы слабого обучаемого F . Мы обсудим их ниже.

18.5.2. Квадратичная потеря и бустинг наименьших квадратов

Предположим, что в качестве потери используется квадратичная ошибка, $\ell(y, \hat{y}) = (y - \hat{y})^2$. В этом случае i -й член целевой функции на шаге m принимает вид:

$$\ell(y_i, f_{m-1}(\mathbf{x}_i) + \beta F(\mathbf{x}_i; \theta)) = (y_i - f_{m-1}(\mathbf{x}_i) - \beta F(\mathbf{x}_i; \theta))^2 = (r_{im} - \beta F(\mathbf{x}_i; \theta))^2, \quad (18.19)$$

где $r_{im} = y_i - f_{m-1}(\mathbf{x}_i)$ – невязка текущей модели на i -м наблюдении. Эту целевую функцию можно оптимизировать, просто положив $\beta = 1$ и подогнав F к невязкам. Получающийся метод называется **бустингом наименьших квадратов** [BY03].

На рис. 18.6 приведен пример этого процесса, причем в качестве слабого обучаемого используется дерево регрессии глубины 2. Слева показан результат подгонки слабого обучаемого к невязкам, а справа – текущий сильный обучаемый. Мы видим, что каждый новый слабый обучаемый, добавленный в ансамбль, исправляет ошибки, допущенные предыдущими версиями модели.

18.5.3. Экспоненциальная потеря и AdaBoost

Предположим, что нас интересует бинарная классификация, т. е. предсказание $\tilde{y}_i \in \{-1, +1\}$. Предположим, что слабый обучаемый вычисляет распределение:

$$p(y = 1|\mathbf{x}) = \frac{e^{F(\mathbf{x})}}{e^{-F(\mathbf{x})} + e^{F(\mathbf{x})}} = \frac{1}{1 + e^{-2F(\mathbf{x})}}, \quad (18.20)$$

т. е. $F(\mathbf{x})$ возвращает половину логарифма отношения шансов. Из формулы (10.13) мы знаем, что отрицательное логарифмическое правдоподобие равно

$$\ell(\tilde{y}, F(\mathbf{x})) = \log(1 + e^{-2\tilde{y}F(\mathbf{x})}). \quad (18.21)$$

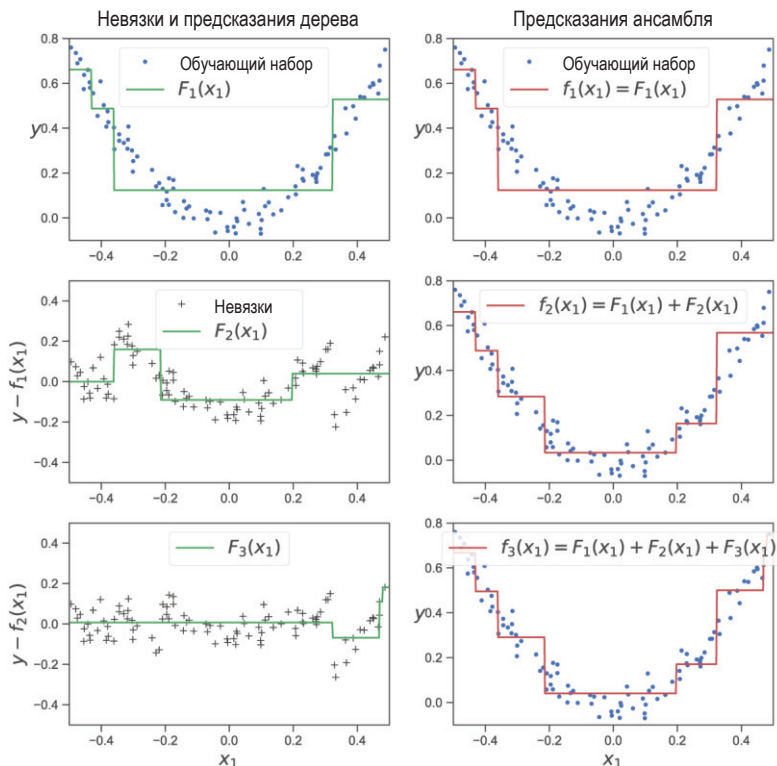


Рис. 18.6 ❖ Бустинг с использованием дерева регрессии глубины 2 применительно к одномерному набору данных. На основе рис. 7.9 из работы [Gér19]. Построено программой по адресу figures.problml.ai/book1/18.6

Эту функцию можно минимизировать, сделав так, чтобы зазор $m(\mathbf{x}) = \tilde{y}F(\mathbf{x})$ был максимально возможным. Из рис. 18.7 видно, что логарифмическая потеря – гладкая верхняя граница ступенчатой потери. Видно также, что отрицательные зазоры штрафуются сильнее, чем положительные, что нам и нужно (поскольку положительный зазор означает, что классификация уже правильная).

Однако можно использовать и другие функции потерь. В этом разделе мы рассмотрим **экспоненциальную потерю**:

$$\ell(\tilde{y}, F(\mathbf{x})) = \exp(-\tilde{y}F(\mathbf{x})). \quad (18.22)$$

Из рис. 18.7 видно, что это также гладкая верхняя граница ступенчатой потери. Для генеральной совокупности (с бесконечным размером выборки) оптимальное решение в случае экспоненциальной потери такое же, как в случае логарифмической. Чтобы убедиться в этом, просто приравняем производную экспоненциальной потери (для любого \mathbf{x}) нулю:

$$\frac{\partial}{\partial F(\mathbf{x})} \mathbb{E}[e^{-\tilde{y}f(\mathbf{x})} | \mathbf{x}] = \frac{\partial}{\partial F(\mathbf{x})} [p(\tilde{y} = 1 | \mathbf{x})e^{-F(\mathbf{x})} + p(\tilde{y} = -1 | \mathbf{x})e^{F(\mathbf{x})}] \quad (18.23)$$

$$= -p(\tilde{y} = 1|\mathbf{x})e^{-F(\mathbf{x})} + p(\tilde{y} = -1|\mathbf{x})e^{F(\mathbf{x})} \quad (18.24)$$

$$= 0 \Rightarrow \frac{p(\tilde{y} = 1|\mathbf{x})}{p(\tilde{y} = -1|\mathbf{x})} = e^{2F(\mathbf{x})}. \quad (18.25)$$

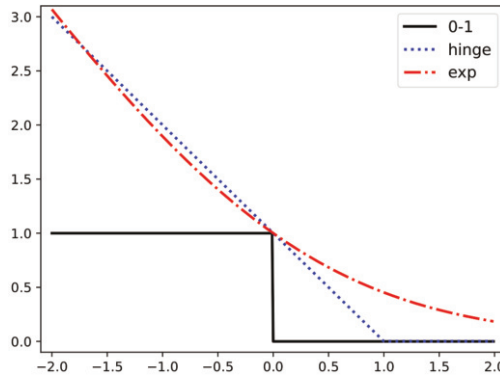


Рис. 18.7 ❖ Применение различных функций потерь для бинарной классификации. По горизонтальной оси откладывается зазор $m(\mathbf{x}) = \tilde{y}F(\mathbf{x})$, по вертикальной – потеря. При вычислении логарифмической потери используется логарифм по основанию 2. Построено программой по адресу figures.probl.ai/book1/18.7

Однако оказывается, что экспоненциальную потерю проще оптимизировать в бустинговой постановке. (Случай логарифмической потери мы рассмотрим в разделе 18.5.4.) Сначала рассмотрим, как обучить m -го слабого обучаемого F_m при использовании экспоненциальной потери. На шаге m нужно минимизировать функцию

$$\mathcal{L}_m(F) = \sum_{i=1}^N \exp[-\tilde{y}_i(f_{m-1}(\mathbf{x}_i) + \beta F(\mathbf{x}_i))] = \sum_{i=1}^N \omega_{i,m} \exp(-\beta \tilde{y}_i F(\mathbf{x}_i)), \quad (18.26)$$

где $\omega_{i,m} \triangleq \exp(-\tilde{y}_i f_{m-1}(\mathbf{x}_i))$ – вес, назначенный примеру i , а $\tilde{y}_i \in \{-1, +1\}$. Эту целевую функцию можно переписать в виде:

$$\mathcal{L}_m = e^{-\beta} \sum_{\tilde{y}_i = F(\mathbf{x}_i)} \omega_{i,m} + e^{\beta} \sum_{\tilde{y}_i \neq F(\mathbf{x}_i)} \omega_{i,m} \quad (18.27)$$

$$= (e^{\beta} - e^{-\beta}) \sum_{i=1}^N \omega_{i,m} \mathbb{I}(\tilde{y}_i \neq F(\mathbf{x}_i)) + e^{-\beta} \sum_{i=1}^N \omega_{i,m}. \quad (18.28)$$

Следовательно, оптимальная функция, которую следует добавить в ансамбль, имеет вид:

$$F_m = \operatorname{argmin}_F \sum_{i=1}^N \omega_{i,m} \mathbb{I}(\tilde{y}_i \neq F(\mathbf{x}_i)). \quad (18.29)$$

Ее можно найти, применив слабого обучаемого к взвешенному варианту набора данных с весами $\omega_{i,m}$.

Осталось только найти размер обновления, β . Подставляя F_m в L_m и решая уравнение относительно β , находим

$$\beta_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}, \quad (18.30)$$

где

$$\text{err}_m = \frac{\sum_{i=1}^N \omega_i \mathbb{I}(\tilde{y}_i \neq F_m(\mathbf{x}_i))}{\sum_{i=1}^N \omega_{i,m}}. \quad (18.31)$$

Поэтому обновление в целом принимает вид:

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F_m(\mathbf{x}). \quad (18.32)$$

После обновления сильного обучаемого нужно пересчитать веса для следующей итерации:

$$\omega_{i,m+1} = e^{-\tilde{y}_i f_m(\mathbf{x}_i)} = e^{-\tilde{y}_i f_{m-1}(\mathbf{x}_i) - \tilde{y}_i \beta_m F_m(\mathbf{x}_i)} = \omega_{i,m} e^{-\tilde{y}_i \beta_m F_m(\mathbf{x}_i)}. \quad (18.33)$$

Если $\tilde{y}_i = F_m(\mathbf{x}_i)$, то $\tilde{y}_i F_m(\mathbf{x}_i) = 1$, а если $\tilde{y}_i \neq F_m(\mathbf{x}_i)$, то $\tilde{y}_i F_m(\mathbf{x}_i) = -1$. Отсюда $-\tilde{y}_i F_m(\mathbf{x}_i) - 2\mathbb{I}(\tilde{y}_i \neq F_m(\mathbf{x}_i)) - 1$, так что обновление принимает вид:

$$\omega_{i,m+1} = \omega_{i,m} e^{\beta_m (2\mathbb{I}(\tilde{y}_i \neq F_m(\mathbf{x}_i)) - 1)} = \omega_{i,m} e^{2\beta_m (2\mathbb{I}(\tilde{y}_i \neq F_m(\mathbf{x}_i)) - 1)} e^{-\beta_m}. \quad (18.34)$$

Так как множитель $e^{-\beta_m}$ во всех примерах одинаков, его можно опустить. Если затем положить $\alpha_m = 2\beta_m$, то обновление принимает вид:

$$\omega_{i,m+1} = \begin{cases} \omega_{i,m} e^{\alpha_m}, & \text{если } \tilde{y}_i \neq F_m(\mathbf{x}_i) \\ \omega_{i,m} & \text{в противном случае} \end{cases}. \quad (18.35)$$

Таким образом, мы видим, что веса неправильно классифицированных примеров экспоненциально увеличиваются. Получившийся алгоритм 8 называется **Adaboost.M1** [FS96]¹.

Многоклассовое обобщение экспоненциальной потери и похожий на adaboost алгоритм ее минимизации, **SAMME** (stagewise additive modeling using a multiclass exponential loss function – поэтапное аддитивное моделирование с использованием многоклассовой экспоненциальной функции потери),

¹ В работе [FHT00] он назван **дискретным AdaBoost**, потому что в нем предполагается, что базовый классификатор F_m возвращает бинарную метку класса. Если вместо этого F_m возвращает вероятность, то можно использовать модифицированный алгоритм, известный под названием **настоящий AdaBoost**. Детали см. в [FHT00].

описан в работе [Has+09]. Он реализован в библиотеке `scikit_learn` (класс `AdaBoostClassifier`).

Алгоритм 8. Алгоритм Adaboost.M1 для бинарной классификации с экспоненциальной потерей

```

1   $\omega_i = 1/N$ ;
2  for  $m = 1 : M$  do
3      Обучить классификатор  $F_m(\mathbf{x})$  на обучающем наборе с весами  $\mathbf{w}$ ;
4      Вычислить  $\text{err}_m = \frac{\sum_{i=1}^N \omega_{i,m} \mathbb{I}(\tilde{y}_i \neq F_m(\mathbf{x}_i))}{\sum_{i=1}^N \omega_{i,m}}$ ;
5      Вычислить  $\alpha_m = \log[(1 - \text{err}_m)/\text{err}_m]$ ;
6      Положить  $\omega_i \leftarrow \omega_i \exp[\alpha_m \mathbb{I}(\tilde{y}_i \neq F_m(\mathbf{x}_i))]$ ;
7  Вернуть  $f(\mathbf{x}) = \text{sgn} [\sum_{m=1}^M \alpha_m F_m(\mathbf{x})]$ ;
```

18.5.4. LogitBoost

Проблема экспоненциальной потери в том, что она назначает слишком большой вес неправильно классифицированным примерам, и это ясно показывает экспоненциальный рост в левой части рис. 18.7. Поэтому метод очень уязвим к выбросам (неправильно помеченным примерам). Кроме того, $e^{-\tilde{y}f}$ не является логарифмом никакой функции вероятности для бинарных переменных $\tilde{y} \in \{-1, +1\}$, следовательно, мы не можем восстановить оценки вероятности по $f(\mathbf{x})$.

Естественная альтернатива – использовать логарифмическую потерю, обсуждавшуюся в разделе 18.5.3. Она штрафует за ошибки лишь линейно, как видно из рис. 18.7. Это также означает, что мы сможем извлечь вероятности из окончательной обученной функции по формуле:

$$p(y = 1|\mathbf{x}) = \frac{e^{f(\mathbf{x})}}{e^{-f(\mathbf{x})} + e^{f(\mathbf{x})}} = \frac{1}{1 + e^{-2f(\mathbf{x})}}. \quad (18.36)$$

Цель – минимизировать ожидаемую логарифмическую потерю:

$$L_m(F) = \sum_{i=1}^N \log[1 + \exp(-2\tilde{y}_i(f_{m-1}(\mathbf{x}) + F(\mathbf{x}_i)))]. \quad (18.37)$$

Применив обновление Ньютона к этой целевой функции (похожее на IRLS), мы можем вывести алгоритм 9, который называется **logitBoost** [FHT00]. Его ключевая особенность – умение слабого обучаемого F решать задачу взвешенных наименьших квадратов. Этот метод можно обобщить на многоклассовую постановку, как описано в работе [FHT00].

Алгоритм 9. Алгоритм LogitBoost для бинарной классификации с логарифмической потерей

```

1   $\omega_i = 1/N, \pi_i = 1/2;$ 
2  for  $m = 1 : M$  do
3      Вычислить рабочую характеристику  $z_i = \frac{y_i - \pi_i}{\pi_i(1 - \pi_i)};$ 
4      Вычислить веса  $\omega_i = \pi_i(1 - \pi_i);$ 
5       $F_m = \operatorname{argmin}_F \sum_{i=1}^N \omega_i (z_i - F(\mathbf{x}_i))^2;$ 
6      Обновить  $f(\mathbf{x}) \leftarrow f(\mathbf{x}) + \frac{1}{2} F_m(\mathbf{x});$ 
7      Вычислить  $\pi_i = 1/(1 + \exp(-2f(\mathbf{x}_i)));$ 
8  Вернуть  $f(\mathbf{x}) = \operatorname{sgn} [\sum_{m=1}^M F_m(\mathbf{x})];$ 
  
```

18.5.5. Градиентный бустинг

Вместо того чтобы выводить новые версии бустинга для каждой функции потерь, можно построить обобщенную версию, известную под названием **градиентный бустинг** [Fri01; Mas+00]. Чтобы понять, как она устроена, предположим, что мы ищем $\hat{f} = \operatorname{argmin}_f \mathcal{L}(f)$, выполняя градиентный спуск в пространстве функций. Поскольку функции – бесконечномерные объекты, будем представлять их значениями на обучающем наборе, $\mathbf{f} = (f(\mathbf{x}_1), \dots, f(\mathbf{x}_N))$. На шаге m положим \mathbf{g}_m равной градиенту $\mathcal{L}(f)$, вычисленному при $\mathbf{f} = \mathbf{f}_{m-1}$:

$$\mathbf{g}_{im} = \left[\frac{\partial \ell(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right]_{f=\mathbf{f}_{m-1}}. \quad (18.38)$$

Градиенты некоторых распространенных функций потерь приведены в табл. 18.1. Затем производится обновление:

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \beta_m \mathbf{g}_m, \quad (18.39)$$

где β_m – длина шага, выбираемая следующим образом:

$$\beta_m = \operatorname{argmin}_{\beta} \mathcal{L}(\mathbf{f}_{m-1} - \beta \mathbf{g}_m). \quad (18.40)$$

Таблица 18.1. Некоторые часто используемые функции потерь, их градиенты и минимизаторы на генеральной совокупности F . Для задач бинарной классификации предполагается, что $\tilde{y}_i \in \{-1, +1\}$ и $\pi_i = \sigma(2f(\mathbf{x}_i))$. Для задач регрессии предполагается, что $\tilde{y}_i \in \mathbb{R}$. На основе [HTF09, стр. 360] и [BH07, стр. 483]

Название	Потеря	$\partial \ell(y_i, f(\mathbf{x}_i)) / \partial f(\mathbf{x}_i)$
Квадратичная ошибка	$\frac{1}{2}(y_i - f(\mathbf{x}_i))^2$	$y_i - f(\mathbf{x}_i)$
Абсолютная ошибка	$ y_i - f(\mathbf{x}_i) $	$\operatorname{sgn}(y_i - f(\mathbf{x}_i))$
Экспоненциальная потеря	$\exp(-\tilde{y}_i f(\mathbf{x}_i))$	$-\tilde{y}_i \exp(-\tilde{y}_i f(\mathbf{x}_i))$
Бинарная логарифмическая потеря	$\log(1 + e^{-\tilde{y}_i f_i})$	$y_i - \pi_i$
Многоклассовая логарифмическая потеря	$-\sum_c y_{ic} \log \pi_{ic}$	$y_{ic} - \pi_{ic}$

В такой форме алгоритм не особенно полезен, потому что оптимизирует f только на фиксированном множестве N точек, т. е. мы не обучаем функцию, способную к обобщению. Однако его можно модифицировать, обучив слабого обучаемого аппроксимировать отрицательный сигнал градиента. То есть будем применять обновление:

$$F_m = \operatorname{argmin}_F \sum_{i=1}^N (-g_{im} - F(\mathbf{x}_i))^2. \quad (18.41)$$

Алгоритм в целом приведен в алгоритме 10. Мы опустили шаг линейного поиска β_m – необязательный, как показано в работе [ВН07]. Однако мы ввели скорость обучения, или **коэффициент усадки** $0 < \nu \leq 1$, чтобы контролировать величину обновлений в целях регуляризации.

Алгоритм 10. Градиентный бустинг

```

1  Инициализировать  $f_0(\mathbf{x}) = \operatorname{argmin}_F \sum_{i=1}^N L(y_i, F(\mathbf{x}_i))$ ;
2  for  $m = 1 : M$  do
3      Вычислить невязку градиента  $r_{im} = - \left[ \frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right]_{f(\mathbf{x}_i) = f_{m-1}(\mathbf{x}_i)}$ ;
4      Использовать слабого обучаемого для вычисления  $F_m = \operatorname{argmin}_F \sum_{i=1}^N (r_{im} - F(\mathbf{x}_i))^2$ ;
5      Обновить  $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu F_m(\mathbf{x})$ ;
6  Вернуть  $f(\mathbf{x}) = f_M(\mathbf{x})$ 
```

Применяя этот алгоритм с квадратичной потерей, мы возвращаемся к L2Boosting, потому что $-g_{im} = y_i - f_{m-1}(\mathbf{x}_i)$ – просто невязка. Но его можно применять и с другими функциями потерь, например абсолютной потерей или потерей Хьюбера (раздел 5.1.5.3), что полезно для задач робастной регрессии.

Для классификации можно использовать логарифмическую потерю. Тогда мы получим алгоритм **BinomialBoost** [ВН07]. Его преимущество перед LogitBoost в том, что не нужно выполнять обучение с весами; нужно просто применить модель регрессии типа черного ящика к вектору градиента. Для многоклассовой классификации мы можем обучить C отдельных деревьев регрессии, используя псевдоневязку вида:

$$-g_{icm} = \frac{\partial \ell(y_i, f_{1m}(\mathbf{x}_i), \dots, f_{Cm}(\mathbf{x}_i))}{\partial f_{cm}(\mathbf{x}_i)} = \mathbb{I}(y_i = c) - \pi_{ic}. \quad (18.42)$$

Хотя деревья обучаются по отдельности, их предсказания объединяются посредством преобразования softmax:

$$p(y = c|\mathbf{x}) = \frac{e^{f_c(\mathbf{x})}}{\sum_{c'=1}^C e^{f_{c'}(\mathbf{x})}}. \quad (18.43)$$

Если набор данных велик, то можно использовать стохастический вариант, в котором на каждой итерации производится подвыборка (без возвращения)

случайной доли данных для передачи дереву регрессии. Он называется **стохастическим градиентным бустингом** [Fri99]. Мало того что он быстрее, так еще и обобщается лучше, потому что подвыборка данных – это форма регуляризации.

18.5.5.1. Градиентный бустинг деревьев

На практике градиентный бустинг почти всегда предполагает, что слабый обучаемый – это дерево регрессии вида

$$F_m(\mathbf{x}) = \sum_{j=1}^{J_m} w_{jm} \mathbb{I}(\mathbf{x} \in R_{jm}), \quad (18.44)$$

где w_{jm} – предсказанный выход для области R_{jm} . (В общем случае w_{jm} может быть вектором.) Эта комбинация называется **деревьями регрессии с градиентным бустингом**, или просто **градиентным бустингом деревьев**. (Один из вариантов называется MART – «multivariate additive regression trees» (многомерные аддитивные деревья регрессии) [FM03].)

Чтобы воспользоваться этим в алгоритме градиентного бустинга, мы сначала находим хорошие области R_{jm} для дерева m , применяя стандартное обучение деревьев регрессии (см. раздел 18.1) на невязках, затем находим веса для каждого листа, решая задачу:

$$\hat{w}_{jm} = \operatorname{argmin}_w \sum_{\mathbf{x}_i \in R_{jm}} \ell(y_i, f_{m-1}(\mathbf{x}_i) + w). \quad (18.45)$$

Для квадратичной ошибки (используемой в сочетании с градиентным бустингом) оптимальным весом \hat{w}_{jm} будет просто среднее невязок в листе.

18.5.5.2. XGBoost

XGBoost (<https://github.com/dmlc/xgboost>) (extreme gradient boosting – экстремальный градиентный бустинг) – очень эффективная и широко используемая реализация градиентного бустинга деревьев, в которой добавлено несколько улучшений сверх описанных в разделе 18.5.5.1. Детали можно найти в работе [CG16], но вкратце речь идет о следующем: добавляется регуляризатор сложности дерева, в котором используется аппроксимация потери второго порядка (из работы [FHT00]) вместо простой линейной аппроксимации; производится выборка признаков во внутренних узлах (как в случайных лесах), и применяются различные методы информатики (например, реализация вычислений вне памяти для больших наборов данных) с целью улучшить масштабируемость¹.

Если говорить о деталях, то XGBoost оптимизирует следующую регуляризованную целевую функцию:

¹ Есть и другие популярные пакеты для градиентного бустинга деревьев, например: **CatBoost** (<https://catboost.ai/>) и **LightGBM** (<https://github.com/Microsoft/LightGBM>).

$$\mathcal{L}(f) = \sum_{i=1}^N \ell(y_i, f(\mathbf{x}_i)) + \Omega(f), \quad (18.46)$$

где

$$\Omega(f) = \gamma J + \frac{1}{2} \lambda \sum_{j=1}^J w_j^2 \quad (18.47)$$

– регуляризатор, в котором J – число листьев, а $\gamma \geq 0$ и $\lambda \geq 0$ – коэффициенты регуляризации. На m -м шаге потеря равна

$$\mathcal{L}_m(F_m) = \sum_{i=1}^N \ell(y_i, f_{m-1}(\mathbf{x}_i) + F_m(\mathbf{x}_i)) + \Omega(F_m) + \text{const}. \quad (18.48)$$

Мы можем вычислить разложение в ряд Тейлора до членов второго порядка:

$$\mathcal{L}_m(F_m) \approx \sum_{i=1}^N \left[\ell(y_i, f_{m-1}(\mathbf{x}_i) + g_{im}F_m(\mathbf{x}_i) + \frac{1}{2} h_{im}F_m^2(\mathbf{x}_i)) \right] + \Omega(F_m) + \text{const}, \quad (18.49)$$

где h_{im} – гессииан:

$$h_{im} = \left[\frac{\partial^2 \ell(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)^2} \right]_{f=f_{m-1}}. \quad (18.50)$$

В случае деревьев регрессии имеем $F(\mathbf{x}) = w_{q(\mathbf{x})}$, где $q: \mathbb{R}^D \rightarrow \{1, \dots, J\}$ определяет, какому листовому узлу принадлежит \mathbf{x} , а $w \in \mathbb{R}^J$ – веса листьев. Тогда формулу (18.49) можно переписать следующим образом, опустив члены, не зависящие от F_m :

$$\mathcal{L}_m(q, \mathbf{w}) \approx \sum_{i=1}^N \left[g_{im}F_m(\mathbf{x}_i) + \frac{1}{2} h_{im}F_m^2(\mathbf{x}_i) \right] + \gamma J + \frac{1}{2} \lambda \sum_{j=1}^J w_j^2 \quad (18.51)$$

$$= \sum_{j=1}^J \left[\left(\sum_{i \in I_j} g_{im} \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_{im} + \lambda \right) w_j^2 \right] + \gamma J, \quad (18.52)$$

где $I_j = \{i: q(\mathbf{x}_i) = j\}$ – множество индексов точек, приписанных j -му узлу.

Определим $G_{jm} = \sum_{i \in I_j} g_{im}$ и $H_{jm} = \sum_{i \in I_j} h_{im}$. Тогда предыдущую формулу можно упростить:

$$\mathcal{L}_m(q, \mathbf{w}) = \sum_{j=1}^J \left[G_{jm} w_j + \frac{1}{2} (H_{jm} + \lambda) w_j^2 \right] + \gamma J. \quad (18.53)$$

Это выражение квадратично зависит от w_j , поэтому оптимальные веса имеют вид:

$$w_j^* = - \frac{G_{jm}}{H_{jm} + \lambda}. \quad (18.54)$$

Тогда потеря при вычислении различных древовидных структур q равна

$$\mathcal{L}_m(q, \mathbf{w}^*) = -\frac{1}{2} \sum_{j=1}^I \frac{G_{jm}^2}{H_{jm} + \lambda} + \gamma J. \quad (18.55)$$

Эту потерю можно жадно оптимизировать, применяя рекурсивную процедуру разделения узлов, описанную в разделе 18.1. Именно, для данного листа j рассматривается разделение его на левую и правую часть, $I = I_L \cup I_R$. Выигрыш (уменьшение потери) при таком разделении можно вычислить следующим образом:

$$\text{gain} = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{(H_L + H_R) + \lambda} \right] - \gamma, \quad (18.56)$$

где $G_L = \sum_{i \in I_L} g_{im}$, $G_R = \sum_{i \in I_R} g_{im}$, $H_L = \sum_{i \in I_L} h_{im}$ и $H_R = \sum_{i \in I_R} h_{im}$. Таким образом, мы видим, что разделять узел бессмысленно, если выигрыш меньше γ .

Быстрое приближенное вычисление этой целевой функции, не требующее сортировки признаков (для выбора оптимального порога разделения), описано в работе [CG16].

18.6. ИНТЕРПРЕТАЦИЯ АНСАМБЛЕЙ ДЕРЕВЬЕВ

Деревья популярны, потому что допускают интерпретацию. К сожалению, ансамбли деревьев (в форме бэггинга, случайных лесов или бустинга) таким свойством уже не обладают. Но есть простые методы, позволяющие все-таки понять, какая функция была обучена.

18.6.1. Важность признаков

Для одного решающего дерева T в работе [BFO84] предложена следующая мера **важности признака** k :

$$R_k(T) = \sum_{j=1}^{I-1} G_j \mathbb{I}(v_j = k), \quad (18.57)$$

где суммирование производится по всем нелистовым (внутренним) узлам, G_j – выигрыш в верности (уменьшение стоимости) в узле j , а $v_j = k$, если в узле j используется признак k . Мы можем получить более надежную оценку, усреднив по всем деревьям в ансамбле:

$$R_k = \frac{1}{M} \sum_{m=1}^M R_k(T_m). \quad (18.58)$$

Вычисленные оценки можно нормировать, так чтобы наибольшее значение было равно 100 %. Ниже приведено несколько примеров.

Пример на рис. 18.8 относится к оцениванию важности признаков для классификатора, обученного различать цифры 0 и 8 из набора MNIST. Мы видим, что он уделяет наибольшее внимание тем частям изображения, в которых эти классы различаются.

На рис. 18.9 показан график относительной важности признаков в наборе данных для распознавания спама (раздел 18.4). Неудивительно, что самыми важными являются слова «george» (имя получателя) и «hp» (компания, в которой он работает), а также символы ! и \$. (Отметим, что информативным является как присутствие, так и отсутствие этих признаков.)

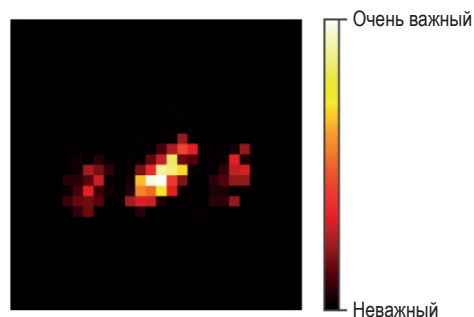


Рис. 18.8 ❖ Важность признаков классификатора в виде случайного леса, обученного распознавать цифры из набора MNIST, принадлежащие классам 0 и 8. На основе рис. 7.6 из [Gér19]. Построено программой по адресу figures.problm.ai/book1/18.8

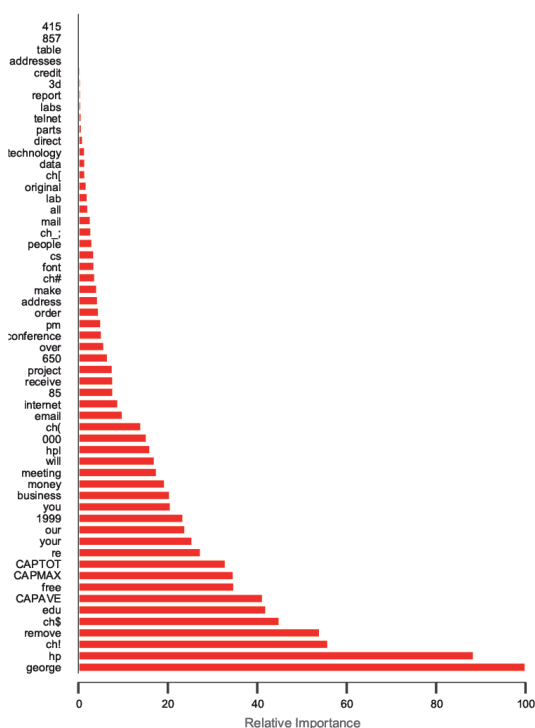


Рис. 18.9 ❖ Важность признаков для классификатора, методом градиентного бустинга обученного отличать спамные почтовые сообщения от нормальных. Набор данных содержит X обучающих примеров с Y признаками, соответствующими частоте токенов. На основе рис. 10.6 из работы [HTF09]. Построено программой по адресу figures.problm.ai/book1/18.9

18.6.2. Графики частичной зависимости

Идентифицировав наиболее релевантные входные признаки, мы можем попытаться оценить их влияние на выход. **Графиком частичной зависимости** для признака k называется график зависимости функции

$$\bar{f}_k(x_k) = \frac{1}{N} \sum_{n=1}^N f(\mathbf{x}_{n,-k}, x_k) \quad (18.59)$$

от x_k . Таким образом, мы исключаем путем суммирования все признаки, кроме k . В случае бинарного классификатора это выражение можно преобразовать в логарифм отношения шансов, $\log p(y = 1|x_k)/p(y = 0|x_k)$, перед тем как строить график. На рис. 18.10а это проиллюстрировано для четырех разных признаков из набора данных для распознавания спама. Мы видим, что по мере возрастания частоты «!» и «remove» возрастает и вероятность спама. Наоборот, при возрастании частоты «edu» или «hp» вероятность спама уменьшается.

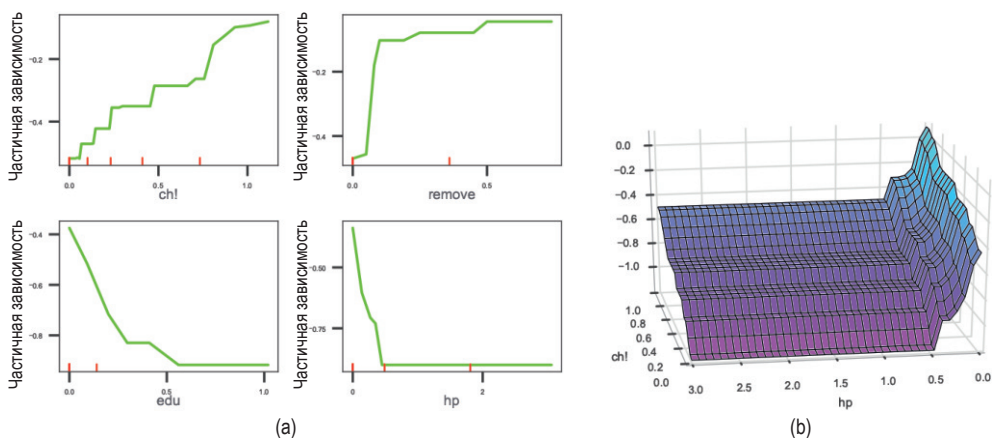


Рис. 18.10 ❖ (а) Частичная зависимость логарифма отношения шансов класса «спам» для четырех важных предикторов. Красные риски на горизонтальной оси – децили эмпирического распределения признака. (б) Совместная частичная зависимость логарифма отношения шансов от признаков hp и $!$. На основе рис. 10.6–10.8 из работы [HTF09]. Построено программой по адресу figures.probml.ai/book1/18.10

Можно также попытаться уловить эффекты взаимодействия между признаками j и k , вычислив выражение:

$$\bar{f}_{jk}(x_j, x_k) = \frac{1}{N} \sum_{n=1}^N f(\mathbf{x}_{n,-jk}, x_j, x_k). \quad (18.60)$$

Для примера со спамом его график для признаков «hp» и «!» показан на рис. 18.10б. Мы видим, что более высокая частота «!» делает сообщение больше похожим на спам, причем вероятность этого значительно возрастает, если слово «hp» отсутствует.

Часть **V**



ЗА ПРЕДЕЛАМИ ОБУЧЕНИЯ С УЧИТЕЛЕМ

Глава 19

Обучение при меньшем числе помеченных примеров

Во многих моделях МО, особенно в нейронных сетях, параметров гораздо больше, чем имеется помеченных обучающих примеров. Например, в CNN ResNet (раздел 14.3.4) с 50 слоями 23 млн параметров. А в моделях трансформеров (раздел 15.5) их может быть еще больше. Конечно, эти параметры сильно коррелированы, поэтому не являются независимыми «степенями свободы». Но все равно такие большие модели обучаются медленно и, что еще важнее, склонны к переобучению. Проблема становится особенно серьезной, если нет большого размеченного обучающего набора. В этой главе мы обсудим некоторые способы решения проблемы, помимо таких общих методов регуляризации, как ранняя остановка, уменьшение весов и прореживание (см. раздел 13.5).

19.1. ПРИРАЩЕНИЕ ДАННЫХ

Предположим, что имеется всего один небольшой размеченный набор данных. Иногда можно создать искусственно модифицированные версии входных векторов, улавливающие вариации, которые мы ожидаем увидеть на этапе тестирования, не меняя при этом оригинальных меток. Это называется **приращением данных** (data augmentation)¹. Ниже мы приведем несколько примеров, а затем обсудим, почему этот подход работает.

19.1.1. Примеры

Для задач классификации изображений в число стандартных методов приращения данных входят случайное кадрирование, изменение масштаба

¹ Термин «приращение данных» используется также в статистике, где обозначает добавление в модель вспомогательных латентных переменных с целью ускорить сходимость алгоритмов апостериорного вывода [DM01].

и зеркальное отражение (см. рис. 19.1). В работе [GVZ16] приведен более сложный пример, когда поверх изображения размещаются символы текста, что позволяет создавать очень большие наборы реалистичных текстовых данных. Эта техника была использована для обучения современных систем нахождения и распознавания текста на изображениях. Из других примеров приращения данных упомянем искусственное добавление фонового шума к ясным речевым сигналам и случайную замену символов или слов в текстовых документах.



Рис. 19.1 ❖ Иллюстрация случайного кадрирования и изменения масштаба изображения. Построено программой по адресу figures.problml.ai/book1/19

Если мы можем позволить себе многократное обучение и тестирование модели на различных версиях данных, то можно выяснить, какие приращения дают наилучший эффект, применяя такие методы оптимизации, как обучение с подкреплением (см., например, [Cub+19]) или байесовскую оптимизацию (см., например, [Lim+19]); этот подход называется **AutoAugment**. Можно также обучиться комбинированию нескольких приращений, это называется **AugMix** [Hen+20].

Примеры приращения в обработке естественного языка см., например, в работе [Fen+21].

19.1.2. Теоретическое обоснование

Приращение данных часто значительно улучшает качество (верность предсказаний, робастность и т. д.). На первый взгляд может показаться, что мы получаем что-то даром, потому что никаких дополнительных данных не предоставили. Однако механизм приращения данных можно рассматривать как способ алгоритмического включения априорных знаний.

Чтобы убедиться в этом, вспомним, что при стандартном ERM-обучении мы стремимся минимизировать эмпирический риск:

$$R(f) = \int \ell(f(\mathbf{x}), \mathbf{y}) p^*(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y}, \quad (19.1)$$

где $p^*(\mathbf{x}, \mathbf{y})$ аппроксимируется эмпирическим распределением:

$$p_{\mathcal{D}}(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^N \delta(\mathbf{x} - \mathbf{x}_n) \delta(\mathbf{y} - \mathbf{y}_n). \quad (19.2)$$

Приращение данных можно интерпретировать как замену эмпирического распределения следующим алгоритмически сглаженным распределением:

$$p_D(\mathbf{x}, y | A) = \frac{1}{N} \sum_{n=1}^N p(\mathbf{x} | \mathbf{x}_n, A) \delta(y - y_n), \quad (19.3)$$

где A – алгоритм приращения данных, который порождает пример \mathbf{x} по обучающему примеру \mathbf{x}_n , не изменяя метку последнего («семантику»). (Очень простой пример дает гауссово ядро, $p(\mathbf{x} | \mathbf{x}_n, A) = \mathcal{N}(\mathbf{x} | \mathbf{x}_n, \sigma^2 \mathbf{I})$.) Это называется **локальной минимизацией риска** (vicinal risk minimization) [Cha+01], потому что мы минимизируем риск в окрестности (vicinity) каждой обучающей точки \mathbf{x} . Дополнительные сведения о таком подходе см. в работах [Zha+17b; CDL19; Dao+19].

19.2. ПЕРЕНОС ОБУЧЕНИЯ

Этот раздел написан в соавторстве с Колином Раффелем.

Для многих задач, в которых данных не хватает, верхнеуровневая структура выглядит почти так же, как в задачах с достаточным объемом данных. Рассмотрим, к примеру, задачу **точной визуальной классификации** птиц, находящихся под угрозой вымирания. Учитывая, что такие птицы по определению встречаются редко, маловероятно, что удастся собрать много разнообразных помеченных изображений. Однако структурные черты птиц во многом одинаковы – у большинства из них есть крылья, оперение, клюв, когти и т. д. Поэтому можно ожидать, что если сначала обучить модель на большом наборе данных о птицах вообще, а потом продолжить обучение на небольшом наборе данных о птицах, находящихся под угрозой вымирания, то качество будет лучше, чем при обучении только на малом наборе.

Это называется **переносом обучения**, поскольку мы переносим информацию с одного набора данных на другой с помощью разделяемого набора параметров. Точнее, сначала выполняется **фаза предобучения**, в которой модель обучается с параметрами θ на большом исходном наборе данных \mathcal{D}_p ; он может быть размеченным или неразмеченным. Затем наступает черед **фазы дообучения** на малом **целевом наборе** представляющих интерес данных \mathcal{D}_q . Ниже мы обсудим обе фазы более подробно, а дополнительные сведения можно найти, например, в недавних обзорах [Tan+18; Zhu+19].

19.2.1. Дообучение

Временно предположим, что уже имеется предобученный классификатор $p(y | \mathbf{x}, \theta_p)$, например СНС, который хорошо работает для входов $\mathbf{x} \in \mathcal{X}_p$ (например, естественных изображений) и выходов $y \in \mathcal{Y}_p$ (например, меток ImageNet), причем данные выбираются из распределения $p(\mathbf{x}, y)$, похожего на то, что использовалось на этапе обучения. Мы хотим создать новую модель

$q(y|\mathbf{x}, \boldsymbol{\theta}_q)$, которая хорошо работает для входов $\mathbf{x} \in \mathcal{X}_q$ (например, изображений птиц) и выходов $y \in \mathcal{Y}_q$ (например, уточненных меток птиц), причем распределение данных $q(\mathbf{x}, y)$ может отличаться от p .

Мы предполагаем, что множество возможных входов то же самое, т. е. $\mathcal{X}_q \approx \mathcal{X}_p$ (например, это RGB-изображения), или что можно легко преобразовать входы из домена p во входы из домена q (например, преобразовать RGB-изображения в полутоновые, отбросив каналы цветности и оставив только канал яркости). (Если это не так, то, возможно, придется использовать какой-нибудь метод адаптации домена, который модифицирует модели, производя отображение модальностей, как обсуждается в разделе 19.2.5.)

Однако выходные домены обычно различны, т. е. $\mathcal{Y}_q \neq \mathcal{Y}_p$. Например, \mathcal{Y}_p может быть метками ImageNet, а \mathcal{Y}_q медицинскими метками (скажем, типами диабетической ретинопатии [Arc+19]). В этом случае нужно «транслировать» выход предобученной модели в новый домен. Это легко сделать для нейронных сетей: мы просто «отсекаем» последний слой оригинальной модели и добавляем вместо него слой с новыми метками классов, как показано на рис. 19.2. Например, предположим, что $p(y|\mathbf{x}, \boldsymbol{\theta}_p) = S(y|\mathbf{W}_2\mathbf{h}(\mathbf{x}; \boldsymbol{\theta}_1) + \mathbf{b}_2)$, где $\boldsymbol{\theta}_p = (\mathbf{W}_2, \mathbf{b}_2, \boldsymbol{\theta}_1)$. Тогда можно построить распределение $q(y|\mathbf{x}, \boldsymbol{\theta}_q) = S(y|\mathbf{W}_3^T\mathbf{h}(\mathbf{x}, \boldsymbol{\theta}_1) + \mathbf{b}_3)$, где $\boldsymbol{\theta}_q = (\mathbf{W}_3, \mathbf{b}_3, \boldsymbol{\theta}_1)$ и $\mathbf{h}(\mathbf{x}, \boldsymbol{\theta}_1)$ – разделяемый нелинейный экстрактор признаков.

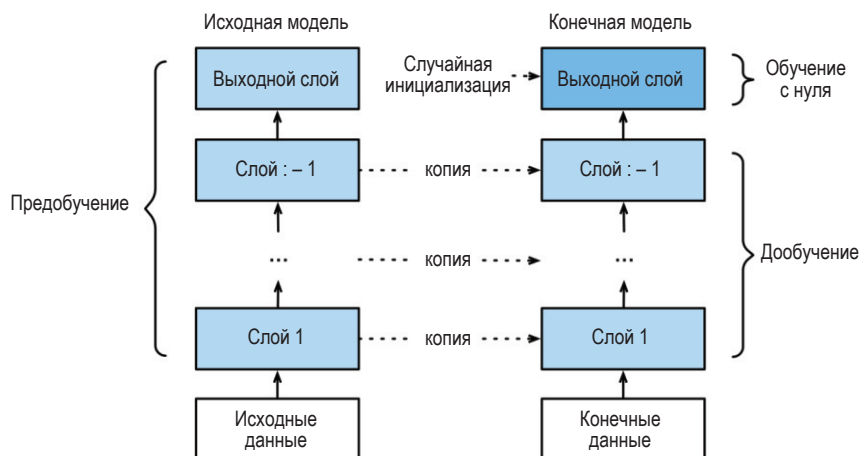


Рис. 19.2 ❖ Дообучение модели на новом наборе данных. Конечный выходной слой обучается с нуля, поскольку ему может соответствовать другой набор меток. Остальные слои инициализируются своими предыдущими параметрами, а затем могут быть обновлены с малой скоростью обучения. На основе рис. 13.2.1 из работы [Zha+20]. Печатается с разрешения Астона Чжана

Подвергнув модель такой «хирургической операции», мы сможем дообучить новую модель с параметрами $\boldsymbol{\theta}_q = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_3)$, где $\boldsymbol{\theta}_1$ – параметры экстрактора признаков, а $\boldsymbol{\theta}_3$ – параметры последнего линейного слоя, который отображает признаки в новый набор меток. Если рассматривать $\boldsymbol{\theta}_1$ как «замороженные параметры», то результирующая модель $q(y|\mathbf{x}, \boldsymbol{\theta}_q)$ линейна

относительно своих параметров, так что мы имеем задачу выпуклой оптимизации, для которой имеется много простых и эффективных методов обучения (см. часть II). Это особенно полезно в постановке с длинными хвостами, когда некоторые классы встречаются очень редко [Kan+20]. Однако линейный «декодер» может быть излишне ограничительным, поэтому можно разрешить и дообучение параметров θ_1 , но с меньшей скоростью обучения, чтобы предотвратить слишком сильное отдаление от значений, обученных на наборе \mathcal{D}_p .

19.2.2. Адаптеры

Дообучение всех параметров предобученной модели может оказаться слишком медленным, поскольку параметров зачастую чересчур много, а кроме того, обучение, возможно, придется проводить с малой скоростью, чтобы низкоровневые экстракторы признаков не ушли слишком далеко от априорных значений. Кроме того, для каждой новой задачи придется обучать новую модель, что затруднит разделение задач. Альтернативный подход – не трогать предобученную модель вообще, но добавить новые параметры, чтобы модифицировать ее внутреннее поведение, настроив таким образом процесс извлечения признаков для каждой задачи. Эта идея **адаптеров** исследовалась в нескольких работах (например, [RBV17; RBV18; Hou+19]).

На рис. 19.3а показаны адаптеры для трансформерных сетей (раздел 15.5), предложенные в работе [Hou+19]. Идея в том, чтобы вставить два неглубоких МСП в каждый слой трансформера: один после слоя многопутевого внимания, а второй после слоев прямого распространения. Заметим, что в этих МСП имеются прямые связи, так что их можно инициализировать для реализации тождественного отображения. Если признаки в слое трансформера имеют размерность D , а размер слоев МСП в адаптере равен M , то в каждый слой добавляется $O(DM)$ новых параметров. Эти адаптерные МСП, а также параметры слоя нормировки и конечный выходной слой обучаются для каждой новой задачи, но все остальные параметры заморожены. Эмпирически на нескольких тестах NLP было показано, что качество результирующей сети получается лучше, чем при дообучении, при этом число новых параметров составляет всего от 1 до 10 % от количества первоначальных.

На рис. 19.3б показаны адаптеры для остаточных сетей (раздел 14.3.4), предложенные в работах [RBV17; RBV18]. Идея в том, чтобы добавить к внутренним слоям СНС сверточный слой α размера 1×1 , играющий ту же роль, что МСП-адаптер в случае трансформера. Его можно добавить последовательно или параллельно, как показано на диаграмме. Если обозначить адаптерный слой $\rho(\mathbf{x})$, то можно будет определить последовательный адаптер в виде

$$\rho(\mathbf{x}) = \mathbf{x} + \text{diag}_1(\alpha) \circledast \mathbf{x} = \text{diag}_1(\mathbf{I} + \alpha) \circledast \mathbf{x}, \quad (19.4)$$

где $\text{diag}_L(\mathbf{A}) \in \mathbb{R}^{L \times L \times C \times D}$ преобразует матрицу $\mathbf{A} \in \mathbb{R}^{C \times D}$ в банк поканальных фильтров. (Для простоты мы опустили пакетную нормировку.) Вставив такой адаптер после регулярного сверточного слоя $\mathbf{f} \circledast \mathbf{x}$, получим

$$\mathbf{y} = \rho(\mathbf{f} \circledast \mathbf{x}) = (\text{diag}_1(\mathbf{I} + \alpha) \circledast \mathbf{f}) \circledast \mathbf{x}. \quad (19.5)$$

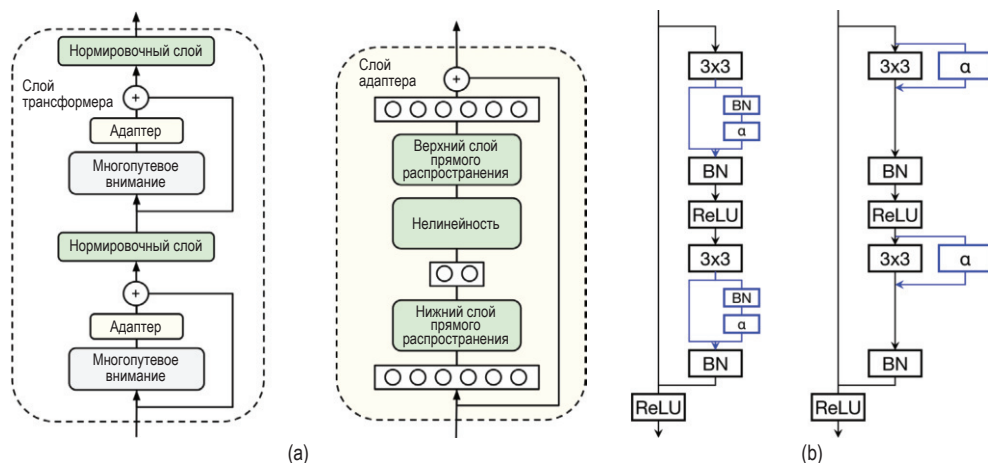


Рис. 19.3 ❖ (а) Добавление адаптерных слоев в трансформер. На основе рис. 2 из работы [Hou+19]. Печатается с разрешения Нила Хоулси. (б) Добавление адаптерных слоев в сеть ResNet. На основе рис. 2 из работы [RBV18]. Печатается с разрешения Сильвестра-Элвиса Ребаффи

Это можно интерпретировать как низкоранговое мультипликативное возмущение оригинального фильтра f .

Параллельный адаптер можно определить так:

$$y = f \circledast x + \text{diag}_1(\alpha) \circledast x = (f + \text{diag}_L(\alpha)) \circledast x. \quad (19.6)$$

Это можно интерпретировать как низкоранговое аддитивное возмущение оригинального фильтра f . В обоих случаях если положить $\alpha = 0$, то адаптерные слои можно будет инициализировать тождественным преобразованием. Кроме того, оба метода требуют $O(C^2)$ параметров на один слой.

19.2.3. Предобучение с учителем

Предобучать модель можно как с учителем, так и без него; основные требования – чтобы модель могла обучиться базовой структуре предметной области и чтобы решаемая задача была в достаточной степени похожа на последующую задачу дообучения. У понятия «похожие задачи» нет строгого определения, но на практике домен задачи предобучения часто шире, чем задачи дообучения (например, предобучение производится на птицах всех видов, а дообучение – только на птицах под угрозой вымирания).

Самая незамысловатая форма переноса обучения – случай, когда имеется большой размеченный набор данных, подходящий для предобучения. Например, очень часто используют набор данных ImageNet (раздел 1.5.1.2) для предобучения СНС, которую затем можно использовать для различных задач и наборов данных (см., например, работу [Kol+19]). Набор ImageNet включает 1.28 млн естественных изображений, с каждым из которых ассоциирован один из 1000 классов. Классы представляют самые разные объекты, в том

числе животных, еду, здания, музыкальные инструменты, одежду и т. д. Сами изображения также отличаются разнообразием в том смысле, что объекты могут быть повернуты под разными углами, быть разного размера и располагаться на разном фоне. Такое разнообразие форм и масштабов может частично объяснить, почему этот набор стал задачей предобучения де факто для переноса обучения в компьютерном зрении (см. демонстрационный код по адресу code.problml.ai/book1/finetune_cnn_torch).

Однако было показано, что предобучение на наборе ImageNet не столь полезно, если домен задачи дообучения резко отличается от естественных изображений (например, если это медицинские изображения [Rag+19]). А в некоторых случаях оно если и полезно (например, при обучении систем обнаружения объектов), то скорее как способ ускорения работы (благодаря хорошему выбору начальных параметров оптимизации), нежели нечто существенно значимое, в том смысле, что сравнимого качества на последующей задаче можно достичь и обучая модель с нуля, если потратить на это достаточно времени [HGD19].

В приложениях, не связанных с компьютерным зрением, предобучение с учителем встречается реже. Заметное исключение – предобучение на данных естественно-языкового вывода (когда требуется понять, вытекает одно предложение из другого или противоречит ему) с целью обучения векторных представлений предложений [Con+17], хотя этот подход почти полностью вытеснен методами обучения без учителя (раздел 19.2.4). Еще одно не относящееся к компьютерному зрению приложение переноса обучения – предобучение системы распознавания речи на большом корпусе размеченных английских текстов с последующим дообучением на языках, для которых имеется меньше ресурсов [Ard+20].

19.2.4. Предобучение без учителя (самостоятельное обучение)

Предобучение без учителя используется все чаще, поскольку непомеченные данные бывает легко собрать; например, это могут быть непомеченные изображения или текстовые документы из веба.

В течение недолгого времени было модно предобучать глубокие нейронные сети с помощью целевой функции без учителя (например, ошибки реконструкции, обсуждаемой в разделе 20.3) на размеченном наборе данных (т. е. игнорируя метки), а затем переходить к стандартному обучению с учителем [HOT06; Vin+10b; Erh+10]. Хотя эту технику также называют предобучением без учителя, она отличается от той формы предобучения, которая обсуждается в этом разделе в контексте переноса обучения, – когда используется (большой) неразмеченный набор данных для предобучения, а затем модель дообучается на другом (меньшем) размеченном наборе.

Задачи предобучения с непомеченными данными часто называют **самостоятельным обучением** (self-supervised), а не обучением без учителя. Этот термин призван подчеркнуть, что метки создаются самим алгоритмом, а не

предоставляются извне человеком, как в стандартном обучении с учителем. И обучение с учителем, и самостоятельное обучение – дискриминантные задачи, поскольку требуется предсказать выходы по входам. Напротив, методы обучения без учителя, рассмотренные, в частности, в главе 20, являются порождающими, потому что предсказывают выходы безусловно.

Существует много испытанных эвристик самостоятельного обучения (см., например, обзоры [GR18; JT19; Ren19] и обширный список статей по адресу <https://github.com/jason718/awesome-self-supervised-learning>). Можно выделить по крайней мере три широкие группы, которые будут рассмотрены ниже.

19.2.4.1. Задачи подстановки

Один из подходов к самостоятельному обучению – решение **задачи подстановки** (imputation). В этом случае мы разбиваем входной вектор \mathbf{x} на две части, $\mathbf{x} = (\mathbf{x}_h, \mathbf{x}_v)$, а затем пытаемся предсказать скрытую часть \mathbf{x}_h по видимой части \mathbf{x}_v , пользуясь моделью вида $\hat{\mathbf{x}}_h = f(\mathbf{x}_v, \mathbf{x}_h = 0)$. Можно считать, что это задача «заполнения пропусков»; в англоязычной части сообщества NLP употребляется также термин **cloze task**. На рис. 19.4 приведены примеры из области компьютерного зрения, а в разделе 15.7.2 – из области NLP.

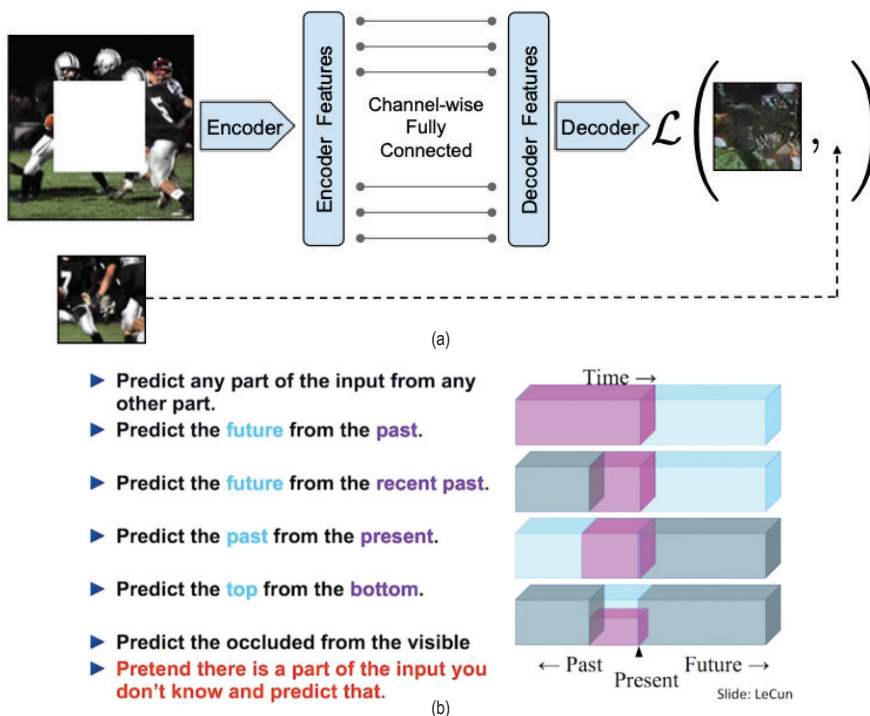


Рис. 19.4 ❖ (а) Контекстный кодировщик для самостоятельного обучения. Взято из работы [Pat+16]. Печатается с разрешения Дипака Патхака. (б) Другие замещающие задачи для самообучения. Взято из работы [LeC18]. Печатается с разрешения Яна Лекуна

19.2.4.2. Замещающие задачи

Другой подход к самостоятельному обучению – использование **замещающих** (proxy task), или **оправдывающих** (pretext task), **задач**. В этом случае мы создаем пары входов ($\mathbf{x}_1, \mathbf{x}_2$), а затем обучаем сиамскую сеть (рис. 16.5а), т. е. классификатор вида $p(y|\mathbf{x}_1, \mathbf{x}_2) = p(y|r[f(\mathbf{x}_1), f(\mathbf{x}_2)])$, где $f(\mathbf{x})$ – функция, выполняющая «**обучение представлений**» [BCV13], а y – метка, улавливающая связь между \mathbf{x}_1 и \mathbf{x}_2 , которую предсказывает $r(f_1, f_2)$. Например, предположим, что \mathbf{x}_1 – патч изображения, а $\mathbf{x}_2 = t(\mathbf{x}_1)$ – некоторое контролируемое нами преобразование \mathbf{x}_1 , скажем случайное вращение; тогда мы определяем у как угол поворота этого вращения [GSK18].

19.2.4.3. Сопоставительные задачи

В настоящее время самый популярный подход к самостоятельному обучению связан с использованием различных **сопоставительных задач** (contrastive task). Идея в том, чтобы создать пары семантически похожих примеров, применяя различные методы приращения данных (раздел 19.1), а затем сделать так, чтобы расстояние между их представлениями было меньше (в пространстве погружений), чем расстояние между двумя несвязанными примерами. Это та же идея, что в глубоком обучении метрики (раздел 16.2.2), – единственное различие заключается в том, что алгоритм самостоятельно создает похожие пары, а не полагается на внешнюю меру сходства, например метки. Примеры будут приведены в разделах 19.2.4.4 и 19.2.4.5.

19.2.4.4. SimCLR

В этом разделе мы обсудим метод **SimCLR** – Simple contrastive learning of visual representations (простое сопоставительное обучение визуальных представлений) [Che+20b; Che+20c]. Он обеспечивает лучшее на сегодняшний день качество в задачах переноса обучения и обучения с частичным привлечением учителя. Основная идея следующая. Каждый вход $\mathbf{x} \in \mathbb{R}^D$ преобразуется в два дополненных «вида» $\mathbf{x}_1 = t_1(\mathbf{x})$, $\mathbf{x}_2 = t_2(\mathbf{x})$, которые являются его «семантически эквивалентными» версиями, генерируемыми некоторыми преобразованиями t_1, t_2 . Например, если \mathbf{x} – изображение, то это могли бы быть его небольшие возмущения, скажем случайные кадрирования (см. раздел 19.1). Кроме того, мы производим выборку «отрицательных» примеров $\bar{\mathbf{x}}_1, \dots, \bar{\mathbf{x}}_n \in N(\mathbf{x})$ из набора данных, представляющего «семантически различные» изображения (на практике это другие примеры в мини-пакете). Затем мы определяем некоторое отображение признаков $F: \mathbb{R}^D \rightarrow \mathbb{R}^E$, где D – размер входа, а E – размер погружения. Далее мы пытаемся максимизировать сходство похожих видов и одновременно минимизировать сходство различных видов для каждого входа \mathbf{x} :

$$J = F(t_1(\mathbf{x}))^T F(t_2(\mathbf{x})) - \log \sum_{\mathbf{x}_i^- \in N(\mathbf{x})} \exp[F(\mathbf{x}_i^-)^T F(t_1(\mathbf{x}))]. \quad (19.7)$$

На практике в качестве меры сходства используется коэффициент Отиаи, поэтому мы нормируем порожденные F представления по норме ℓ_2 , прежде

чем вычислять скалярные произведения, но в формуле выше это действие опущено (см. иллюстрацию на рис. 19.5a). (На этом рисунке предполагается, что $F(\mathbf{x}) = g(f(\mathbf{x}))$, где промежуточное представление $\mathbf{h} = f(\mathbf{x})$ впоследствии будет использовано для дообучения, а g – дополнительное преобразование, примененное в процессе обучения.)

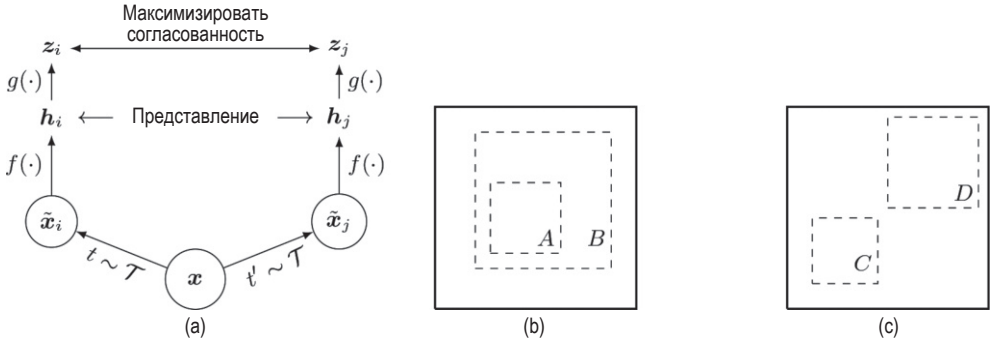


Рис. 19.5 ❖ (a) Иллюстрация обучения SimCLR. T – множество стохастических преобразований, сохраняющих семантику (приращений данных). (b–c) Преимущества случайных кадрирований. Сплошные прямоугольники представляют оригинальное изображение, штриховые – случайные кадрирования. На рисунке (b) модель вынужденно предсказывает локальный вид A по глобальному виду B (и наоборот). На рисунке (c) модель вынужденно предсказывает внешний вид соседних видов (C, D). На основе рис. 2–3 из работы [Che+20b]. Печатается с разрешения Тин Чена

Интересно, что это можно интерпретировать как форму условной **модели на основе энергии** вида

$$p(\mathbf{x}_2|\mathbf{x}_1) = \frac{\exp[-\Psi(\mathbf{x}_2|\mathbf{x}_1)]}{Z(\mathbf{x}_1)}, \quad (19.8)$$

где $\Psi(\mathbf{x}_2|\mathbf{x}_1) = -F(\mathbf{x}_2)^T F(\mathbf{x}_1)$ – энергия, а

$$Z(\mathbf{x}) = \int \exp[-\Psi(\mathbf{x}^-|\mathbf{x})] d\mathbf{x}^- = \int \exp[F(\mathbf{x}^-)^T F(\mathbf{x})] d\mathbf{x}^- \quad (19.9)$$

– нормировочная постоянная, называемая **функцией разбиения**. Условное логарифмическое правдоподобие при такой модели имеет вид:

$$\log p(\mathbf{x}_2|\mathbf{x}_1) = -F(\mathbf{x}_2)^T F(\mathbf{x}_1) - \log \int \exp[F(\mathbf{x}^-)^T F(\mathbf{x}_1)] d\mathbf{x}^-. \quad (19.10)$$

Единственное отличие от формулы (19.7) в том, что мы заменяем интеграл верхней границей Монте-Карло, выведенной по отрицательным примерам. Поэтому сопоставительное обучение можно рассматривать как оценку максимального правдоподобия условной порождающей модели на основе энергии [Gra+20]. Дополнительные сведения о таких моделях можно найти во втором томе этой книги, [Mur22].

Важнейшим условием успеха SimCLR является выбор методов приращения данных. Применяя случайное кадрирование, можно заставить модель предсказывать локальные виды по глобальным, а также соседние виды одного и того же изображения (см. рис. 19.5). После кадрирования восстанавливается исходный размер всех изображений. Кроме того, некоторые случайно выбранные изображения подвергаются зеркальному отражению¹.

SimCLR требует обучения на большом пакете, чтобы множество отрицательных примеров было достаточно разнообразным. Если это невозможно, то можно использовать банк прошлых (отрицательных) погружений и обновлять его, применяя экспоненциальное скользящее усреднение (раздел 4.4.2.2). Это называется **импульсным сопоставительным обучением** (momentum contrastive learning), или **MoCo** [He+20].

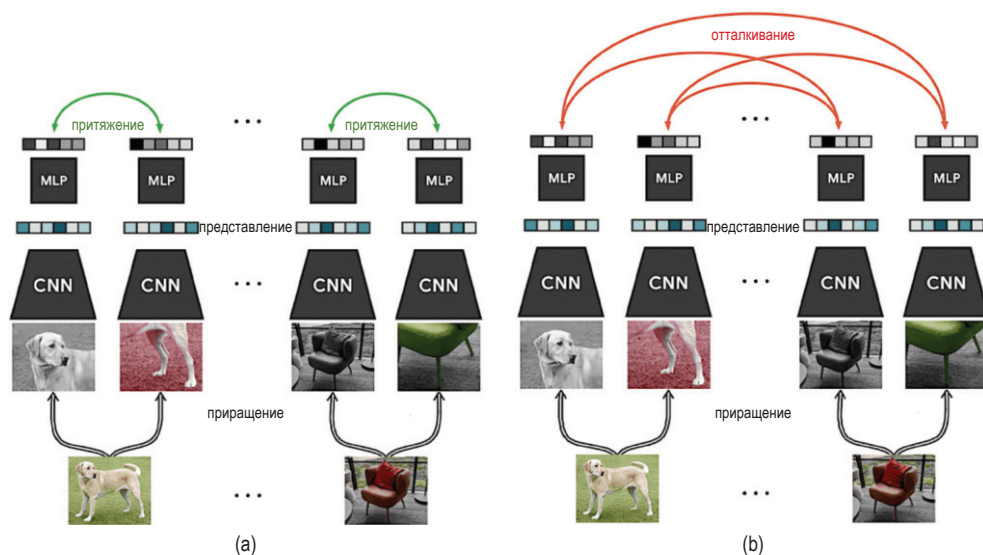


Рис. 19.6 ❖ Визуализация обучения SimCLR. Каждое входное изображение в мини-пакете случайно модифицируется двумя разными способами (кадрирование с последующим изменением размера, отражение и искажение цветов), а затем подается на вход сиамской сети. Погружения (последний слой) для каждой пары, выведенные по одному и тому же изображению, вынужденно близки, тогда как погружения для всех остальных пар вынужденно далеки. Из статьи <https://ai.googleblog.com/2020/04/advancing-self-supervised-and-semi.html>. Печатается с разрешения Тин Ченя

¹ Оказывается, что отличить положительные кадрирования (полученные из одного и того же изображения) от отрицательных (полученных из разных отображений) зачастую можно, просто проанализировав гистограммы цветов. Чтобы не допустить такого «обмана», добавляют также случайное искажение цветов, перекрывая тем самым «короткий путь». Показано, что такая комбинация случайного кадрирования и искажения цветов работает лучше, чем каждый метод в отдельности.

19.2.4.5. CLIP

В этом разделе мы опишем технологию **CLIP** – Contrastive Language-Image Pre-training (сопоставительное предобучение на парах изображение–текст) [Rad+]. Это сопоставительный подход к обучению представлений на большом (объемом 400 МБ) корпусе пар (изображение, текст), полученных из веба. Пусть \mathbf{x}_i – i -е изображение и \mathbf{y}_i – соответствующий ему текст. Вместо того чтобы пытаться предсказать точные слова, ассоциированные с изображением, проще решить, правда ли, что \mathbf{y}_i с большей вероятностью является правильным текстом, чем какая-то другая текстовая строка \mathbf{y}_j из того же мини-пакета. Аналогично модель может попытаться решить, правда ли, что изображение \mathbf{x}_i с большей вероятностью соответствует тексту \mathbf{y}_i , чем изображение \mathbf{x}_j .

Точнее, пусть $f_I(\mathbf{x}_i)$ – погружение изображения, $f_T(\mathbf{y}_j)$ – погружение текста, $\mathbf{I}_i = f_I(\mathbf{x}_i)/\|f_I(\mathbf{x}_i)\|_2$ – нормированное на единицу погружение изображения, а $\mathbf{T}_j = f_T(\mathbf{y}_j)/\|f_T(\mathbf{y}_j)\|_2$ – нормированное на единицу погружение текста. Определим вектор попарных логитов (оценок сходства):

$$L_{ij} = \mathbf{I}_i^T \mathbf{T}_j. \quad (19.11)$$

Теперь обучим параметры двух функций погружения f_I и f_T с целью минимизировать следующую потерю, усредненную по мини-пакетам размера N :

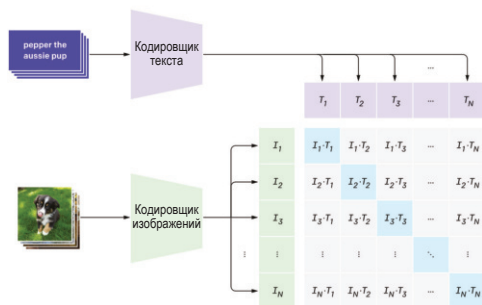
$$J = \frac{1}{2} \left[\sum_{i=1}^N \text{CE}(\mathbf{L}_{i,:}, \mathbf{1}_i) + \sum_{j=1}^N \text{CE}(\mathbf{L}_{:,j}, \mathbf{1}_j) \right], \quad (19.12)$$

где CE – потеря перекрестной энтропии

$$\text{CE}(\mathbf{p}, \mathbf{q}) = - \sum_{k=1}^K p_k \log q_k, \quad (19.13)$$

а $\mathbf{1}_i$ – унитарный код метки i , см. иллюстрацию на рис. 19.7а. (На практике нормированные погружения умножаются на параметр температуры, который тоже обучается; он контролирует остроту softmax.)

1. Сопоставительное предобучение



2. Создать классификатор набора данных по тексту метки



3. Использование для предсказания с первой попытки

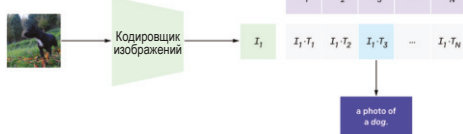


Рис. 19.7 ❖ Модель CLIP. На основе рис. 1 из работы [Rad+].
Печатается с разрешения Алека Рэдфорда

В своей работе авторы рассматривали сеть ResNet (раздел 14.3.4), а также трансформер изображений (раздел 15.5.6) в качестве функции f_I и трансформер текста (раздел 15.5) в качестве f_T . Они использовали очень большой мини-пакет размера $N \sim 32k$ и обучали сеть на протяжении многих дней с использованием сотен GPU.

Обученную модель можно использовать для **классификации** изображения x **без примеров** следующим образом. Во-первых, каждая из K возможных меток классов для заданного набора данных преобразуется в текстовую строку y_k , которая, возможно, встречалась в вебе. Например, «dog» преобразуется в «a photo of a dog». Во-вторых, вычисляются нормированные погружения $I \propto f_I(x)$ и $T_k \propto f_T(y_k)$. В-третьих, вычисляются вероятности softmax:

$$p(y = k|x) = \mathcal{S}([I^T T_1, \dots, I^T T_k])_k. \quad (19.14)$$

Смотрите иллюстрацию на рис. 19.7b. (Похожий подход был принят в работе по визуальным n -граммам [Li+17].)

Удивительно, что этот подход может давать такие же хорошие результаты, как стандартное обучение с учителем на задачах типа классификации ImageNet даже без явного обучения на конкретных размеченных наборах данных. Конечно, изображения, включенные в ImageNet, взяты из веба и были найдены с помощью веб-поиска по тексту, так что модель видела хорошие примеры раньше. Тем не менее ее способность к обобщению на новые задачи и робастность к дрейфу распределения впечатляют (см. примеры в оригинальной статье).

Но у этого подхода есть и недостаток – чувствительность к способу преобразования меток классов в текстовую форму. Например, чтобы модель работала для классификации пищевых продуктов, необходимо использовать строки вида «a photo of guacamole, a type of food», «a photo of ceviche, a type of food» и т. д.

Устраняющие неоднозначность фразы типа «a type of food» (пищевой продукт) в настоящее время добавляются вручную для каждого набора данных в отдельности. Это называется **конструированием описаний** (prompt engineering) и необходимо, потому что первичные имена классов могут иметь разную семантику в разных классах (а иногда даже в одном классе).

19.2.5. Адаптация домена

Рассмотрим задачу, в которой имеются входные данные из разных доменов, например **исходного домена** \mathcal{X}_s и **целевого домена** \mathcal{X}_t , но общее множество выходных меток, \mathcal{Y} . (Эта задача «двойственна» по отношению к переносу обучения, потому что входные домены различны, а выходные одинаковы.) Например, доменами могут быть изображения из системы компьютерной графики и реальные изображения или отзывы о товарах и отзывы о фильмах. Мы предполагаем, что для целевого домена нет помеченных примеров. Наша цель – обучить модель на исходном домене, а затем модифицировать ее параметры, так чтобы модель работала на целевом домене. Это называется **адаптацией домена** (без учителя) (см., например, обзор [KL19]).

Общий подход к этой проблеме – обучить исходный классификатор, так чтобы он не мог отличить, из какого распределения поступил вход: исходного или целевого. В этом случае он сможет использовать только признаки, общие для обоих доменов. Это называется **сопоставительным обучением доменов** (domain adversarial learning) [Gan+16]. Формально пусть $d_n \in \{s, t\}$ – метка, определяющая, поступил ли пример n из домена s или t . Мы хотим оптимизировать целевую функцию

$$\min_{\phi} \max_{\theta} \frac{1}{N_s + N_t} \sum_{n \in \mathcal{D}_s, \mathcal{D}_t} \ell(d_n, f_{\theta}(\mathbf{x}_n)) + \frac{1}{N_s} \sum_{m \in \mathcal{D}_s} \ell(y_m, g_{\phi}(f_{\theta}(\mathbf{x}_m))), \quad (19.15)$$

где $N_s = |\mathcal{D}_s|$, $N_t = |\mathcal{D}_t|$, $f: \mathcal{X}_s \cup \mathcal{X}_t \rightarrow \mathcal{H}$ и $g: \mathcal{H} \rightarrow \mathcal{Y}_t$. Функция (19.15) минимизирует потерю на основной задаче классификации y , но *максимизирует* потерю на вспомогательной задаче классификации исходного домена d . Это можно реализовать с помощью **трюка обращения знака градиента**, связанного с порождающими сопоставительными сетями (generative adversarial network – GAN). О некоторых других подходах к адаптации домена см., например, работы [Csu17; Wu+19].

19.3. ОБУЧЕНИЕ С ЧАСТИЧНЫМ ПРИВЛЕЧЕНИЕМ УЧИТЕЛЯ

Этот раздел написан в соавторстве с Колином Раффелем.

Многие из недавних успешных приложений машинного обучения относятся к категории обучения с учителем, когда для обучения модели доступен большой набор помеченных примеров. Однако на практике получение таких помеченных данных часто обходится дорого. Рассмотрим автоматическое распознавание речи: современные наборы данных включает тысячи часов аудиозаписей [Pan+15; Ard+20]. Процесс аннотирования произносимых слов во много раз медленнее, чем режим реального времени, и может занимать очень много времени (и соответственно стоить). Хуже того, в некоторых приложениях к разметке данных необходимо привлекать специалистов (скажем, врачей, если речь идет о медицинском приложении), а это еще больше увеличивает цену.

Обучение с частичным привлечением учителя может уменьшить потребность в помеченных данных за счет привлечения непомеченных. Общая цель обучения с частичным привлечением учителя – дать модели возможность обучиться общей структуре распределения данных на непомеченных данных, а помеченные использовать только для обучения тонким деталям конкретной задачи. Если в стандартном обучении с учителем мы предполагаем, что имеется доступ к примерам из совместного распределения данных и меток $\mathbf{x}, y \sim p(\mathbf{x}, y)$, то в обучении с частичным привлечением учителя предполагается, что дополнительно имеется доступ к маргинальному распределению \mathbf{x} , а именно $\mathbf{x} \sim p(\mathbf{x})$. Кроме того, в общем случае предполагается,

что таких немеченных примеров гораздо больше, поскольку получить их, как правило, проще. Возвращаясь к распознаванию речи, отметим, что часто намного дешевле просто записать разговор людей (это немеченные данные), чем переводить записанную речь в текст. Обучение с частичным привлечением учителя хорошо приспособлено для сценария, когда уже собрано много немеченных данных, и желательно обойтись без того, чтобы помечать их все.

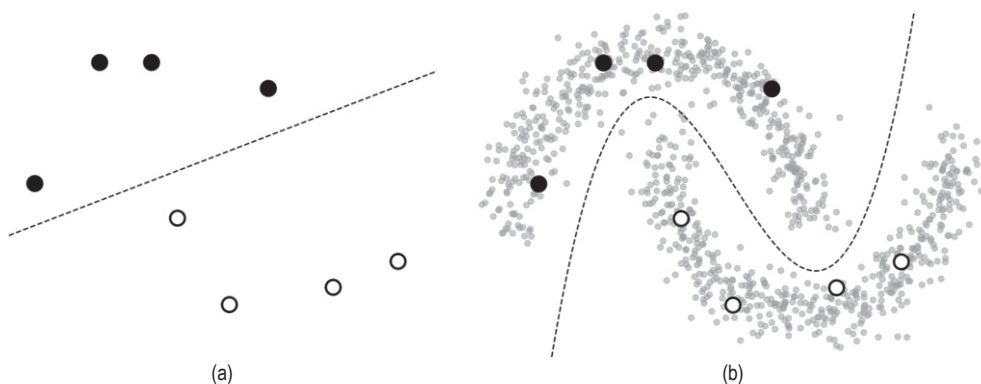


Рис. 19.18 ❖ Преимущества обучения с частичным привлечением учителя для решения задачи бинарной классификации. Помеченные точки из каждого класса обозначены черными и белыми кружочками соответственно. (a) Решающая граница, которую можно было бы обучить, имея только немеченные данные. (b) Решающая граница, которую можно было бы обучить, имея также много немеченных примеров, показанных маленькими серыми кружочками

19.3.1. Самообучение и псевдопометка

Ранний и прямолинейный подход к обучению с частичным привлечением учителя называется **самообучением** (self-training)¹ [Scu65; Agr70; McL75]. Основная идея самообучения – использовать саму модель для вывода предсказания по немеченным данным, а затем использовать эти предсказания как метки для последующего обучения. Самообучение стало признанным методом обучения с частичным привлечением учителя в силу своей простоты и универсальной применимости; это значит, что оно применимо к любой модели, которая умеет генерировать предсказания для немеченных данных. Недавно вошло в моду называть этот подход **псевдопометкой** (pseudo-labeling) [Lee13], потому что выведенные для немеченных данных метки лишь «псевдоправильны» по сравнению с истинными целевыми метками, применяемыми в обучении с учителем.

Алгоритмически самообучение обычно сводится к одной из двух процедур. В первом случае псевдометки сначала предсказываются для всего

¹ Не путать с «самостоятельным обучением» (self-supervised learning) (см. раздел 19.2.4). – Прим. перев.

набора непомеченных данных и модель переобучается (возможно, с нуля) до сходимости на комбинации помеченных и непомеченных (псевдопомеченных) данных. Затем непомеченные данные заново помечаются моделью, и процесс повторяется, пока не будет найдено подходящее решение. Во втором случае вместо этого непрерывно генерируются предсказания на случайно выбранных пакетах непомеченных данных, и модель сразу же обучается на этих псевдометках. Оба подхода часто применяются на практике; первый «офлайновый» вариант особенно успешен при наличии огромного количества непомеченных данных [Yal+19; Xie+20], тогда как «онлайновый» подход часто используется как один из компонентов более изощренных методов обучения с частичным привлечением учителя [Soh+20]. Ни у одного варианта нет принципиальных преимуществ перед другим. Офлайновое самообучение может приводить к обучению модели на «устаревших» псевдометках, так как они обновляются только после сходимости модели. Зато онлайновая псевдопометка влечет повышенные вычислительные затраты, потому что подразумевает постоянную «переразметку» непомеченных данных.

Самообучению присуща очевидная проблема: если модель генерирует неправильные предсказания для непомеченных данных, а затем переобучается на этих неправильных предсказаниях, то она будет становиться все хуже и хуже, пока в итоге не обучится совершенно неверному решению. Эта проблема получила название **предвзятость подтверждения** [TV17], потому что модель раз за разом подтверждает свое собственное (неверное) мнение о решающем правиле.

Распространенный способ уменьшить предвзятость подтверждения – использовать «метрику выбора» [RHS05], которая применяет эвристики, пытаясь оставить только правильные псевдометки. Например, если модель выводит вероятности каждого из возможных классов, в роли метрики выбора часто выступает правило оставлять лишь те псевдометки, для которых наибольшая вероятность класса выше порога [Yar95; RHS05]. Если оценки вероятностей классов хорошо откалиброваны, то эта метрика выбора будет сохранять только метки, которые с высокой вероятностью правильны (по крайней мере, согласно модели). В зависимости от предметной области можно придумать и более сложные метки выбора.

19.3.2. Минимизация энтропии

Самообучение неявно поощряет модель к выводу низкоэнтропийных (т. е. с высокой степенью уверенности) предсказаний. Этот эффект особенно нагляден при онлайновом обучении с потерей перекрестной энтропии, когда модель минимизирует следующую функцию потерь \mathcal{L} на непомеченных данных:

$$\mathcal{L} = -\max_c \log p_\theta(y = c|\mathbf{x}), \quad (19.16)$$

где $p_\theta(y|\mathbf{x})$ – модельное распределение вероятностей классов при условии входа \mathbf{x} . Эта функция достигает минимума, когда модель назначает всю вероятность одному классу c^* , т. е. $p(y = c^*|\mathbf{x}) = 1$ и $p(y \neq c^*|\mathbf{x}) = 0$.

С этим методом обучения с частичным привлечением учителя тесно связан метод **минимизации энтропии** [GB05], который минимизирует следующую функцию потерь:

$$\mathcal{L} = -\sum_{c=1}^C p_{\theta}(y = c|\mathbf{x}) \log p(y = c|\mathbf{x}). \quad (19.17)$$

Заметим, что эта функция также достигает минимума, когда модель назначает всю вероятность одному классу. Мы можем сделать потерю минимизации энтропии (19.17) эквивалентной потере онлайнowego самообучения (19.16), заменив первый член $p_{\theta}(y = c|\mathbf{x})$ унитарным вектором, который назначает вероятность 1 классу с наибольшей вероятностью. Иными словами, онлайнговое самообучение минимизирует перекрестную энтропию между выходом модели и «жесткой» целью $\arg\max p_{\theta}(y|\mathbf{x})$, тогда как при минимизации энтропии используется «мягкая» цель $p_{\theta}(y|\mathbf{x})$. Компромиссом между этими двумя крайностями является настройка «температуры» целевого распределения, для чего нужно возвести каждую вероятность в степень $1/T$ и перенормировать; эта идея лежит в основе метода **mixmatch** из работ [Ber+19a; Ber+19b; Xie+19]. При $T = 1$ он эквивалентен минимизации энтропии, а при $T \rightarrow 0$ становится жестким онлайнговым самообучением. На рис. 19.9 сравниваются все три функции потерь.

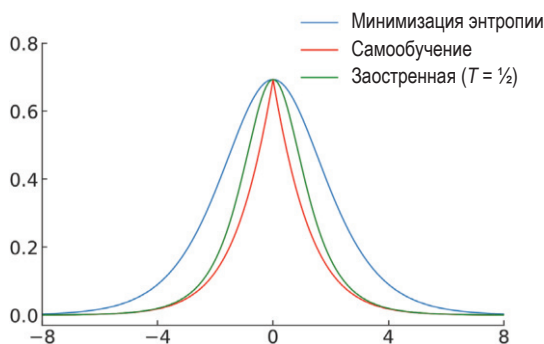


Рис. 19.9 ❖ Сравнение трех функций потерь: минимизации энтропии, самообучения и «заостренной» минимизации энтропии для задачи бинарной классификации

19.3.2.1. Кластерное допущение

Почему минимизация энтропии – хорошая идея? Основное допущение многих методов обучения с частичным привлечением учителя заключается в том, что решающая граница между классами должна проходить в области низкой плотности многообразия данных. По сути дела, это означает, что данные, соответствующие разным классам, должны объединяться в кластеры. Поэтому хорошая решающая граница должна не пересекать кластеры, а разделять их. Можно считать, что методы обучения с частичным привлечением учителя, которые делают такое «**кластерное допущение**», используют непомеченные

данные для оценки формы многообразия данных и отодвигания решающей границы подальше от него.

Минимизация энтропии – один из таких методов. Чтобы убедиться в этом, предположим сначала, что решающая граница между двумя классами «гладкая», т. е. не бывает так, что модель где-то резко меняет класс. На практике это действительно так для простых и (или) регуляризированных моделей. В этом случае, если решающая граница пересекает область высокой плотности данных, то она по необходимости порождает высокоэнтропийные предсказания для некоторых примеров из распределения данных. Поэтому минимизация энтропии поощряет модель помещать решающую границу в области низкой плотности пространства входов, чтобы избежать перехода от одного класса к другому в той части пространства, из которой могут выбираться данные. Это поведение наглядно представлено на рис. 19.10.

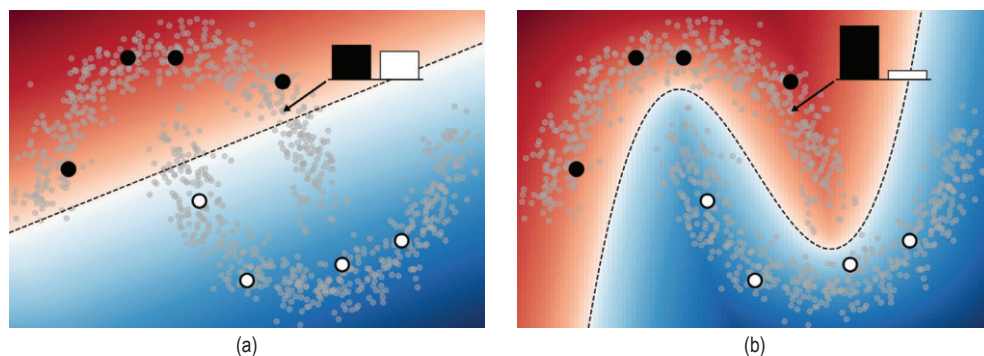


Рис. 19.10 ❖ Как минимизация энтропии обеспечивает выполнение кластерного допущения. Классификатор назначает более высокую вероятность классу 1 (черные точки) или 2 (белые точки) в красной и синей области соответственно. Предсказанные вероятности классов для одного конкретного непомеченного примера показаны на столбиковой диаграмме. На рисунке (а) решающая граница пересекает области высокой плотности данных, так что классификатор вынужден порождать высокоэнтропийные предсказания. На рисунке (б) классификатор избегает областей высокой плотности, поэтому может назначать низкоэнтропийные предсказания большинству непомеченных данных

19.3.2.2. Взаимная информация между входом и выходом

Другое обоснование минимизации энтропии в качестве целевой функции было предложено в работе Бридла, Хэдинга и Маккея [ВНМ92], где было показано, что это естественное следствие максимизации взаимной информации (раздел 6.3) между данными и меткой (т. е. между входом и выходом модели). Если обозначить вход \mathbf{x} , а выход y , то взаимную информацию между входом и выходом можно записать в виде:

$$\mathcal{I}(y; \mathbf{x}) = \iint p(y, \mathbf{x}) \log \frac{p(y, \mathbf{x})}{p(y)p(\mathbf{x})} dy d\mathbf{x} \quad (19.18)$$

$$= \int \int p(y|\mathbf{x})p(\mathbf{x}) \log \frac{p(y, \mathbf{x})}{p(y)p(\mathbf{x})} dy d\mathbf{x} \quad (19.19)$$

$$= \int p(\mathbf{x})d\mathbf{x} \int p(y|\mathbf{x}) \log \frac{p(y|\mathbf{x})}{p(y)} dy \quad (19.20)$$

$$= \int p(\mathbf{x})d\mathbf{x} \int p(y|\mathbf{x}) \log \frac{p(y|\mathbf{x})}{\int p(\mathbf{x})p(y|\mathbf{x})d\mathbf{x}} dy. \quad (19.21)$$

Заметим, что первый интеграл эквивалентен взятию математического ожидания \mathbf{x} , а второй – суммированию по всем возможным значениям класса y . Пользуясь этими соотношениями, получаем

$$\mathcal{J}(y; \mathbf{x}) = \mathbb{E}_{\mathbf{x}} \left[\sum_{i=1}^L p(y_i|\mathbf{x}) \log \frac{p(y_i|\mathbf{x})}{\mathbb{E}_{\mathbf{x}}[p(y_i|\mathbf{x})]} \right] \quad (19.22)$$

$$= \mathbb{E}_{\mathbf{x}} \left[\sum_{i=1}^L p(y_i|\mathbf{x}) \log p(y_i|\mathbf{x}) \right] - \mathbb{E}_{\mathbf{x}} \left[\sum_{i=1}^L p(y_i|\mathbf{x}) \log \mathbb{E}_{\mathbf{x}}[p(y_i|\mathbf{x})] \right] \quad (19.23)$$

$$= \mathbb{E}_{\mathbf{x}} \left[\sum_{i=1}^L p(y_i|\mathbf{x}) \log p(y_i|\mathbf{x}) \right] - \sum_{i=1}^L \mathbb{E}_{\mathbf{x}}[p(y_i|\mathbf{x})] \log \mathbb{E}_{\mathbf{x}}[p(y_i|\mathbf{x})]. \quad (19.24)$$

Поскольку изначально мы стремились *максимизировать* взаимную информацию, а обычно *минимизируем* функции потерь, можно преобразовать это выражение в подходящую функцию потерь, поставив знак минус:

$$\mathcal{J}(y; \mathbf{x}) = -\mathbb{E}_{\mathbf{x}} \left[\sum_{i=1}^L p(y_i|\mathbf{x}) \log p(y_i|\mathbf{x}) \right] + \sum_{i=1}^L \mathbb{E}_{\mathbf{x}}[p(y_i|\mathbf{x})] \log \mathbb{E}_{\mathbf{x}}[p(y_i|\mathbf{x})]. \quad (19.25)$$

Первый член – это в точности целевая функция минимизации энтропии в математическом ожидании. Второй член говорит, что мы должны максимизировать энтропию ожидаемого предсказания класса, т. е. среднее предсказание класса по нашему обучающему набору. Это поощряет модель предсказывать все возможные классы с равной вероятностью; это приемлемо, только если мы заранее знаем, что все классы равновероятны.

19.3.3. Совместное обучение

Совместное обучение [BM98] также похоже на самообучение, но делается дополнительное предположение о существовании двух взаимодополняющих «видов» (т. е. независимых наборов признаков) данных, которые можно по отдельности использовать для обучения разумной модели. После раздельного обучения двух моделей на разных видах непомеченные данные классифицируются каждой моделью для получения псевдометок-кандидатов. Если некоторая псевдометка получает от одной модели предсказание с низкой энтропией (что говорит о высокой степени уверенности) и предсказание с высокой энтропией (низкой степенью уверенности) от другой, то эта псев-

допомеченная точка добавляется в обучающий набор для модели с низкой степенью уверенности. Затем этот процесс повторяется с новыми, большими наборами данных. Процедура сохранения только тех псевдометок, в которых одна из моделей уверена, идеально подходит для постепенного построения обучающих наборов с правильно помеченными данными.

Совместное обучение основано на сильном допущении о том, что существует два информативных, но независимых вида данных, но для многих задач это может быть не так. Алгоритм **обучения втроем** (Tri-Training) [ZL05] обходит эту трудность за счет использования *трех* моделей, которые сначала обучаются на результатах независимой выборки (с возвращением) из помеченных данных. В идеале первоначальное обучение на разных наборах помеченных данных приводит к моделям, предсказания которых не всегда согласованы. Затем для непомеченных данных генерируются псевдометки – независимо каждой из трех моделей. Если для некоторого непомеченного примера две модели дают одну и ту же псевдометку, то она добавляется в обучающий набор для третьей модели. Это можно рассматривать как метрику выбора, потому что псевдометки сохраняются, только если две (по-разному инициализированных) модели согласны на правильной метке. Затем модели переобучаются на комбинации помеченных данных и новых псевдометок, и весь процесс повторяется.

19.3.4. Распространение меток на графах

Если два примера чем-то «похожи», то можно ожидать, что им будут назначены одинаковые метки. Эта идея получила название **«допущение многообразия»**. **Распространение меток** – это метод обучения с частичным привлечением учителя, в котором допущение многообразия используется для назначения меток непомеченным данным. Алгоритм распространения меток сначала строит граф, в котором вершинами являются примеры данных, а веса ребер представляют степень сходства. Для вершин, соответствующих помеченным данным, метки известны, а для непомеченных данных неизвестны. Далее алгоритм распространяет известные метки вдоль ребер графа таким образом, чтобы расхождение между метками соседей данного узла было минимально. Это позволяет угадать метки для непомеченных данных, а затем обычным образом использовать для обучения модели с учителем.

Формально базовый алгоритм распространения меток [ZG02] работает следующим образом. Обозначим $w_{i,j}$ неотрицательный вес ребра, соединяющего вершины x_i и x_j ; это будет мера сходства двух (помеченных или непомеченных) примеров. В предположении, что имеется M помеченных примеров и N непомеченных, определим матрицу переходов \mathbf{T} размера $(M + N) \times (M + N)$, элементы которой равны:

$$\mathbf{T}_{i,j} = \frac{w_{i,j}}{\sum_k w_{k,j}}. \quad (19.26)$$

Элемент $\mathbf{T}_{i,j}$ представляет вероятность распространения метки узла j в узел i . Далее определим матрицу \mathbf{Y} размера $(M + N) \times C$, где C – количество

возможных классов. i -я строка \mathbf{Y} представляет распределение вероятностей классов для примера i . Затем будем повторять следующие шаги, пока значения элементов \mathbf{Y} не перестанут заметно изменяться. Сначала используем матрицу переходов \mathbf{T} для распространения меток из \mathbf{Y} , полагая $\mathbf{Y} \leftarrow \mathbf{T}\mathbf{Y}$. Затем перенормируем строки \mathbf{Y} , полагая $\mathbf{Y}_{i,c} \leftarrow \mathbf{Y}_{i,c} / \sum_k \mathbf{Y}_{i,k}$. Наконец, заменяем строки \mathbf{Y} , соответствующие помеченным примерам, их унитарным представлением (т. е. $\mathbf{Y}_{i,c} = 1$, если истинная метка примера i равна c , и 0 в противном случае). После достижения сходимости для каждого примера в \mathbf{Y} выбирается метка, соответствующая классу с наибольшей вероятностью.

Этот алгоритм итеративно использует сходство примеров (закодированное в весах, на основе которых строилась матрица переходов), чтобы распространять информацию от (фиксированных) меток на непомеченные данные. На каждой итерации распределение меток для данного примера вычисляется как взвешенное среднее распределений меток для всех связанных с ними примеров, причем веса соответствуют весам ребер в \mathbf{T} . Можно показать, что эта процедура сходится к единственной фиксированной точке, а стоимость вычислений определяется стоимостью обращения матрицы вероятностей переходов между непомеченными примерами [ZG02].

Подход в целом можно рассматривать как форму **трансдуктивного обучения**, так как его цель – предсказать метки для фиксированного неразмеченного набора данных, а не обучить модель, допускающую обобщение. Однако с помощью индуцированной пометки мы можем затем выполнить **индуктивное обучение** обычным образом.

Успех алгоритма распространения меток сильно зависит от понятия сходства, использованного для построения весов ребер, соединяющих различные вершины (примеры данных). Для простых данных достаточно измерения евклидова расстояния между точками. Но для сложных многомерных данных евклидово расстояние необязательно отражает вероятность того, что две точки принадлежат одному классу. Веса сходства можно также назначать произвольно, сообразуясь со знаниями о предметной области. Несколько примеров построения графа сходства см. в работе [Zhu05, глава 3]. Из недавних работ, в которых этот подход используется в сочетании с глубоким обучением, упомянем [BRR18; Isc+19].

19.3.5. Регуляризация по согласованности

Идея **регуляризации по согласованности** (consistency regularization) проста: небольшое возмущение данного примера (или самой модели) не должно вызывать значительного изменения выхода модели. Поскольку при таком измерении согласованности используются только выходы модели (а не истинные метки), эту идею легко применить к непомеченным данным и, следовательно, создать функции потерь, подходящие для обучения с частичным привлечением учителя. Впервые эта идея была предложена в системе «обучения с псевдоансамблями» [BAP14], а вскоре были разработаны ее варианты [LA16; SJT16].

В самой общей форме и сама модель $p_\theta(y|x)$, и преобразования, применяемые к входам, могут быть стохастическими. Например, в задачах ком-

пьютерного зрения вход можно преобразовывать с помощью приращения данных, скажем, случайных поворотов или прибавления шума к входному изображению, а сеть может содержать стохастические компоненты типа прореживания (раздел 13.5.4) или зашумления весов [Gra11]. В широко распространенной и простой форме регуляризации по согласованности сначала производится выборка $\mathbf{x}' \sim q(\mathbf{x}'|\mathbf{x})$ (где $q(\mathbf{x}'|\mathbf{x})$ – распределение, индуцированное стохастическими преобразованиями ввода), а затем минимизируется потеря $\|p_\theta(y|\mathbf{x}) - p_\theta(y|\mathbf{x}')\|^2$. На практике первый член $p_\theta(y|\mathbf{x})$ обычно считается фиксированным (т. е. градиенты через него не распространяются). В обучении с частичным привлечением учителя комбинированная функция потерь на пакете помеченных данных $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_M, y_M)$ и непомеченных данных $\mathbf{x}_1, \dots, \mathbf{x}_N$ имеет вид:

$$\mathcal{L}(\theta) = -\sum_{i=1}^M \log p_\theta(y = y_i | \mathbf{x}_i) + \lambda \sum_{j=1}^N \|p_\theta(y|\mathbf{x}_j) - p_\theta(y|\mathbf{x}'_j)\|^2, \quad (19.27)$$

где λ – скалярный гиперпараметр, который балансирует важность потери на непомеченных данных и для простоты \mathbf{x}'_j обозначает пример, выбранный из распределения $q(\mathbf{x}'|\mathbf{x}_j)$.

Базовая форма регуляризации по согласованности (19.27) открывает много возможностей для проектных решений, от выбора которых зависит успех этого подхода к обучению с частичным привлечением учителя. Во-первых, важен выбор гиперпараметра λ . Если он слишком велик, то модель может не придать достаточный вес обучению задачи с учителем, а вместо этого начнет подкреплять собственные плохие предсказания (как в случае предвзятости подтверждения в самообучении). Поскольку модель часто плохо ведет себя в начале обучения еще до того, как была обучена на большом объеме помеченных данных, на практике часто инициализируют λ нулем и увеличивают значение по ходу продвижения обучения.

Второе важное соображение – какие случайные преобразования применяются к входу, т. е. $q(\mathbf{x}'|\mathbf{x})$. Вообще говоря, эти преобразования следует проектировать, так чтобы они не изменяли метку \mathbf{x} . Как было отмечено выше, чаще всего применяют зависящие от предметной области способы приращения данных. Недавно было показано, что использование приращений, которые существенно искажают вход (но все-таки не изменяют метку), может принести особо ценные плоды [Xie+19; Ber+19b; Soh+20].

Использование приращения данных требует экспертных знаний для определения того, какие виды преобразований сохраняют метки и подходят для данной задачи. Альтернативная техника, называемая **виртуальным состязательным обучением** (virtual adversarial training – VAT), заключается в применении к входам аналитически найденного возмущения, призванного максимально изменить выход модели. Точнее, VAT вычисляет возмущение, которое аппроксимирует $\delta = \arg\max_{\delta} \mathbb{KL}(p_\theta(y|\mathbf{x}) \| p_\theta(y|\mathbf{x} + \delta))$. Для аппроксимации мы производим выборку \mathbf{d} из многомерного гауссова распределения, инициализируем $\delta = \mathbf{d}$, а затем полагаем

$$\delta \leftarrow \nabla_{\delta} \mathbb{KL}(p_\theta(y|\mathbf{x}) \| p_\theta(y|\mathbf{x} + \delta))|_{\delta=\xi\mathbf{d}}, \quad (19.28)$$

где ξ – небольшая константа, обычно 10^{-6} . Затем алгоритм VAT полагает

$$\mathbf{x}' = \mathbf{x} + \varepsilon \frac{\delta}{\|\delta\|_2} \quad (19.29)$$

и продолжает работу, как обычно в схеме регуляризации по согласованности (по формуле (19.27)). Здесь ε – скалярный гиперпараметр, определяющий L2-норму возмущения, прибавленного к \mathbf{x} .

Регуляризация по согласованности может также сильно влиять на геометрические свойства целевой функции обучения и на траекторию СГС, поэтому качество может особенно выиграть от нестандартных процедур обучения. Например, евклидовы расстояния между весами на различных эпохах обучения существенно больше для целевых функций, в которых используется регуляризация по согласованности. В работе [AIW19] показано, что **вариант стохастического усреднения весов** (stochastic weight averaging – SWA) [Izm+18] может достигать лучшего на сегодняшний день качества в задачах обучения с частичным привлечением учителя благодаря удачному использованию геометрических свойств регуляризации по согласованности.

Последнее, что нужно учитывать при использовании регуляризации по согласованности, – функция для измерения различия между выходом сети с возмущениями и без них. В формуле (19.27) используется квадрат расстояния L2 (называемый также оценкой Бриера), и это довольно популярный выбор [SJT16; TV17; LA16; Ber+19a]. Также часто используется расхождение Кульбака–Лейблера $\mathbb{KL}(p_\theta(y|\mathbf{x})\|p_\theta(y|\mathbf{x}'))$ по аналогии с потерей перекрестной энтропии (т. е. расхождением между истинной меткой и предсказанием) для помеченных примеров [Miy+18; Ber+19b; Xie+19]. Градиент потери квадратичной ошибки стремится к нулю, когда предсказания модели на возмущенных и невозмущенных входах начинают все больше различаться, в предположении, что в качестве нелинейности на выходе используется softmax. Поэтому потенциальное преимущество такой потери – то, что модель не обновляется, когда ее предсказания очень нестабильны. Однако расхождение КЛ имеет такой же масштаб, как потеря перекрестной энтропии для помеченных данных, что делает интуитивно более понятной настройку гиперпараметра λ потери на непомеченных примерах. Сравнение двух функций потерь приведено на рис. 19.11.

19.3.6. Глубокие порождающие модели*

Порождающие модели предлагают естественный способ воспользоваться непомеченными данными путем обучения модели маргинального распределения, для чего нужно минимизировать функцию $\mathcal{L}_U = -\sum_n \log p_\theta(\mathbf{x}_n)$. Есть разные подходы к применению порождающих моделей в обучении с частичным привлечением учителя, различающиеся способом использования модели $p_\theta(\mathbf{x}_n)$.

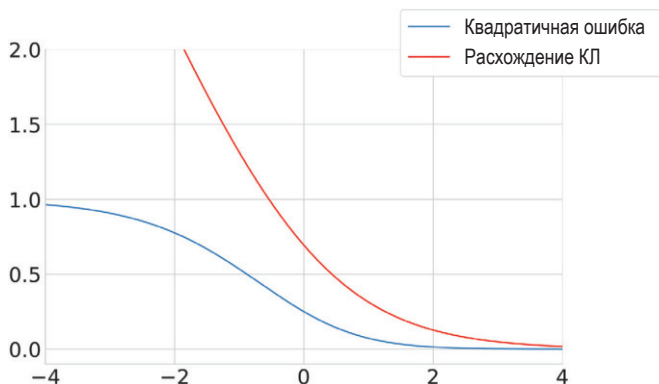


Рис. 19.11 ❖ Сравнение потерь, равных квадратичной ошибке и расхождению Кульбака–Лейблера, в регуляризации по согласованности. Этот рисунок относится к задаче бинарной классификации, где предполагается, что выход модели для невозмущенного входа равен 1. На рисунке показана потеря для конкретного значения логита (т. е. предактивации, поданной выходной сигмоидной нелинейности) возмущенного входа. Когда логит стремится к плюс бесконечности, модель предсказывает метку класса 1 (в полном согласии с предсказанием для невозмущенного входа). А когда логит стремится к минус бесконечности, модель предсказывает класс 0. Потеря, равная квадратичной ошибке, насыщается, т. е. стремится к горизонтальной асимптоте (и, стало быть, имеет нулевые градиенты), когда модель предсказывает один или другой класс с высокой вероятностью, а потеря, равная расхождению КЛ, неограниченно растет по мере того, как модель предсказывает класс 0 все с большей уверенностью

19.3.6.1. Вариационные автокодировщики

В разделе 20.3.5 мы опишем вариационный автокодировщик (variational autoencoder – VAE), который определяет вероятностную модель совместного распределения данных \mathbf{x} и латентных переменных \mathbf{z} . Предполагается, что для генерирования данных сначала производится выборка $\mathbf{z} \sim p(\mathbf{z})$, а затем выборка $\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$. Для обучения VAE использует кодировщик $q_{\lambda}(\mathbf{z}|\mathbf{x})$, чтобы аппроксимировать апостериорное распределение, и декодер $p_{\theta}(\mathbf{x}|\mathbf{z})$ для аппроксимации правдоподобия. Кодировщик и декодер обычно являются глубокими нейронными сетями. Параметры кодировщика и декодера можно совместно обучить путем максимизации вариационной нижней границы (evidence lower bound – ELBO) данных.

В качестве маргинального распределения латентных переменных $p(\mathbf{z})$ часто выбирается какое-нибудь простое распределение типа гауссова с диагональной ковариационной матрицей. На практике это может сделать латентные переменные \mathbf{z} более пригодными для последующих задач классификации в силу следующих причин: \mathbf{z} , как правило, имеет меньшую размерность, чем \mathbf{x} ; \mathbf{z} строится путем каскадных нелинейных преобразований, и измерения латентных переменных проектируются независимыми. Иными словами, латентные переменные могут дать (обученное) представление, в котором данные легче разделяются. В работе [Kin+14] этот подход назван

M1 и показано, что латентные переменные действительно позволяют обучить более сильные модели, если метки разрежены. (Общая идея обучения представлений без учителя с целью упростить решение последующих задач классификации более подробно описана в разделе 19.2.4.)

Альтернативный подход к использованию VAE, также предложенный в работе [Kin+14] и названный **M2**, имеет вид:

$$p_{\theta}(\mathbf{x}, y) = p_{\theta}(y)p_{\theta}(\mathbf{x}|y) = p_{\theta}(y) \int p_{\theta}(\mathbf{x}|y, \mathbf{z})p_{\theta}(\mathbf{z})d\mathbf{z}, \quad (19.30)$$

где \mathbf{z} – латентная переменная, $p_{\theta}(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ – латентное априорное распределение, $p_{\theta}(y) = \text{Cat}(y|\boldsymbol{\pi})$ – априорное распределение меток, а $p_{\theta}(\mathbf{x}|y, \mathbf{z}) = p(\mathbf{x}|f_{\theta}(y, \mathbf{z}))$ – правдоподобие, например гауссово, с параметрами, вычисленными f (глубокой нейронной сетью). Главное новшество этого подхода – предположение о том, что данные генерируются в соответствии как с латентной переменной класса y , так и с непрерывной латентной переменной \mathbf{z} . Переменная класса y является наблюдаемой для помеченных данных и ненаблюдаемой для непомеченных.

Чтобы вычислить правдоподобие для *помеченных данных*, $p_{\theta}(\mathbf{x}, y)$, нам нужно исключить \mathbf{z} , что можно сделать, воспользовавшись сетью вывода вида

$$q_{\phi}(\mathbf{z}|y, \mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\phi}(y, \mathbf{x}), \text{diag}(\sigma_{\phi}^2(\mathbf{x}))). \quad (19.31)$$

Затем мы используем следующую вариационную нижнюю границу:

$$\begin{aligned} \log p_{\theta}(\mathbf{x}, y) &\geq \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x}, y)}[\log p_{\theta}(\mathbf{x}|y, \mathbf{z}) + \log p_{\theta}(y) \\ &\quad + \log p_{\theta}(\mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x}, y)] = -\mathcal{L}(\mathbf{x}, y), \end{aligned} \quad (19.32)$$

как обычно в VAE (см. раздел 20.3.5). Единственное различие в том, что наблюдаются данные двух видов: \mathbf{x} и y .

Чтобы вычислить правдоподобие для *непомеченных данных*, $p_{\theta}(\mathbf{x})$, нужно исключить \mathbf{z} и y ; это можно сделать, воспользовавшись сетью вывода вида:

$$q_{\phi}(\mathbf{z}, y|\mathbf{x}) = q_{\phi}(\mathbf{z}|\mathbf{x})q_{\phi}(y|\mathbf{x}); \quad (19.33)$$

$$q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\phi}(\mathbf{x}), \text{diag}(\sigma_{\phi}^2(\mathbf{x}))); \quad (19.34)$$

$$q_{\phi}(y|\mathbf{x}) = \text{Cat}(y|\boldsymbol{\pi}_{\phi}(\mathbf{x})). \quad (19.35)$$

Отметим, что $q_{\phi}(y|\mathbf{x})$ работает как дискриминантный классификатор, который подставляет отсутствующие метки. Затем мы используем следующую вариационную нижнюю границу:

$$\log p_{\theta}(\mathbf{x}) \geq \mathbb{E}_{q_{\phi}(\mathbf{z}, y|\mathbf{x})}[\log p_{\theta}(\mathbf{x}|y, \mathbf{z}) + \log p_{\theta}(y) + \log p_{\theta}(\mathbf{z}) - \log q_{\phi}(\mathbf{z}, y|\mathbf{x})] \quad (19.36)$$

$$= -\sum_y q_{\phi}(y|\mathbf{x})\mathcal{L}(\mathbf{x}, y) + \mathbb{H}(q_{\phi}(y|\mathbf{x})) = -\mathcal{U}(\mathbf{x}). \quad (19.37)$$

Отметим, что дискриминантный классификатор $q_{\phi}(y|\mathbf{x})$ используется только для вычисления логарифмического правдоподобия непомеченных данных, что нежелательно. Поэтому мы можем прибавить дополнительную

потерю классификации на помеченных данных и получить полную целевую функцию:

$$\mathcal{L}(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}_L}[\mathcal{L}(x, y)] + \mathbb{E}_{x \sim \mathcal{D}_U}[\mathcal{U}(x)] + \alpha \mathbb{E}_{(x,y) \sim \mathcal{D}_L}[-\log q_\phi(y|x)], \quad (19.38)$$

где α – гиперпараметр, контролирующий относительные веса порождающего и дискриминантного обучения.

Конечно, вероятностная модель, используемая в M2, – лишь один из многих способов произвести декомпозицию зависимостей между наблюдаемыми данными, метками классов и непрерывными латентными переменными. Есть также много других способов приближенного вывода, помимо вариационного. Какой метод выбрать, зависит от задачи, но в целом главное преимущество порождающего подхода в том, что можно включить знания о предметной области. Скажем, можно смоделировать механизм отсутствующих данных, потому что отсутствие метки может нести информации о самих данных (например, люди часто не хотят отвечать на вопрос о здоровье в анкете, если страдают каким-то заболеванием).

19.3.6.2. Порождающие состязательные сети

Порождающие состязательные сети (generative adversarial network – GAN), более подробно описанные во втором томе этой книги, [Mur22]), – это популярный класс порождающих моделей, которые обучаются неявной модели распределения данных. Они состоят из сети-генератора, которая отображает примеры из простого латентного распределения в пространство данных, и сети-критика, которая пытается отличить выходы генератора от примеров из истинного распределения данных. Генератор обучается порождать примеры, который критик классифицирует как «настоящие».

Поскольку стандартные GAN не порождают обученного латентного представления данного примера и не обучаются явной модели распределения данных, мы не можем использовать те же подходы, которые использовались для VAE. Вместо этого обучение с частичным привлечением учителя с применением GAN обычно производится путем модификации критика, так чтобы он выводил либо метку класса, либо «подлог», а не просто классифицировал примеры как настоящие или подложные [Sal+16; Ode16]. Для помеченных настоящих данных критик обучается выводить подходящую метку класса, а для непомеченных настоящих данных – вероятность каждой из меток классов. Как и при обучении стандартной GAN, критик обучается классифицировать выходы генератора как подложные, а генератор обучается обманывать критика.

Формально обозначим $p_\theta(y|x)$ критика с $C + 1$ выходами, соответствующими C классам и классу «подлог», а $G(z)$ генератор, который принимает на входе примеры из априорного распределения $p(z)$. Предположим, что используется стандартная для GAN потеря перекрестной энтропии, предложенная в работе [Goo+14]. Тогда потеря критика равна

$$\begin{aligned} -\mathbb{E}_{x,y \sim p(x,y)} \log p_\theta(y|x) - \mathbb{E}_{x \sim p(x)} \log[1 - p_\theta(y = C + 1|x)] \\ - \mathbb{E}_{z \sim p(z)} \log p_\theta(y = C + 1|G(z)). \end{aligned} \quad (19.39)$$

Наша цель здесь – максимизировать вероятность правильного класса для помеченных примеров, минимизировать вероятность подложного класса для настоящих непомеченных примеров и максимизировать вероятность подложного класса для сгенерированных примеров. Потеря генератора проще:

$$\mathbb{E}_{z \sim p(z)} \log p_{\theta}(y = C + 1 | G(z)). \quad (19.40)$$

Диаграмма системы обучения GAN с частичным привлечением учителя показана на рис. 19.12.

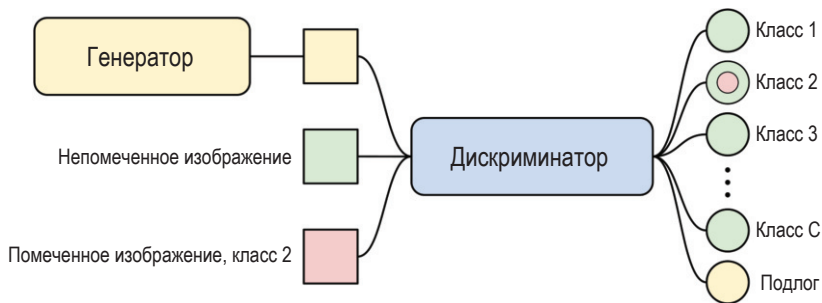


Рис. 19.12 ❖ Диаграмма обучения GAN с частичным привлечением учителя. Дискриминатор обучается выводить класс помеченных примеров (красный), «подлог» для выходов генератора (желтый) и какую-то метку для непомеченных данных (зеленый)

19.3.6.3. Нормализующие потоки

Нормализующие потоки (более подробно описаны во втором томе этой книги, [Mur22]) – это вычислительно реализуемый способ определения глубоких порождающих моделей. Точнее, они определяют обратимое отображение $f_{\theta} : \mathcal{X} \rightarrow \mathcal{Z}$ с параметрами θ из пространства данных \mathcal{X} в латентное пространство \mathcal{Z} . Плотность в пространстве данных можно записать, начав с плотности в латентном пространстве и используя формулу замены переменных:

$$p(x) = p(f(x)) \cdot \left| \det \left(\frac{\partial f}{\partial x} \right) \right|. \quad (19.41)$$

Мы можем обобщить эту идею это на обучение с частичным привлечением учителя, как предложено в работе [Izm+20]. Для меток классов $y \in \{1 \dots C\}$ можно в качестве латентного распределения, обусловленного меткой k , задать гауссово распределение со средним μ_k и ковариацией Σ_k : $p(z|y = k) = \mathcal{N}(z|\mu_k, \Sigma_k)$. Тогда маргинальное распределение z является смесью гауссовых. Правдоподобие помеченных данных равно

$$p_X(x|y = k) = \mathcal{N}(f(x)|\mu_k, \Sigma_k) \cdot \left| \det \left(\frac{\partial f}{\partial x} \right) \right|, \quad (19.42)$$

а правдоподобие для данных с неизвестной меткой равно $p(x) = \sum_k p(x|y = k)p(y = k)$.

Для обучения с частичным привлечением учителя мы можем затем максимизировать совместное правдоподобие помеченных данных \mathcal{D}_l и непомеченных данных \mathcal{D}_u :

$$p(\mathcal{D}_l, \mathcal{D}_u|\theta) = \prod_{(x_i, y_i) \in \mathcal{D}_l} p(x_i, y_i) \prod_{x_j \in \mathcal{D}_u} p(x_j) \quad (19.43)$$

при условии параметров θ биективной функции f , которая обучает модель плотности для байесовского классификатора.

Для данной тестовой точки x прогнозное распределение модели имеет вид:

$$p_x(x|y) = \frac{p(x|y)p(y)}{p(x)} = \frac{\mathcal{N}(f(x)|\mu_y, \Sigma_y)}{\sum_{k=1}^c \mathcal{N}(f(x)|\mu_k, \Sigma_k)}. \quad (19.44)$$

Мы можем делать предсказания для тестовой точки x с помощью решающего правила Байеса $y = \operatorname{argmax}_{c \in \{1, \dots, c\}} p(y = c|x)$.

19.3.7. Сочетание самостоятельного обучения и обучения с частичным привлечением учителя

Самостоятельное обучение и обучение с частичным привлечением учителя можно сочетать. Например, в работе [Che+20c] используется метод SimCLR (раздел 19.2.4.4), чтобы с частичным привлечением учителя обучить представления на небольшом размеченном наборе данных (как при переносе обучения, раздел 19.2), а затем применить обученную модель к исходному неразмеченному набору данных и дистиллировать предсказания из этой модели-учителя T в модель-ученика S . (**Дистилляцией знаний** называется подход, заключающийся в обучении одной модели на предсказаниях другой, впервые предложенный в работе [HVD14].) То есть после дообучения T авто-ры обучают S путем минимизации потери:

$$\mathcal{L}(T) = - \sum_{x_i \in \mathcal{D}} \left(\sum_y p^T(y|x_i; \tau) \log p^S(y|x_i; \tau) \right), \quad (19.45)$$

где $\tau > 0$ – параметр температуры, применяемый к выходу softmax и нужный для сглаживания меток. Если S имеет такую же форму, как T , то это называется самообучением (см. раздел 19.3.1). Но обычно ученик S меньше учителя T . (Например, T может быть моделью высокой емкости, а S – ее облегченной версией для работы в телефоне.) Как работает подход в целом, показано на рис. 19.13.

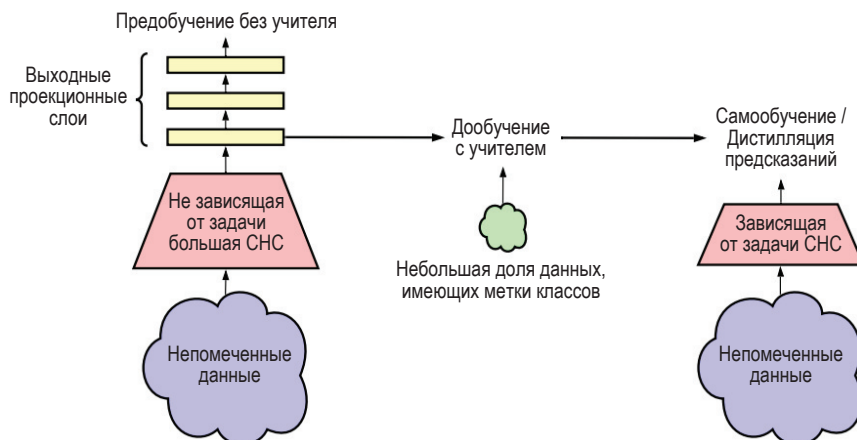


Рис. 19.13 ❖ Сочетание обучения с частичным привлечением учителя на непомеченных данных (слева), дообучения с учителем (в центре) и самообучения на псевдопомеченных данных (справа). На основе рис. 3 из работы [Che+20c].
Печатается с разрешения Тин Чена

19.4. АКТИВНОЕ ОБУЧЕНИЕ

В активном обучении цель заключается в том, чтобы найти истинное прогнозируемое отображение $y = f(\mathbf{x})$, опросив как можно меньше точек (\mathbf{x}, y) . Существует три основных варианта. В случае **синтеза запросов** алгоритм может выбрать любой вход \mathbf{x} и запросить соответствующий выход $y = f(\mathbf{x})$. В случае **активного обучения с пулом** имеется большой, но фиксированный набор непомеченных примеров, а алгоритм должен запросить метку для одного или нескольких из них. Наконец, в **поточном активном обучении** данные поступают непрерывно, а алгоритм должен решить, хочет он запросить метку для текущего входа или нет.

Имеются различные тесно связанные задачи. Цель **байесовской оптимизации** – оценить местоположение глобального оптимума $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} f(\mathbf{x})$, задав минимальное число вопросов; как правило, мы обучаем суррогатную модель (поверхность отклика) на промежуточных запросах (\mathbf{x}, y) , чтобы решить, какой вопрос задать следующим. При **планировании эксперимента** цель – вывести вектор параметров некоторой модели, используя тщательно подобранные примеры $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, т. е. мы хотим оценить $p(\theta|\mathcal{D})$, используя как можно меньше данных. (Это можно рассматривать как обобщенную форму активного обучения без учителя.)

В этом разделе мы дадим краткий обзор активного обучения с пулом. Дополнительные сведения см. в обзоре [Set12].

19.4.1. Подход на основе теории принятия решений

В подходе к активному обучению на основе теории принятия решений, предложенном в работах [КНВ07; РМ01], мы определяем полезность запрашивания \mathbf{x} в терминах **ценности информации**:

$$U(\mathbf{x}) \triangleq \mathbb{E}_{p(y|\mathbf{x}, \mathcal{D})} \left[\min_a R(a|\mathcal{D}) - R(a|\mathcal{D}, (\mathbf{x}, y)) \right], \quad (19.46)$$

где $R(a|\mathcal{D}) = \mathbb{E}_{p(\theta|\mathcal{D})} [\ell(\theta, a)]$ – апостериорная ожидаемая потеря, связанная с выбором некоторого будущего действия a при условии наблюдавшихся до сих пор данных \mathcal{D} . К сожалению, вычислять $U(\mathbf{x})$ для каждого \mathbf{x} очень дорого, так как для каждого возможного отклика y , который мы могли бы наблюдать, необходимо обновить наши ожидания при условии (\mathbf{x}, y) , чтобы понять, какое влияние он мог бы оказать на будущие решения (аналогично технике поиска с заглядыванием вперед применительно к ожидаемым состояниям).

19.4.2. Теоретико-информационный подход

Теоретико-информационный подход к активному обучению с учителем позволяет обойтись без зависящих от задачи функций потери, а сосредоточиться на обучении модели так хорошо, как получится. В частности, в работе [Lin56] предложено определять полезность опрашивания \mathbf{x} в терминах **информационного выигрыша**, т. е. приобретения информации о параметрах θ , а значит, снижения энтропии:

$$U(\mathbf{x}) \triangleq \mathbb{H}(p(\theta|\mathcal{D})) - \mathbb{E}_{p(y|\mathbf{x}, \mathcal{D})} [\mathbb{H}(p(\theta|\mathcal{D}, \mathbf{x}, y))]. \quad (19.47)$$

(Заметим, что первый член вообще не зависит от \mathbf{x} , но мы включили его для удобства в будущем.) В упражнении 19.1 вам будет предложено доказать, что эта целевая функция идентична ожидаемому изменению в апостериорном распределении параметров, которое имеет вид:

$$U'(\mathbf{x}) \triangleq \mathbb{E}_{p(y|\mathbf{x}, \mathcal{D})} [\mathbb{KL}(p(\theta|\mathcal{D}, \mathbf{x}, y) \| p(\theta|\mathcal{D}))]. \quad (19.48)$$

Пользуясь симметрией взаимной информации, мы можем переписать формулу (19.47) в виде:

$$U(\mathbf{x}) = \mathbb{H}(p(\theta|\mathcal{D})) - \mathbb{E}_{p(y|\mathbf{x}, \mathcal{D})} [\mathbb{H}(p(\theta|\mathcal{D}, \mathbf{x}, y))] \quad (19.49)$$

$$= \mathbb{I}(\theta, y|\mathcal{D}, \mathbf{x}) \quad (19.50)$$

$$= \mathbb{H}(p(y|\mathbf{x}, \mathcal{D})) - \mathbb{E}_{p(\theta|\mathcal{D})} [\mathbb{H}(p(y|\mathbf{x}, \theta))]. \quad (19.51)$$

Преимущество этого подхода в том, что теперь нужно рассуждать только о недостоверности прогнозного распределения выходов y , а не параметров θ .

У формулы (19.51) есть любопытная интерпретация. Первый член предпочитает примеры \mathbf{x} , для которых имеется неопределенность в предсказанной метке. Использование только его в качестве критерия отбора называется **выборкой максимальной энтропии** [SW87]. Однако при этом возможны проблемы с примерами, которые внутренне неоднозначны или неправильно помечены. Второй член в формуле (19.51) противодействует такому поведению, потому что предпочитает примеры \mathbf{x} , для которых метка может быть предсказана с достаточной определенностью, если известно θ ; это позволяет избежать выборки принципиально трудных для предсказания примеров. Иными словами, формула (19.51) отбирает примеры \mathbf{x} , для которых модель делает уверенные и при этом весьма разнообразные предсказания. Поэтому подход получил название **байесовского активного обучения по несогласию** (Bayesian active learning by disagreement – BALD) [Hou+12].

Этот метод можно использовать для обучения классификаторов для других предметных областей, где получить проставленные экспертами метки трудно, например медицинских или астрономических изображений [Wal+20].

19.4.3. Пакетное активное обучение

До сих пор мы предполагали жадную, или **близорукую**, стратегию, когда выбирается единственный пример \mathbf{x} , как будто он последний в мире. Но иногда бюджет позволяет собрать набор из B примеров, назовем их (\mathbf{X}, \mathbf{Y}) . В таком случае критерий информационного выигрыша принимает вид $U(\mathbf{X}) = \mathbb{H}(p(\theta|\mathcal{D})) - \mathbb{E}_{p(\mathbf{Y}|\mathbf{X}, \mathcal{D})}[\mathbb{H}(p(\theta|\mathbf{Y}, \mathbf{X}, \mathcal{D}))]$. К сожалению, оптимизация этой функции – NP-трудная задача относительно длины горизонта B [KLQ95; KG05].

По счастью, при некоторых условиях жадная стратегия почти оптимальна, и сейчас мы объясним, почему это так. Прежде всего, заметим, что для любого заданного \mathbf{X} функция информационного выигрыша $f(\mathbf{Y}) \triangleq \mathbb{H}(p(\theta|\mathcal{D})) - \mathbb{H}(p(\theta|\mathbf{Y}, \mathbf{X}, \mathcal{D}))$ отображает множество меток \mathbf{Y} в скаляр. Ясно, что $f(\emptyset) = 0$ и что f не убывает, т. е. $f(Y^{\text{large}}) \geq f(Y^{\text{small}})$ в силу принципа «информация лишней не бывает». Кроме того, в работе [KG05] доказано, что функция f **субмодулярная**. Следовательно, последовательный жадный подход отличается от оптимального не более чем на постоянный множитель. Объединив эту жадную технику с целевой функцией BALD, мы получим метод **BatchBALD** [KAG19].

19.5. МЕТАОБУЧЕНИЕ

Дисциплина **метаобучения**, или **обучения учению** [TP97], занимается обучением нескольких взаимосвязанных функций (часто называемых «задачами»). Например, предположим, что имеется множество J взаимосвязанных наборов данных, $\{\mathcal{D}^j : j = 1 \dots J\}$, где $\mathcal{D}^j = \{(\mathbf{x}_n^j, y_n^j) : n = 1 \dots N_j\}$, $\mathbf{x}_n^j \sim p(\mathbf{x})$, $y_n^j = f^j(\mathbf{x}_n^j)$. (Например, $p(\mathbf{x})$ могло бы быть распределением изображений, f^1 – функцией, отображающей изображения в породы собак, f^2 – функцией, ото-

бражающей изображения в типы автомобилей, и т. д.) Мы можем применить этот внутренний алгоритм для генерирования размеченного набора $\mathcal{D}_{\text{meta}} = \{(\mathcal{D}^j, f^j) : j = 1 \dots J\}$, который можно использовать для обучения внешнего «метаалгоритма», или функции \hat{M} , отображающей набор данных в функцию предсказания, $\hat{f}^j = \hat{M}(\mathcal{D}^j)$ (см. [Min99]). Для сравнения: в традиционном обучении с учителем обучается функция \hat{f} , которая отображает один пример в метку, $\hat{y}_n = \hat{f}(\mathbf{x}_n)$.

Существует много других подходов к метаобучению (см., например, обзор [Van18]). В разделе 19.5.1 мы обсудим популярный подход **MAML**. У метаобучения также много приложений, например **обучение на малом числе примеров** (few-shot learning) (см. раздел 19.6).

19.5.1. Метаобучение, не зависящее от модели (MAML)

Естественный подход к метаобучению – использовать иерархическую байесовскую модель, как показано на рис. 19.14. Предполагается, что параметры для каждой задачи θ_j имеют общее априорное распределение, $p(\theta_j|\phi)$, которое можно использовать для объединения статистической силы от нескольких задач с малым объемом обучающих данных.

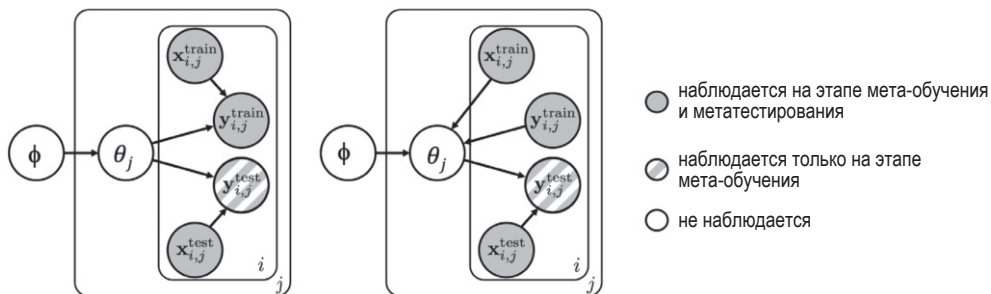


Рис. 19.14 ❖ Графовая модель, соответствующая MAML. Слева: порождающая модель. Справа: в процессе метаобучения каждый параметр задачи θ_j обновляется с использованием своего локального набора данных. Индексы j относятся к задачам (метанаборам данных), а индексы i – к экземплярам внутри каждой задачи. Сплошные узлы наблюдаемы всегда, заштрихованные – только на этапе метаобучения (но не на этапе тестирования). На основе рис. 1 из работы [FXL18]. Печатается с разрешения Челси Финн

Можно было бы выполнить байесовский вывод для параметров задачи θ_j и разделяемых гиперпараметров ϕ , но более эффективно использовать следующую эмпирическую байесовскую аппроксимацию (раздел 4.6.5.3):

$$\phi^* = \operatorname{argmax}_{\phi} \frac{1}{J} \sum_{j=1}^J \log p(\mathcal{D}_{\text{valid}}^i | \hat{\theta}_j(\phi, \mathcal{D}_{\text{train}}^i)), \quad (19.52)$$

где $\hat{\theta}_j = \hat{\theta}(\phi, \mathcal{D}_{\text{train}}^j)$ – точечная оценка параметров для задачи j на основе набора данных $\mathcal{D}_{\text{train}}^j$ и априорного распределения ϕ , а для аппроксимации маргинального правдоподобия используется перекрестная проверка (раздел 5.2.4).

Чтобы вычислить точечную оценку параметров задачи, $\hat{\theta}_j$, мы используем K -шаговую процедуру градиентного подъема, отправляясь от ϕ со скоростью обучения ρ . Можно показать, что это эквивалентно приближенной оценке MAP с использованием гауссова априорного распределения с центром в ϕ , где сила априорного распределения управляется количеством шагов градиента [San96; Gra+18]. (Это пример **быстрой адаптации** весов конкретной задачи, отталкиваясь от разделяемого априорного распределения ϕ .)

Таким образом, мы видим, что обучение априорного распределения ϕ эквивалентно обучению хорошего инициализатора для обучения конкретной задачи. Это называется **модельнезависимым метаобучением** (model-agnostic meta-learning – MAML) [FAL17].

19.6. ОБУЧЕНИЕ НА МАЛОМ ЧИСЛЕ ПРИМЕРОВ

Люди часто обучаются предсказывать, располагая совсем небольшим числом помеченных примеров. Это называется **обучением на малом числе примеров** (few-shot learning – FSL). В предельном случае человек или система обучаются на одном примере из каждого класса (one-shot learning) или даже вообще без помеченных примеров (zero-shot learning).

Распространенный подход к оценке методов FSL – использовать **классификацию с C классами и N примерами** (C -way N -shot classification), когда от системы ожидают, что она обучится классифицировать по C классам, имея всего N примеров из каждого класса. Обычно N и C очень малы, например на рис. 19.15 показан случай, когда имеется $C = 3$ класса по $N = 2$ примеров в каждом. Раз объем данных, принадлежащих новому домену (в данном случае – утки, дельфины и куры), так мал, нет надежды обучиться с нуля. Поэтому придется прибегнуть к метаобучению.

В процессе обучения метаалгоритм M обучается на помеченном опорном наборе из группы j и возвращает предиктор f^j , который затем оценивается на опросном наборе из группы j , не пересекающемся с опорным. M оптимизируется на всех J группах. Наконец, мы можем применить M к нашему новому помеченному опорному набору и получить предиктор f^{test} , который применяется к опросному набору из тестового домена. Эта схема показана на рис. 19.15. Мы видим, что между классами из обеих обучающих задач ({кошка, овца, свинья} и {собака, акула, лев}) и классами из тестовой задачи ({утка, дельфин, курица}) нет пересечений. Таким образом, алгоритм M должен научиться предсказывать классы изображений вообще, а не какой-то конкретный набор меток.

Существует много подходов к обучению на малом числе примеров. Один такой метод мы обсудим в разделе 19.6.1. Другие методы см., например, в работе [Wan+20b].

Обучающая задача 1

Опорный набор



Опросный набор

**Обучающая задача 2** · · ·

Опорный набор



Опросный набор

**Тестовая задача 1** · · ·

Опорный набор



Опросный набор

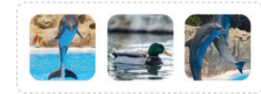


Рис. 19.15 ❖ Применение метаобучения для обучения на малом числе примеров. Здесь каждая задача – классификация с тремя классами и двумя примерами. Из работы <https://bit.ly/3rrvSjw>. Copyright (2019) Borealis AI. Печатается с разрешения Саймона Принса и Эйприл Купер

19.6.1. Сопоставляющие сети

Один из подходов к обучению на малом числе примеров – обучить метрику расстояния на каком-то другом наборе данных, а затем использовать $d_{\theta}(\mathbf{x}, \mathbf{x}')$ в классификаторе по ближайшим соседям. По существу, мы таким образом определяем полупараметрическую модель вида $p_{\theta}(y|\mathbf{x}, S)$, где S – небольшой помеченный набор данных (называемый опорным набором), а θ – параметры функции расстояния. Этот подход широко используется в задачах **детальной классификации**, когда имеется много разных, но визуально похожих категорий, например изображений лиц в галерее или изображений товаров в каталоге.

Обобщением этого подхода является задача обучения функции вида

$$p_{\theta}(y|\mathbf{x}, S) = \mathbb{P}\left(y = \sum_{n \in S} a_{\theta}(\mathbf{x}, \mathbf{x}_n; S) y_n\right), \quad (19.53)$$

где $a_{\theta}(\mathbf{x}, \mathbf{x}_n, S) \in \mathbb{R}^+$ – то или иное адаптивное ядро сходства. Например, можно использовать **ядро внимания** вида

$$a(\mathbf{x}, \mathbf{x}_n; S) = \frac{\exp(c(f(\mathbf{x}), g(\mathbf{x}_n)))}{\sum_{n'=1}^N \exp(c(f(\mathbf{x}), g(\mathbf{x}_{n'})))}, \quad (19.54)$$

где $c(\mathbf{u}, \mathbf{v})$ – коэффициент Отиаи. (При желании можно сделать f и g одинаковыми.) Интуитивно понятно, что ядро внимания будет сравнивать \mathbf{x} с \mathbf{x}_n в контексте всех помеченных примеров, что дает неявный сигнал о том, какие измерения признаков релевантны. (Более подробно механизмы внимания обсуждаются в разделе 15.4.) Это называется **сопоставляющей сетью** (matching network) [Vin+16] (см. иллюстрацию на рис. 19.16).

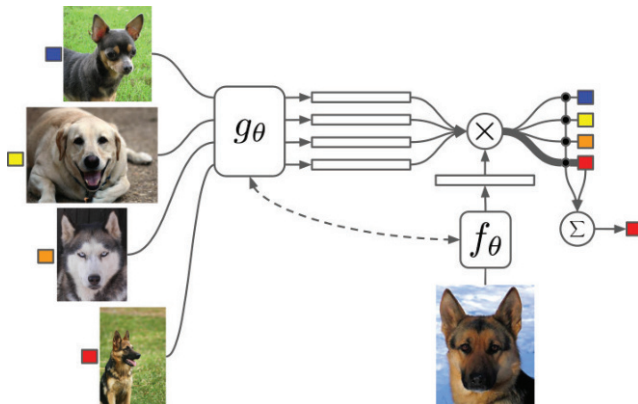


Рис. 19.16 ❖ Сопоставляющая сеть для обучения на одном примере.
На основе рис. 1 из работы [Vin+16]. Печатается с разрешения Ориола Виньялса

Мы можем обучить функции f и g на нескольких небольших наборах данных, как при метаобучении (раздел 19.5). Точнее, пусть \mathcal{D} – большой размеченный набор данных (например, ImageNet), а $p(\mathcal{L})$ – распределение его меток. Создадим задачу, произведя выборку из малого набора меток (скажем, 25), $\mathcal{L} \sim p(\mathcal{L})$, затем выборку из малого опорного набора примеров из \mathcal{D} с этими метками, $\mathcal{S} \sim \mathcal{L}$, и, наконец, выборку из малого тестового набора с теми же метками, $\mathcal{T} \sim \mathcal{L}$. Затем обучим модель предсказывать тестовые метки на опорном наборе, т. е. оптимизировать следующую целевую функцию:

$$\mathcal{L}(\theta; \mathcal{D}) = \mathbb{E}_{\mathcal{L} \sim p(\mathcal{L})} \left[\mathbb{E}_{\mathcal{S} \sim \mathcal{L}, \mathcal{T} \sim \mathcal{L}} \left[\sum_{(\mathbf{x}, y) \in \mathcal{T}} \log p_\theta(y | \mathbf{x}, \mathcal{S}) \right] \right]. \quad (19.55)$$

После обучения мы замораживаем θ и применяем формулу (19.53) к тестовому опорному набору \mathcal{S} .

19.7. ОБУЧЕНИЕ СО СЛАБЫМ УЧИТЕЛЕМ

Термин «**обучение со слабым учителем**» (weakly supervised learning) относится к ситуации, где метки, ассоциированные с каждым вектором признаков в обучающем наборе, неточны.

Один из возможных сценариев – когда для каждого примера имеется распределение меток, а не единственная метка. К счастью, мы все же можем выполнить обучение методом максимального правдоподобия: нужно просто минимизировать перекрестную энтропию:

$$\mathcal{L}(\theta) = - \sum_n \sum_y p(y | \mathbf{x}_n) \log q_\theta(y | \mathbf{x}_n), \quad (19.56)$$

где $p(y | \mathbf{x}_n)$ – распределение меток для примера n , а $q_\theta(y | \mathbf{x}_n)$ – предсказанное распределение. На самом деле часто даже полезно искусственно заменять

точные метки «мягкой» версией, где дельта-функция заменена распределением, которое назначает 90 % массы вероятности наблюдаемой метке, а оставшуюся равномерно распределяет между остальными вариантами. Это называется **сглаживанием меток** и является полезной формой регуляризации (см., например, [МКН19]).

Другой сценарий – когда имеется множество, или **мешок**, экземпляров $\mathbf{x}_n = \{\mathbf{x}_{n,1}, \dots, \mathbf{x}_{n,B}\}$ и одна метка y_n для всего пакета, а не для его отдельных членов, y_{nb} . Часто предполагается, если какой-нибудь элемент мешка положителен, то и весь мешок помечается как положительный, т. е. $y_n = \bigvee_{b=1}^B y_{nb}$, но мы не знаем, какой именно элемент стал причиной положительного исхода. Однако если все элементы отрицательны, то мешок в целом отрицателен. Это называется **обучением на нескольких экземплярах** (MIL) [DLLP97]. (Недавний пример такого подхода в контексте обучения оценок риска заболеть COVID-19 см. в работе [MKS21].) Для решения задачи MIL можно использовать различные алгоритмы, зависящие от того, какие предположения мы делаем о корреляции между метками в каждом мешке, и от ожидаемой доли положительных примеров (см., например, [KF05]).

Еще один сценарий называется **дальнее управление** (distant supervision) [Min+09], он часто встречается при обучении систем извлечения информации. Идея в том, что имеется некоторый факт, например «Женаты(В,М)», достоверность которого не подлежит сомнению (потому что он хранится в базе данных). И этот факт мы используем для пометки любого предложения (в неразмеченном корпусе обучающих документов), в котором упомянуты сущности В и М, как положительный пример отношения «Женаты». Например, предложение «В и М пригласили 100 человек на свою свадьбу» будет помечено как положительное. Но такая эвристика может давать и ложноположительные результаты, например предложение «В и М пошли поужинать» тоже будет помечено как положительное. Таким образом, результирующие метки будут зашумленными. Вопрос о том, как бороться с зашумленностью меток, обсуждается в разделе 10.4.

19.8. УПРАЖНЕНИЯ

Упражнение 19.1 [уравнение информационного выигрыша].

Рассмотрим следующие две целевых функции для вычисления полезности опрашивания примера \mathbf{x} в активном обучении:

$$U(\mathbf{x}) \triangleq \mathbb{H}(p(\boldsymbol{\theta}|\mathcal{D})) - \mathbb{E}_{p(y|\mathbf{x},\mathcal{D})}[\mathbb{H}(p(\boldsymbol{\theta}|\mathcal{D}, \mathbf{x}, y))]; \quad (19.57)$$

$$U'(\mathbf{x}) \triangleq \mathbb{E}_{p(y|\mathbf{x},\mathcal{D})}[\mathbb{KL}(p(\boldsymbol{\theta}|\mathcal{D}, \mathbf{x}, y)||p(\boldsymbol{\theta}|\mathcal{D}))]. \quad (19.58)$$

Докажите, что они равны.

Глава 20

Понижение размерности

Широко распространенная форма обучения без учителя – **понижение размерности**, когда требуется обучить отображение из видимого пространства высокой размерности, $\mathbf{x} \in \mathbb{R}^D$, в латентное пространство меньшей размерности, $\mathbf{z} \in \mathbb{R}^L$. Это может быть либо параметрическая модель $\mathbf{z} = f(\mathbf{x}; \boldsymbol{\theta})$, применимая к любому входу, либо непараметрическое отображение, когда вычисляется **погружение** \mathbf{z}_n для всех входов \mathbf{x}_n , принадлежащих набору данных, и только для них. Второй подход чаще всего используется для визуализации данных, а первый еще и как шаг предобработки для других видов алгоритмов обучения. Например, мы могли бы понизить размерность, обучив погружение \mathbf{x} в \mathbf{z} , а затем обучив на этом погружении простой линейный классификатор, который отображает \mathbf{z} в y .

20.1. Метод главных компонент

Самая простая и популярная форма понижения размерности – **метод главных компонент** (principal components analysis – **PCA**). Основная идея – найти линейную ортогональную проекцию данных высокой размерности $\mathbf{x} \in \mathbb{R}^D$ в подпространство меньшей размерности $\mathbf{z} \in \mathbb{R}^L$, такое что низкоразмерное представление является «хорошей аппроксимацией» исходных данных в следующем смысле: если в результате проецирования (**кодирования**) \mathbf{x} мы получаем $\mathbf{z} = \mathbf{W}^T \mathbf{x}$ и затем выполняем обратное проецирование (**декодирование**) \mathbf{z} и получаем $\hat{\mathbf{x}} = \mathbf{W} \mathbf{z}$, то хотелось бы, чтобы $\hat{\mathbf{x}}$ было близко к \mathbf{x} в смысле метрики ℓ_2 . Формально мы можем определить следующую **ошибку реконструкции**, или **искажение**:

$$\mathcal{L}(\mathbf{W}) \triangleq \frac{1}{N} \sum_{n=1}^N \|\mathbf{x}_n - \text{decode}(\text{encode}(\mathbf{x}_n; \mathbf{W}); \mathbf{W})\|_2^2, \quad (20.1)$$

где этапы кодирования и декодирования – линейные отображения.

В разделе 20.1.2 мы покажем, что эту целевую функцию можно минимизировать, положив $\hat{\mathbf{W}} = \mathbf{U}_L$, где \mathbf{U}_L образована L собственными векторами

эмпирической ковариационной матрицы с наибольшими собственными значениями:

$$\hat{\Sigma} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T = \frac{1}{N} \mathbf{X}_c^T \mathbf{X}_c, \quad (20.2)$$

где \mathbf{X}_c – центрированная версия матрицы плана размера $N \times D$. В разделе 20.2.2 мы покажем, что это эквивалентно максимизации правдоподобия латентной линейной гауссовой модели; этот метод называется вероятностным PCA.

20.1.1. Примеры

Прежде чем переходить к деталям, приведем несколько примеров.

На рис. 20.1 показан очень простой пример, когда двумерные данные проецируются на одномерную прямую. Это направление улавливает наибольшую изменчивость данных.

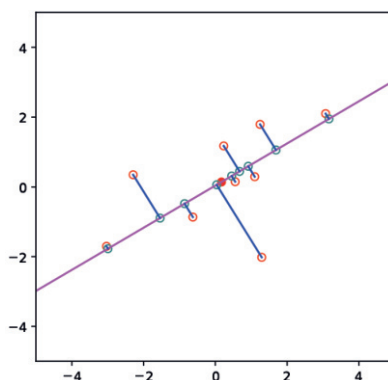


Рис. 20.1 ❖ Иллюстрация PCA в случае, когда двумерная плоскость проецируется на одномерную прямую. Кружочками обозначены оригинальные точки данных, крестиками – реконструкции. Красная звездочка – среднее данных. Построено программой по адресу figures.probl.ai/book1/20.1

На рис. 20.2 мы показываем, что происходит при проецировании некоторых изображений цифры 9 из набора MNIST на плоскость. Хотя размерность входов очень велика ($28 \times 28 = 784$ измерения), количество «эффективных степеней свободы» гораздо меньше, потому что пиксели коррелированы, а многие цифры внешне похожи. Поэтому мы можем представить каждое изображение точкой в линейном пространстве низкой размерности.

В общем случае бывает трудно интерпретировать латентные измерения, на которые проецируются данные. Однако, глядя на несколько точек, спроецированных на данное направление, и на примеры, из которых они взяты,

мы видим, что первая главная компонента (горизонтальное направление), скорее всего, отражает ориентацию цифры, а вторая компонента (вертикальное направление) – толщину линии.

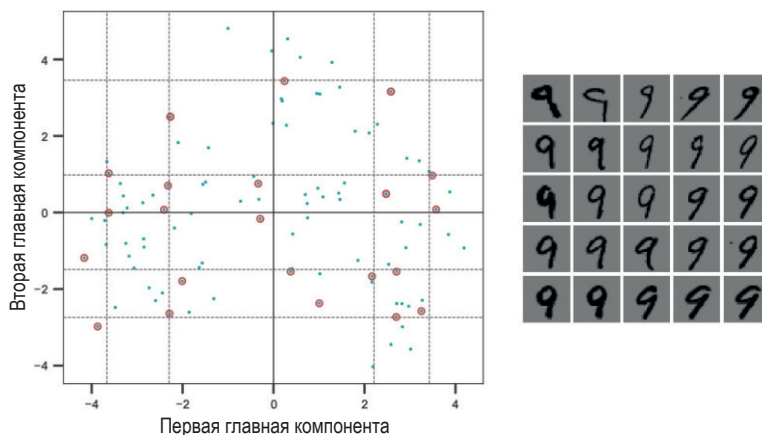


Рис. 20.2 ❖ Применение PCA к цифрам класса 9 из набора MNIST. Сетка образована квантилями 5, 25, 50, 75, 95 % распределения данных в каждом направлении. Кружочки – ближайшие к вершинам сетки спроецированные изображения. На основе рис. 14.23 из работы [HTF09]. Построено программой по адресу figures.problml.ai/book1/20.2

На рис. 20.3 показано применение PCA еще к одному набору изображений – базе данных лиц, составленной компанией Olivetti, в которую входят полутоновые изображения размера 64×64 . Мы проецируем их на трехмерное подпространство. Результирующие базисные векторы (столбцы матрицы проекции \mathbf{W}) показаны в виде изображений на рис. 20.3b; они называются **собственными лицами** [Tur13] по причинам, которые будут объяснены в разделе 20.1.2. Мы видим, что главные моды изменения данных связаны



Рис. 20.3 ❖ (a) Несколько изображений размером 64×64 пикселя, случайно выбранных из базы данных лиц, принадлежащей компании Olivetti. (b) Среднее и первые три главных компоненты, представленные в виде изображений. Построено программой по адресу figures.problml.ai/book1/20.3

с общим освещением, а также с различиями в области бровей. Взяв достаточное число измерений (но меньше изначальных 4096), мы сможем подать представление $\mathbf{z} = \mathbf{W}^T \mathbf{x}$ на вход классификатора по ближайшему соседу для распознавания лиц: это быстрее и надежнее, чем работать в пространстве пикселей [MWP98].

20.1.2. Вывод алгоритма

Пусть имеется неразмеченный набор данных $\mathcal{D} = \{\mathbf{x}_n : n = 1 \dots N\}$, где $\mathbf{x}_n \in \mathbb{R}^D$. Мы можем представить его в виде матрицы данных \mathbf{X} размера $N \times D$. Будем предполагать, что $\bar{\mathbf{x}} = (1/N) \sum_{n=1}^N \mathbf{x}_n = \mathbf{0}$, этого можно добиться центрированием данных.

Мы хотели бы аппроксимировать каждый пример \mathbf{x}_n представлением низкой размерности, $\mathbf{z}_n \in \mathbb{R}^L$. Предположим, что каждый \mathbf{x}_n можно «объяснить» в терминах взвешенной комбинации базисных функций $\mathbf{w}_1, \dots, \mathbf{w}_L$, где все $\mathbf{w}_k \in \mathbb{R}^D$, а веса определяются векторами $\mathbf{z}_n \in \mathbb{R}^L$, т. е. $\mathbf{x}_n \approx \sum_{k=1}^L z_{nk} \mathbf{w}_k$. Вектор \mathbf{z}_n – это низкоразмерное представление \mathbf{x}_n , которое называется **латентным вектором**, потому что состоит из латентных, или «скрытых», значений, которые непосредственно в данных не наблюдаются. Совокупность этих латентных переменных называется **латентными факторами**.

Измерить ошибку такой аппроксимации можно следующим образом:

$$\mathcal{L}(\mathbf{W}, \mathbf{Z}) = \frac{1}{N} \|\mathbf{X} - \mathbf{Z}\mathbf{W}^T\|_F^2 = \frac{1}{N} \|\mathbf{X}^T - \mathbf{W}\mathbf{Z}^T\|_F^2 = \frac{1}{N} \sum_{n=1}^N \|\mathbf{x}_n - \mathbf{W}\mathbf{z}_n\|^2, \quad (20.3)$$

где строки \mathbf{Z} – низкоразмерные версии строк \mathbf{X} . Эта величина называется (средней) **ошибкой реконструкции**, поскольку мы аппроксимируем каждый \mathbf{x}_n вектором $\hat{\mathbf{x}}_n = \mathbf{W}\mathbf{z}_n$.

Мы хотим минимизировать эту ошибку при условии, что \mathbf{W} – ортогональная матрица. Ниже будет показано, что оптимальное решение получается, когда $\hat{\mathbf{W}} = \mathbf{U}_L$, где \mathbf{U}_L содержит L собственных векторов эмпирической ковариационной матрицы с наибольшими собственными значениями.

20.1.2.1. Базовый случай

Начнем с оценки наилучшего одномерного решения, $\mathbf{w}_1 \in \mathbb{R}^D$. Остальные базисные векторы $\mathbf{w}_2, \mathbf{w}_3, \dots$ мы найдем позже.

Обозначим коэффициенты при первом базисном векторе для каждой из точек данных $\tilde{\mathbf{z}}_1 = [z_{11}, \dots, z_{N1}] \in \mathbb{R}^N$. Ошибка реконструкции равна

$$\mathcal{L}(\mathbf{w}_1, \tilde{\mathbf{z}}_1) = \frac{1}{N} \sum_{n=1}^N \|\mathbf{x}_n - z_{n1} \mathbf{w}_1\|^2 = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - z_{n1} \mathbf{w}_1)^T (\mathbf{x}_n - z_{n1} \mathbf{w}_1) \quad (20.4)$$

$$= \frac{1}{N} \sum_{n=1}^N [\mathbf{x}_n^T \mathbf{x}_n - 2z_{n1} \mathbf{w}_1^T \mathbf{x}_n + z_{n1}^2 \mathbf{w}_1^T \mathbf{w}_1] \quad (20.5)$$

$$= \frac{1}{N} \sum_{n=1}^N [\mathbf{x}_n^T \mathbf{x}_n - 2z_{n1} \mathbf{w}_1^T \mathbf{x}_n + z_{n1}^2], \quad (20.6)$$

так как $\mathbf{w}_1^\top \mathbf{w}_1 = 1$ (в силу предположения об ортонормированности). Приравнявая производные по z_{n1} нулю, получаем

$$\frac{\partial}{\partial z_{n1}} \mathcal{L}(\mathbf{w}_1, \tilde{\mathbf{z}}_1) = \frac{1}{N} [-2\mathbf{w}_1^\top \mathbf{x}_n + 2z_{n1}] = 0 \Rightarrow z_{n1} = \mathbf{w}_1^\top \mathbf{x}_n. \quad (20.7)$$

Следовательно, оптимальное вложение получается при ортогональном проецировании данных на направление \mathbf{w}_1 (см. рис. 20.1а). Подстановка в (20.4) дает потерю для таких весов:

$$\mathcal{L}(\mathbf{w}_1) = \mathcal{L}(\mathbf{w}_1, \tilde{\mathbf{z}}_1^*(\mathbf{w}_1)) = \frac{1}{N} \sum_{n=1}^N [\mathbf{x}_n^\top \mathbf{x}_n - z_{n1}^2] = \text{const} - \frac{1}{N} \sum_{n=1}^N z_{n1}^2. \quad (20.8)$$

Чтобы решить это уравнение относительно \mathbf{w}_1 , заметим, что

$$\mathcal{L}(\mathbf{w}_1) = -\frac{1}{N} \sum_{n=1}^N z_{n1}^2 = -\frac{1}{N} \sum_{n=1}^N \mathbf{w}_1^\top \mathbf{x}_n \mathbf{x}_n^\top \mathbf{w}_1 = -\mathbf{w}_1^\top \hat{\Sigma} \mathbf{w}_1, \quad (20.9)$$

где Σ – эмпирическая ковариационная матрица (поскольку мы предположили, что данные центрированы). Эта функция тривиально оптимизируется, если положить $\|\mathbf{w}_1\| \rightarrow \infty$, поэтому наложим ограничение $\|\mathbf{w}_1\| = 1$ и вместо этого оптимизируем функцию:

$$\tilde{\mathcal{L}}(\mathbf{w}_1) = \mathbf{w}_1^\top \hat{\Sigma} \mathbf{w}_1 + \lambda_1 (\mathbf{w}_1^\top \mathbf{w}_1 - 1), \quad (20.10)$$

где λ_1 – множитель Лагранжа (см. раздел 8.5.1). Приравнявая производные нулю, получаем

$$\frac{\partial}{\partial \mathbf{w}_1} \tilde{\mathcal{L}}(\mathbf{w}_1) = 2\hat{\Sigma} \mathbf{w}_1 - 2\lambda_1 \mathbf{w}_1; \quad (20.11)$$

$$\hat{\Sigma} \mathbf{w}_1 = \lambda_1 \mathbf{w}_1. \quad (20.12)$$

Поэтому оптимальное направление, на которое следует проецировать данные, определяется собственным вектором ковариационной матрицы. Умножая слева на \mathbf{w}_1^\top (и пользуясь тем, что $\mathbf{w}_1^\top \mathbf{w}_1 = 1$), находим

$$\mathbf{w}_1^\top \hat{\Sigma} \mathbf{w}_1 = \lambda_1. \quad (20.13)$$

Поскольку мы хотим максимизировать эту величину (минимизировать потерю), то выбираем собственный вектор, соответствующий наибольшему собственному значению.

20.1.2.2. Оптимальный вектор весов максимизирует дисперсию спроецированных данных

Прежде чем продолжить, сделаем интересное наблюдение. Поскольку данные центрированы, имеем

$$\mathbb{E}[z_{n1}] = \mathbb{E}[\mathbf{x}_n^\top \mathbf{w}_1] = \mathbb{E}[\mathbf{x}_n]^\top \mathbf{w}_1 = 0. \quad (20.14)$$

Следовательно, дисперсия спроецированных данных равна

$$\mathbb{V}[\tilde{\mathbf{z}}_1] = \mathbb{E}[\tilde{\mathbf{z}}_1^2] - (\mathbb{E}[\tilde{\mathbf{z}}_1])^2 = \frac{1}{N} \sum_{n=1}^N z_{n1}^2 - 0 = -\mathcal{L}(\mathbf{w}_1) + \text{const}. \quad (20.15)$$

Поэтому *минимизация* ошибки реконструкции эквивалентна *максимизации* дисперсии спроецированных данных:

$$\underset{\mathbf{w}_1}{\operatorname{argmin}} \mathcal{L}(\mathbf{w}_1) = \underset{\mathbf{w}_1}{\operatorname{argmax}} \mathbb{V}[\tilde{\mathbf{z}}_1(\mathbf{w}_1)]. \quad (20.16)$$

Именно по этой причине часто говорят, что РСА находит направления максимальной дисперсии (см. иллюстрацию на рис. 20.4). Однако формулировку с минимальной ошибкой проще понять, и она более общая.

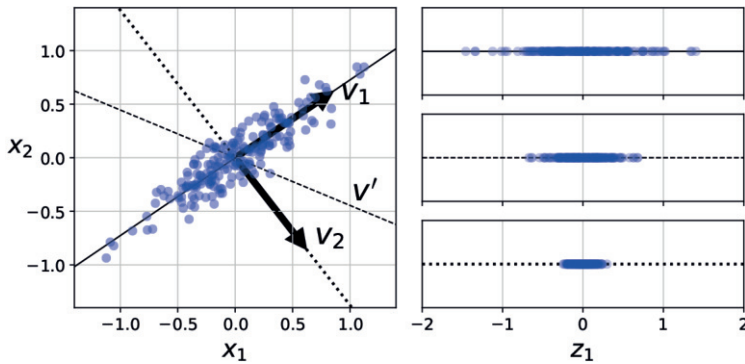


Рис. 20.4 ❖ Дисперсия точек, спроецированных на разные одномерные векторы. v_1 – первая главная компонента, которая максимизирует дисперсию проекции. v_2 – вторая главная компонента, ее направление ортогонально v_1 . Наконец, v_0 – какой-то вектор между v_1 и v_2 . На основе рис. 8.7 из работы [Gér19]. Построено программой по адресу figures.probml.ai/book1/20.4

20.1.2.3. Шаг индукции

Теперь найдем другое направление \mathbf{w}_2 , чтобы продолжить минимизацию ошибки реконструкции, теперь уже при условиях $\mathbf{w}_1^\top \mathbf{w}_2 = 0$ и $\mathbf{w}_2^\top \mathbf{w}_2 = 1$. Ошибка равна

$$\mathcal{L}(\mathbf{w}_1, \tilde{\mathbf{z}}_1, \mathbf{w}_2, \tilde{\mathbf{z}}_2) = \frac{1}{N} \sum_{n=1}^N \|\mathbf{x}_n - z_{n1} \mathbf{w}_1 - z_{n2} \mathbf{w}_2\|^2. \quad (20.17)$$

Оптимизация относительно \mathbf{w}_1 и \mathbf{z}_1 дает то же решение, что и прежде. В упражнении 20.3 вам будет предложено доказать, что из уравнения $\partial \mathcal{L} / \partial z_2 = 0$ следует, что $z_{n2} = \mathbf{w}_2^\top \mathbf{x}_n$. После подстановки получаем

$$\mathcal{L}(\mathbf{w}_2) = \frac{1}{N} \sum_{n=1}^N [\mathbf{x}_n^T \mathbf{x}_n - \mathbf{w}_1^T \mathbf{x}_n \mathbf{x}_n^T \mathbf{w}_1 - \mathbf{w}_2^T \mathbf{x}_n \mathbf{x}_n^T \mathbf{w}_2] = \text{const} - \mathbf{w}_2^T \hat{\Sigma} \mathbf{w}_2. \quad (20.18)$$

Опуская постоянный член, подставляя оптимальный \mathbf{w}_1 и добавляя ограничения, получаем

$$\tilde{\mathcal{L}}(\mathbf{w}_2) = \mathbf{w}_2^T \hat{\Sigma} \mathbf{w}_2 + \lambda_2 (\mathbf{w}_2^T \mathbf{w}_2 - 1) + \lambda_{12} (\mathbf{w}_2^T \mathbf{w}_1 - 0). \quad (20.19)$$

В упражнении 20.3 вам будет предложено доказать, что решение дает собственный вектор, соответствующий второму по величине собственному значению:

$$\hat{\Sigma} \mathbf{w}_2 = \lambda_2 \mathbf{w}_2. \quad (20.20)$$

Продолжая рассуждение таким же образом, можно показать, что $\hat{\mathbf{W}} = \mathbf{U}_L$.

20.1.3. Вычислительные трудности

В этом разделе мы обсудим различные практические трудности, связанные с применением РСА.

20.1.3.1. Ковариационная матрица и корреляционная матрица

Мы все время имели дело со спектральным разложением ковариационной матрицы. Однако лучше вместо нее использовать корреляционную матрицу. Причина в том, что в противном случае РСА может быть «введен в заблуждение» направлениями, в которых дисперсия высока просто из-за шкалы измерений. На рис. 20.5 приведен такой пример. На левом рисунке диапазон значений по вертикальной оси шире, чем по горизонтальной. Из-за этого первая главная компонента выглядит не вполне «естественно». На правом

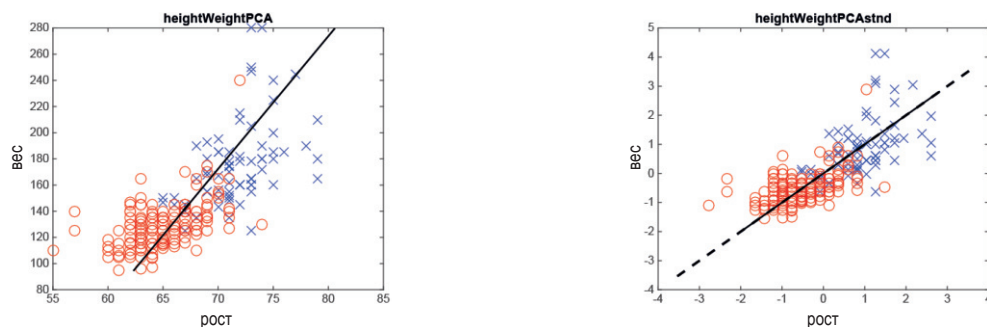


Рис. 20.5. Влияние стандартизации на РСА применительно к набору данных о росте и весе (красный=женщина, синий=мужчина). Слева: РСА для первичных данных. Справа: РСА для стандартизованных данных. Построено программой по адресу figures.problmlai/book1/20.5

рисунке показаны результаты PCA после стандартизации данных (что эквивалентно использованию корреляционной матрицы вместо ковариационной) – выглядит гораздо лучше.

20.1.3.2. Работа с данными высокой размерности

Мы представили PCA как задачу нахождения собственных векторов ковариационной матрицы $\mathbf{X}^T\mathbf{X}$ размера $D \times D$. Если $D > N$, то быстрее работать с матрицей Грама $\mathbf{X}\mathbf{X}^T$ размера $N \times N$. Покажем, как это сделать.

Предположим сначала, что \mathbf{U} – ортогональная матрица, образованная собственными векторами $\mathbf{X}\mathbf{X}^T$, собственные значения которых составляют матрицу $\mathbf{\Lambda}$. По определению имеем $(\mathbf{X}\mathbf{X}^T)\mathbf{U} = \mathbf{U}\mathbf{\Lambda}$. Умножение слева на \mathbf{X}^T дает

$$(\mathbf{X}^T\mathbf{X})(\mathbf{X}^T\mathbf{U}) = (\mathbf{X}^T\mathbf{U})\mathbf{\Lambda}, \quad (20.21)$$

откуда видно, что собственные векторы $\mathbf{X}^T\mathbf{X}$ образуют матрицу $\mathbf{V} = \mathbf{X}^T\mathbf{U}$, а собственные значения, как и раньше, составляют матрицу $\mathbf{\Lambda}$. Однако эти собственные векторы не нормированы, так как $\|\mathbf{v}_j\|^2 = \mathbf{u}_j^T\mathbf{X}\mathbf{X}^T\mathbf{u}_j = \lambda_j\mathbf{u}_j^T\mathbf{u}_j = \lambda_j$. Нормированные собственные векторы образуют матрицу:

$$\mathbf{V} = \mathbf{X}^T\mathbf{U}\mathbf{\Lambda}^{-1/2}. \quad (20.22)$$

Это дает альтернативный способ вычисления PCA-базиса. А также открывает возможность воспользоваться ядерным трюком, как будет объяснено в разделе 20.4.6.

20.1.3.3. Вычисление PCA с использованием SVD

В этом разделе мы продемонстрируем эквивалентность PCA, вычисляемого с помощью собственных векторов (раздел 20.1), и усеченного SVD¹.

Пусть $\mathbf{U}_\Sigma\mathbf{\Lambda}_\Sigma\mathbf{U}_\Sigma^T$ – первые L векторов из спектрального разложения ковариационной матрицы $\Sigma \propto \mathbf{X}^T\mathbf{X}$ (предполагается, что \mathbf{X} центрирована). Напомним (см. раздел 20.1.2), что оптимальную оценку проекционных весов \mathbf{W} дают первые L собственных значений, так что $\mathbf{W} = \mathbf{U}_\Sigma$.

Пусть теперь $\mathbf{U}_X\mathbf{S}_X\mathbf{V}_X^T \approx \mathbf{X}$ – L -усеченная SVD-аппроксимация матрицы данных \mathbf{X} . Из формулы (7.184) мы знаем, что правосингулярные векторы \mathbf{X} являются собственными векторами $\mathbf{X}^T\mathbf{X}$, так что $\mathbf{V}_X = \mathbf{U}_\Sigma = \mathbf{W}$. (Кроме того, собственные значения ковариационной матрицы связаны с сингулярными числами матрицы данных соотношением $\lambda_k = s_k^2/N$.)

Теперь предположим, что нас интересуют спроецированные точки (называемые также главными компонентами или оценками главных компонент), а не матрица проекции. Имеем

$$\mathbf{Z} = \mathbf{X}\mathbf{W} = \mathbf{U}_X\mathbf{S}_X\mathbf{V}_X^T\mathbf{V}_X = \mathbf{U}_X\mathbf{S}_X. \quad (20.23)$$

Наконец, если мы хотим приближенно реконструировать данные, то имеем

$$\hat{\mathbf{X}} = \mathbf{Z}\mathbf{W}^T = \mathbf{U}_X\mathbf{S}_X\mathbf{V}_X^T. \quad (20.24)$$

¹ Более обстоятельное объяснение можно найти по адресу <https://bit.ly/2I566OK>.

Это в точности то же самое, что усеченная SVD-аппроксимация (раздел 7.5.5).

Таким образом, PCA можно реализовать либо как спектральное разложение Σ , либо как сингулярное разложение \mathbf{X} . Последнее часто предпочтительнее из соображений вычислительного характера. Для задач очень большой размерности можно использовать рандомизированный алгоритм SVD, см., например, [HMT11; SKT14; DM16]. Например, рандомизированный решатель в библиотеки `sklearn` требует времени $O(NL^2) + O(L^3)$ для N примеров и L главных компонент, тогда как точный алгоритм SVD требует времени $O(ND^2) + O(D^3)$.

20.1.4. Выбор числа латентных измерений

В этом разделе мы обсудим, как выбирать число латентных измерений L в PCA.

20.1.4.1. Ошибка реконструкции

Определим ошибку реконструкции на наборе данных \mathcal{D} при использовании модели с L измерениями:

$$\mathcal{L}_L = \frac{1}{|\mathcal{D}|} \sum_{n \in \mathcal{D}} \|\mathbf{x}_n - \hat{\mathbf{x}}_n\|^2, \quad (20.25)$$

где реконструкция вычисляется по формуле $\hat{\mathbf{x}}_n = \mathbf{W}\mathbf{z}_n + \boldsymbol{\mu}$, где $\mathbf{z}_n = \mathbf{W}^T(\mathbf{x}_n - \boldsymbol{\mu})$, $\boldsymbol{\mu}$ – эмпирическое среднее, а \mathbf{W} оценивается так же, как и выше. На рис. 20.6а показан график зависимости \mathcal{L}_L от L на обучающем наборе данных MNIST. Как видим, кривая быстро спадает, а это значит, что большую часть эмпирической корреляции пикселей можно уловить с помощью небольшого числа факторов.

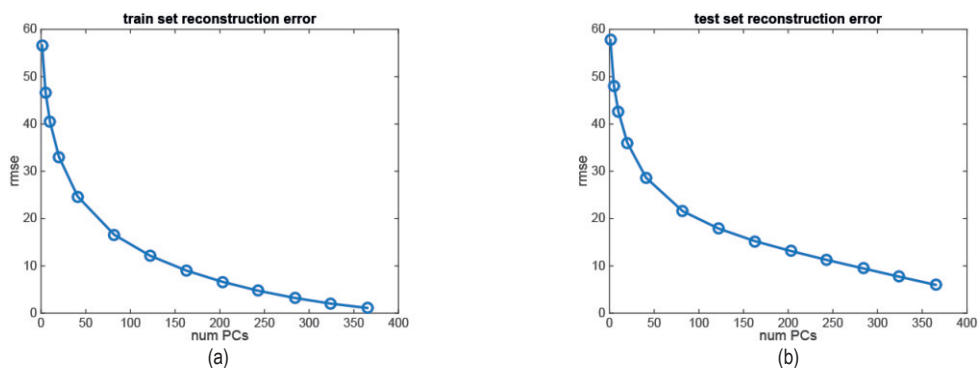


Рис. 20.6 ❖ Зависимость ошибки реконструкции на наборе MNIST от числа латентных измерений в PCA. (a) Обучающий набор. (b) Тестовый набор. Построено программой по адресу figures.probml.ai/book1/20.6

Конечно, если взять $L = \text{rank}(\mathbf{X})$, то мы получим нулевую ошибку реконструкции на обучающем наборе. Чтобы избежать переобучения, естественно построить график ошибки реконструкции на тестовом наборе. Он показан на рис. 20.6b. Здесь мы видим, что ошибка уменьшается, даже когда модель усложняется! Таким образом, мы не наблюдаем обычной U-образной кривой, ожидаемой при обучении с учителем. Проблема в том, что PCA не настоящая порождающая модель данных: при увеличении числа латентных измерений она будет аппроксимировать тестовые данные более точно. (Аналогичная проблема возникает, если построить график ошибки реконструкции на тестовом наборе, используя кластеризацию методом К средних, обсуждаемую в разделе 21.3.7.) Некоторые решения этой проблемы мы обсудим ниже.

20.1.4.2. Графики каменистой осыпи

Стандартная альтернатива графику зависимости реконструкции ошибки от L – **график каменистой осыпи** (scree plot), отражающий зависимость собственных значений λ_j , упорядоченных по убыванию абсолютной величины, от j . Можно показать (упражнение 20.4), что

$$L_L = \sum_{j=L+1}^D \lambda_j. \quad (20.26)$$

Таким образом, при увеличении числа измерений собственные значения уменьшаются, а вместе с ними и ошибка реконструкции, как показано на рис. 20.7a¹. С этой величиной связана также **доля объясненной дисперсии**, определяемая как

$$F_L = \frac{\sum_{j=1}^L \lambda_j}{\sum_{j'=1}^{L^{\max}} \lambda_{j'}}. \quad (20.27)$$

Она отражает ту же информацию, что график каменистой осыпи, но возрастает вместе с L (см. рис. 20.7b).

20.1.4.3. Правдоподобие профиля

Хотя график ошибки реконструкции не имеет U-образной формы, часто кривая содержит «колено», или «угловой изгиб», где ошибка резко переходит от сравнительно больших значений к сравнительно малым. Идея в том, что для $L < L^*$, где L^* – «истинная» латентная размерность (или число кластеров), скорость убывания функции ошибки будет высокой, а для $L > L^*$ выигрыш будет меньше, потому что модель и так уже достаточно сложна, чтобы уловить истинное распределение.

¹ Название связано с тем, что «график выглядит как склон горы, а “каменистая осыпь” – это те обломки камней, которые упали с горы и лежат у ее подножия». (Цитируется по статье Кеннета Джанда по адресу <https://bit.ly/2kqG1yW>.)

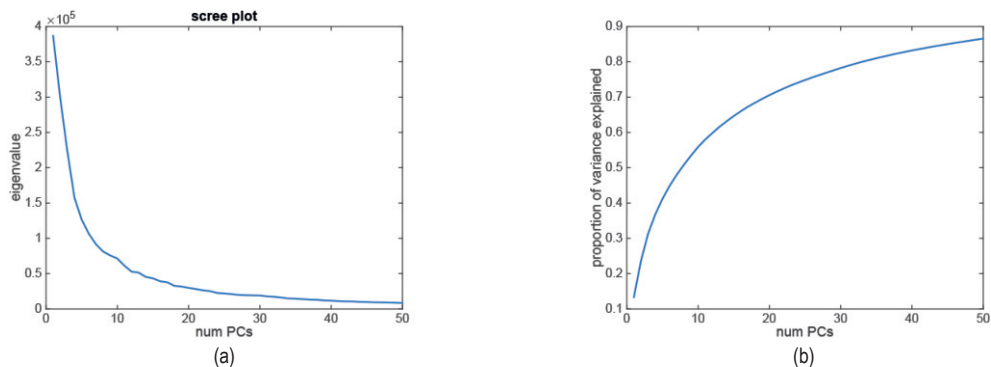


Рис. 20.7 ❖ (а) График каменной осыпи для того же обучающего набора, что на рис. 20.6а. (б) Доля объясненной дисперсии. Построено программой по адресу figures.problml.ai/book1/20.7

Один из способов автоматизировать обнаружение этого изменения градиента кривой – вычислить **правдоподобие профиля** (profile likelihood), как предложено в работе [ZG06]. Идея в следующем. Пусть λ_L – некоторая мера ошибки, вносимой моделью размера L , так что $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{L_{\max}}$. В случае РСА это собственные значения, но идея применима также к ошибке реконструкции при кластеризации методом К средних (см. раздел 21.3.7). Теперь рассмотрим разбиение этих значений на две группы – для $k < L$ и $k > L$, где L – некоторая пороговая величина, которую мы хотим найти. Чтобы измерить качество L , мы будем использовать простую модель точки перехода, где $\lambda_k \sim \mathcal{N}(\mu_1, \sigma^2)$, если $k \leq L$, и $\lambda_k \sim \mathcal{N}(\mu_2, \sigma^2)$, если $k > L$. (Важно, что σ^2 в обеих моделях одинаково, чтобы предотвратить переобучение в случае, когда в одном режиме данных меньше, чем в другом.) Внутри каждого режима предполагается, что λ_k независимы и одинаково распределены, что, очевидно, неправильно, но для наших целей годится. Эту модель можно обучить для каждого $L = 1 : L_{\max}$, разбив данные на части и вычислив MLE с применением объединенной оценки дисперсии:

$$\mu_1(L) = \frac{\sum_{k \leq L} \lambda_k}{L}; \quad (20.28)$$

$$\mu_2(L) = \frac{\sum_{k > L} \lambda_k}{L_{\max} - L}; \quad (20.29)$$

$$\sigma^2(L) = \frac{\sum_{k \leq L} (\lambda_k - \mu_1(L))^2 + \sum_{k > L} (\lambda_k - \mu_2(L))^2}{L_{\max}}. \quad (20.30)$$

Затем мы можем вычислить логарифмическое правдоподобие профиля:

$$\ell(L) = \sum_{k=1}^L \log \mathcal{N}(\lambda_k | \mu_1(L), \sigma^2(L)) + \sum_{k=L+1}^{L_{\max}} \log \mathcal{N}(\lambda_k | \mu_2(L), \sigma^2(L)). \quad (20.31)$$

Смотрите иллюстрацию на рис. 20.8. Мы видим, что величина $L^* = \arg\max \ell(L)$ корректно определена.

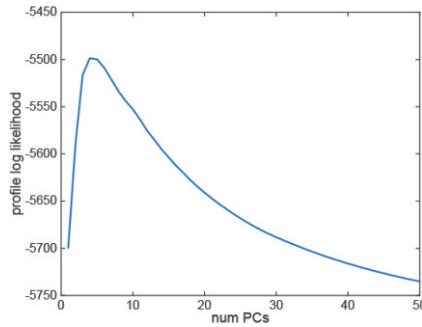


Рис. 20.8 ❖ Правдоподобие профиля, соответствующее модели PCA на рис. 20.6(а). Построено программой по адресу figures.problm.ai/book1/20.8

20.2. ФАКТОРНЫЙ АНАЛИЗ*

PCA – простой метод вычисления линейного представления данных низкой размерности. В этом разделе мы опишем обобщение PCA – **факторный анализ** (ФА). Он основан на вероятностной модели, а это значит, что его можно рассматривать как строительный блок для более сложных моделей, например смеси моделей ФА в разделе 20.2.6 или нелинейной модели ФА в разделе 20.3.5. PCA является частным предельным случаем ФА, как обсуждается в разделе 20.2.2.

20.2.1. Порождающая модель

Факторный анализ соответствует следующей линейно-гауссовой порождающей модели латентных переменных:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0); \quad (20.32)$$

$$p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}), \quad (20.33)$$

где \mathbf{W} – **матрица факторных нагрузок** размера $D \times L$, а $\boldsymbol{\Psi}$ – диагональная ковариационная матрица размера $D \times D$.

ФА можно рассматривать как низкоранговую версию гауссова распределения. Чтобы убедиться в этом, заметим, что индуцированное маргинальное распределение $p(\mathbf{x}|\boldsymbol{\theta})$ является гауссовым (см. вывод в (3.38)):

$$p(\mathbf{x}|\boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}) \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) d\mathbf{z} \quad (20.34)$$

$$= \mathcal{N}(\mathbf{x}|\mathbf{W}\boldsymbol{\mu}_0 + \boldsymbol{\mu}, \boldsymbol{\Psi} + \mathbf{W}\boldsymbol{\Sigma}_0\mathbf{W}^T). \quad (20.35)$$

Следовательно, $E[\mathbf{x}] = \mathbf{W}\boldsymbol{\mu}_0 + \boldsymbol{\mu}$ и $\text{Cov}[\mathbf{x}] = \mathbf{W}\text{Cov}[\mathbf{z}]\mathbf{W}^T + \boldsymbol{\Psi} = \mathbf{W}\boldsymbol{\Sigma}_0\mathbf{W}^T + \boldsymbol{\Psi}$. Отсюда видно, что можно без ограничения общности положить $\boldsymbol{\mu}_0 = 0$, так как мы всегда можем включить $\mathbf{W}\boldsymbol{\mu}_0$ в $\boldsymbol{\mu}$. Аналогично можно без ограничения общности положить $\boldsymbol{\Sigma}_0 = \mathbf{I}$, введя при необходимости новую матрицу весов $\tilde{\mathbf{W}} = \mathbf{W}\boldsymbol{\Sigma}_0^{-1/2}$. После этих упрощений имеем

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}); \quad (20.36)$$

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}); \quad (20.37)$$

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi}). \quad (20.38)$$

Например, предположим, что $L = 1$, $D = 2$ и $\boldsymbol{\Psi} = \sigma^2\mathbf{I}$. Порождающий процесс в этом случае показан на рис. 20.9. Его можно рассматривать как изотропный гауссов «аэрозольный баллончик», представляющий правдоподобие $p(\mathbf{x}|\mathbf{z})$, который «скользит» вдоль прямой $\mathbf{W}\mathbf{z} + \boldsymbol{\mu}$ при изменении одномерного латентного априорного распределения \mathbf{z} . Это индуцирует вытянутое (и, следовательно, коррелированное) гауссово распределение на плоскости. То есть индуцированное распределение имеет форму $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I})$.

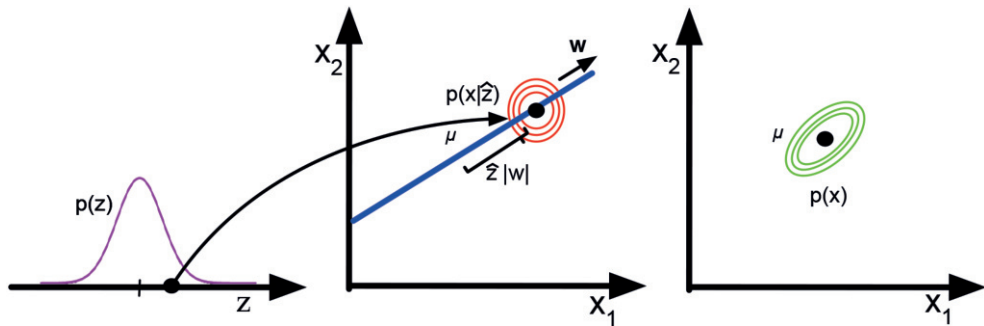


Рис. 20.9 ❖ Иллюстрация порождающего процесса ФА, где $L = 1$ латентных измерений генерируют $D = 2$ наблюдаемых измерений; предполагается, что $\boldsymbol{\Psi} = \sigma^2\mathbf{I}$. Латентный фактор принимает значение $z \in \mathbb{R}$, выбранное из распределения $p(z)$; оно отображается на двумерное смещение $\boldsymbol{\delta} = z\mathbf{w}$, где $\mathbf{w} \in \mathbb{R}^2$, которое прибавляется к $\boldsymbol{\mu}$ для определения гауссова распределения $p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu} + \boldsymbol{\delta}; \sigma^2\mathbf{I})$. Интегрируя по z , мы «сдвигаем» этот круговой гауссов «аэрозольный баллончик» вдоль оси главной компоненты \mathbf{w} , что индуцирует в пространстве \mathbf{x} эллиптические гауссовы линии уровня с центром в $\boldsymbol{\mu}$. На основе рис. 12.9 из работы [Bis06]

В общем случае ФА аппроксимирует ковариационную матрицу видимого вектора низкоранговым разложением:

$$\mathbf{C} = \text{Cov}[\mathbf{x}] = \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi}. \quad (20.39)$$

Количество параметров при этом всего $O(LD)$, что обеспечивает гибкий компромисс между гауссовым распределением с полной ковариационной матрицей с $O(D^2)$ параметрами и диагональной ковариационной матрицей с $O(D)$ параметрами.

Из формулы (20.39) мы видим, что следует наложить на Ψ ограничение диагональности, иначе можно было бы положить $\mathbf{W} = 0$ и тем самым игнорировать латентные факторы, но сохранить возможность моделировать любую ковариацию. Маргинальная дисперсия каждой видимой переменной равна $\mathbb{V}[x_d] = \sum_{k=1}^L w_{dk}^2 + \psi_d$, где первый член – дисперсия вследствие общих факторов, а второй член ψ_d называется **уникальностью** и описывает дисперсию, характерную для данного измерения.

Мы можем оценивать параметры модели ФА с помощью ЕМ-алгоритма (см. раздел 20.2.3). Обучив модель, мы можем вычислить вероятностные латентные погружения, используя распределение $p(\mathbf{z}|\mathbf{x})$. По формуле Байеса для гауссовых распределений, имеем

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mathbf{W}^T \mathbf{C}^{-1}(\mathbf{x} - \boldsymbol{\mu}), \mathbf{I} - \mathbf{W}^T \mathbf{C}^{-1} \mathbf{W}), \quad (20.40)$$

где \mathbf{C} определено формулой (20.39).

20.2.2. Вероятностный РСА

В этом разделе мы рассмотрим частный случай модели факторного анализа, когда столбцы \mathbf{W} ортонормированы, $\Psi = \sigma^2 \mathbf{I}$ и $\boldsymbol{\mu} = \mathbf{0}$. Это модель называется **вероятностным анализом главных компонент** (РРСА) [ТВ99], или **разумным** (sensible) **РСА** [Row97]. Маргинальное распределение видимых переменных имеет вид:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z}, \sigma^2 \mathbf{I}) \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}) d\mathbf{z} = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{C}), \quad (20.41)$$

где

$$\mathbf{C} = \mathbf{W}\mathbf{W}^T + \sigma^2 \mathbf{I}. \quad (20.42)$$

Логарифмическое правдоподобие для РРСА имеет вид:

$$\log p(\mathbf{X}|\boldsymbol{\mu}, \mathbf{W}, \sigma^2) = -\frac{ND}{2} \log(2\pi) - \frac{N}{2} \log |\mathbf{C}| - \frac{1}{2} \sum_{n=1}^N (\mathbf{x}_n - \boldsymbol{\mu})^T \mathbf{C}^{-1} (\mathbf{x}_n - \boldsymbol{\mu}). \quad (20.43)$$

Оценка максимального правдоподобия для $\boldsymbol{\mu}$ равна $\bar{\mathbf{x}}$. Подстановка дает

$$\log p(\mathbf{X}|\boldsymbol{\mu}, \mathbf{W}, \sigma^2) = -\frac{N}{2} [D \log(2\pi) + \log |\mathbf{C}| + \text{tr}(\mathbf{C}^{-1} \mathbf{S})], \quad (20.44)$$

где $\mathbf{S} = (1/N) \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T$ – эмпирическая ковариационная матрица.

В работах [ТВ99; Row97] показано, что максимум этой целевой функции должен удовлетворять соотношению

$$\mathbf{W} = \mathbf{U}_L (\mathbf{L}_L - \sigma^2 \mathbf{I})^{1/2} \mathbf{R}, \quad (20.45)$$

где \mathbf{U}_L – матрица размера $D \times L$, столбцами которой являются L собственных векторов \mathbf{S} с наибольшими собственными значениями, \mathbf{L}_L – диагональная матрица $L \times L$, содержащая собственные значения, \mathbf{R} – произвольная ортого-

нальная матрица $L \times L$, в качестве которой можно без ограничения общности взять $\mathbf{R} = \mathbf{I}$. В пределе при отсутствии шума, когда $\sigma^2 = 0$, мы видим, что $\mathbf{W}_{\text{mle}} = \mathbf{U}_L \mathbf{L}_L^{1/2}$, т. е. пропорциональна решению PCA.

MLE для дисперсии наблюдений равна

$$\sigma^2 = \frac{1}{D - L} \sum_{i=L+1}^D \lambda_i; \quad (20.46)$$

это среднее искажение, ассоциированное с отброшенными измерениями. Если $L = D$, то оценка шума равна 0, потому что модель схлопывается к $\mathbf{z} = \mathbf{x}$.

Для вычисления правдоподобия $p(\mathbf{X}|\boldsymbol{\mu}, \mathbf{W}, \sigma^2)$ нам необходимо вычислить \mathbf{C}^{-1} и $\log |\mathbf{C}|$, где \mathbf{C} – матрица $D \times D$. Чтобы сделать это эффективно, мы можем воспользоваться леммой об обращении матрицы и написать:

$$\mathbf{C}^{-1} = \sigma^{-2} [\mathbf{I} - \mathbf{W} \mathbf{M}^{-1} \mathbf{W}^T], \quad (20.47)$$

где матрица \mathbf{M} размера $L \times L$ равна

$$\mathbf{M} = \mathbf{W}^T \mathbf{W} + \sigma^2 \mathbf{I}. \quad (20.48)$$

Подставляя MLE для \mathbf{W} из формулы (20.45) (и считая, что $\mathbf{R} = \mathbf{I}$), находим

$$\mathbf{M} = \mathbf{U}_L (\mathbf{L}_L - \sigma^2 \mathbf{I}) \mathbf{U}_L^T + \sigma^2 \mathbf{I}, \quad (20.49)$$

откуда

$$\mathbf{C}^{-1} = \sigma^{-2} [\mathbf{I} - \mathbf{U}_L (\mathbf{L}_L - \sigma^2 \mathbf{I}) \boldsymbol{\Lambda}_L^{-1} \mathbf{U}_L^T]; \quad (20.50)$$

$$\log |\mathbf{C}| = (D - L) \log \sigma^2 + \sum_{j=1}^L \log \lambda_j. \quad (20.51)$$

Таким образом, мы можем вообще избежать обращения матриц (т. е. $\boldsymbol{\Lambda}_L^{-1} = \text{diag}(1/\lambda_j)$).

Чтобы использовать PPCA в качестве альтернативы PCA, мы должны вычислить апостериорное среднее $\mathbb{E}[\mathbf{z}|\mathbf{x}]$, что эквивалентно модели кодировщика. По формуле Байеса для гауссовых распределений имеем

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mathbf{M}^{-1} \mathbf{W}^T (\mathbf{x} - \boldsymbol{\mu}), \sigma^2 \mathbf{M}^{-1}), \quad (20.52)$$

где \mathbf{M} определена в формуле (20.48). В пределе при $\sigma^2 = 0$ апостериорное среднее с параметрами MLE принимает вид:

$$\mathbb{E}[\mathbf{z}|\mathbf{x}] = (\mathbf{W}^T \mathbf{W})^{-1} \mathbf{W}^T (\mathbf{x} - \bar{\mathbf{x}}), \quad (20.53)$$

что является ортогональной проекцией данных в латентное пространство, как в стандартном PCA.

20.2.3. EM-алгоритм для ФА/PPCA

В этом разделе мы опишем метод вычисления MLE для модели ФА с применением EM-алгоритма, основанный на работах [RT82; GH96].

20.2.3.1. EM-алгоритм для ФА

На Е-шаге вычисляются апостериорные погружения:

$$p(\mathbf{z}_i | \mathbf{x}_i, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_i | \mathbf{m}_i, \boldsymbol{\Sigma}_i); \quad (20.54)$$

$$\boldsymbol{\Sigma}_i \triangleq (\mathbf{I}_L + \mathbf{W}^T \boldsymbol{\Psi}^{-1} \mathbf{W})^{-1}; \quad (20.55)$$

$$\mathbf{m}_i \triangleq \boldsymbol{\Sigma}_i (\mathbf{W}^T \boldsymbol{\Psi}^{-1} (\mathbf{x}_i - \boldsymbol{\mu})). \quad (20.56)$$

На М-шаге проще всего оценить $\boldsymbol{\mu}$ и \mathbf{W} одновременно, для чего определим $\tilde{\mathbf{W}} = (\mathbf{W}, \boldsymbol{\mu})$, $\tilde{\mathbf{z}} = (\mathbf{z}, 1)$. Также определим

$$\mathbf{b}_i \triangleq \mathbb{E}[\tilde{\mathbf{z}} | \mathbf{x}_i] = [\mathbf{m}_i, 1]; \quad (20.57)$$

$$\mathbf{C}_i \triangleq \mathbb{E}[\tilde{\mathbf{z}} \tilde{\mathbf{z}}^T | \mathbf{x}_i] = \begin{pmatrix} \mathbb{E}[\mathbf{z} \mathbf{z}^T | \mathbf{x}_i] & \mathbb{E}[\mathbf{z} | \mathbf{x}_i] \\ \mathbb{E}[\mathbf{z} | \mathbf{x}_i]^T & 1 \end{pmatrix}. \quad (20.58)$$

Тогда М-шаг будет выглядеть следующим образом:

$$\hat{\mathbf{W}} = \left[\sum_i \mathbf{x}_i \mathbf{b}_i^T \right] \left[\sum_i \mathbf{C}_i \right]^{-1}; \quad (20.59)$$

$$\hat{\boldsymbol{\Psi}} = \frac{1}{N} \text{diag} \left\{ \sum_i (\mathbf{x}_i - \hat{\mathbf{W}} \mathbf{b}_i) \mathbf{x}_i^T \right\}. \quad (20.60)$$

Заметим, что это обновления «простого» EM-алгоритма. Гораздо более быстрая версия этого алгоритма, основанная на ECM, описана в работе [ZY08].

20.2.3.2. EM-алгоритм для (P)РСА

ЕМ-алгоритм можно также использовать для обучения модели РРСА, что составляет полезную альтернативу методам на основе собственных векторов. Идея опирается на вероятностную формулировку РСА. Однако алгоритм будет работать и в пределе, когда шум нулевой, $\sigma^2 = 0$, как показано в работе [Row97].

Конкретно обозначим $\tilde{\mathbf{Z}} = \mathbf{Z}^T$ матрицу размера $L \times N$, столбцами которой являются апостериорные средние (представления низкой размерности). Аналогично обозначим $\tilde{\mathbf{X}} = \mathbf{X}^T$ матрицу, столбцами которой являются оригинальные данные. Из формулы (20.52) при $\sigma^2 = 0$ имеем

$$\tilde{\mathbf{Z}} = (\mathbf{W}^T \mathbf{W})^{-1} \mathbf{W}^T \tilde{\mathbf{X}}. \quad (20.61)$$

В этом состоит Е-шаг. Заметим, что это просто ортогональная проекция данных. Из формулы (20.59) выводим М-шаг:

$$\hat{\mathbf{W}} = \left[\sum_i \mathbf{x}_i \mathbb{E}[\mathbf{z}_i]^T \right] \left[\sum_i \mathbb{E}[\mathbf{z}_i] \mathbb{E}[\mathbf{z}_i]^T \right]^{-1}, \quad (20.62)$$

где использован тот факт, что $\boldsymbol{\Sigma} = \text{Cov}[\mathbf{z}_i | \mathbf{x}_i, \boldsymbol{\theta}] = \mathbf{0I}$, когда $\sigma^2 = 0$.

Сравним это выражение с MLE для линейной регрессии с несколькими выходами (формула (11.2)), которая имеет вид $\mathbf{W} = (\sum_i \mathbf{y}_i \mathbf{x}_i^T) (\sum_i \mathbf{x}_i \mathbf{x}_i^T)^{-1}$. Таким образом, мы видим, что М-шаг похож на линейную регрессию, только наблюдаемые входы заменены математическими ожиданиями латентных переменных.

Вот как выглядит полный алгоритм:

$$\tilde{\mathbf{Z}} = (\mathbf{W}^T \mathbf{W})^{-1} \mathbf{W}^T \tilde{\mathbf{X}} \text{ (Е-шаг);} \quad (20.63)$$

$$\mathbf{W} = \tilde{\mathbf{X}} \tilde{\mathbf{Z}}^T (\tilde{\mathbf{Z}} \tilde{\mathbf{Z}}^T)^{-1} \text{ (М-шаг).} \quad (20.64)$$

В работе [TB99] показано, что единственной устойчивой неподвижной точкой ЕМ-алгоритма является глобально оптимальное решение. Это значит, что ЕМ-алгоритм сходится к решению, для которого на \mathbf{W} натянуто то самое линейное подпространство, которое определено первыми L собственными векторами. Но если мы хотим, чтобы \mathbf{W} была ортогональной и содержала собственные векторы в порядке убывания собственных значений, то должны ортогонализировать результирующую матрицу (для чего не требуется больших затрат). Альтернативно можно модифицировать ЕМ-алгоритм, так чтобы он порождал базис главных компонент непосредственно [AO03].

У этого алгоритма есть простая физическая аналогия в случае $D = 2$ и $L = 1$ [Row97]. Рассмотрим точки в \mathbb{R}^2 , прикрепленные пружинами к жесткому стержню, ориентация которого определяется вектором \mathbf{w} . Обозначим z_i место, в котором i -я пружина крепится к стержню. На Е-шаге мы оставляем стержень неподвижным, а точкам прикрепления позволяем перемещаться по нему, чтобы минимизировать энергию пружин (которая пропорциональна сумме квадратов невязок). На М-шаге точки неподвижны, а стержень может вращаться с той же целью – минимизировать энергию пружин (см. иллюстрацию на рис. 20.10).

20.2.3.3. Преимущества

ЕМ-алгоритм для PCA имеет следующие преимущества перед методами на основе собственных векторов:

- ЕМ-алгоритм может работать быстрее. В частности, в предположении, что $N, D \gg L$, наиболее затратной частью является операция проецирования на Е-шаге, так что общее время составляет $O(TLND)$, где T – количество итераций. В работе [Row97] экспериментально показано, что количество итераций обычно очень мало (в среднем 3.6) независимо от N и D . (Результат зависит от отношения собственных значений эмпирической ковариационной матрицы.) Это гораздо быстрее, чем время порядка $O(\min(ND^2, DN^2))$, необходимое при прямолинейном применении методов на основе собственных векторов, хотя в более изобретательных методах такого рода, например в алгоритме Ланцоша, время работы сопоставимо с ЕМ;

- ЕМ-алгоритм можно реализовать в онлайнном режиме, т. е. мы можем обновлять оценку \mathbf{W} по мере поступления данных потоком;
- ЕМ-алгоритм позволяет легко обрабатывать отсутствие данных (см., например, [IR10; DJ15]);
- ЕМ-алгоритм можно обобщить для обработки смесей моделей РРСА/ФА (см. раздел 20.2.6);
- У ЕМ-алгоритма имеются модификации – вариационный и вариационный байесовский ЕМ – для обучения более сложных моделей (см., например, раздел 20.2.7).

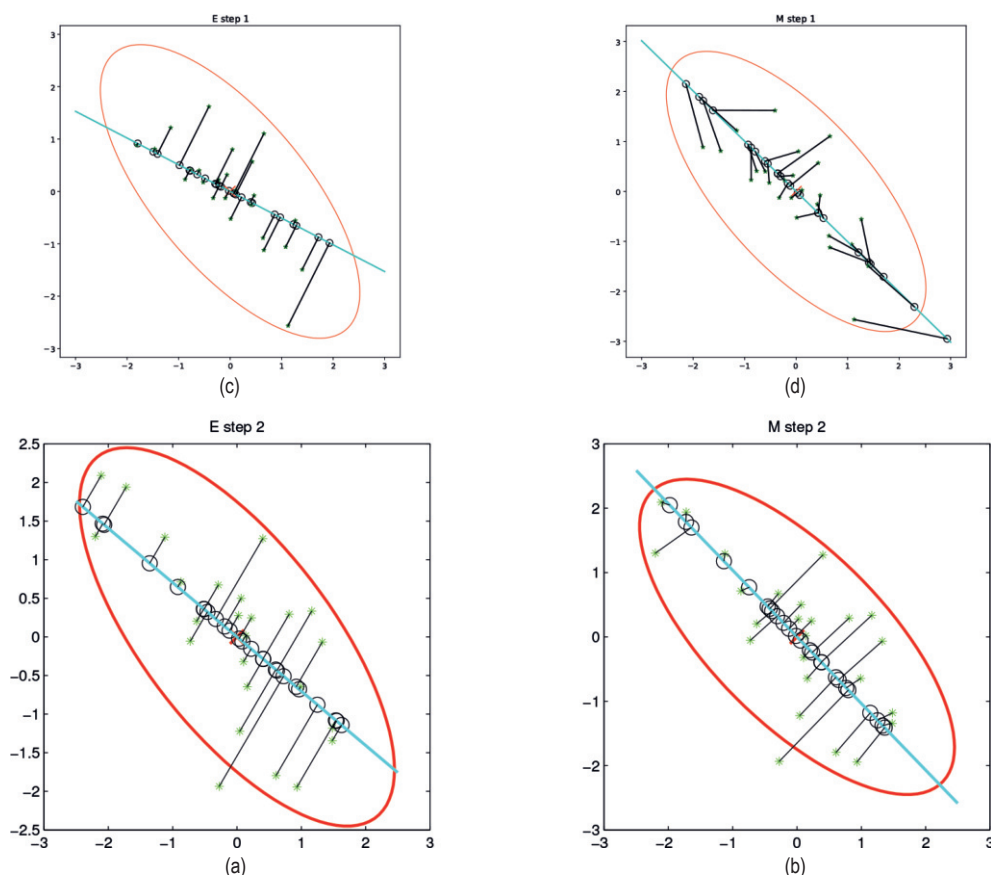


Рис. 20.10 ❖ ЕМ-алгоритм для РСА при $D = 2$ и $L = 1$. Зелеными звездочками обозначены оригинальные точки данных, черными кружочками – их реконструкции. Вектор весов \mathbf{w} представлен синей прямой. (а) Начинаем со случайного начального значения \mathbf{w} . Е-шаг представлен ортогональными проекциями. (б) На М-шаге обновляем положение стержня \mathbf{w} , оставляя проекции на стержень (черные кружочки) фиксированными. (с) Другой Е-шаг. Черные кружочки могут перемещаться по стержню, но сам стержень неподвижен. (д) Другой М-шаг. На основе рис. 12.12 из работы [Bis06]. Построено программой по адресу figures.probl.ai/book1/20.10

20.2.4. Неидентифицируемость параметров

Параметры модели ФА не идентифицируемы. Чтобы убедиться в этом, рассмотрим модель с весами \mathbf{W} и ковариацией наблюдений Ψ . Имеем

$$\text{Cov}[\mathbf{x}] = \mathbf{W}\mathbb{E}[\mathbf{z}\mathbf{z}^T]\mathbf{W}^T + \mathbb{E}[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^T] = \mathbf{W}\mathbf{W}^T + \Psi. \quad (20.65)$$

Теперь рассмотрим другую модель с весами $\tilde{\mathbf{W}} = \mathbf{W}\mathbf{R}$, где \mathbf{R} – произвольная ортогональная матрица поворота, удовлетворяющая условию $\mathbf{R}\mathbf{R}^T = \mathbf{I}$. Правдоподобие при этом не изменяется, потому что

$$\text{Cov}[\mathbf{x}] = \tilde{\mathbf{W}}\mathbb{E}[\mathbf{z}\mathbf{z}^T]\tilde{\mathbf{W}}^T + \mathbb{E}[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^T] = \mathbf{W}\mathbf{R}\mathbf{R}^T\mathbf{W}^T + \Psi = \mathbf{W}\mathbf{W}^T + \Psi. \quad (20.66)$$

Геометрически умножение \mathbf{W} на ортогональную матрицу эквивалентно повороту \mathbf{z} перед генерированием \mathbf{x} ; но, поскольку \mathbf{z} выбирается из изотропного гауссова распределения, на правдоподобии это никак не отражается. Следовательно, мы не можем однозначно определить \mathbf{W} , а значит, не можем и однозначно идентифицировать латентные факторы.

Ниже обсуждается несколько способов нарушить эту симметрию.

- **Сделать столбцы \mathbf{W} ортонормированными.** Пожалуй, это самое простое решение проблемы идентифицируемости. Такой подход принят в PCA. В этом случае получающаяся апостериорная оценка будет единственной с точностью до перестановки латентных измерений. (В PCA неоднозначность упорядочения разрешается сортировкой измерений в порядке убывания собственных значений \mathbf{W} .)
- **Сделать \mathbf{W} нижнетреугольной.** Еще один способ разрешить неоднозначность с точностью до перестановки, популярной в байесовском сообществе (см., например, [LW04c]), – гарантировать, что первый видимый признак генерируется только первым латентным фактором, второй видимый признак – только первыми двумя латентными факторами и т. д. Например, если $L = 3$ и $D = 4$, то соответствующая матрица факторной нагрузки имеет вид:

$$\mathbf{W} = \begin{pmatrix} w_{11} & 0 & 0 \\ w_{21} & w_{22} & 0 \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix}. \quad (20.67)$$

Мы также требуем, чтобы $w_{kk} > 0$ для $k = 1, \dots, L$. Общее число параметров в этой матрице с наложенными ограничениями равно $D + DL - L(L - 1)/2$, т. е. равно числу однозначно идентифицируемых параметров в ФА¹. Недостаток этого метода в том, что первые L ви-

¹ Мы получаем D параметров для Ψ и DL параметров для \mathbf{W} , но нужно исключить $L(L - 1)/2$ степеней свободы, имеющих источником \mathbf{R} , так как это размерность пространства ортогональных матриц размера $L \times L$. Действительно, в первом столбце \mathbf{R} имеется $L - 1$ свободных параметров (потому что вектор-столбец должен быть нормирован на единичную длину), во втором столбце – $L - 2$ свободных параметров (так как он должен быть ортогонален первому) и т. д.

димых переменных, называемые **основополагающими**, влияют на интерпретацию латентных факторов и потому должны тщательно выбираться.

- **Априорные распределения весов, поощряющие разреженность.** Вместо того чтобы заранее определять, какие элементы \mathbf{W} должны быть нулевыми, мы можем поощрить обнуление элементов с помощью ℓ_1 -регуляризации [ZHT06], автоматического определения релевантности [Bis99; AB08] или спайковых априорных распределений [Rat+09]. Это называется **разреженным факторным анализом**. Однозначная оценка MAP при этом не гарантируется, зато решения допускают интерпретацию.
- **Выбор информативной матрицы поворота.** Существует целый ряд эвристических методов, пытающихся найти матрицы поворота \mathbf{R} , которые можно использовать для модификации \mathbf{W} (а значит, и латентных факторов) с целью повысить интерпретируемость. Как правило, они стремятся сделать матрицу разреженной (хотя бы приближенно). В частности, популярен метод **varimax** [Kai58].
- Использование негауссовых априорных распределений для латентных факторов. Если заменить априорное распределение латентных переменных, $p(\mathbf{z})$, негауссовым, то иногда удастся однозначно идентифицировать как \mathbf{W} , так и латентные факторы. Детали см., например, в работе [KKH19].

20.2.5. Нелинейный факторный анализ

Модель ФА предполагает, что наблюдаемые данные можно смоделировать так, будто они являются результатом линейного отображения низкоразмерного множества гауссовых факторов. Это предположение можно ослабить, допустив, что отображение \mathbf{z} в \mathbf{x} описывается нелинейной моделью, например нейронной сетью. То есть модель принимает вид:

$$p(\mathbf{x}) = \int d\mathbf{z} \mathcal{N}(\mathbf{x} | f(\mathbf{w}, \boldsymbol{\theta}), \boldsymbol{\Psi}) \mathcal{N}(\mathbf{z} | \mathbf{0}, \mathbf{I}). \quad (20.68)$$

Это называется **нелинейным факторным анализом**. К сожалению, мы больше не можем вычислить апостериорное распределение или MLE точно, поэтому вынуждены использовать приближенные методы. В разделе 20.3.5 обсуждаются вариационные автокодировщики – самый распространенный способ аппроксимации нелинейной модели ФА.

20.2.6. Смеси факторных анализаторов

Модель факторного анализа (раздел 20.2) предполагает, что наблюдаемые данные можно смоделировать так, будто они являются результатом линейного отображения низкоразмерного множества гауссовых факторов. Один из способов ослабить это допущение – предположить, что модель линейна только локально, а полная модель является взвешенной комбинацией моде-

лей ФА; это называется **смесью факторных анализаторов**. Полная модель данных оказывается смесью линейных многообразий, которую можно использовать для аппроксимации всего искривленного многообразия.

Формально пусть латентный индикатор $m_n \in \{1, \dots, K\}$ определяет, какое подпространство (кластер) следует использовать для генерирования данных. Если $m_n = k$, то производится выборка z_n из гауссова априорного распределения, к ней применяется матрица \mathbf{W}_k и прибавляется шум, где \mathbf{W}_k отображает L -мерное подпространство в D -мерное видимое пространство¹. Точнее, модель имеет вид:

$$p(\mathbf{x}_n | \mathbf{z}_n, m_n = k, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k + \mathbf{W}_k \mathbf{z}_n, \boldsymbol{\Psi}_k); \quad (20.69)$$

$$p(\mathbf{z}_n | \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_n | \mathbf{0}, \mathbf{I}); \quad (20.70)$$

$$p(\mathbf{m}_n | \boldsymbol{\theta}) = \text{Cat}(m_n | \boldsymbol{\pi}). \quad (20.71)$$

Это называется **смесью факторных анализаторов** (mixture of factor analysers – MFA) [GH96]. Соответствующее распределение в видимом пространстве имеет вид:

$$\begin{aligned} p(\mathbf{x} | \boldsymbol{\theta}) &= \sum_k p(c = k) \int d\mathbf{z} p(\mathbf{z} | c) p(\mathbf{x} | \mathbf{z}, c) \\ &= \sum_k \pi_k \int d\mathbf{z} \mathcal{N}(\mathbf{z} | \boldsymbol{\mu}_k, \mathbf{I}) \mathcal{N}(\mathbf{x} | \mathbf{W}_k \mathbf{z}, \sigma^2 \mathbf{I}). \end{aligned} \quad (20.72)$$

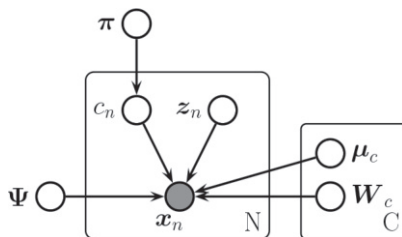


Рис. 20.11 ❖ Смесь факторных анализаторов в виде вероятностной графовой модели

В частном случае, когда $\boldsymbol{\Psi}_k = \sigma^2 \mathbf{I}$, получаем смесь моделей PPCA (хотя в этой ситуации трудно обеспечить ортогональность \mathbf{W}_k). На рис. 20.12 приведен пример применения этого метода к двумерным данным.

Эту идею можно рассматривать как низкоранговую версию смеси гауссовых распределений. В частности, эта модель требует $O(KLD)$ параметров, а не $O(KD^2)$ параметров, необходимых для смеси гауссовых распределений с полной ковариационной матрицей. Это может уменьшить риск переобучения.

¹ Если допустить зависимость \mathbf{z}_n от m_n , то подпространства смогут иметь разные размерности, как предложено в работе [KS15].

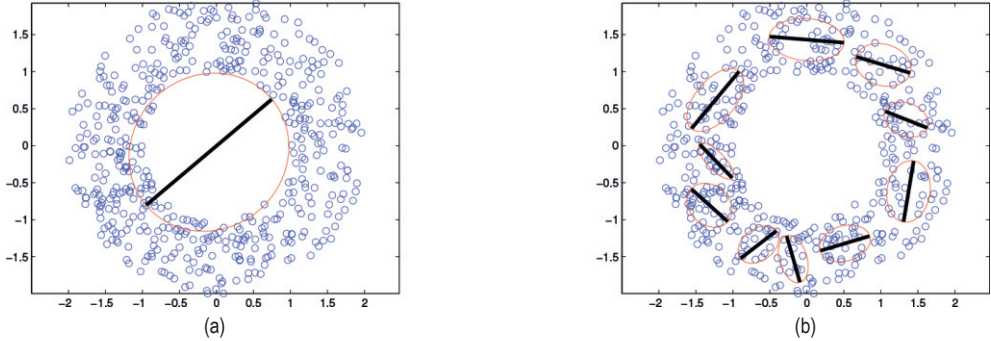


Рис. 20.12 ❖ Смесь моделей PPCA, обученных на двумерном наборе данных с использованием $L = 1$ латентных измерений. (a) $K = 1$ компонент смеси. (b) $K = 10$ компонент смеси. Построено программой по адресу figures.problm.ai/book1/20.12

20.2.7. Факторный анализ экспоненциального семейства

До сих пор мы предполагали, что наблюдаемые данные вещественные, т. е. $\mathbf{x}_n \in \mathbb{R}^D$. Если требуется моделировать другие виды данных (например, бинарные или категориальные), то можно просто заменить гауссово выходное распределение подходящим членом экспоненциального семейства, где естественные параметры определяются линейной функцией от \mathbf{z}_n . То есть используется распределение

$$p(\mathbf{x}_n | \mathbf{z}_n) = \exp(\mathcal{T}(\mathbf{x})^\top \boldsymbol{\theta} + h(\mathbf{x}) - g(\boldsymbol{\theta})), \quad (20.73)$$

где предполагается, что матрица естественных параметров размера $N \times D$ определяется низкоранговым разложением $\boldsymbol{\Theta} = \mathbf{Z}\mathbf{W}$, где \mathbf{Z} имеет размер $N \times L$, а \mathbf{W} – размер $L \times D$. Результирующая модель называется **факторным анализом экспоненциального семейства**.

В отличие от линейно-гауссова ФА, мы не можем вычислить точное апостериорное распределение $p(\mathbf{z}_n | \mathbf{x}_n, \mathbf{W})$ из-за отсутствия сопряженности между правдоподобием экспоненциального семейства и априорным гауссовым распределением. Более того, мы также не можем вычислить точное маргинальное правдоподобие, а значит, не можем найти оптимальную MLE.

В работе [CDS02] предложен метод покоординатного подъема для детерминированного варианта этой модели, известный под названием **РСА экспоненциального семейства**. В нем поочередно вычисляется точечная оценка \mathbf{z}_n и \mathbf{W} . Это можно рассматривать как вырожденный вариант вариационного ЕМ-алгоритма, где на Е-шаге используется дельта-функция в качестве апостериорного распределения \mathbf{z}_n . В работе [GS08] представлен улучшенный алгоритм, который находит глобальный оптимум, а в работе [Ude+16] – его обобщение, названное **обобщенными низкоранговыми моделями**, которое покрывает многие типы функций потерь.

Однако часто бывает предпочтительно использовать вероятностную версию модели, а не вычислять точечные оценки латентных факторов. В этом случае мы должны представить апостериорное распределение, используя невырожденное распределение во избежание переобучения, поскольку число латентных переменных пропорционально числу примеров [WCS08]. По счастью, можно использовать невырожденное апостериорное распределение, например гауссово, оптимизировав вариационную нижнюю границу. Примеры будут приведены ниже.

20.2.7.1. Пример: бинарный PCA

Рассмотрим факторизованное правдоподобие Бернулли:

$$p(\mathbf{x}|\mathbf{z}) = \prod_d \text{Ber}(x_d | \sigma(\mathbf{w}_d^T \mathbf{z})). \quad (20.74)$$

Предположим, что наблюдается $N = 150$ битовых векторов длины $D = 6$. Каждый пример сгенерирован путем выбора одного из трех бинарных векторов-прототипов с последующим случайным инвертированием битов. Данные показаны на рис. 20.13а. Эти данные можно аппроксимировать с помощью вариационного ЕМ-алгоритма (детали см. в работе [Tip98]). Мы будем использовать $L = 2$ латентных измерений, что позволяет визуализировать латентное пространство. На рис. 20.13b показан график $E[\mathbf{z}_n | \mathbf{x}_n, \hat{\mathbf{W}}]$. Мы видим, что проекции точек группируются в три кластера, как и следовало ожидать. На рис. 20.13с показана реконструированная версия данных, вычисляемая следующим образом:

$$p(\hat{x}_{nd} = 1 | \mathbf{x}_n) = \int d\mathbf{z}_n p(\mathbf{z}_n | \mathbf{x}_n) p(\hat{x}_{nd} | \mathbf{z}_n). \quad (20.75)$$

Если задать для этих вероятностей порог 0.5 (что соответствует оценке MAP), то получим «очищенную от шума» версию данных, изображенную на рис. 20.13d.

20.2.7.2. Пример: категориальный PCA

Мы можем обобщить модель из раздела 20.2.7.1 на обработку категориальных данных, воспользовавшись следующим правдоподобием:

$$p(\mathbf{x}|\mathbf{z}) = \prod_d \text{Cat}(x_d | \mathcal{S}(\mathbf{W}_d \mathbf{z})). \quad (20.76)$$

Это называется **категориальным PCA (CatPCA)**. Вариационный ЕМ-алгоритм для обучения этой модели описан в работе [Kha+10].

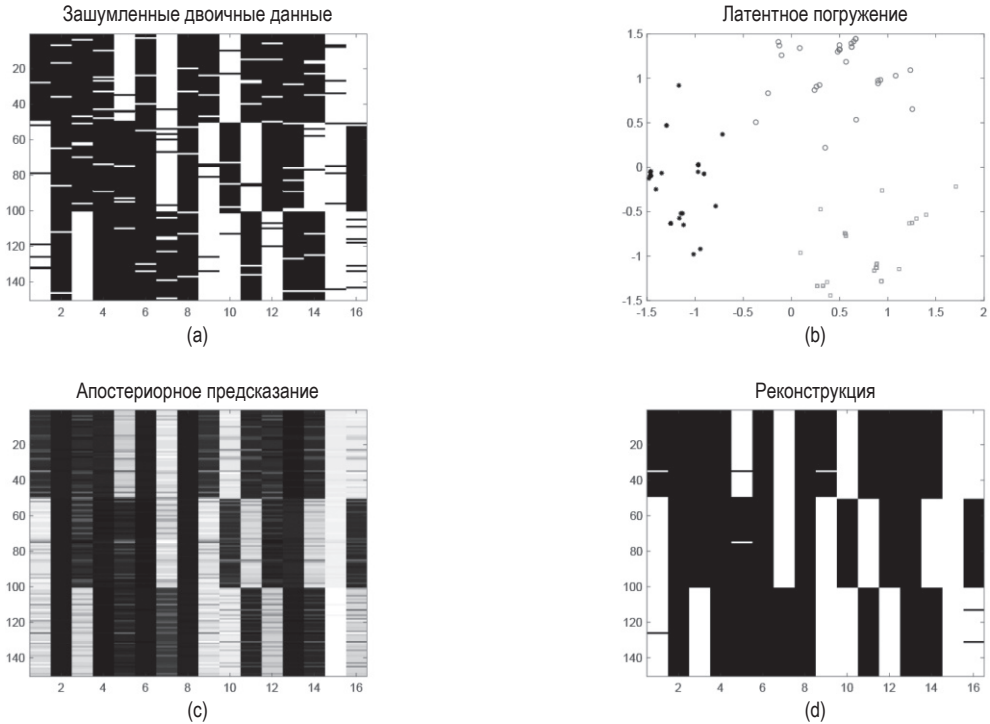


Рис. 20.13 ❖ (а) 150 синтетических 16-мерных битовых векторов. (б) Двумерное погружение, которому обучился бинарный PCA с помощью вариационного ЕМ-алгоритма. Цвет точек показывает, из какого «истинного» прототипа они были сгенерированы. (с) Предсказанная вероятность того, что бит поднят. (д) Предсказания после применения порога. Построено программой по адресу figures.probml.ai/book1/20.13

20.2.8. Модели факторного анализа для парных данных

В этом разделе мы обсудим модели линейно-гауссова факторного анализа, когда имеются два вида наблюдаемых переменных, $\mathbf{x} \in \mathbb{R}^{D_x}$ и $\mathbf{y} \in \mathbb{R}^{D_y}$, объединенных в пары. Часто они соответствуют различным датчикам или модальностям (например, изображение и звук). В своем изложении мы следуем работе [Vir10].

20.2.8.1. PCA с учителем

В методе **PCA с учителем** [Yu+06] мы моделируем совместное распределение $p(\mathbf{x}, \mathbf{y})$ разделяемым низкоразмерным представлением с использованием следующей линейно-гауссовой модели:

$$p(\mathbf{z}_n) = \mathcal{N}(\mathbf{z}_n | \mathbf{0}, \mathbf{I}_L); \quad (20.77)$$

$$p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_n | \mathbf{W}_x \mathbf{z}_n, \sigma_x^2 \mathbf{I}_{D_x}); \quad (20.78)$$

$$p(\mathbf{y}_n | \mathbf{z}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_n | \mathbf{W}_y \mathbf{z}_n, \sigma_y^2 \mathbf{I}_{D_y}). \quad (20.79)$$

Это показано в виде графовой модели на рис. 20.14а. Интуитивно понятно, что \mathbf{z}_n – разделяемое латентное подпространство, которое улавливает общие признаки \mathbf{x}_n и \mathbf{y}_n . Члены дисперсии σ_x и σ_y контролируют, какое значение модель придает двум разным сигналам. Если наложить априорное распределение на параметры $\boldsymbol{\theta} = (\mathbf{W}_x, \mathbf{W}_y, \sigma_x, \sigma_y)$, то мы придем к модели **байесовской факторной регрессии** из работы [Wes03].

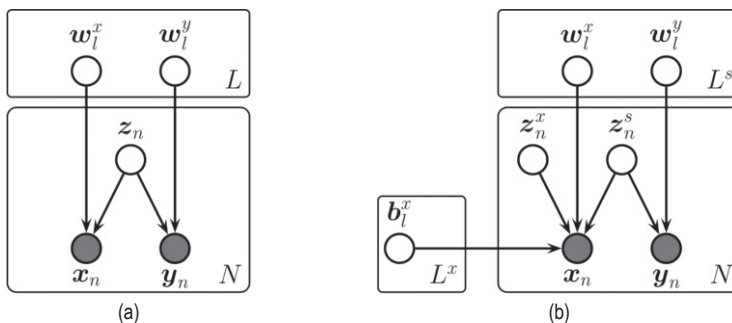


Рис. 20.14 ❖ Модели гауссовых латентных факторов для парных данных.
(а) PCA с учителем. (б) Метод частичных наименьших квадратов

Мы можем исключить \mathbf{z}_n и получить распределение $p(\mathbf{y}_n | \mathbf{x}_n)$. Если \mathbf{y}_n – скаляр, то оно принимает вид:

$$p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_n | \mathbf{x}_n^T \mathbf{v}, \mathbf{w}_y^T \mathbf{C} \mathbf{w}_y + \sigma_y^2); \quad (20.80)$$

$$\mathbf{C} = (\mathbf{I} + \sigma_x^{-2} \mathbf{W}_x^T \mathbf{W}_x)^{-1}; \quad (20.81)$$

$$\mathbf{v} = \sigma_x^{-2} \mathbf{W}_x \mathbf{C} \mathbf{w}_y. \quad (20.82)$$

Чтобы применить это к задаче классификации, мы можем использовать ePCA с учителем [Guo09], когда гауссово распределение $p(\mathbf{y} | \mathbf{z})$ заменяется моделью логистической регрессии.

Эта модель симметрична относительно \mathbf{x} и \mathbf{y} . Если наша цель – предсказать \mathbf{y} по \mathbf{x} с помощью латентного представления \mathbf{z} , то можно увеличить вес члена правдоподобия для \mathbf{y} , как предложено в работе [Ris+08]. Это дает

$$p(\mathbf{X}, \mathbf{Y}, \mathbf{Z} | \boldsymbol{\theta}) = p(\mathbf{Y} | \mathbf{Z}, \mathbf{W}_y) p(\mathbf{X} | \mathbf{Z}, \mathbf{W}_x)^\alpha p(\mathbf{Z}), \quad (20.83)$$

где $\alpha \leq 1$ контролирует относительную важность моделирования обоих источников. Значение α можно выбрать с помощью перекрестной проверки.

20.2.8.2. Метод частичных наименьших квадратов

Еще один способ улучшить качество предсказаний в задачах обучения с учителем – разрешить входам \mathbf{x} иметь собственный «частный» источник шума,

который не зависит от целевой переменной, так как не всякая вариативность \mathbf{x} релевантна целям предсказания. Это можно сделать, введя дополнительную латентную переменную \mathbf{z}_n^x только для входов, отличающуюся от переменной \mathbf{z}_n^s , опосредующей связь между \mathbf{x}_n и \mathbf{y}_n . В гауссовом случае полная модель принимает вид:

$$p(\mathbf{z}_n) = \mathcal{N}(\mathbf{z}_n^s | \mathbf{0}, \mathbf{I}) \mathcal{N}(\mathbf{z}_n^x | \mathbf{0}, \mathbf{I}); \quad (20.84)$$

$$p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_n | \mathbf{W}_x \mathbf{z}_n^s + \mathbf{B}_x \mathbf{z}_n^x, \sigma_x^2 \mathbf{I}); \quad (20.85)$$

$$p(\mathbf{y}_n | \mathbf{z}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_n | \mathbf{W}_y \mathbf{z}_n^s, \sigma_y^2 \mathbf{I}). \quad (20.86)$$

Смотрите рис. 20.14b. MLE для $\boldsymbol{\theta}$ в этой модели эквивалентна методу **частичных наименьших квадратов** (partial least squares – PLS) [Gus01; Nou+02; Sun+09].

20.2.8.3. Канонический корреляционный анализ

Иногда мы хотим использовать полностью симметричную модель, чтобы можно было уловить зависимость между \mathbf{x} и \mathbf{y} , разрешив в то же время зависящие от предметной области или «частные» источники шума. Это можно сделать, заведя латентную переменную \mathbf{z}_n^x только для \mathbf{x}_n , латентную переменную \mathbf{z}_n^y только для \mathbf{y}_n и разделяемую латентную \mathbf{z}_n^s . В гауссовом случае полная модель принимает вид:

$$p(\mathbf{z}_n) = \mathcal{N}(\mathbf{z}_n^s | \mathbf{0}, \mathbf{I}) \mathcal{N}(\mathbf{z}_n^x | \mathbf{0}, \mathbf{I}); \quad (20.87)$$

$$p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_n | \mathbf{W}_x \mathbf{z}_n^s + \mathbf{B}_x \mathbf{z}_n^x, \sigma_x^2 \mathbf{I}); \quad (20.88)$$

$$p(\mathbf{y}_n | \mathbf{z}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_n | \mathbf{W}_y \mathbf{z}_n^s + \mathbf{B}_y \mathbf{z}_n^y, \sigma_y^2 \mathbf{I}), \quad (20.89)$$

где матрицы \mathbf{W}_x и \mathbf{W}_y имеют размер $L^s \times D$, \mathbf{B}_x – размер $L^x \times D$, а \mathbf{B}_y – размер $L^y \times D$ (см. вероятностную графовую модель на рис. 20.15).

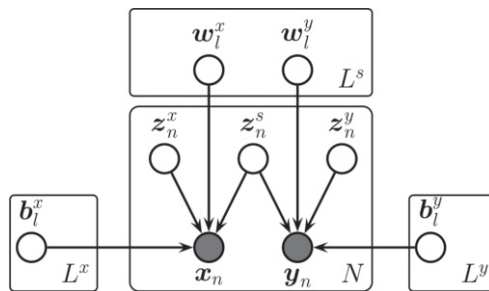


Рис. 20.15 ❖ Канонический корреляционный анализ в виде вероятностной графовой модели

Исключив все латентные переменные, получим следующее распределение видимых данных (предполагается, что $\sigma_x = \sigma_y = \sigma$):

$$p(\mathbf{x}_n, \mathbf{y}_n) = \int d\mathbf{z}_n p(\mathbf{z}_n) p(\mathbf{x}_n, \mathbf{y}_n | \mathbf{z}_n) = \mathcal{N}(\mathbf{x}_n, \mathbf{y}_n | \boldsymbol{\mu}, \mathbf{W}\mathbf{W}^\top + \sigma^2 \mathbf{I}), \quad (20.90)$$

где $\boldsymbol{\mu} = (\boldsymbol{\mu}_x; \boldsymbol{\mu}_y)$, а $\mathbf{W} = [\mathbf{W}_x; \mathbf{W}_y]$. Таким образом, индуцированная ковариация описывается следующей низкоранговой матрицей:

$$\mathbf{W}\mathbf{W}^\top = \begin{pmatrix} \mathbf{W}_x \mathbf{W}_x^\top & \mathbf{W}_x \mathbf{W}_y^\top \\ \mathbf{W}_y \mathbf{W}_x^\top & \mathbf{W}_y \mathbf{W}_y^\top \end{pmatrix}. \quad (20.91)$$

В работе [BJ05] показано, что MLE для этой модели эквивалентна классическому статистическому методу, который называется **каноническим корреляционным анализом** (canonical correlation analysis – **CCA**) [Hot36]. Однако подход на основе вероятностных графовых моделей позволяет легко обобщить его на несколько видов наблюдений (это называется **обобщенным CCA**) [Hog61]) или на нелинейные модели (**глубокий CCA** [WLL16; SNM16]), или на CCA экспоненциального семейства [KVK10]. Дальнейшее обсуждение CCA и его обобщений см. в работе [Uur+17].

20.3. Автокодировщики

Мы можем рассматривать PCA (раздел 20.1) и факторный анализ (раздел 20.2) как обучение (линейного) отображения $\mathbf{x} \rightarrow \mathbf{z}$, называемого **кодировщиком**, f_e , и другого (линейного) отображения $\mathbf{z} \rightarrow \mathbf{x}$, называемого **декодером**, f_d . Полная функция реконструкции имеет вид $r(\mathbf{x}) = f_d(f_e(\mathbf{x}))$. Модель обучается минимизировать потерю $\mathcal{L}(\boldsymbol{\theta}) = \|\mathbf{r}(\mathbf{x}) - \mathbf{x}\|_2^2$. В общем случае мы можем использовать $\mathcal{L}(\boldsymbol{\theta}) = -\log p(\mathbf{x} | r(\mathbf{x}))$.

В этом разделе мы рассмотрим случай, когда кодировщик и декодер – нелинейные отображения, реализованные нейронными сетями. Такая система называется **автокодировщиком** (AE). Если взять МСП с одним скрытым слоем, то получится модель, показанная на рис. 20.16. Скрытые блоки в середине можно рассматривать как низкоразмерное **сужение** между входом и его реконструкцией.

Конечно, если скрытый слой достаточно широк, то ничто не мешает этой модели обучиться тождественной функции. Чтобы предотвратить такое вырожденное решение, нужно наложить на модель какие-то ограничения. Самое простое – использовать узкий скрытый слой, когда $L \ll D$; это называется **сужающим** (undercomplete) представлением. Противоположный подход – взять $L \gg D$ (**расширяющее** (overcomplete) представление), но применить какую-то другую регуляризацию, например прибавить к входам шум, принудительно сделать активацию скрытых блоков разреженной или штрафовать производные скрытых блоков. Все эти варианты мы обсудим ниже.

20.3.1. Автокодировщики с сужением

Начнем с рассмотрения частного случая **линейного автокодировщика** с одним скрытым слоем, блоки которого вычисляются как $\mathbf{z} = \mathbf{W}_1 \mathbf{x}$, а выход

реконструируется по формуле $\hat{\mathbf{x}} = \mathbf{W}_2 \mathbf{z}$, где \mathbf{W}_1 – матрица $L \times D$, \mathbf{W}_2 – матрица $D \times L$ и $L < D$. Тогда $\hat{\mathbf{x}} = \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} = \mathbf{W} \mathbf{x}$ – выход модели. Если мы обучим эту модель минимизировать квадратичную ошибку реконструкции, $\mathcal{L}(\mathbf{W}) = \sum_{n=1}^N \|\mathbf{x}_n - \mathbf{W} \mathbf{x}_n\|_2^2$, то можно показать [BH89; KJ95], что \mathbf{W} – ортогональная проекция на первые L собственных векторов эмпирической ковариационной матрицы данных. Таким образом, этот случай эквивалентен PCA.

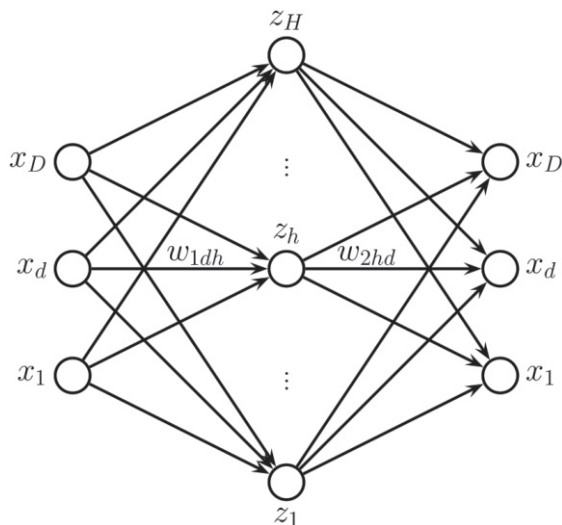


Рис. 20.16 ❖ Автокодировщик с одним скрытым слоем

Если включить в автокодировщик нелинейности, то получится модель, строго более мощная, чем PCA, это доказано в работе [JHG00]. Такие методы могут обучиться очень полезным низкоразмерным представлениям данных.

Рассмотрим обучение автокодировщика на наборе данных Fashion MNIST. Мы рассматриваем как архитектуру МСП (с тремя слоями и сужением размера 30), так и архитектуру СНС (с тремя слоями и трехмерным сужением с 64 каналами). Используется модель правдоподобия Бернулли и бинарная перекрестная энтропия в качестве потери. На рис. 20.17 показаны некоторые тестовые изображения и их реконструкции. Как видим, модель СНС реконструирует изображения точнее, чем МСП. Однако обе модели малы и обучались только на протяжении пяти эпох; результаты можно улучшить, если взять модели побольше и обучать их подольше.

На рис. 20.18 показаны первых два (из 30) латентных измерения, порожденные автокодировщиком с архитектурой МСП. Точнее, построена диаграмма tSNE-погружений (см. раздел 20.4.10), а цветом закодированы метки классов. Также показаны некоторые изображения из набора данных, на основе которых построены погружения. Мы видим, что метод неплохо справился с разделением классов без всякого привлечения учителя. Также мы видим, что латентные пространства моделей МСП и СНС очень похожи (по крайней мере, на этой двумерной проекции).



Рис. 20.17 ❖ Результаты применения автокодировщика к набору данных Fashion MNIST. В верхнем ряду показаны первые пять изображений из контрольного набора, в нижнем ряду – их реконструкции. (a) Модель МСП (обученная с 20 эпохами). Кодировщик является МСП с архитектурой 784-100-30. Декодер – его зеркальное отражение. (b) Модель ЧСЧ (обученная с пятью эпохами). Кодировщик – модель ЧСЧ с архитектурой Conv2D(16, 3x3, same, selu), MaxPool2D(2x2), Conv2D(32, 3x3, same, selu), MaxPool2D(2x2), Conv2D(64, 3x3, same, selu), MaxPool2D(2x2). Декодер – зеркальное отражение с транспонированной сверткой и без слоев max-пулинга. На основе рис. 17.4 из работы [Gér19]. Построено программой по адресу figures.probml.ai/book1/20.17

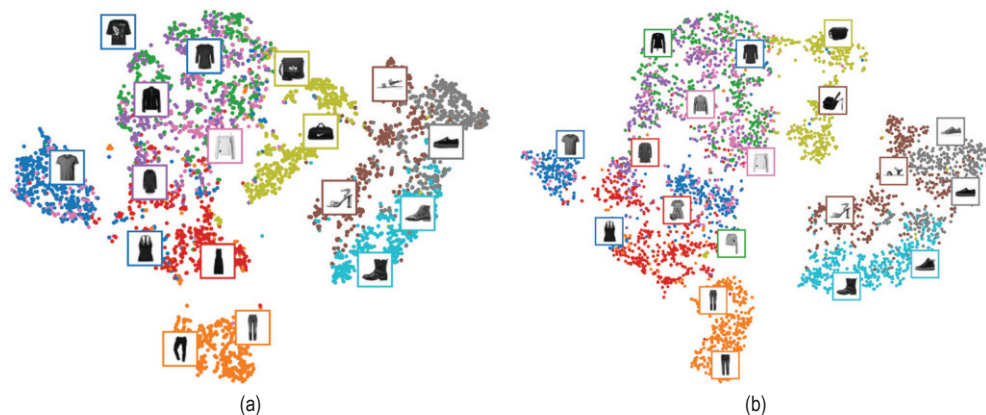


Рис. 20.18 ❖ Диаграмма tSNE первых двух латентных измерений контрольного набора Fashion MNIST, полученная с использованием автокодировщика. (a) МСП. (b) ЧСЧ. На основе рис. 17.5 из работы [Gér19]. Построено программой по адресу figures.probml.ai/book1/20.18

20.3.2. Шумоподавляющие автокодировщики

Полезный способ управления емкостью автокодировщика – прибавить шум к его входу, а затем обучить модель реконструировать чистую (неискаженную) версию оригинальных данных. Это называется **шумоподавляющим автокодировщиком** (DAE) [Vin+10a].

Мы можем реализовать эту идею, прибавив гауссов шум или воспользовавшись бернуллиевым прореживанием. На рис. 20.19 показаны реконструкции нескольких зашумленных изображений, вычисленных с помощью DAE. Как видим, модель умеет «придумывать» детали, отсутствующие во входных дан-

ных, потому что видела похожие изображения в прошлом и умеет сохранять эту информацию в своих параметрах.



Рис. 20.19 ❖ Шумоподавляющий автокодировщик (с архитектурой МСП), примененный к зашумленным изображениям из контрольного набора Fashion MNIST. (a) Гауссов шум. (b) Бернуллиево прореживание. Верхний ряд: вход. Нижний ряд: выход. На основе рис. 17.9 из работы [Gér19]. Построено программой по адресу figures.problml.ai/book1/20.19

Предположим, что DAE обучен с применением гауссова искажения и квадратичной ошибки реконструкции, т. е. $p_c(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}|\mathbf{x}, \sigma^2\mathbf{I})$ и $\ell(\mathbf{x}, r(\tilde{\mathbf{x}})) = \|\mathbf{e}\|_2^2$, где $\mathbf{e}(\mathbf{x}) = r(\tilde{\mathbf{x}}) - \mathbf{x}$ — невязка для примера \mathbf{x} . Тогда можно доказать [AB14] удивительный результат: при $\sigma \rightarrow 0$ (и при условии, что модель достаточно мощная, а данных хватает) невязки аппроксимируют **функцию вклада**, равную логарифму вероятности данных, т. е. $\mathbf{e}(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$. То есть DAE обучается **векторному полю**, соответствующему градиенту логарифмической плотности данных. Таким образом, точки, близкие к многообразию данных, будут проецироваться на него с помощью процесса выборки (см. иллюстрацию на рис. 20.20).

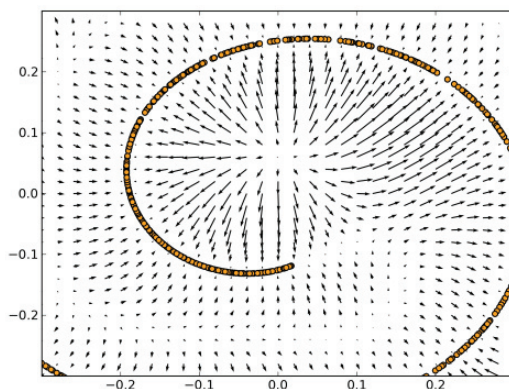


Рис. 20.20 ❖ Невязку шумоподавляющего автокодировщика, $\mathbf{e}(\mathbf{x}) = r(\tilde{\mathbf{x}}) - \mathbf{x}$, можно обучить векторному полю, соответствующему функции вклада. Стрелки направлены в сторону областей с более высокой вероятностью. Длина стрелки пропорциональна $\|\mathbf{e}(\mathbf{x})\|$, поэтому в точках вблизи одномерного многообразия данных (представленные кривой) стрелки короче. На основе рис. 5 из работы [AB14]. Печатается с разрешения Гийома Алена

20.3.3. Сжимающие автокодировщики

Другой способ регуляризации автокодировщиков заключается в прибавлении штрафующего члена

$$\Omega(\mathbf{z}, \mathbf{x}) = \lambda \left\| \frac{\partial f_e(\mathbf{x})}{\partial \mathbf{x}} \right\|_F^2 = \lambda \sum_k \|\nabla_{\mathbf{x}} h_k(\mathbf{x})\|_2^2 \quad (20.92)$$

к потере реконструкции, где h_k – значение k -го скрытого блока погружения. То есть мы штрафует норму Фробениуса якобиана кодировщика. Такой автокодировщик называется **сжимающим** (SAE) [Rif+11]. (Линейный оператор с якобианом \mathbf{J} называется **сжатием**, если $\|\mathbf{J}\mathbf{x}\| \leq 1$ для всех входов \mathbf{x} с единичной нормой.)

Чтобы понять, почему это полезно, взглянем на рис. 20.20. Мы можем аппроксимировать криволинейное многообразие низкой размерности последовательностью локально линейных многообразий. Эти линейные аппроксимации можно вычислить, зная якобиан кодировщика в каждой точке. Поощряя наличие у них свойства сжатия, мы гарантируем, что модель будет «подталкивать» входы, оказывающиеся вне многообразия, к возврату на него.

Есть и другой способ интерпретации SAE. Чтобы минимизировать штраф, модель должна стремиться сделать кодировщик постоянной функцией. Но если бы он был строго постоянным, то игнорировал бы вход, поэтому стоимость реконструкции была бы высока. Поэтому оба члена вместе поощряют модель обучиться представлению, в котором лишь немногие блоки изменяются в ответ на даже самые значительные изменения входов.

Одно из вырожденных решений заключается в том, что кодировщик просто обучается умножать вход на небольшую постоянную ϵ (которая уменьшает якобиан), а затем декодер делит на то же ϵ (так что реконструкция идеальна). Чтобы избежать этого, мы можем связать веса кодировщика и декодера, сделав так, чтобы матрица весов ℓ -го слоя f_d была результатом транспонирования матрицы весов ℓ -го слоя f_e , но оставить несвязанными члены смещения. К сожалению, SAE обучаются медленно из-за затрат на вычисление якобиана.

20.3.4. Разреженные автокодировщики

Еще один способ регуляризовать автокодировщики состоит в том, чтобы ввести поощряющий разреженность штраф латентных активаций вида $\Omega(\mathbf{z}) = \lambda \|\mathbf{z}\|_1$. (Это называется **регуляризацией активности**.)

Альтернативный способ реализовать разреженность, зачастую приносящий лучшие результаты, – использовать логистические блоки, а затем для каждого блока k вычислить ожидаемую долю времени, в течение которого он включен при обработке мини-пакета (обозначим ее q_k), и сделать так, чтобы она была близка к целевому значению p (эта идея предложена в работе [GBB11]). Именно, используем регуляризатор $\Omega(\mathbf{z}_{1:L}, \mathbf{1:N}) = \lambda \sum_k \text{KL}(\mathbf{p}|q_k)$ для

латентных измерений $1 \dots L$ и примеров $1 \dots N$, где $\mathbf{p} = (p, 1 - p)$ – желаемое целевое распределение, а $\mathbf{q}_k = (q_k, 1 - q_k)$ – эмпирическое распределение для блока k , вычисленное по формуле $q_k = (1/N) \sum_{n=1}^N \mathbb{I}(z_{n,k} = 1)$.

На рис. 20.21 показан результат при обучении автокодировщика с архитектурой МСП (с 300 скрытыми блоками) на наборе Fashion MNIST. Если положить $\lambda = 0$ (т. е. не включать штраф, поощряющий разреженность), то среднее значение активации будет около 0.4 и большую часть времени большинство нейронов частично активировано. Со штрафом по норме ℓ_1 большинство блоков все время выключено, т. е. они вообще не используются. Со штрафом на основе расхождения КЛ в среднем 70 % нейронов выключено, но, в отличие от случая ℓ_1 , мы не видим постоянно выключенных блоков (средний уровень активации равен 0.1). Последний паттерн разреженности похож на то, что мы наблюдаем в биологическом мозге (см., например, [Bey+19]).

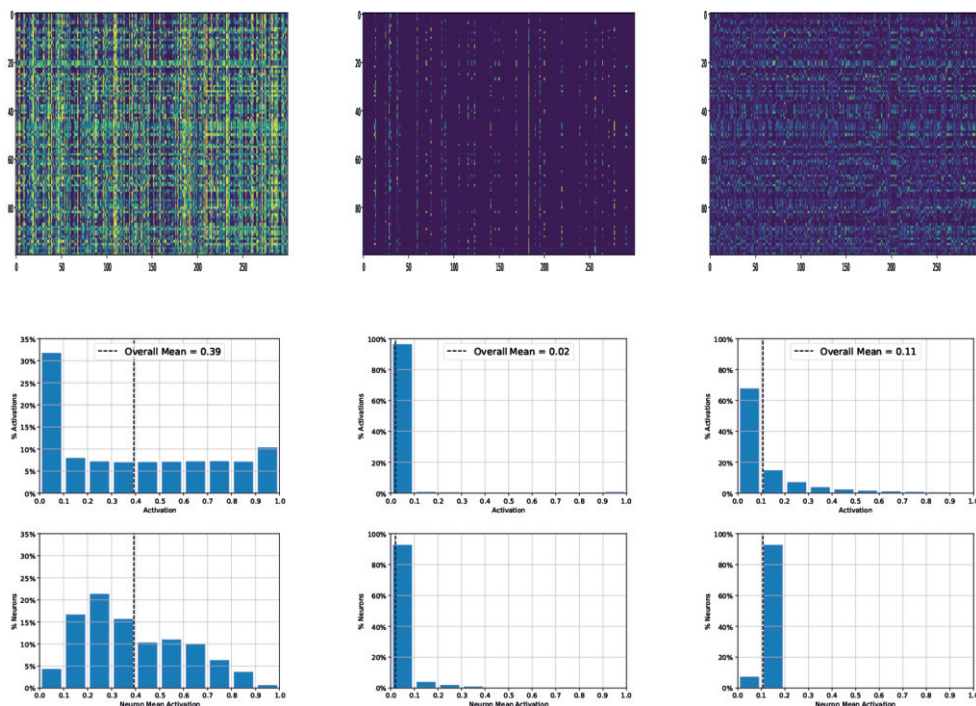


Рис. 20.21 ❖ Активность нейрона (в слое сужения) для автокодировщика применительно к набору данных Fashion MNIST. Показаны результаты для трех моделей с разными видами штрафа, поощряющего разреженность: штраф отсутствует (левая колонка), штраф по норме ℓ_1 (средняя колонка), штраф на основе расхождения КЛ (правая колонка). Верхний ряд: тепловая карта 300 активаций нейронов (столбцы) на 100 примерах (строки). Средний ряд: гистограмма уровней активации, построенная по этой тепловой карте. Нижний ряд: гистограмма средней активации нейронов, усредненной по всем примерам в контрольном наборе. На основе рис. 17.11 из работы [Gér19]. Построено программой по адресу figures.problml.ai/book1/20.21

20.3.5. Вариационные автокодировщики

В этом разделе мы обсудим **вариационный автокодировщик** (variational autoencoder – **VAE**) [KW14; RMW14; KW19a], который можно рассматривать как вероятностную версию детерминированного автокодировщика (раздел 20.3). Принципиальное преимущество VAE состоит в том, что это порождающая модель, способная создавать новые примеры, тогда как автокодировщик просто вычисляет погружения входных векторов.

Во всех подробностях VAE будут обсуждаться во втором томе этой книги, [Mur22]. Но если говорить вкратце, то VAE объединяет две идеи. Во-первых, мы создаем нелинейное обобщение порождающей модели факторного анализа, т. е. заменяем $p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z}, \sigma^2\mathbf{I})$ распределением

$$p_{\theta}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|f_d(\mathbf{z}; \theta), \sigma^2\mathbf{I}), \quad (20.93)$$

где f_d – декодер. Для бинарных наблюдений следует использовать правдоподобие Бернулли:

$$p(\mathbf{x}|\mathbf{z}, \theta) = \prod_{i=1}^D \text{Ber}(x_i|f_d(\mathbf{z}; \theta), \sigma^2\mathbf{I}). \quad (20.94)$$

Во-вторых, мы создаем еще одну модель, $q(\mathbf{z}|\mathbf{x})$, называемую **распознающей сетью**, или **сетью вывода**, которая одновременно с порождающей моделью обучается выполнять приближенный апостериорный вывод. Если мы предполагаем, что апостериорное распределение гауссово с диагональной ковариационной матрицей, то получаем

$$q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|f_{e,\mu}(\mathbf{x}; \phi), \text{diag}(f_{e,\sigma}(\mathbf{x}; \phi))), \quad (20.95)$$

где f_e – кодировщик (см. схему на рис. 20.22).

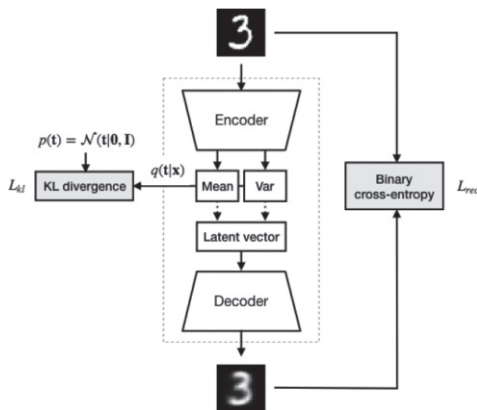


Рис. 20.22 ❖ Схема работы VAE.

Из статьи <http://krasserm.github.io/2018/07/27/dfc-vae/>.

Печатается с разрешения Мартина Крассера

Идея обучать сеть вывода «инвертировать» порождающую сеть, а не выполнять алгоритм оптимизации для вывода латентного кода называется **амортизированным выводом**. Эта идея впервые была предложена для **машины Гельмгольца** в работе [Day+95]. Однако в той статье не была представлена единственная унифицированная целевая функция для вывода и порождения, а вместо этого для обучения использовался метод пробуждения-засыпания, в котором поочередно выполняется оптимизация то порождающей модели, то модели вывода. Напротив, VAE оптимизирует вариационную нижнюю границу логарифмического правдоподобия, и такой подход теоретически правильнее, потому что опирается на одну унифицированную целевую функцию.

20.3.5.1. Обучение VAE

Мы не можем точно вычислить маргинальное правдоподобие $p(\mathbf{x}|\boldsymbol{\theta})$, необходимое для обучения MLE, потому что апостериорный вывод в нелинейной модели ФА – вычислительно неразрешимая задача. Однако мы можем использовать сеть вывода для вычисления приближенного апостериорного распределения, $q(\mathbf{z}|\mathbf{x})$. Затем его можно использовать для вычисления **вариационной нижней границы** (evidence lower bound – ELBO). Для одного примера \mathbf{x} она равна

$$\mathcal{E}(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})] \quad (20.96)$$

$$= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})}[\log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})] - \mathbb{KL}(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})||p(\mathbf{z})). \quad (20.97)$$

Это можно интерпретировать как ожидаемое логарифмическое правдоподобие плюс регуляризатор, который штрафует апостериорное распределение за слишком большое отклонение от априорного. (Это отличается от подхода, описанного в разделе 20.3.4, где мы применяли штраф на основе расхождения КЛ к агрегированному апостериорному распределению в каждом мини-пакете.)

ELBO – это нижняя граница логарифмического маргинального правдоподобия (иначе говоря, свидетельства – отсюда и слово «evidence» в англоязычном термине), как легко видеть из неравенства Йенсена:

$$\mathcal{E}(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{x}) = \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} d\mathbf{z} \quad (20.98)$$

$$\leq \log \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} d\mathbf{z} = \log p_{\boldsymbol{\theta}}(\mathbf{x}). \quad (20.99)$$

Таким образом, при фиксированных параметрах сети вывода $\boldsymbol{\phi}$ увеличение ELBO должно приводить к увеличению логарифмического правдоподобия данных, как в случае EM-алгоритма (раздел 8.7.2).

20.3.5.2. Перепараметризация

В этом разделе мы обсудим, как вычислить ELBO и ее градиент. Для простоты будем предполагать, что сеть вывода оценивает параметры гауссова апостериорного распределения. Так как $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ гауссово, можно написать

$$\mathbf{z} = f_{e,\mu}(\mathbf{x}; \boldsymbol{\phi}) + f_{e,\sigma}(\mathbf{x}; \boldsymbol{\phi}) \odot \boldsymbol{\varepsilon}, \quad (20.100)$$

где $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Отсюда

$$\mathcal{E}(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{x}) = \mathbb{E}_{\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} [\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z} = \mu_{\boldsymbol{\phi}}(\mathbf{x}) + \sigma_{\boldsymbol{\phi}}(\mathbf{x}) \odot \boldsymbol{\varepsilon})] - \mathbb{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p(\mathbf{z})). \quad (20.101)$$

Теперь математическое ожидание не зависит от параметров модели, поэтому мы можем безопасно переместить градиенты внутрь и обычным образом использовать для обучения обратное распространение, минимизируя $-\mathbb{E}_{\mathbf{x} \sim \mathcal{D}} [\mathcal{E}(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{x})]$ относительно $\boldsymbol{\theta}$ и $\boldsymbol{\phi}$. Это называется **трюком перепараметризации** (см. иллюстрацию на рис. 20.23).

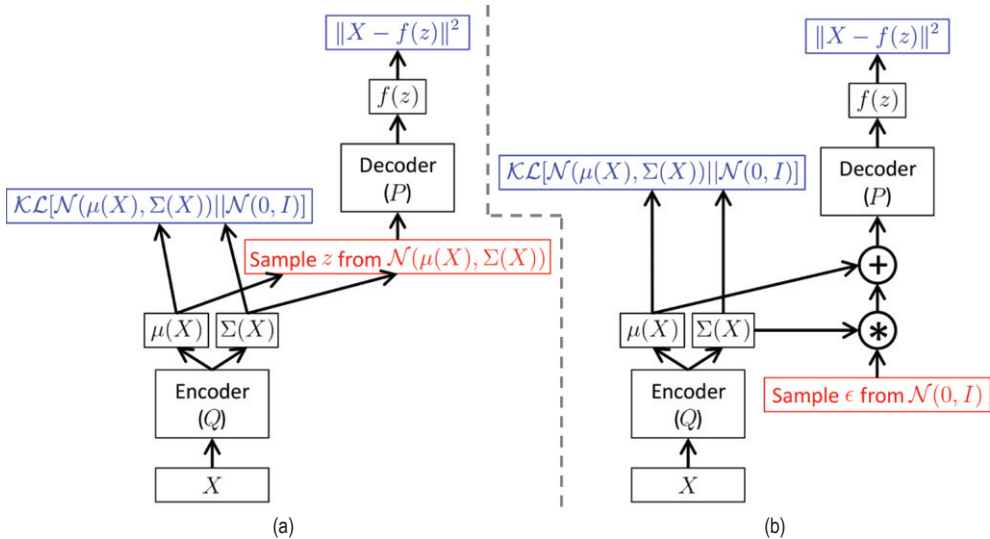


Рис. 20.23 ❖ Граф вычислений для VAE, где $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$, $p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}|f(\mathbf{z}), \sigma^2\mathbf{I})$ и $q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) = \mathcal{N}(\mathbf{z}|\mu(\mathbf{x}), \Sigma(\mathbf{x}))$. В красных прямоугольниках показаны недифференцируемые операции выборки, а в синих – слои потери (предполагается, что правдоподобия и априорные распределения гауссовы). (a) Без перепараметризации. (b) С перепараметризацией. Градиенты могут течь от выходной потери назад в декодер и затем в кодировщик. На основе рис. 4 из работы [Doe16].
Печатается с разрешения Карла Дёрша

Первый член в ELBO можно аппроксимировать, произведя выборку $\boldsymbol{\varepsilon}$, масштабируя ее на выход сети вывода, чтобы получить \mathbf{z} , а затем вычислив $\log p(\mathbf{x}|\mathbf{z})$ с помощью сети декодера.

Второй член в ELBO – расхождение КЛ двух гауссовых распределений, его можно найти в замкнутой форме. Именно, подставив $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ и $q(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}))$ в формулу (6.33), получим

$$\mathbb{KL}(q||p) = \sum_{k=1}^K \left[\log \left(\frac{1}{\sigma_k} \right) + \frac{\sigma_k^2 + (\mu_k - 0)^2}{2 \cdot 1} - \frac{1}{2} \right]$$

$$= -\frac{1}{2} \sum_{k=1}^K [\log \sigma_k^2 - \sigma_k^2 - \mu_k^2 + 1]. \quad (20.102)$$

20.3.5.3. Сравнение VAE с автокодировщиками

VAE очень похожи на автокодировщики. Действительно, порождающая модель, $p_\theta(\mathbf{x}|\mathbf{z})$, действует как декодер, а сеть вывода, $q_\phi(\mathbf{z}|\mathbf{x})$, – как кодировщик. Способность обеих моделей к реконструкции схожа, в чем можно убедиться, сравнив рис. 20.24а и б.

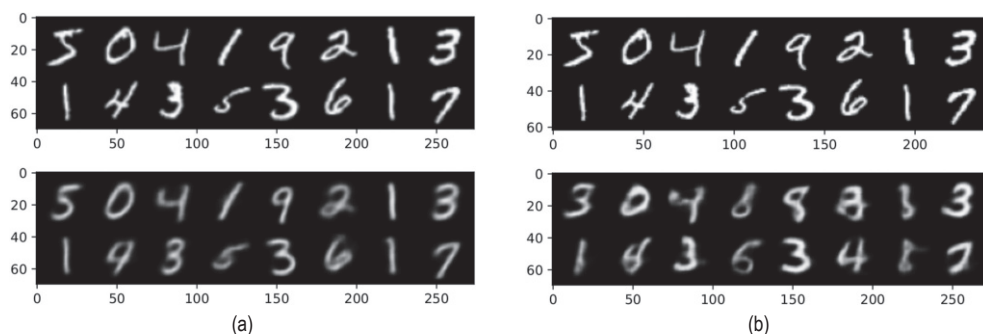


Рис. 20.24 ❖ Реконструкция цифр из набора MNIST с помощью 20-мерного латентного пространства. Верхний ряд: входные изображения. Нижний ряд: реконструкции. (а) VAE. (б) Детерминированный АЕ. Построено программой по адресу figures.probl.ai/book1/20.24

Главное преимущество VAE в том, что его можно использовать для генерирования новых данных из случайного шума. Именно, мы выбираем \mathbf{z} из гауссова априорного распределения $\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$, а затем пропускаем его через декодер с целью получить $\mathbb{E}[\mathbf{x}|\mathbf{z}] = f_d(\mathbf{z}; \theta)$. Декодер VAE обучается преобразовывать точки в пространстве погружения (порожденные возмущением закодированных входов) в разумные выходы. Для сравнения отметим, что декодер детерминированного автокодировщика получает в качестве входов только точные закодированные представления обучающих примеров, поэтому не знает, что делать со случайными входами вне того множества, на котором он обучался. Поэтому стандартный автокодировщик не способен создавать новые примеры. Эта разница становится особенно наглядной, если сравнить рис. 20.25а и б.

Причина, по которой VAE лучше ведет себя на выборке, заключается в том, что погружения изображений представляют собой гауссовы распределения в латентном пространстве, тогда как АЕ погружает отображения в точки, ведущие себя как дельта-функции. Преимущество латентного *распределения* в том, что оно поощряет локальную гладкость, так как заданное изображение может отображаться в несколько расположенных рядом мест в зависимости от стохастической процедуры выборки. Напротив, в случае АЕ латентное пространство обычно не гладкое, поэтому изображения из разных классов часто оказываются по соседству друг с другом. Это различие можно увидеть, сравнив рис. 20.26а и б.

812 ❖ Понижение размерности

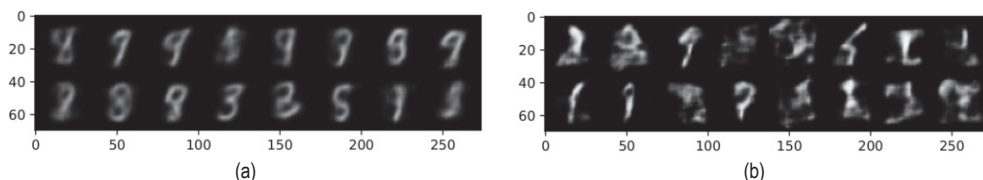


Рис. 20.25 ❖ Выборка цифр из набора MNIST с помощью 20-мерного латентного пространства. (a) VAE. (b) Детерминированный АЕ. Построено программой по адресу figures.probl.ai/book1/20.25

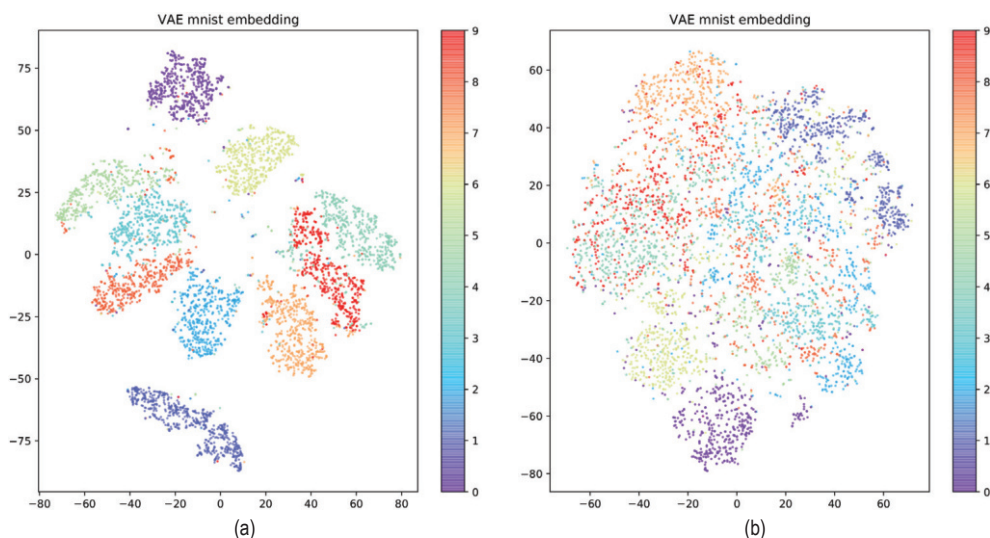


Рис. 20.26 ❖ Проекция tSNE 20-мерного латентного пространства. (a) VAE. (b) Детерминированный АЕ. Построено программой по адресу figures.probl.ai/book1/20.26

Мы можем воспользоваться гладкостью латентного пространства для **интерполяции изображений**. Вместо работы в пространстве пикселей мы можем работать в латентном пространстве модели. Именно, пусть \mathbf{x}_1 и \mathbf{x}_2 – два изображения, а $\mathbf{z}_1 = \mathbb{E}_q(\mathbf{z}|\mathbf{x}_1)[\mathbf{z}]$ и $\mathbf{z}_2 = \mathbb{E}_q(\mathbf{z}|\mathbf{x}_2)[\mathbf{z}]$ – их кодированные представления. Мы можем сгенерировать промежуточное изображение между этими двумя якорными, вычислив $\mathbf{z} = \lambda \mathbf{z}_1 + (1 - \lambda) \mathbf{z}_2$, где $0 \leq \lambda \leq 1$, а затем декодировать его, вычислив $\mathbb{E}[\mathbf{x}|\mathbf{z}]$. Это называется **латентной интерполяцией**. (Теоретическим обоснование линейной интерполяции является тот факт, что обученное многообразие имеет приближенно нулевую кривизну, как показано в работе [SKTF18].) VAE более пригоден для интерполяции в латентном пространстве, чем АЕ, потому что его латентное пространство более гладкое и модель может генерировать изображения почти из любой точки латентного пространства. Это различие наглядно видно при сравнении рис. 20.27а и b.

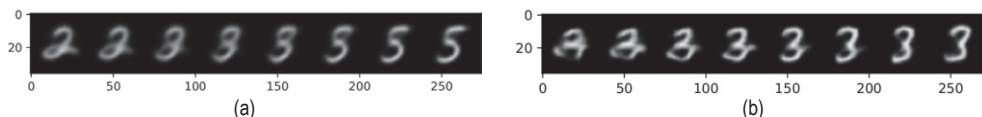


Рис. 20.27 ❖ Линейная интерполяция левого и правого изображений в 20-мерном латентном пространстве. (a) VAE. (b) Детерминированный AE. Построено программой по адресу figures.problml.ai/book1/20.27

20.4. ОБУЧЕНИЕ МНОГООБРАЗИЙ*

В этом разделе мы обсудим проблему восстановления истинной низкоразмерной структуры в наборе данных высокой размерности. Часто предполагается, что эта структура представляет собой искривленное многообразие (что это такое, объясняется ниже), поэтому задача называется **обучением многообразия**, или **нелинейным понижением размерности**. Основное отличие от таких методов, как автокодировщики (раздел 20.3), состоит в том, что нас будут интересовать непараметрические методы, вычисляющие погружение для каждой точки в обучающем наборе, а не обучение общей модели, способной вычислять погружение любого входного вектора. То есть обсуждаемые методы не поддерживают (по крайней мере, без усилий) **обобщение на вневыборочные примеры**. Однако они легко обучаются и весьма гибки. Такие методы могут быть полезны для обучения без учителя (обнаружение знаний), визуализации данных и как шаг предобработки в обучении с учителем.

20.4.1. Что такое многообразие?

Грубо говоря, многообразие – это локально евклидово топологическое пространство. Один из простейших примеров – поверхность Земли, представляющая собой искривленную двумерную поверхность, погруженную в трехмерное пространство. В каждой точке поверхности Земля кажется плоской.

Более формально d -мерным **многообразием** \mathcal{X} называется пространство, для каждой точки которого $x \in \mathcal{X}$ существует окрестность, топологически эквивалентная d -мерному евклидову пространству; она называется **касательным пространством** и обозначается $\mathcal{T}_x = \mathcal{T}_x \mathcal{X}$ (см. рис. 20.28).

Римановым многообразием называется дифференцируемое многообразие, в котором для каждой точки x касательного пространства определено скалярное произведение; предполагается, что оно гладко зависит от x . Скалярное произведение индуцирует понятие расстояния, угла и объема. Совокупность всех скалярных произведений называется **римановой метрикой**. Можно показать, что любое достаточно гладкое риманово многообразие можно погрузить в евклидово пространство, возможно, большей размерности; при этом риманово скалярное произведение в точке становится евклидовым скалярным произведением в этом касательном пространстве.

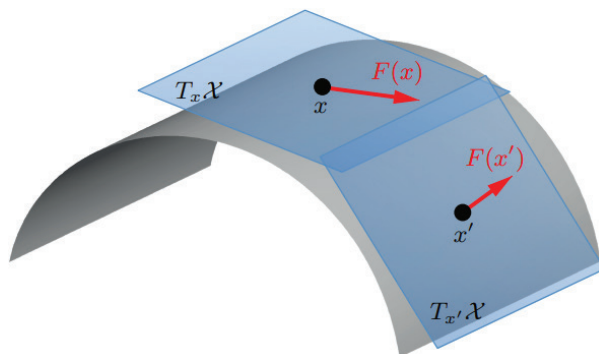


Рис. 20.28 ❖ Иллюстрация касательного пространства и касательных векторов в двух разных точках двумерного искривленного многообразия. На основе рис. 1 из работы [Bro+17a]. Печатается с разрешения Михаэля Бронштейна

20.4.2. Гипотеза многообразия

Большинство «естественно возникающих» наборов данных высокой размерности лежит на многообразии низкой размерности. Это предположение называется **гипотезой многообразия** [FMN16]. Например, рассмотрим случай изображения. На рис. 20.29а показано одно изображение размера 64×57 . Это вектор в 3648-мерном пространстве, в котором каждое измерение соответствует яркости пикселя. Допустим, мы пытаемся сгенерировать изображение путем выборки случайной точки в этом пространстве; практически невероятно, что оно будет похоже на изображение цифры (см. рис. 20.29b). Однако пиксели не являются независимыми, так как порождены некоторой низкоразмерной структурой, а именно формой цифры 6.

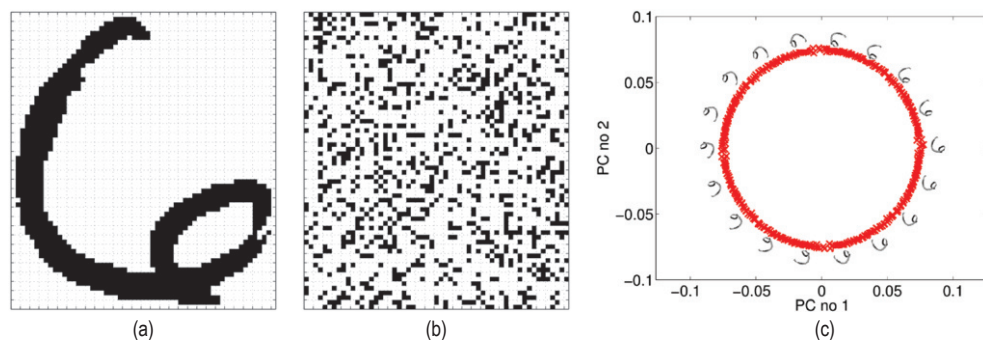


Рис. 20.29 ❖ Многообразие изображения. (a) Изображение цифры 6 из набора данных USPS размера $64 \times 57 = 3648$. (b) Случайный пример из пространства $\{0,1\}^{3648}$, представленный в виде изображения. (c) Набор данных, созданный поворотом оригинального изображения на один градус 360 раз. Мы проецируем эти данные на первые две главные компоненты, чтобы выявить стоящее за ними двумерное круговое многообразие. На основе рис. 1 из работы [Law12].

Печатается с разрешения Нила Лоуренса

Изменяя форму, мы будем генерировать другие изображения. Часто пространство вариаций формы можно охарактеризовать с помощью многообразия низкой размерности. Это показано на рис. 20.29с, где мы применяем PCA (раздел 20.1) для проецирования набора данных из 360 изображений, по одному на каждую немного повернутую версию цифры 6, в двумерное пространство. Видно, что большая часть вариации данных улавливается искривленным двумерным многообразием. Говорят, что **внутренняя размерность d** данных равна 2, хотя **размерность объемлющего пространства D** равна 3648.

20.4.3. Подходы к обучению многообразий

Далее в этом разделе мы поговорим о том, как обучать многообразия на данных. Было предложено много разных алгоритмов, в которых делаются различные предположения о природе многообразия и которые различаются своими вычислительными свойствами. В следующих разделах мы обсудим некоторые из них. Дополнительные сведения см., например, в работе [Bur10].

Таблица 20.1. Перечень некоторых подходов к понижению размерности.
Если метод выпуклый, то в скобках указано, требует ли он решения разреженной или плотной задачи нахождения собственных значений

Метод	Параметрический	Выпуклый	Раздел
PCA/классическое ММШ	Нет	Да (плотный)	Раздел 20.1
Ядерный PCA	Нет	Да (плотный)	Раздел 20.4.6
Isomap	Нет	Да (плотный)	Раздел 20.4.5
LLE	Нет	Да (разреженный)	Раздел 20.4.8
Лапласовы собственные отображения	Нет	Да (разреженный)	Раздел 20.4.9
tSNE	Нет	Нет	Раздел 20.4.10
Автокодировщик	Да	Нет	Раздел 20.3

Методы можно классифицировать, как показано в табл. 20.1. Термин «непараметрический» относится к методам, которые обучаются низкоразмерному погружению z_i для каждой точки данных x_i , но не обучаются отображению, которое можно было бы применить к вневыборочной точке. (Однако в работе [Ben+04b] обсуждается вопрос о том, как распространить многие из этих методов за пределы обучающего набора путем обучения ядра.)

Далее мы сравним работу некоторых из этих методов на двух наборах данных: набор из 1000 трехмерных точек, выбранных из двумерного многообразия «рулет с кремом», и набор 1797 64-мерных точек, выбранных из набора цифр UCI (см. иллюстрацию на рис. 20.30). Мы будем обучать двумерное многообразие, чтобы данные можно было визуализировать.

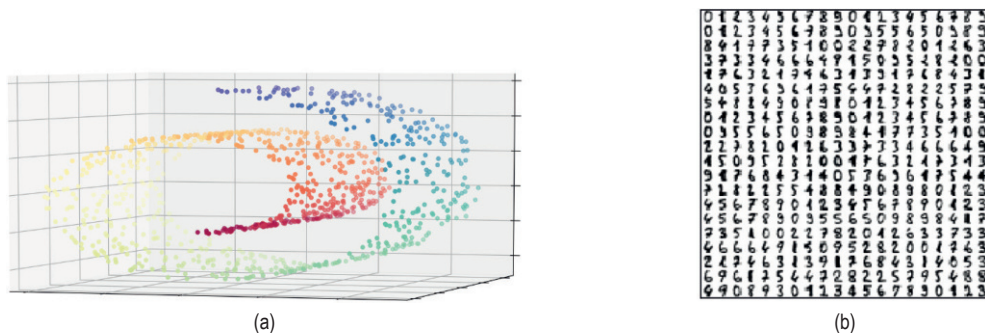


Рис. 20.30 ❖ Некоторые данные, сгенерированные по низкоразмерным многообразиям. (а) Двумерное многообразие «рулет с кремом», погруженное в трехмерное пространство. Построено программой по адресу figures.probl.ai/book1/20.30. (б) Выборка цифр из набора данных UCI размера $8 \times 8 = 64$. Построено программой по адресу figures.probl.ai/book1/20.30

20.4.4. Многомерное шкалирование

Простейший подход к обучению многообразий – **многомерное шкалирование** (ММШ, англ. MDS). Идея в том, чтобы попытаться найти множество векторов низкой размерности $\{z_i \in \mathbb{R}^L : i = 1 \dots N\}$ такое, что попарные расстояния между этими векторами максимально похожи на множество попарных различий $\mathbf{D} = \{d_{ij}\}$, предоставленное пользователем. Ниже будет описано несколько вариантов ММШ, один из которых эквивалентен PCA.

20.4.4.1. Классическое ММШ

Допустим, что мы начинаем с матрицы данных \mathbf{X} размера $N \times D$ со строками \mathbf{x}_i . Определим центрированную матрицу Грама (сходства) следующим образом:

$$\tilde{K}_{ij} = \langle \mathbf{x}_i - \bar{\mathbf{x}}, \mathbf{x}_j - \bar{\mathbf{x}} \rangle. \quad (20.103)$$

В матричной нотации имеем $\tilde{\mathbf{K}} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}^T$, где $\tilde{\mathbf{X}} = \mathbf{C}_N\mathbf{X}$ и $\mathbf{C}_N = \mathbf{I}_N - (1/N)\mathbf{1}_N\mathbf{1}_N^T$ – центрирующая матрица.

Теперь определим **деформацию** (strain) множества погружений:

$$\mathcal{L}_{\text{strain}}(\mathbf{Z}) = \sum_{i,j} (\tilde{K}_{ij} - \langle \tilde{\mathbf{z}}_i, \tilde{\mathbf{z}}_j \rangle)^2 = \|\tilde{\mathbf{K}} - \tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^T\|_F^2, \quad (20.104)$$

где $\tilde{\mathbf{z}}_i = \mathbf{z}_i - \bar{\mathbf{z}}$ – центрированный вектор погружения. Интуитивно понятно, что это способ измерить, насколько хорошо сходство в пространстве данных высокой размерности, \tilde{K}_{ij} , соответствует сходству в пространстве погружения низкой размерности, $\langle \tilde{\mathbf{z}}_i, \tilde{\mathbf{z}}_j \rangle$. Минимизация этой потери называется **классическим ММШ**.

Из раздела 7.5 мы знаем, что аппроксимацией наилучшего ранга L для матрицы является ее представление в виде усеченного сингулярного разложе-

ния, $\tilde{\mathbf{K}} = \mathbf{USV}^T$. Поскольку матрица $\tilde{\mathbf{K}}$ положительно полуопределенная, имеем $\mathbf{V} = \mathbf{U}$. Поэтому оптимальное погружение удовлетворяет соотношению

$$\tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^T = \mathbf{USU}^T = (\mathbf{US}^{1/2})(\mathbf{S}^{1/2}\mathbf{U}^T). \quad (20.105)$$

Таким образом, мы можем взять в качестве векторов погружения строки матрицы $\tilde{\mathbf{Z}} = \mathbf{US}^{1/2}$.

Теперь опишем, как применить классическое ММШ к набору данных, для которого имеются только евклидовы расстояния, а не первичные признаки. Сначала вычислим матрицу квадратов евклидовых расстояний, $\mathbf{D}^{(2)} = \mathbf{D} \odot \mathbf{D}$, содержащую следующие элементы:

$$D_{ij}^{(2)} = \|\mathbf{x}_i - \mathbf{x}_j\|^2 = \|\mathbf{x}_i - \bar{\mathbf{x}}\|^2 + \|\mathbf{x}_j - \bar{\mathbf{x}}\|^2 - 2\langle \mathbf{x}_i - \bar{\mathbf{x}}, \mathbf{x}_j - \bar{\mathbf{x}} \rangle \quad (20.106)$$

$$= \|\mathbf{x}_i - \bar{\mathbf{x}}\|^2 + \|\mathbf{x}_j - \bar{\mathbf{x}}\|^2 - 2\tilde{K}_{ij}. \quad (20.107)$$

Мы видим, что $\mathbf{D}^{(2)}$ отличается от $\tilde{\mathbf{K}}$ только постоянными, зависящими от строки и столбца (и множителем -2). Поэтому $\tilde{\mathbf{K}}$ можно вычислить, дважды центрировав $\mathbf{D}^{(2)}$ с помощью формулы (7.89), в результате чего получаем $\tilde{\mathbf{K}} = -\frac{1}{2}\mathbf{C}_N\mathbf{D}^{(2)}\mathbf{C}_N$. Иными словами,

$$\tilde{K}_{ij} = -\frac{1}{2}\left(d_{ij}^2 - \frac{1}{N}\sum_{l=1}^N d_{il}^2 - \frac{1}{N}\sum_{l=1}^N d_{jl}^2 + \frac{1}{N^2}\sum_{l=1}^N\sum_{m=1}^N d_{lm}^2\right). \quad (20.108)$$

Затем погружения можно вычислить, как и раньше.

Оказывается, что классическое ММШ эквивалентно PCA (раздел 20.1). Чтобы убедиться в этом, обозначим $\tilde{\mathbf{K}} = \mathbf{U}_L\mathbf{S}_L\mathbf{U}_L^T$ усеченное сингулярное разложение ранга L центрированной матрицы ядра. ММШ-погружение имеет вид $\mathbf{Z}_{\text{MDS}} = \mathbf{U}_L\mathbf{S}_L^{1/2}$. Теперь рассмотрим усеченное сингулярное разложение ранга L центрированной матрицы данных $\tilde{\mathbf{X}} = \mathbf{U}_X\mathbf{S}_X\mathbf{V}_X^T$. PCA-погружение имеет вид $\mathbf{Z}_{\text{PCA}} = \mathbf{U}_X\mathbf{S}_X$. Имеем

$$\tilde{\mathbf{K}} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}^T = \mathbf{U}_X\mathbf{S}_X\mathbf{V}_X^T\mathbf{V}_X\mathbf{S}_X\mathbf{U}_X^T = \mathbf{U}_X\mathbf{S}_X^2\mathbf{U}_X^T = \mathbf{U}_L\mathbf{S}_L\mathbf{U}_L^T. \quad (20.109)$$

Отсюда $\mathbf{U}_X = \mathbf{U}_L$ и $\mathbf{S}_X = \mathbf{S}_L^2$, следовательно, $\mathbf{Z}_{\text{PCA}} = \mathbf{Z}_{\text{MDS}}$.

20.4.4.2. Метрическое ММШ

В классическом ММШ предполагается, что расстояния евклидовы. Мы можем обобщить его на любую метрику различия, определив **функцию напряжения**:

$$\mathcal{L}_{\text{stress}}(\mathbf{Z}) = \sqrt{\frac{\sum_{i < j} (d_{i,j} - \hat{d}_{ij})^2}{\sum_{ij} d_{ij}^2}}, \quad (20.110)$$

где $\hat{d}_{ij} = \|\mathbf{z}_i - \mathbf{z}_j\|$. Это называется **метрическим ММШ**. Заметим, что это не та же целевая функция, что в классическом ММШ, поэтому, даже если d_{ij} – евклидовы расстояния, результат все равно будет другим.

Для решения этой задачи оптимизации можно воспользоваться градиентным спуском. Однако лучше использовать алгоритм граничной оптимизации (раздел 8.7) **SMACOF** [Lee77] (Scaling by MAjorizing a COMplication Function – шкалирование путем мажоризации функции сложности). (Именно этот метод реализован в библиотеке `scikit-learn`.) Результаты его применения к нашему сквозному примеру показаны на рис. 20.31.



Рис. 20.31 ❖ Результаты применения метрического ММШ к (а) Рулету с кремом. Построено программой по адресу figures.problml.ai/book1/20.31. (б) Цифрам из набора UCI. Построено программой по адресу figures.problml.ai/book1/20.31

20.4.4.3. Неметрическое ММШ

Вместо того чтобы пытаться уравнивать расстояния между точками, мы можем поставить во главу угла ранжирование близости точек. Пусть $f(d)$ – монотонное преобразование расстояний в ранги. Определим потерю:

$$\mathcal{L}_{\text{NM}}(\mathbf{Z}) = \sqrt{\frac{\sum_{i < j} (f(d_{i,j}) - \hat{d}_{ij})^2}{\sum_{ij} \hat{d}_{ij}^2}}, \quad (20.111)$$

где $\hat{d}_{ij} = \|\mathbf{z}_i - \mathbf{z}_j\|$. Ее минимизация называется **неметрическим ММШ**.

Эту целевую функцию можно оптимизировать итеративно. Сначала функция f оптимизируется для данного \mathbf{Z} с помощью изотонической регрессии; в результате находится оптимальное монотонное преобразование входных расстояний, соответствующее текущим расстояниям в пространстве погружения. Затем оптимизируются погружения \mathbf{Z} для данной f с применением градиентного спуска. После этого весь процесс повторяется.

20.4.4.4. Отображение Саммона

Цель метрической ММШ – минимизировать сумму квадратов расстояний, поэтому особое внимание уделяется большим расстояниям. Однако для многих методов погружения малые расстояния значат больше, так как именно они улавливают локальную структуру. Один из способов уловить ее – поделить каждый член потери на d_{ij} , повысив тем самым вес малых расстояний:

$$\mathcal{L}_{\text{sammon}}(\mathbf{Z}) = \left(\frac{1}{\sum_{i < j} d_{ij}} \right) \sum_{i \neq j} \frac{(\hat{d}_{ij} - d_{ij})^2}{d_{ij}}. \quad (20.112)$$

Минимизация этой потери приводит к **отображению Саммона**. (Коэффициент перед суммой просто упрощает градиент потери.) К сожалению, эта целевая функция невыпукла и, вероятно, делает слишком сильный акцент на точное сохранение малых расстояний. Ниже мы обсудим лучшие методы улавливания локальной структуры.

20.4.5. Isomap

Если многомерные данные лежат на искривленном многообразии, как в примере рулета с кремом, или вблизи него, то ММШ может рассматривать две точки как близкие, даже если их расстояние вдоль многообразия велико. Это показано на рис. 20.32а.

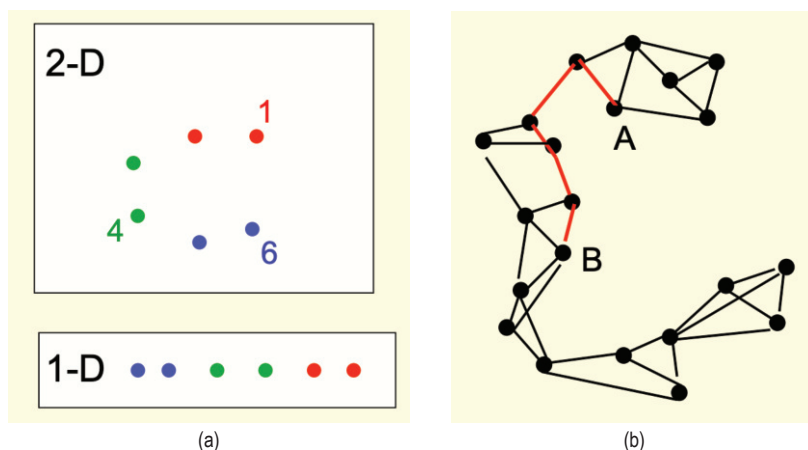


Рис. 20.32 ❖ (а) Если измерять расстояния вдоль многообразия, то окажется, что $d(1, 6) > d(1, 4)$, а если измерять в объемлющем пространстве, то $d(1, 6) < d(1, 4)$. На нижнем рисунке показано истинное одномерное многообразие. (б) Граф K ближайших соседей для нескольких точек данных; красным цветом показан кратчайший путь между A и B в этом графе. Из работы [Hin13].

Печатается с разрешения Джеффри Хинтона

Один из способов уловить эту особенность – построить граф K ближайших соседей между точками данных¹, а затем аппроксимировать расстояние по многообразию между парой точек длиной кратчайшего пути в этом графе, которую можно эффективно вычислить с помощью алгоритма Дijkstra (см. иллюстрацию на рис. 20.32b). После того как это новое расстояние вы-

¹ В библиотеке `scikit-learn` можно воспользоваться функцией `sklearn.neighbors.kneighbors_graph`.

численно, мы можем применить классическое ММШ (т. е. PCA). Этот метод позволяет уловить локальную структуру, не прибегая к поиску локальных оптимумов, и называется он **isomap** [TSL00].

На рис. 20.33 показаны результаты применения этого метода к нашему сквозному примеру. Мы видим, что они вполне разумны. Однако если данные зашумлены, то в графе ближайших соседей могут появиться «ложные» ребра, приводящие к «коротким путям», которые значительно искажают погружение (см. рис. 20.34). Эта проблема называется **топологической неустойчивостью** [BS02]. Уменьшение размера окрестности не решает проблему, потому что тогда многообразие может распасться на большое число несвязных областей. Были предложены другие подходы, см., например, [CC07].

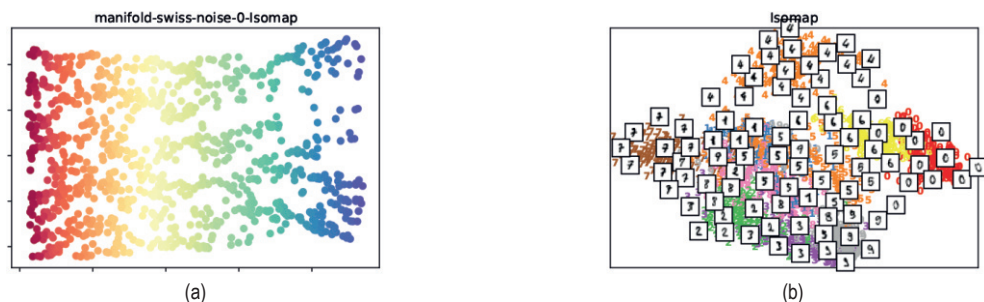


Рис. 20.33 ❖ Результаты применения метода Isomap к (а) Рулету с кремом. Построено программой по адресу figures.problml.ai/book1/20.33. (b) Цифрам из набора UCI. Построено программой по адресу figures.problml.ai/book1/20.33

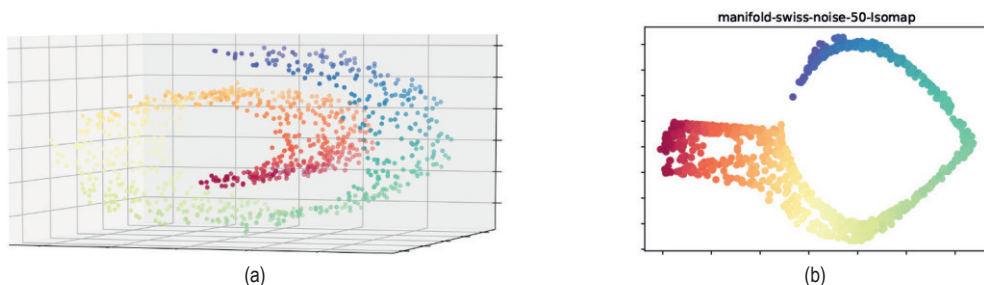


Рис. 20.34 ❖ (а) Зашумленная версия данных рулета с кремом. Мы возмущаем каждую точку, прибавляя случайный шум $\mathcal{N}(0, 0.5^2)$. (b) Результаты применения метода Isomap к этим данным. Построено программой по адресу figures.problml.ai/book1/20.34

20.4.6. Ядерный PCA

PCA (и классическое ММШ) находят наилучшую линейную проекцию данных с целью сохранить попарное сходство между всеми точками. В этом разделе мы рассмотрим нелинейные проекции. Ключевая идея – решить

задачу PCA, вычислив собственные векторы матрицы скалярных произведений (Грама) $\mathbf{K} = \mathbf{X}\mathbf{X}^T$, как в разделе 20.1.3.2, а затем применить ядерный трюк (раздел 17.3.4), который позволит заменить скалярные произведения, например $\mathbf{x}_i^T \mathbf{x}_j$ ядерной функцией, $K_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$. Это называется **ядерным PCA** [SSM98].

Напомним, что, по теореме Мерсера, использование ядра подразумевает наличие некоего истинного пространства признаков, т. е. мы неявно заменяем \mathbf{x}_i вектором $\boldsymbol{\phi}(\mathbf{x}_i) = \boldsymbol{\phi}_i$. Обозначим $\boldsymbol{\Phi}$ соответствующую (воображаемую) матрицу плана, а $\mathbf{K} = \mathbf{X}\mathbf{X}^T$ – матрицу Грама. Наконец, обозначим $\mathbf{S}_\phi = (1/N) \sum_i \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T$ ковариационную матрицу в пространстве признаков. (Пока что будем предполагать, что признаки центрированы.) Из формулы (20.22) имеем, что нормированные собственные векторы \mathbf{S} равны $\mathbf{V}_{\text{kPCA}} = \boldsymbol{\Phi}^T \mathbf{U} \boldsymbol{\Lambda}^{-1/2}$, где \mathbf{U} и $\boldsymbol{\Lambda}$ содержат соответственно собственные векторы и собственные значения \mathbf{K} . Разумеется, мы не можем вычислить \mathbf{V}_{kPCA} , потому что $\boldsymbol{\phi}_i$ потенциально бесконечномерны. Однако можно вычислить проекцию тестового вектора \mathbf{x}_* на пространство признаков:

$$\boldsymbol{\phi}_*^T \mathbf{V}_{\text{kPCA}} = \boldsymbol{\phi}_*^T \boldsymbol{\Phi}^T \mathbf{U} \boldsymbol{\Lambda}^{-1/2} = \mathbf{k}_*^T \mathbf{U} \boldsymbol{\Lambda}^{-1/2}, \quad (20.113)$$

где $\mathbf{k}_* = [\mathcal{K}(\mathbf{x}_*, \mathbf{x}_1), \dots, \mathcal{K}(\mathbf{x}_*, \mathbf{x}_N)]$.

И еще об одной детали нужно побеспокоиться. Ковариационная матрица имеет вид $\mathbf{S} = \boldsymbol{\Phi}^T \boldsymbol{\Phi}$, только если признаки имеют нулевое среднее. Поэтому матрицу Грама $\mathbf{K} = \boldsymbol{\Phi} \boldsymbol{\Phi}^T$ можно использовать, только если $\mathbb{E}[\boldsymbol{\phi}_i] = \mathbf{0}$. К сожалению, нельзя просто вычесть среднее в пространстве признаков, потому что оно может быть бесконечномерным. Но есть один трюк. Определим центрированный вектор признаков как $\tilde{\boldsymbol{\phi}}_i = \boldsymbol{\phi}_i - (1/N) \sum_{j=1}^N \boldsymbol{\phi}_j$. Матрица Грама центрированных векторов признаков равна $\tilde{\mathbf{K}}_{ij} = \tilde{\boldsymbol{\phi}}_i^T \tilde{\boldsymbol{\phi}}_j$. Используя двойное центрирование (7.89), мы можем записать это в матричной форме, $\tilde{\mathbf{K}} = \mathbf{C}_N \mathbf{K} \mathbf{C}_N$, где $\mathbf{C}_N \triangleq \mathbf{I}_N - (1/N) \mathbf{1}_N \mathbf{1}_N^T$ – центрирующая матрица.

Применив kPCA с линейным ядром, мы вернемся к регулярному PCA (классическому ММШ). При этом размерности погружения ограничены $L \times D$. Применяя невырожденное ядро, мы можем использовать до N компонент, так как размер $\boldsymbol{\Phi}$ равен $N \times D^*$, где D^* – (потенциально бесконечная) размерность погруженных векторов признаков. На рис. 20.35 показаны результаты применения этого метода к некоторым данным размерности $D = 2$ с использованием RBF-ядра. Мы проецируем точки единичной сетки на первые 8 компонент и визуализируем соответствующие поверхности с помощью линий уровня. Видно, что первые две компоненты разделяют все три кластера, а следующие компоненты разбивают сами кластеры.

На рис. 20.36 показаны результаты применения kPCA (с RBF-ядром) к нашему сквозному примеру. В этом случае результаты, признаем, не особенно полезны. На самом деле можно показать, что kPCA с RBF-ядром расширяет, а не уменьшает пространство признаков [WSS04], как мы видели на рис. 20.35, что делает этот метод практически бесполезным для понижения размерности. Решение этой проблемы мы обсудим в разделе 20.4.7.

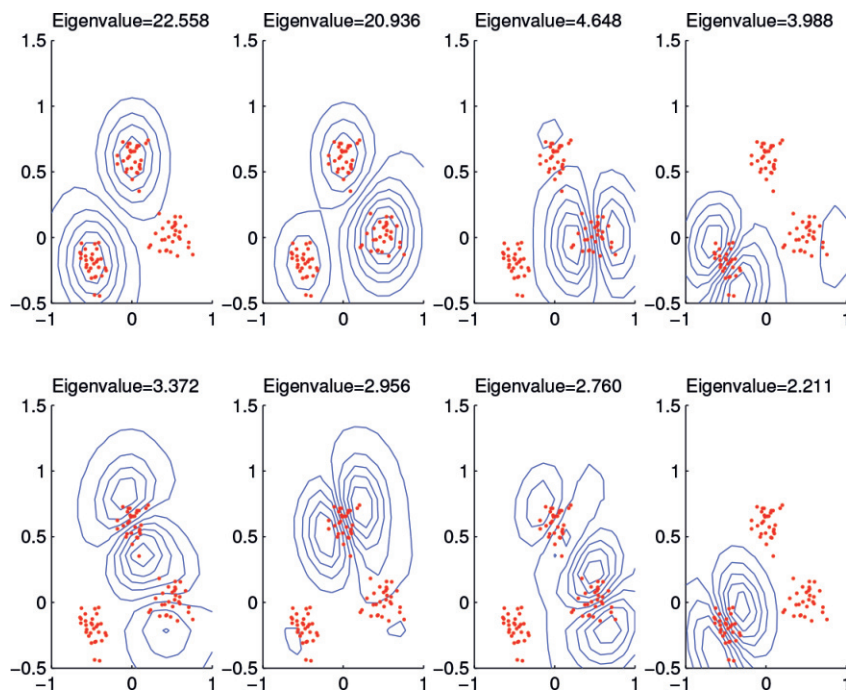


Рис. 20.35 ❖ Визуализация первых восьми базисных функций ядерного PCA для некоторых двумерных данных. Используется RBF-ядро с $\sigma^2 = 0.1$. Построено программой по адресу figures.problml.ai/book1/20.35

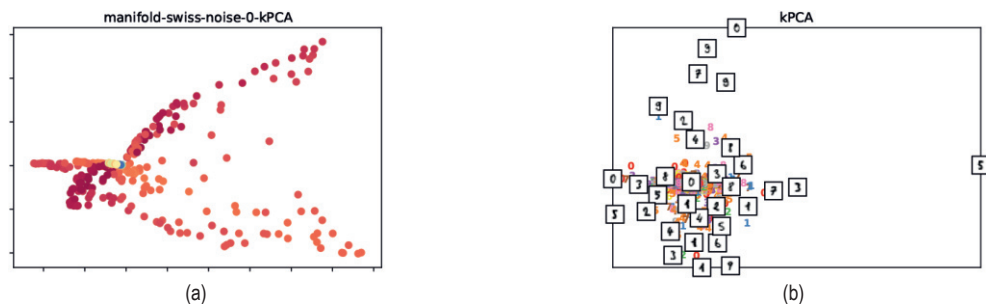


Рис. 20.36 ❖ Результаты применения ядерного PCA к (a) Рулету с кремом. Построено программой по адресу figures.problml.ai/book1/20.36. (b) Цифрам из набора UCI. Построено программой по адресу figures.problml.ai/book1/20.36

20.4.7. Максимальное раскрытие дисперсии

Метод kPCA с некоторыми ядрами, например RBF, может не давать низкоразмерного погружения, о чем было сказано в разделе 20.4.6. Это наблюдение привело к разработке алгоритма **полуопределенного погружения** (semi-

definite embedding) [WSS04], называемого также **развертыванием с максимальной дисперсией** (maximum variance unfolding – MVU), который пытается обучить погружение $\{\mathbf{z}_i\}$ такое, что

$$\max \sum_{ij} \|\mathbf{z}_i - \mathbf{z}_j\|_2^2 \text{ при условии } \|\mathbf{z}_i - \mathbf{z}_j\|_2^2 = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \text{ для всех } (i, j) \in G, \quad (20.114)$$

где G – граф ближайших соседей (как в Isomap). В этом подходе мы пытаемся «развернуть» многообразие данных, сохранив по возможности ограничения ближайших соседей.

Эту задачу можно переформулировать как задачу **полуопределенного программирования** (SDP), определив матрицу ядра $\mathbf{K} = \mathbf{Z}\mathbf{Z}^T$, а затем оптимизировав функцию:

$$\max \text{tr}(\mathbf{K}) \text{ при условиях } \|\mathbf{z}_i - \mathbf{z}_j\|_2^2 = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2, \sum_{ij} K_{ij}, \mathbf{K} \succ 0. \quad (20.115)$$

20.4.8. Локально линейное погружение

Все обсуждавшиеся выше методы опираются на спектральное разложение полной матрицы попарных сходств либо в объемлющем пространстве (PCA), либо в пространстве признаков (kPCA), либо на графе K ближайших соседей (Isomap). В этом разделе мы обсудим **локально линейное погружение** (local linear embedding – LLE) [RS00] – метод, который решает разреженную задачу на собственные значения и таким образом акцентирует внимание на локальной структуре внутри данных.

В LLE предполагается, что многообразие данных в окрестности каждой точки \mathbf{x}_i локально линейно. Наилучшую линейную аппроксимацию можно найти, предсказав \mathbf{x}_i в виде линейной комбинации K ее ближайших соседей с весами реконструкции \mathbf{w}_i . Это можно сделать, решив задачу:

$$\hat{\mathbf{W}} = \min_{\mathbf{W}} \sum_{i=1}^N \left\| \mathbf{x}_i - \sum_{j=1}^N w_{ij} \mathbf{x}_j \right\|^2 \quad (20.116)$$

$$\text{при условии } \begin{cases} w_{ij} = 0, & \text{если } \mathbf{x}_j \notin \text{nbr}(\mathbf{x}_i, K) \\ \sum_{j=1}^N w_{ij} = 1 & \text{для } i = 1 : N \end{cases}. \quad (20.117)$$

Обратите внимание на ограничение – сумма весов должна быть равна 1; это нужно, для того чтобы предотвратить тривиальное решение $\mathbf{W} = \mathbf{0}$. Получающийся вектор весов \mathbf{w}_i образует **барицентрические координаты** \mathbf{x}_i .

Любое линейное отображение этой гиперплоскости в пространство низкой размерности сохраняет веса реконструкции, а значит, и локальную геометрию. Таким образом, мы можем найти низкоразмерные погружения для каждой точки, решив задачу:

$$\hat{\mathbf{Z}} = \underset{\mathbf{Z}}{\operatorname{argmin}} \sum_i \left\| \mathbf{z}_i - \sum_{j=1}^N \hat{w}_{ij} \mathbf{z}_j \right\|_2^2, \quad (20.118)$$

где $\hat{w}_{ij} = 0$, если j не является одним из K ближайших соседей i . Эту потерю можно переписать в виде

$$\mathcal{L}(\mathbf{Z}) = \|\mathbf{Z} - \mathbf{WZ}\|^2 = \mathbf{Z}^T (\mathbf{I} - \mathbf{W})^T (\mathbf{I} - \mathbf{W}) \mathbf{Z}. \quad (20.119)$$

Следовательно, решение дают собственные векторы матрицы $(\mathbf{I} - \mathbf{W})^T (\mathbf{I} - \mathbf{W})$, соответствующие наименьшим ненулевым собственным значениям, как показано в разделе 7.4.8.

На рис. 20.37 показаны результаты применения LLE к нашему сквозному примеру. В данном случае результаты не столь хороши, как полученные с помощью Isomap. Однако метод все же менее чувствителен к появлению коротких путей (шуму).

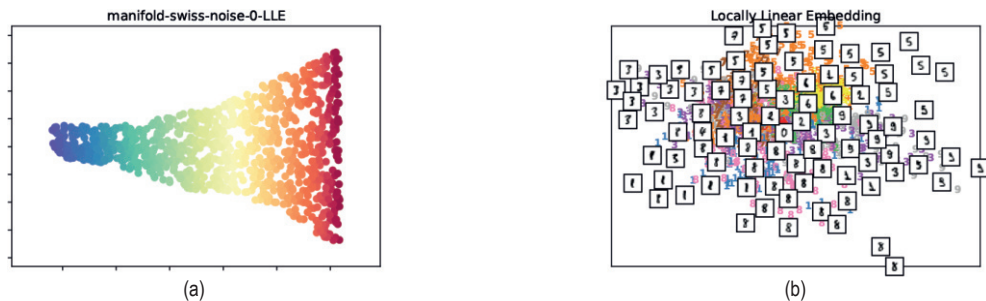


Рис. 20.37 ❖ Результаты применения LLE к (a) Рулету с кремом. Построено программой по адресу figures.problm.ai/book1/20.37. (b) Цифрам из набора UCI. Построено программой по адресу figures.problm.ai/book1/20.37

20.4.9. Лапласовы собственные отображения

В этом разделе мы опишем **лапласовы собственные отображения** (Laplacian eigenmaps), или **спектральное погружение** [BN01]. Идея в том, чтобы вычислить низкоразмерное представление данных, в котором минимизируется взвешенная комбинация расстояний от точки до K ее ближайших соседей. Мы назначаем больший вес первому ближайшему соседу, чем второму и т. д. Детали описаны ниже.

20.4.9.1. Использование собственных векторов лапласиана графа для вычисления погружений

Мы хотим найти погружения, которые минимизируют целевую функцию:

$$\mathcal{L}(\mathbf{Z}) = \sum_{(i,j) \in E} W_{i,j} \|\mathbf{z}_i - \mathbf{z}_j\|_2^2, \quad (20.120)$$

где $W_{ij} = \exp((-1/2\sigma^2)\|\mathbf{x}_i - \mathbf{x}_j\|_2^2)$, если i и j – соседи в графе KNN, и 0 в противном случае. Ограничение $\mathbf{Z}^T \mathbf{D} \mathbf{Z} = \mathbf{I}$ добавлено, чтобы избежать вырожденного решения $\mathbf{Z} = \mathbf{0}$, где \mathbf{D} – диагональная матрица, в которой хранятся степени вершин, $D_{ii} = \sum_j W_{ij}$.

Мы можем переписать приведенную выше целевую функцию следующим образом:

$$\mathcal{L}(\mathbf{Z}) = \sum_{ij} W_{ij} (\|\mathbf{z}_i\|^2 + \|\mathbf{z}_j\|^2 - 2\mathbf{z}_i \mathbf{z}_j^T) \quad (20.121)$$

$$= \sum_i D_{ii} \|\mathbf{z}_i\|^2 + \sum_i D_{jj} \|\mathbf{z}_j\|^2 - 2 \sum_i W_{ij} \mathbf{z}_i \mathbf{z}_j^T \quad (20.122)$$

$$= 2\mathbf{Z}^T \mathbf{D} \mathbf{Z} - 2\mathbf{Z}^T \mathbf{W} \mathbf{Z} = 2\mathbf{Z}^T \mathbf{L} \mathbf{Z}, \quad (20.123)$$

где $\mathbf{L} = \mathbf{D} - \mathbf{W}$ – лапласиан графа (см. раздел 20.4.9.2). Можно показать, что ее оптимизация эквивалентна решению задачи на собственные значения $\mathbf{L} \mathbf{z}_i = \lambda \mathbf{D} \mathbf{z}_i$, в которой требуется найти L наименьших ненулевых собственных значений.

На рис. 20.38 показаны результаты применения этого метода (с RBF-ядром) к нашему сквозному примеру.

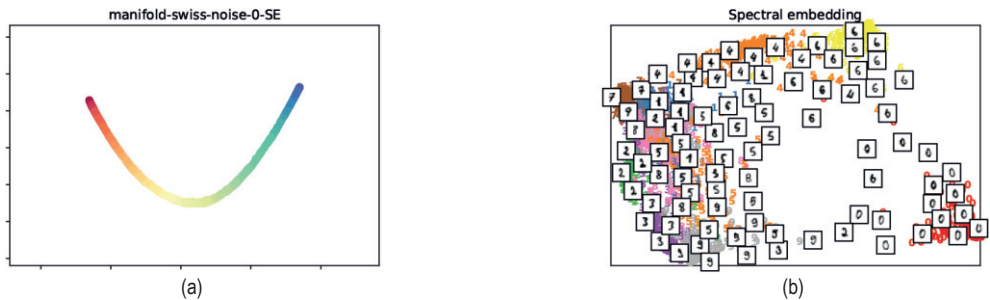


Рис. 20.38 ❖ Результаты применения лапласовых собственных отображений к (a) Рулету с кремем. Построено программой по адресу figures.problml.ai/book1/20.38. (b) Цифрам из набора UCI. Построено программой по адресу figures.problml.ai/book1/20.38

20.4.9.2. Что такое лапласиан графа?

Выше мы видели, что можем вычислить собственные векторы лапласиана графа, чтобы обучить хорошее погружение точек пространства высокой размерности. В этом разделе мы приведем интуитивные соображения, показывающие, почему эта идея работает.

Пусть \mathbf{W} – симметричная матрица весов для графа, в которой $W_{ij} = W_{ji} \geq 0$. Обозначим $\mathbf{D} = \text{diag}(d_i)$ диагональную матрицу, содержащую взвешенные степени вершин, $d_i = \sum_j w_{ij}$. Определим **лапласиан графа** следующим образом:

$$\mathbf{L} \triangleq \mathbf{D} - \mathbf{W}. \quad (20.124)$$

Таким образом, элементы \mathbf{L} равны

$$L_{ij} = \begin{cases} d_i, & \text{если } i = j \\ -1, & \text{если } i \neq j \text{ и } w_{ij} \neq 0 \\ 0 & \text{в противном случае} \end{cases} \quad (20.125)$$

Как эта матрица вычисляется, показано на рис. 20.39.

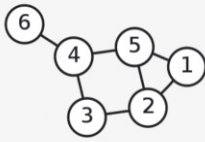
Помеченный граф	Степенная матрица	Матрица смежности	Матрица Лапласа
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

Рис. 20.39 ❖ Матрица Лапласа для неориентированного графа.
Из статьи https://en.wikipedia.org/wiki/Laplacian_matrix.
Печатается с разрешения автора «Википедии» AzaToth

Предположим, что с каждой вершиной i графа ассоциировано значение $f_i \in \mathbb{R}$ (см. пример на рис. 20.40). Тогда лапласиан графа можно использовать как оператор разности, чтобы вычислить дискретную производную этой функции в точке:

$$(\mathbf{L}\mathbf{f})(i) = \sum_{j \in \text{nbr}_i} w_{ij}[f(i) - f(j)], \quad (20.126)$$

где nbr_i – множество соседей вершины i . Мы также можем найти полную меру «гладкости» функции f , вычислив **энергию Дирихле**:

$$\mathbf{f}^T \mathbf{L} \mathbf{f} = \mathbf{f}^T \mathbf{D} \mathbf{f} - \mathbf{f}^T \mathbf{W} \mathbf{f} = \sum_i d_i f_i^2 - \sum_{i,j} f_i f_j w_{ij} \quad (20.127)$$

$$= \frac{1}{2} \left(\sum_i d_i f_i^2 - 2 \sum_{i,j} f_i f_j w_{ij} + \sum_j d_j f_j^2 \right) = \frac{1}{2} \sum_{i,j} w_{ij} (f_i - f_j)^2. \quad (20.128)$$

Путем изучения собственных значений и собственных векторов матрицы Лапласа мы можем выявить различные полезные свойства функции. (Применение методов линейной алгебры к изучению матрицы смежности графа и родственных ей матриц называется **спектральной теорией графов** [Chu97].) Например, мы видим, что \mathbf{L} симметричная и положительно определенная, так как $\mathbf{f}^T \mathbf{L} \mathbf{f} \geq 0$ для всех $\mathbf{f} \in \mathbb{R}^N$, что вытекает из формулы (20.128) в силу предположения о том, что $w_{ij} \geq 0$. Следовательно, \mathbf{L} имеет N неотрицательных вещественных собственных значений $0 \leq \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_N$. Соответствующие собственные векторы образуют ортогональный базис для функции f , определенной на графе, упорядоченный по убыванию гладкости.

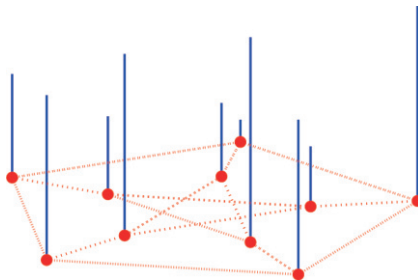


Рис. 20.40 ❖ Положительная функция, определенная на графе.

На основе рис. 1 из работы [Shu+13].

Печатается с разрешения Паскаля Фроссара

В разделе 20.4.9.1 мы обсуждали лапласовы собственные отображения как способ обучить низкоразмерные погружения для векторов данных высокой размерности. Идея заключается в том, чтобы положить $\mathbf{z}_{id} = f_i^d$ равным d -му измерению погружения для входа i , а затем найти базис для этих функций (т. е. погружение точек), который гладко изменяется на графе, и тем самым учесть расстояния между точками в объемлющем пространстве.

У лапласиана графа есть много приложений в МО. Например, в разделе 21.5.1 мы обсудим нормализованные разрезы – способ обучиться кластеризации векторов данных высокой размерности, основанный на попарном сходстве, а в работе [WTN19] описано, как использовать собственные векторы матрицы перехода состояний в целях обучения представлений для обучения с подкреплением.

20.4.10. t-SNE

В этом разделе мы опишем очень популярный невыпуклый метод обучения низкоразмерных погружений, называемый **t-SNE** [MH08]. Он обобщает более ранний метод **стохастического погружения соседей** (SNE) из работы [HR03], поэтому сначала опишем SNE, а потом перейдем к t-SNE.

20.4.10.1. Стохастическое погружение соседей

Основная идея метода стохастического погружения соседей (stochastic neighborhood embedding – SNE) заключается в том, чтобы преобразовать евклидовы расстояния в пространстве высокой размерности в условные вероятности, представляющие сходство. Формально определим p_{ji} как вероятность того, что точка j выбрала бы точку i в качестве своего соседа, если бы соседи выбирались пропорционально их вероятности в соответствии с гауссовым распределением, центрированным в точке \mathbf{x}_i :

$$p_{ji} = \frac{\exp\left(-\frac{1}{2\sigma_i^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\frac{1}{2\sigma_i^2}\|\mathbf{x}_i - \mathbf{x}_k\|^2\right)}. \quad (20.129)$$

Здесь σ_i^2 – дисперсия для точки i , которую можно использовать, чтобы «увеличить» масштаб точек в плотных областях пространства входов и уменьшить в более разреженных областях. (Как оценивать линейные масштабы σ_i^2 , мы обсудим чуть ниже.)

Обозначим \mathbf{z}_i низкоразмерное погружение, представляющее \mathbf{x}_i . Определим сходство в пространстве низкой размерности аналогично:

$$q_{j|i} = \frac{\exp(-\|\mathbf{z}_i - \mathbf{z}_j\|^2)}{\sum_{k \neq i} \exp(-\|\mathbf{z}_i - \mathbf{z}_k\|^2)}. \quad (20.130)$$

В этом случае дисперсия постоянна: ее изменение просто изменило бы масштаб обученного отображения, но не его топологию.

Если погружение хорошее, то $q_{j|i}$ должно соответствовать $p_{j|i}$. Поэтому целевая функция SNE определяется следующим образом:

$$\mathcal{L} = \sum_i \mathbb{KL}(P_i \| Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}, \quad (20.131)$$

где P_i – условное распределение всех остальных точек данных при условии \mathbf{x}_i , Q_i – условное распределение всех остальных латентных точек при условии \mathbf{z}_i , а $\mathbb{KL}(P_i \| Q_i)$ – расхождение Кульбака–Лейблера (раздел 6.2) между распределениями.

Отметим, что эта целевая функция асимметрична. Именно, стоимость велика, если для моделирования большой $p_{j|i}$ используется малая $q_{j|i}$. Поэтому целевая функция предпочитает сближать удаленные точки, а не отдалять близкие. Составить более точное представление о геометрии можно, взглянув на градиент для каждого вектора погружения, который равен

$$\nabla_{\mathbf{z}_i} \mathcal{L}(\mathbf{Z}) = 2 \sum_j (\mathbf{z}_j - \mathbf{z}_i) (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j}). \quad (20.132)$$

Таким образом, точки притягиваются, если вероятности p больше, чем вероятности q , и отталкиваются в противном случае.

Хотя эта функция интуитивно кажется разумной, она не выпукла. Тем не менее ее можно минимизировать методом СГС. На практике полезно прибавить гауссов шум к точкам погружения и постепенно «отжигать» его величину. В работе [Hin13] рекомендуется «тратить длительное время при уровне шума, при котором из горячей плазмы точек начинает формироваться глобальная структура», а только потом приступить к его уменьшению¹.

¹ См. обсуждение отжига и фазовых переходов в обучении без учителя в работах [Ros98; WF20]. См. также обсуждение алгоритма **эластичного погружения** в работе [CP10]; в нем применяются методы теории гомотопий для более эффективной оптимизации модели, связанной одновременно с SNE и лапласовыми собственными отображениями.

20.4.10.2. Симметричное SNE

Существует немного более простой вариант SNE, в котором минимизируется только одно расхождение КЛ между совместным распределением P в пространстве высокой размерности и распределением Q в пространстве низкой размерности:

$$\mathcal{L} = \mathbb{KL}(P\|Q) = \sum_{i < j} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \quad (20.133)$$

Он называется **симметричным SNE**.

Очевидный способ определения p_{ij} таков:

$$p_{ij} = \frac{\exp\left(-\frac{1}{2\sigma^2} \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)}{\sum_{k < l} \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{x}_k - \mathbf{x}_l\|^2\right)}. \quad (20.134)$$

Определить q_{ij} можно аналогично. Соответствующий градиент принимает вид:

$$\nabla_{\mathbf{z}_i} \mathcal{L}(\mathbf{Z}) = 2 \sum_j (\mathbf{z}_j - \mathbf{z}_i) (p_{ij} - q_{ij}). \quad (20.135)$$

Как и раньше, точки притягиваются, если вероятности p больше, чем q , и отталкиваются в противном случае.

Хотя симметричное SNE реализовать немного проще, оно не обладает полезным свойством обычного SNE: данные являются своим собственным оптимальным погружением, если размерность погружения L положить равной размерности объемлющего пространства D . Тем не менее на практике оба метода, похоже, дают сопоставимые результаты на реальных наборах данных, когда $L \ll D$.

20.4.10.3. SNE с t -распределением

Фундаментальная проблема SNE и многих других методов погружения заключается в том, что они пытаются сблизить в пространстве погружения (обычно двумерном) точки, отстоящие довольно далеко друг от друга в пространстве высокой размерности; эта так называемая **проблема скученности** возникает из-за использования квадратичных ошибок (или гауссовых вероятностей).

Одно из возможных решений – использовать в латентном пространстве распределение вероятностей с более тяжелыми хвостами, устранив тем самым нежелательные силы притяжения между точками, отстоящими далеко друг от друга в пространстве высокой размерности. Очевидный выбор – t -распределение Стьюдента (раздел 2.7.1). В алгоритме t -SNE параметр степени свободы задается равным $\nu = 1$, поэтому мы приходим к распределению Коши:

$$q_{ij} = \frac{(1 + \|\mathbf{z}_i - \mathbf{z}_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|\mathbf{z}_k - \mathbf{z}_i\|^2)^{-1}}. \quad (20.136)$$

Можно использовать такую же глобальную целевую функцию на основе расхождения КЛ, как в формуле (20.133). Для t-SNE градиент равен

$$\nabla \mathbf{z}_i \mathcal{L} = 4 \sum_j (p_{ij} - q_{ij})(\mathbf{z}_i - \mathbf{z}_j)(1 + \|\mathbf{z}_i - \mathbf{z}_j\|^2)^{-1}. \quad (20.137)$$

Градиент для симметричного (гауссова) SNE такой же, но в нем отсутствует член $(1 + \|\mathbf{z}_i - \mathbf{z}_j\|^2)^{-1}$. Этот член полезен, потому что $(1 + \|\mathbf{z}_i - \mathbf{z}_j\|^2)^{-1}$ действует так же, как закон обратных квадратов. Следовательно, точки в пространстве погружения ведут себя, как звезды и галактики, образуя много четко разделенных кластеров (галактик), в каждом из которых плотно упаковано много звезд. Это бывает полезно для разделения классов данных без учителя (см. пример на рис. 20.41).

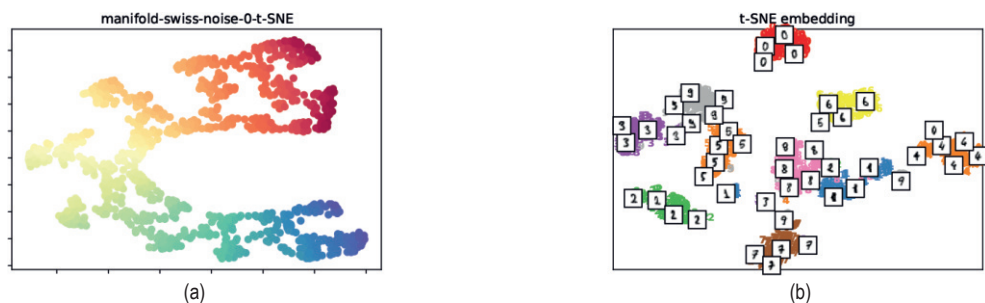


Рис. 20.41 ❖ Результаты применения tSNE к (a) Рулету с кремом. Построено программой по адресу figures.problml.ai/book1/20.41. (b) Цифрам из набора UCI. Построено программой по адресу figures.problml.ai/book1/20.41

20.4.10.4. Выбор линейного масштаба

Важным параметром в t-SNE является локальная полоса пропускания σ_i^2 . Обычно она выбирается, так чтобы P_i имела перплексивность, заданную пользователем¹. Ее можно интерпретировать как гладкую меру эффективного числа соседей.

К сожалению, результаты t-SNE могут быть весьма чувствительны к параметру перплексивности, поэтому разумно прогонять алгоритм с большим числом различных значений. Это показано на рис. 20.42. Входные данные двумерные, поэтому искажений из-за их отображения в двумерное латент-

¹ Перплексивность определяется как $2^{\mathbb{H}(P_i)}$, где $\mathbb{H}(P_i) = -\sum_j p_{ji} \log_2 p_{ji}$ — энтропия; детали см. в разделе 6.1.5. Если радиус вокруг каждой точки велик (велико значение σ_i), то энтропия, а значит, и перплексивность будет высокой.

ное пространство нет. Если перплексивность слишком мала, то метод находит в каждом кластере реально не существующую структуру. Когда перплексивность равна 30 (значение по умолчанию в `scikit-learn`), кластеры кажутся равноотстоящими в пространстве погружения, хотя в пространстве данных расстояния между ними различны. Другие подводные камни, присущие интерпретации графиков t-SNE, приведены в работе [WVJ16].

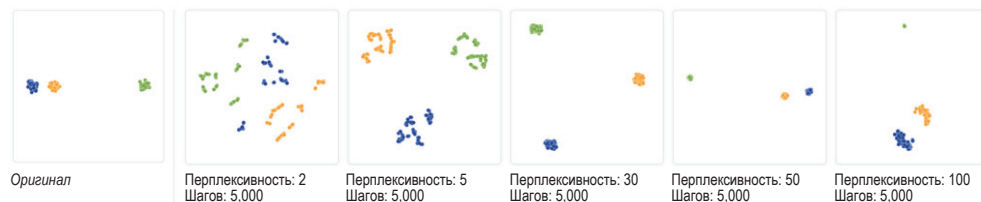


Рис. 20.42 ❖ Эффект изменения параметра перплексивности при применении t-SNE к двумерным данным. Из работы [WVJ16]. Смотрите анимированный вариант этих рисунков по адресу <http://distill.pub/2016/misread-tsne>. Печатается с разрешения Мартина Уоттенберга

20.4.10.5. Вычислительные проблемы

Наивная реализация t-SNE занимает время $O(N^2)$, что следует из члена градиента в формуле (20.137). Ускорить работу можно, воспользовавшись аналогией с моделированием N тел в физике. Именно, для вычисления градиента нужно найти силы, действующие на каждую из N точек со стороны всех остальных. Однако далеко отстоящие друг от друга точки можно объединить в кластеры (с точки зрения вычислений) и аппроксимировать их эффективную силу несколькими репрезентативными точками в каждом кластере. Затем для аппроксимации действующих сил можно воспользоваться **алгоритмом Барнса–Хата** [BH86], который требует времени $O(N \log N)$, как предложено в работе [Маа14]. К сожалению, это хорошо работает только для погружений совсем низкой размерности, скажем $L = 2$.

20.4.10.6. UMAP

Было предложено несколько обобщений tSNE, в которых делается попытка улучшить быстродействие, качество пространства погружений или способность к погружению в пространство размерности больше 2.

Одно из недавних популярных обобщений называется **UMAP** (Uniform Manifold Approximation and Projection – равномерная аппроксимация и проецирование многообразия) (см. [МНМ18]). На верхнем уровне оно похоже на tSNE, но обычно лучше сохраняет глобальную структуру и гораздо быстрее работает. Поэтому становится проще опробовать разные значения гиперпараметров. Интерактивное пособие по UMAP и его сравнение с tSNE см. в работе [CP19].

20.5. Погружения слов

Слова – это категориальные случайные величины, поэтому соответствующие им представления унитарными векторами разрежены. Проблема такого бинарного представления заключается в том, что семантически похожие слова могут иметь сильно различающиеся векторные представления. Например, расстояние Хэмминга между родственными словами «man» и «woman» равно 1 – такое же, как между никак не связанными словами «man» и «banana».

Стандартный способ решения этой проблемы – использовать **погружения слов**, когда каждый разреженный унитарный вектор, $\mathbf{s}_{n,t} \in \{0, 1\}^M$, представляющий t -е слово в документе n , отображается в низкоразмерный плотный вектор, $\mathbf{z}_{n,t} \in \mathbb{R}^D$, так что семантически похожие слова оказываются рядом. Это может существенно снизить разреженность данных. Ниже мы обсудим различные способы обучения таких погружений.

Прежде чем переходить к методам, необходимо определить, что мы понимаем под «семантически похожими» словами. Будем считать, что два слова семантически похожи, если они встречаются в похожих контекстах. Эту **дистрибутивную гипотезу** [Har54] часто описывают фразой (взятой из работы [Fir57]) «узнаешь слово по тому, с кем оно водит компанию». Таким образом, все обсуждаемые ниже методы обучают отображение из контекста слова в вектор погружения для этого слова.

20.5.1. Латентно-семантический анализ и индексирование

В этом разделе мы обсудим простой способ обучиться погружениям слов на основе сингулярного разложения (раздел 7.5) терм-документной матрицы.

20.5.1.1. Латентно-семантическое индексирование

Пусть C_{ij} показывает, сколько раз «терм» i встречается в «контексте» j . Что означает слово «терм», зависит от приложения. В англоязычных текстах это часто уникальные токены, разделенные знаком препинания или пробелом; для простоты будем называть их «словами». Однако текст можно подвергнуть предварительной обработке, удалив слова, которые встречаются очень часто или, наоборот, редко. Возможны и другие виды предобработки, обсуждаемые в разделе 1.5.4.1.

Что понимать под «контекстом», также зависит от приложения. В этом разделе мы подсчитываем, сколько раз слово i встречается в каждом документе $j \in \{1, \dots, N\}$ из некоторого множества, или **корпуса**, документов; в результате получается **терм-документная матрица** \mathbf{C} , показанная на рис. 1.15. (Иногда к счетчикам применяется преобразование TF-IDF, обсуждавшееся в разделе 1.5.4.2.)

Пусть $\mathbf{C} \in \mathbb{R}^{M \times N}$ – матрица счетчиков, а $\hat{\mathbf{C}}$ – ее аппроксимация ранга K , минимизирующая следующую потерю:

$$\mathcal{L}(\hat{\mathbf{C}}) = \|\mathbf{C} - \hat{\mathbf{C}}\|_F = \sum_{ij} (C_{ij} - \hat{C}_{ij})^2. \quad (20.138)$$

Можно показать, что минимизатор описывается усеченным сингулярным разложением ранга K , $\hat{\mathbf{C}} = \mathbf{USV}$. Это означает, что каждый элемент c_{ij} можно представить билинейным произведением:

$$c_{ij} \approx \sum_{k=1}^K u_{ik} s_k v_{jk}. \quad (20.139)$$

Мы определяем \mathbf{u}_i как погружение слова i , а $\mathbf{s} \odot \mathbf{v}_j$ как погружение для контекста j .

Эти погружения можно использовать для **поиска документов**. Идея в том, чтобы вычислить погружение для слов запроса, используя \mathbf{u}_i , и сравнить его с погружениями всех документов или контекстов \mathbf{v}_j . Это называется **латентно-семантическим индексированием** (ЛСИ, англ. LSI) [Dee+90].

Точнее, предполагая, что запрос – это мешок слов w_1, \dots, w_B , мы будем представлять его вектором $\mathbf{q} = (1/B) \sum_{b=1}^B \mathbf{u}_{w_b}$, где \mathbf{u}_{w_b} – погружение слова w_b . Пусть документ j представлен вектором \mathbf{v}_j . Затем мы ранжируем документы по коэффициенту Отиаи между вектором запроса и документом:

$$\text{sim}(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q}^\top \mathbf{d}}{\|\mathbf{q}\| \|\mathbf{d}\|}, \quad (20.140)$$

где $\|\mathbf{q}\| = \sqrt{\sum_i q_i^2}$ – ℓ_2 -норма \mathbf{q} . Это выражение измеряет углы между двумя векторами, как показано на рис. 20.43. Заметим, что для векторов единичной нормы коэффициент Отиаи совпадает со скалярным произведением; он также равен квадрату евклидова расстояния с точностью до знака и не-существенной аддитивной постоянной:

$$\|\mathbf{q} - \mathbf{d}\|^2 = (\mathbf{q} - \mathbf{d})^\top (\mathbf{q} - \mathbf{d}) = \mathbf{q}^\top \mathbf{q} + \mathbf{d}^\top \mathbf{d} - 2\mathbf{q}^\top \mathbf{d} = 2(1 - \text{sim}(\mathbf{q}, \mathbf{d})). \quad (20.141)$$

20.5.1.2. Латентно-семантический анализ

Теперь предположим, что контекст определен более общо, как некоторая локальная окрестность слова $j \in \{1, \dots, M^h\}$, где h – размер окна. Таким образом, C_{ij} показывает, сколько раз слово i встречается в окрестности типа j . Мы можем вычислить сингулярное разложение этой матрицы, как и прежде, и получить $c_{ij} \approx \sum_{k=1}^K u_{ik} s_k v_{jk}$. Определим \mathbf{u}_i как погружение слова i , а $\mathbf{s} \odot \mathbf{v}_j$ как погружение контекста j . Это называется **латентно-семантическим анализом** (ЛСА, англ. LSA) [Dee+90].

Например, допустим, что \mathbf{C} вычисляется на Британском национальном корпусе¹. Для каждого слова найдем K ближайших соседей в пространстве погружения, ранжированных по коэффициенту Отиаи (т. е. нормированному

¹ Этот пример взят из работы [Eis19, стр. 312].

скалярному произведению). Если запрашивается слово «dog» и используется $h = 2$ или $h = 30$, то ближайшими соседями будут

$h=2$: cat, horse, fox, pet, rabbit, pig, animal, mongrel, sheep, pigeon

$h=30$: kennel, puppy, pet, bitch, terrier, rottweiler, canine, cat, to bark

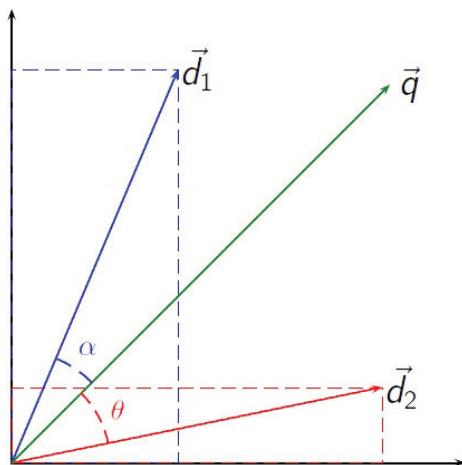


Рис. 20.43 ❖ Сходство между вектором запроса q и двумя векторами документов d_1 и d_2 , вычисленное в виде коэффициента Отиаи. Поскольку угол α меньше θ , мы видим, что запрос больше похож на документ 1. Из статьи https://en.wikipedia.org/wiki/Vector_space_model. Печатается с разрешения автора «Википедии» Riclas

Двухсловное контекстное окно более чувствительно к синтаксису, а 30-словное – к семантике. «Оптимальный» размер контекста h зависит от приложения.

20.5.1.3. Поточечная взаимная информация

На практике ЛСА (и другие подобные методы) дает лучшие результаты, если заменить счетчики C_{ij} **поточечной взаимной информацией** (pointwise mutual information – **PMI**) [CH90], определенной следующим образом:

$$\text{PMI}(i, j) = \log \frac{p(i, j)}{p(i)p(j)}. \quad (20.142)$$

Если между словом i и контекстом j имеется сильная связь, то $\text{PMI}(i, j) > 0$. Если PMI отрицательна, значит i и j совместно встречаются реже, чем если бы были независимы; однако такая отрицательная корреляция может оказаться ненадежной, поэтому обычно используют **положительную PMI**: $\text{PPMI}(i, j) = \max(\text{PMI}(i, j), 0)$. В работе [BL07] показано, что сингулярное разложение матрицы PPMI дает погружения слов, которые хорошо работают во многих задачах, связанных со смыслом слов. Теоретическая модель, объясняющая это эмпирически наблюдаемое качество, описана в разделе 20.5.5.

20.5.2. Word2vec

В этом разделе мы обсудим популярную модель **word2vec** из работ [Mik+13a; Mik+13b]. Это «мелкая» нейронная сеть для предсказания слова по его контексту. В разделе 20.5.5 мы обсудим связи между сингулярным разложением и матрицей PMI.

Существует две версии модели word2vec. Первая называется CBOW (continuous bag of words – непрерывный мешок слов), вторая – скипграммы. Ниже обсуждаются обе.

20.5.2.1. Модель Word2vec CBOW

В модели непрерывного мешка слов (**CBOW**) (см. рис. 20.44a) логарифмическое правдоподобии последовательности слов вычисляется с применением следующей модели:

$$\log p(\mathbf{w}) = \sum_{t=1}^T \log p(w_t | \mathbf{w}_{t-m:t+m}) = \sum_{t=1}^T \log \frac{\exp(\mathbf{v}_{w_t}^\top \bar{\mathbf{v}}_t)}{\sum_{w' \in \mathcal{V}} \exp(\mathbf{v}_{w'}^\top \bar{\mathbf{v}}_t)} \quad (20.143)$$

$$= \sum_{t=1}^T \mathbf{v}_{w_t}^\top \bar{\mathbf{v}}_t - \log \sum_{i \in \mathcal{V}} \exp(\mathbf{v}_i^\top \bar{\mathbf{v}}_t), \quad (20.144)$$

где \mathbf{v}_{w_t} – вектор для слова в позиции w_t , \mathcal{V} – множество всех слов, m – размер контекста и

$$\bar{\mathbf{v}}_t = \frac{1}{2m} \sum_{h=1}^m (\mathbf{v}_{w_{t+h}} + \mathbf{v}_{w_{t-h}}) \quad (20.145)$$

– среднее векторов слов в окне, окружающем слово w_t . Таким образом, мы пытаемся предсказать каждое слово по его контексту. Модель называется CBOW, потому что в ней используется предположение о мешке слов для контекста, а каждое слово представляется непрерывным погружением.

20.5.2.2. Скипграммная модель Word2vec

В модели CBOW каждое слово предсказывается по его контексту. Вариант этой идеи – предсказывать контекст (окружающие слова) по слову. Получается такая целевая функция:

$$-\log p(\mathbf{w}) = -\sum_{t=1}^T \left[\sum_{j=1}^m \log p(w_{t-j} | w_t) + \log p(w_{t+j} | w_t) \right] \quad (20.146)$$

$$= -\sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j} | w_t), \quad (20.147)$$

где m – длина контекстного окна. Мы определяем логарифмическую вероятность другого контекстного слова w_o при условии центрального слова w_c как

$$\log p(w_o|w_c) = \mathbf{u}_o^T \mathbf{v}_c - \log \left(\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^T \mathbf{v}_c) \right), \quad (20.148)$$

где \mathcal{V} – словарь. Здесь \mathbf{u}_i – погружение слова, если оно используется в качестве контекста, а \mathbf{v}_i – погружение слова, если оно используется в качестве центрального (целевого) слова, подлежащего предсказанию. Это **скипграммная модель** (см. иллюстрацию на рис. 20.44b).

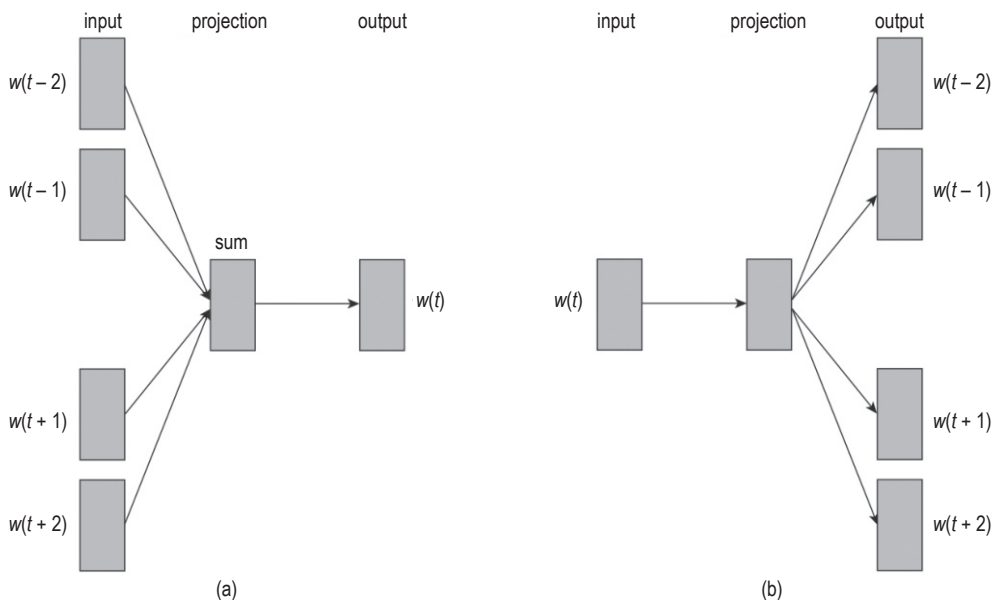


Рис. 20.44 ❖ Модель word2vec с окном размера 2.
(a) Версия CBOW. (b) Скипграммная версия

20.5.2.3. Отрицательная выборка

Вычисление условной вероятности каждого слова по формуле (20.148) обходится дорого из-за необходимости производить нормировку во всем возможным словам в словаре. Поэтому вычислять логарифмическое правдоподобие и его градиент медленно в обеих моделях, CBOW и скипграммной.

В работе [Mik+13b] предложена быстрая аппроксимация, названная **скипграммами с отрицательной выборкой** (skip-gram with negative sampling – **SGNS**). Основная идея – создать набор $K + 1$ контекстных слов для каждого центрального слова w_t и пометить то, которое фактически встречается как положительное, а все остальные как отрицательные. Отрицательные слова называются пропускаемыми, их можно выбрать из распределения униграмм с пересчитанными весами, $p(w) \propto \text{freq}(w)^{3/4}$, смысл которого – перераспределить массу вероятности от частых слов к редким. Тогда условная вероятность аппроксимируется следующим образом:

$$p(w_{t+j}|w_t) = p(D = 1|w_t, w_{t+j}) \prod_{k=1}^K p(D = 0|w_t, w_k), \quad (20.149)$$

где $w_k \sim p(w)$ пропускаемые слова, $D = 1$ – событие, заключающееся в том, что пара слов действительно встречается в данных, а $D = 0$ – событие, заключающееся в том, что пара слов не встречается. Бинарные вероятности равны

$$p(D = 1|w_t, w_{t+j}) = \sigma(\mathbf{u}_{w_{t+j}}^\top \mathbf{v}_{w_t}); \quad (20.150)$$

$$p(D = 0|w_t, w_k) = 1 - \sigma(\mathbf{u}_{w_k}^\top \mathbf{v}_{w_t}). \quad (20.151)$$

Для обучения этой модели нужно только вычислить контексты для каждого центрального слова и множество пропускаемых слов. С контекстными словами ассоциируется метка 1, а с пропускаемыми – метка 0. Затем можно вычислить логарифмическую вероятность данных и оптимизировать векторы погружений \mathbf{u}_i и \mathbf{v}_i для каждого слова, применив метод СГС. Демонстрационный код см. по адресу code.problml.ai/book1/skipgram_torch.

20.5.3. GloVe

Популярную альтернативу скипграммной модели составляет модель **GloVe** из работы [PSM14a]. (GloVe означает «global vectors for word representation» – глобальные векторы для представления слов.) В этом методе используется более простая целевая функция, оптимизировать которую гораздо быстрее.

Напомним, что в скипграммной модели предсказанная условная вероятность того, что слово j встречается в контекстном окне центрального слова i , равна

$$q_{ij} = \frac{\exp(\mathbf{u}_j^\top \mathbf{v}_i)}{\sum_{k \in \mathcal{V}} \exp(\mathbf{u}_k^\top \mathbf{v}_i)}. \quad (20.152)$$

Пусть x_{ij} – количество вхождений слова j в какое-нибудь контекстное окно слова i . (Заметим, что если слово i встречается в окне слова j , то j встречается в окне i , так что $x_{ij} = x_{ji}$.) Тогда формулу (20.147) можно переписать в виде

$$\mathcal{L} = - \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} x_{ij} \log q_{ij}. \quad (20.153)$$

Если положить $p_{ij} = x_{ij}/x_i$ – эмпирическая вероятность вхождения слова j в контекстное окно центрального слова i , то скипграммную потерю можно переписать как потерю перекрестной энтропии:

$$\mathcal{L} = - \sum_{i \in \mathcal{V}} x_i \sum_{j \in \mathcal{V}} p_{ij} \log q_{ij}. \quad (20.154)$$

Проблема этой целевой функции в том, что вычисление q_{ij} стоит дорого из-за необходимости нормировки по всем словам. В GloVe мы работаем

с ненормированными вероятностями, $p'_{ij} = x_{ij}$ и $q'_{ij} = \exp(\mathbf{u}_j^\top \mathbf{v}_i + b_i + c_j)$, где b_i и c_j – члены смещения, улавливающие маргинальные вероятности. Кроме того, мы минимизируем квадратичную потерю, $(\log p'_{ij} - \log q'_{ij})^2$, которая более робастна к ошибкам в оценке малых вероятностей, чем логарифмическая потеря. Наконец, мы увеличиваем веса редких слов, для которых $x_{ij} < c$, где $c = 100$, умножая квадратичные ошибки на $h(x_{ij})$, где $h(x) = (x/c)^{0.75}$, если $x < c$, и $h(x) = 1$ в противном случае. В итоге получаем такую целевую функцию GloVe:

$$\mathcal{L} = -\sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} h(x_{ij})(\mathbf{u}_j^\top \mathbf{v}_i + b_i + c_j - \log x_{ij})^2. \quad (20.155)$$

Пересчитывать x_{ij} можно в офлайн-режиме, а затем оптимизировать эту целевую функцию методом СГС. После обучения мы определяем погружение слова i как среднее \mathbf{v}_i и \mathbf{u}_i .

Эмпирически показано, что GloVe дает примерно такие же результаты, как скипграммная модель, но обучается быстрее. Теоретическое обоснование обоих методов см. в разделе 20.5.5.

20.5.4. Аналогичные слова

Одно из самых примечательных свойств погружений слов, порождаемых моделями word2vec, GloVe и другими подобными методами, – тот факт, что обученное векторное пространство, похоже, улавливает реляционную семантику в терминах простого сложения векторов. Например, рассмотрим **проблему аналогичных слов** «man is to woman as king is to queen»¹, которую часто записывают в виде man:woman::king:queen. Предположим, что даны слова $a=\text{man}$, $b=\text{woman}$, $c=\text{king}$; как найти $d=\text{queen}$? Обозначим $\delta = \mathbf{v}_b - \mathbf{v}_a$ вектор, представляющий концепцию «преобразовать мужской род в женский». Интуитивно кажется, что можно найти слово d , вычислив $\mathbf{v}_d = \mathbf{c} + \delta$, а затем взяв ближайшее к \mathbf{v}_d слово в словаре (см. иллюстрацию этого процесса на рис. 20.45, а код по адресу code.probl.ai/book1/word_analogies_torch).

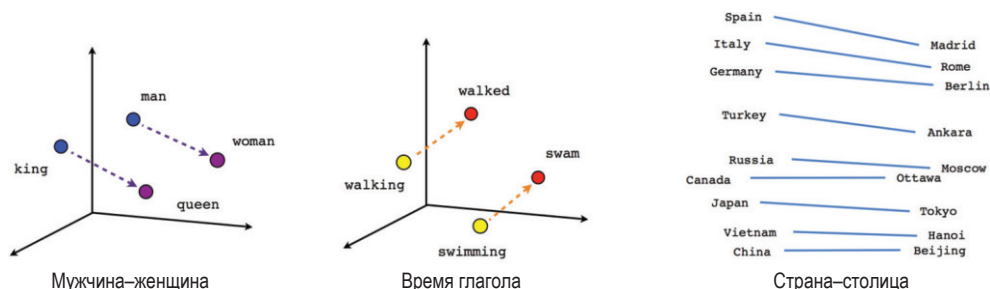


Рис. 20.45 ❖ Визуализация арифметических операций с пространстве погружения word2vec. Из работы <https://www.tensorflow.org/tutorials/representation/word2vec>

¹ Мужчина относится к женщине как король относится к королеве. – Прим. перев.

В работе [PSM14a] высказано предположение, что отношение $a : b :: c : d$ справедливо тогда и только тогда, когда для любого слова w в словаре имеем

$$\frac{p(w|a)}{p(w|b)} \approx \frac{p(w|c)}{p(w|d)}. \quad (20.156)$$

В работе [Aro+16] показано, что это следует из предположений модели RAND-WALK, описанных в разделе 20.5.5 (см. также работы [АН19; EDH19], где даются объяснения работы механизма аналогичных слов, основанные на других допущениях).

20.5.5. Модель погружений слов RAND-WALK

Погружения слов значительно улучшают качество различных моделей NLP по сравнению с унитарными кодировками слов. Естественно спросить, почему они так хорошо работают. В этом разделе мы опишем простую порождающую модель текстовых документов, объясняющую этот феномен. Изложение основано на работе [Aro+16].

Рассмотрим последовательность слов w_1, \dots, w_T . Предполагается, что каждое слово порождено латентным контекстом, или вектором дискурса $\mathbf{z}_t \in \mathbb{R}^D$ с использованием следующей **логарифмически-билинейной языковой модели**, аналогичной той, что приведена в работе [МН07]:

$$p(w_t = w | \mathbf{z}_t) = \frac{\exp(\mathbf{z}_t^\top \mathbf{v}_w)}{\sum_{w'} \exp(\mathbf{z}_t^\top \mathbf{v}_{w'})} = \frac{\exp(\mathbf{z}_t^\top \mathbf{v}_w)}{Z(\mathbf{z}_t)}, \quad (20.157)$$

где $\mathbf{v}_w \in \mathbb{R}^D$ – погружение слова w , а $Z(\mathbf{z}_t)$ – функция разбиения. Мы предполагаем, что $D < M$, числа слов в словаре.

Предположим далее, что погружения слов \mathbf{v}_w имеют изотропное гауссово априорное распределение и что латентная тема \mathbf{z}_t совершает медленное случайное гауссово блуждание (именно поэтому модель называется **RAND-WALK**). В рамках этой модели можно показать, что $Z(\mathbf{z}_t)$ приближенно равно постоянной Z , не зависящей от контекста. Это так называемое **свойство самонормировки** логарифмически-линейных моделей [АК15]. Кроме того, можно показать, что поточечная взаимная информация предсказания модели имеет вид:

$$\mathbb{P}\text{MI}(w, w') = \frac{p(w, w')}{p(w)p(w')} \approx \frac{\mathbf{v}_w^\top \mathbf{v}_{w'}}{D}. \quad (20.158)$$

Таким образом, мы можем обучить модель RAND-WALK, стараясь приравнять предсказанные значения PMI эмпирическим, т. е. минимизировать функцию:

$$\mathcal{L} = \sum_{w, w'} X_{w, w'} (\mathbb{P}\text{MI}(w, w') - \mathbf{v}_w^\top \mathbf{v}_{w'})^2, \quad (20.159)$$

где $X_{w,w'}$ – сколько раз w и w' встречаются рядом. Эту целевую функцию можно рассматривать как взвешенную по частоте потерю сингулярного разложения (20.138). (О других связях между погружениями слов и сингулярным разложением см. [LG14].)

Можно использовать дополнительные аппроксимации, чтобы показать, что отрицательное логарифмическое правдоподобие RAND-WALK эквивалентно целевым функциям вариантов модели word2vec (CBOW и скипграммного). Из этого подхода можно также вывести целевую функцию для GloVE.

20.5.6. Контекстуальные погружения слов

Рассмотрим предложения «I was eating an apple»¹ и «I bought a new phone from Apple»². Слово «apple» имеет разный смысл, но его погружение из числа обсуждавшихся в разделе 20.5 фиксировано и не может уловить этого различия. В разделе 15.7 мы обсуждали **контекстуальные погружения**, когда погружение слова является функцией от всех слов в его контексте (обычно в предложении). Это позволяет значительно улучшить результаты и в настоящее время является стандартным подходом к представлению данных естественного языка, а также шагом предобработки, выполняемым перед переносом обучения (см. раздел 19.2).

20.6. УПРАЖНЕНИЯ

Упражнение 20.1 [ЕМ-алгоритм для факторного анализа].

Выведите обновления в ЕМ-алгоритме для модели факторного анализа. Для простоты можете предположить, что $\mu = \mathbf{0}$ фиксировано.

Упражнение 20.2 [ЕМ-алгоритм для смеси факторных анализаторов*].

Выведите обновления в ЕМ-алгоритме для смеси факторных анализаторов.

Упражнение 20.3 [вывод второй главной компоненты].

а. Пусть

$$J(\mathbf{v}_2, \mathbf{z}_2) = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - z_{i1}\mathbf{v}_1 - z_{i2}\mathbf{v}_2)^T (\mathbf{x}_i - z_{i1}\mathbf{v}_1 - z_{i2}\mathbf{v}_2). \quad (20.160)$$

Покажите, что из $\partial J / \partial \mathbf{z}_2 = \mathbf{0}$ следует, что $z_{i2} = \mathbf{v}_2^T \mathbf{x}_i$.

б. Покажите, что значение \mathbf{v}_2 , доставляющее минимум функции

$$\tilde{J}(\mathbf{v}_2) = -\mathbf{v}_2^T \mathbf{C} \mathbf{v}_2 + \lambda_2 (\mathbf{v}_2^T \mathbf{v}_2 - 1) + \lambda_{12} (\mathbf{v}_2^T \mathbf{v}_1 - 0), \quad (20.161)$$

равно собственному вектору \mathbf{C} , соответствующему второму по величине собственному значению. *Указание:* вспомните, что $\mathbf{C} \mathbf{v}_1 = \lambda_1 \mathbf{v}_1$ и $\partial \mathbf{x}^T \mathbf{A} \mathbf{x} / \partial \mathbf{x} = (\mathbf{A} + \mathbf{A}^T) \mathbf{x}$.

¹ Я ел яблоко. – Прим. перев.

² Я купил новый телефон Apple. – Прим. перев.

Упражнение 20.4 [вывод невязки для PCA*].

а. Докажите, что

$$\left\| \mathbf{x}_i - \sum_{j=1}^K z_{ij} \mathbf{v}_j \right\|^2 = \mathbf{x}_i^T \mathbf{x}_i - \sum_{j=1}^K \mathbf{v}_j^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{v}_j. \quad (20.162)$$

Указание: сначала рассмотрите случай $K = 2$. Используйте тот факт, что $\mathbf{v}_j^T \mathbf{v}_j = 1$ и $\mathbf{v}_j^T \mathbf{v}_k = 0$ для $k \neq j$. Вспомните также, что $z_{ij} = \mathbf{x}_i^T \mathbf{v}_j$.

б. Теперь покажите, что

$$J_K \triangleq \frac{1}{n} \sum_{i=1}^n \left(\mathbf{x}_i^T \mathbf{x}_i - \sum_{j=1}^K \mathbf{v}_j^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{v}_j \right) = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^T \mathbf{x}_i - \sum_{j=1}^K \lambda_j. \quad (20.163)$$

Указание: вспомните, что $\mathbf{v}_j^T \mathbf{C} \mathbf{v}_j = \lambda_j \mathbf{v}_j^T \mathbf{v}_j = \lambda_j$.

с. Если $K = d$, то нет никакого усечения, так что $J_d = 0$. Воспользуйтесь этим, чтобы показать, что ошибка, связанная только с использованием $K < d$ членов, имеет вид:

$$J_K = \sum_{j=K+1}^d \lambda_j. \quad (20.164)$$

Указание: разбейте сумму $\sum_{j=1}^d \lambda_j$ на две части: $\sum_{j=1}^K \lambda_j$ и $\sum_{j=K+1}^d \lambda_j$.

Упражнение 20.5 [PCA посредством последовательного понижения порядка].

Пусть $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ – первые k собственных векторов с наибольшими собственными значениями матрицы $\mathbf{C} = (1/n) \mathbf{X}^T \mathbf{X}$, т. е. главные базисные векторы. Для них имеет место равенство:

$$\mathbf{v}_j^T \mathbf{v}_k = \begin{cases} 0, & \text{если } j \neq k \\ 1, & \text{если } j = k \end{cases}. \quad (20.165)$$

Мы опишем метод последовательного нахождения \mathbf{v}_j .

Как было показано в основном тексте, \mathbf{v}_1 является первым главным собственным вектором \mathbf{C} и для него имеет место равенство $\mathbf{C} \mathbf{v}_1 = \lambda_1 \mathbf{v}_1$. Теперь определим $\tilde{\mathbf{x}}_i$ как ортогональную проекцию \mathbf{x}_i на пространство, ортогональное \mathbf{v}_1 :

$$\tilde{\mathbf{x}}_i = \mathbf{P}_{\perp \mathbf{v}_1} \mathbf{x}_i = (\mathbf{I} - \mathbf{v}_1 \mathbf{v}_1^T) \mathbf{x}_i. \quad (20.166)$$

Определим матрицу **пониженного порядка** $d - 1$ $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_n]$, получающуюся в результате удаления из d -мерных данных компоненты, лежащей в направлении первого главного собственного вектора:

$$\tilde{\mathbf{X}} = (\mathbf{I} - \mathbf{v}_1 \mathbf{v}_1^T)^T \mathbf{X} = (\mathbf{I} - \mathbf{v}_1 \mathbf{v}_1^T) \mathbf{X}. \quad (20.167)$$

а. Пользуясь тем, что $\mathbf{X}^T \mathbf{X} \mathbf{v}_1 = n \lambda_1 \mathbf{v}_1$ (и, следовательно, $\mathbf{v}_1^T \mathbf{X}^T \mathbf{X} = n \lambda_1 \mathbf{v}_1^T$) и $\mathbf{v}_1^T \mathbf{v}_1 = 1$, покажите, что ковариация матрицы пониженного порядка равна

$$\tilde{\mathbf{C}} \triangleq \frac{1}{n} \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} = \frac{1}{n} \mathbf{X}^T \mathbf{X} - \lambda_1 \mathbf{v}_1 \mathbf{v}_1^T. \quad (20.168)$$

- б. Пусть \mathbf{u} – главный собственный вектор $\tilde{\mathbf{C}}$. Объясните, почему $\mathbf{u} = \mathbf{v}_2$. (Можете предполагать, что \mathbf{u} имеет единичную норму.)
- с. Предположим, что имеется простой метод нахождения первого собственного вектора и собственного значения положительно определенной матрицы, который обозначим $[\lambda, \mathbf{u}] = f(\mathbf{C})$. Напишите псевдокод нахождения первых K главных базисных векторов \mathbf{X} , в котором должна использоваться только функция f и простые арифметические операции с векторами, но не должны использоваться сингулярное разложение или функция `eig`. *Указание:* это должна быть простая итеративная процедура, занимающая 2–3 строки. На вход подаются \mathbf{C} , K и функция f , а на выходе должно быть \mathbf{v}_j и λ_j для $j = 1 \dots K$.

Упражнение 20.6 [члены дисперсии PPCA].

Напомним, что в модели PPCA $\mathbf{C} = \mathbf{W}\mathbf{W}^T + \sigma^2 \mathbf{I}$. Мы покажем, что эта модель правильно улавливает дисперсию данных вдоль главных осей и аппроксимирует дисперсию во всех остальных направлениях одним и тем же средним значением σ^2 .

Рассмотрим дисперсию прогнозного распределения $p(\mathbf{x})$ вдоль некоторого направления, заданного единичным вектором \mathbf{v} , где $\mathbf{v}^T \mathbf{v} = 1$, которая равна $\mathbf{v}^T \mathbf{C} \mathbf{v}$.

- а. Сначала предположим, что \mathbf{v} ортогонален главному подпространству и, следовательно, $\mathbf{v}^T \mathbf{U} = \mathbf{0}$. Покажите, что $\mathbf{v}^T \mathbf{C} \mathbf{v} = \sigma^2$.
- б. Теперь предположим, что \mathbf{v} параллелен главному подпространству и, следовательно, $\mathbf{v} = \mathbf{u}_i$ для некоторого собственного вектора \mathbf{u}_i . Покажите, что $\mathbf{v}^T \mathbf{C} \mathbf{v} = (\lambda_i - \sigma^2) + \sigma^2 = \lambda_i$.

Упражнение 20.7 [апостериорный вывод в PPCA*].

Выведите $p(\mathbf{z}_n | \mathbf{x}_n)$ для модели PPCA.

Упражнение 20.8 [подстановка в модель ФА*].

Выведите выражение для $p(\mathbf{x}_h | \mathbf{x}_v, \boldsymbol{\theta})$ для модели ФА, где $\mathbf{x} = (\mathbf{x}_h, \mathbf{x}_v)$ разбиение вектора данных.

Упражнение 20.9 [эффективное вычисление плотности PPCA].

Выведите выражение для $p(\mathbf{x} | \hat{\mathbf{W}}, \sigma^2)$ в модели PPCA, основываясь на подстановке в оценки MLE и используя лемму об обращении матрицы.

Глава 21

Кластеризация

21.1. ВВЕДЕНИЕ

Кластеризация – широко распространенная форма обучения без учителя. Есть два основных подхода. В первом случае входом является набор примеров данных $\mathcal{D} = \{\mathbf{x}_n : n = 1 \dots N\}$, где $\mathbf{x}_n \in \mathcal{X}$, а \mathcal{X} , как правило, совпадает с \mathbb{R}^D . Во втором случае на вход подается $N \times N$ попарных метрик несходства $D_{ij} \geq 0$. В обоих случаях цель – отнести похожие примеры к одному и тому же кластеру.

Как часто бывает при обучении без учителя, оценить качество алгоритма кластеризации трудно. Если мы поместили какие-то примеры, то можем воспользоваться сходством (или совпадением) меток как метрикой при решении вопроса о том, следует ли относить примеры к одному кластеру или нет. Если меток нет, но метод основан на порождающей модели данных, то в качестве метрики можно использовать логарифмическое правдоподобие. Примеры обоих подходов будут приведены ниже.

21.1.1. Оценивание выхода методов кластеризации

Проверка кластерных структур – самая трудная и обескураживающая часть кластерного анализа. Без целенаправленных усилий в этом направлении кластерный анализ остался бы черной магией, доступной истинным поборникам, обладающим опытом и незаурядным мужеством.

— Джейн и Дьюбс [JD88]

Кластеризация – это способ обучения без учителя, поэтому оценить выход любого метода проблематично [Kle02; LWG12]. При использовании вероятностных моделей мы всегда можем оценить правдоподобие данных, но есть два недостатка: во-первых, так нельзя непосредственно оценить кластеризацию, выявленную моделью, а во-вторых, такой способ неприменим к невероятностным методам. Поэтому мы обсудим некоторые меры качества, не основанные на правдоподобии.

Интуитивно понятно, что цель кластеризации – поместить похожие точки в один и тот же кластер, причем так, чтобы непохожие точки оказались в разных кластерах. Есть несколько способов измерения этих величин; см., например, [JD88; KR90]. Однако полезность этих внутренних критериев ограничена. Альтернативный способ – положиться на какую-то внешнюю форму данных, с помощью которой можно проверить качество метода. Например, если с каждым объектом ассоциирована метка, то можно считать, что объекты с одинаковой меткой похожи. А затем использовать описываемые ниже метки для количественной оценки качества кластеров. (Если меток нет, но имеется эталонная кластеризация, то можно вывести метки из нее.)

21.1.1.1. Чистота

Пусть N_{ij} – количество объектов в кластере i , принадлежащих классу j , и пусть $N_i = \sum_{j=1}^C N_{ij}$ – общее число объектов в кластере i . Определим $p_{ij} = N_{ij}/N_i$; это эмпирическое распределение меток классов для кластера i . Определим **чистоту** кластера как $p_i \triangleq \max_j p_{ij}$, а полную чистоту кластеризации как

$$\text{purity} \triangleq \sum_i \frac{N_i}{N} p_i. \quad (21.1)$$

Так, на рис. 21.1 чистота равна

$$\frac{6}{17} \frac{5}{6} + \frac{6}{17} \frac{4}{6} + \frac{5}{17} \frac{3}{5} = \frac{5 + 4 + 3}{17} = 0.71. \quad (21.2)$$

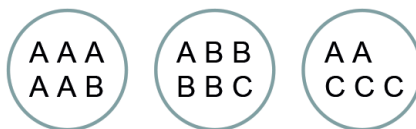


Рис. 21.1 ❖ Три кластера, содержащие помеченные объекты

Чистота принимает значения в диапазоне от 0 (плохая) до 1 (хорошая). Но мы можем тривиально добиться чистоты 1, поместив каждый объект в отдельный кластер, т. е. эта мера не штрафует за количество кластеров.

21.1.1.2. Индекс Рэнда

Пусть $U = \{u_1, \dots, u_R\}$ и $V = \{v_1, \dots, v_C\}$ – два разбиения множества N точек. Например, U может быть оценочной кластеризацией, а V – эталонной кластеризацией, построенной на основе меток классов. Теперь определим таблицу сопряженности 2×2 , содержащую следующие числа: TP – количество пар, оказавшихся в одном кластере и в U , и в V (истинно положительные результаты); TN – количество пар, оказавшихся в разных кластерах в U и в V (истинно отрицательные результаты); FN – количество пар, оказавшихся в разных кластерах в U , но в одном кластере в V (ложноотрицательные результаты); FP – количество пар, оказавшихся в одном кластере в U , но в разных класте-

рах в V (ложноположительные результаты). В качестве сводной статистики часто применяется **индекс Рэнда**:

$$R \triangleq \frac{TP + TN}{TP + FP + FN + TN}. \quad (21.3)$$

Это можно интерпретировать как долю правильных решений о кластеризации. Очевидно, что $0 \leq R \leq 1$.

Например, рассмотрим рис. 21.1. Три кластера содержат соответственно 6, 6 и 5 точек, поэтому количество «положительных» результатов (когда пары объектов оказываются в одном и том же кластере вне зависимости от метки), равно

$$TP + FP = \binom{6}{2} + \binom{6}{2} + \binom{5}{2} = 40. \quad (21.4)$$

Из них количество истинно положительных результатов равно

$$TP = \binom{5}{2} + \binom{4}{2} + \binom{3}{2} + \binom{2}{2} = 20, \quad (21.5)$$

где последние два члена взяты из кластера 3: имеется $\binom{3}{2}$ пар с меткой C и $\binom{2}{2}$ пар с меткой A . Таким образом, $FP = 40 - 20 = 20$. Аналогично можно показать, что $FN = 24$ и $TN = 72$. Следовательно, индекс Рэнда равен $(20 + 72)/(20 + 20 + 24 + 72) = 0.68$.

Индекс Рэнда достигает нижней границы 0, только если $TP = TN = 0$, что бывает редко. Можно определить **скорректированный индекс Рэнда** [HA85] следующим образом:

$$AR \triangleq \frac{\text{индекс} - \text{ожидаемый индекс}}{\text{макс. индекс} - \text{ожидаемый индекс}}. \quad (21.6)$$

Здесь модель случайности основана на обобщенном гипергеометрическом распределении, т. е. оба разбиения формируются случайным образом с соблюдением условия, что в каждом присутствует оригинальное число классов и объектов, а затем вычисляется ожидаемое значение $TP + TN$. Эту модель можно использовать для вычисления статистической значимости индекса Рэнда.

Индекс Рэнда сопоставляет ложноположительным и ложноотрицательным результатам одинаковые веса. Можно использовать и другие сводные статистики для задач принятия альтернативных решений, например F -меру (раздел 5.1.4).

21.1.1.3. Взаимная информация

Еще один способ измерить качество кластера – вычислить взаимную информацию между двумя потенциальными разбиениями U и V , как предложено

в работе [VD99]. Для этого обозначим $p_{UV}(i, j) = |u_i \cap v_j|/N$ вероятность того, что случайно выбранный объект принадлежит кластеру u_i в U и кластеру v_j в V . Кроме того, обозначим $p_U(i) = |u_i|/N$ вероятность того, что случайно выбранный объект принадлежит кластеру u_i в U ; аналогично определим $p_V(j) = |v_j|/N$. Тогда имеем

$$\mathbb{I}(U, V) = \sum_{i=1}^R \sum_{j=1}^C p_{UV}(i, j) \log \frac{p_{UV}(i, j)}{p_U(i)p_V(j)}. \quad (21.7)$$

Эта величина принимает значения от 0 до $\min\{\mathbb{H}(U), \mathbb{H}(V)\}$. К сожалению, максимального значения можно достичь, взяв много маленьких кластеров, имеющих низкую энтропию. Чтобы компенсировать этот недостаток, можно использовать **нормированную взаимную информацию**:

$$NMI(U, V) \triangleq \frac{\mathbb{I}(U, V)}{(\mathbb{H}(U) + \mathbb{H}(V))/2}. \quad (21.8)$$

Эта величина заключена между 0 и 1. Ее вариант, скорректированный с учетом случайности (при определенной модели случайных данных), описан в работе [VEB09]. Еще один вариант, называемый **вариацией информации**, описан в работе [Mei05].

21.2. ИЕРАРХИЧЕСКАЯ АГЛОМЕРАТИВНАЯ КЛАСТЕРИЗАЦИЯ

Распространенной формой кластеризации является **иерархическая агломеративная кластеризация** (hierarchical agglomerative clustering – **НАС**). На вход алгоритма подается матрица различий $N \times N$ с элементами $D_{ij} \geq 0$, а выходом является древовидная структура, в которой мало различающиеся группы i и j объединяются в группу следующего уровня.

Например, рассмотрим множество из пяти входных точек на рис. 21.2а, $\mathbf{x}_n \in \mathbb{R}^2$. Для определения несходства точек будем использовать **манхэттенское расстояние**:

$$d_{ij} = \sum_{k=1}^2 |x_{ik} - x_{jk}|. \quad (21.9)$$

Начнем с дерева с N листьями, каждый из которых соответствует кластеру, содержащему всего одну точку. Далее вычисляем пару ближайших точек и объединяем их. Мы видим, что точки (1,3) и (4,5) находятся на расстоянии 1 друг от друга, поэтому они объединяются первыми. Затем мы измеряем различие между множествами {1, 3}, {4, 5} и {2}, применяя ту же меру (см. детали ниже), группируем их и повторяем. Получающееся в результате двоичное дерево называется **дендрограммой**, оно показано на рис. 21.2b. Об-

резая это дерево на различной высоте, мы можем создать разное количество вложенных кластеров.

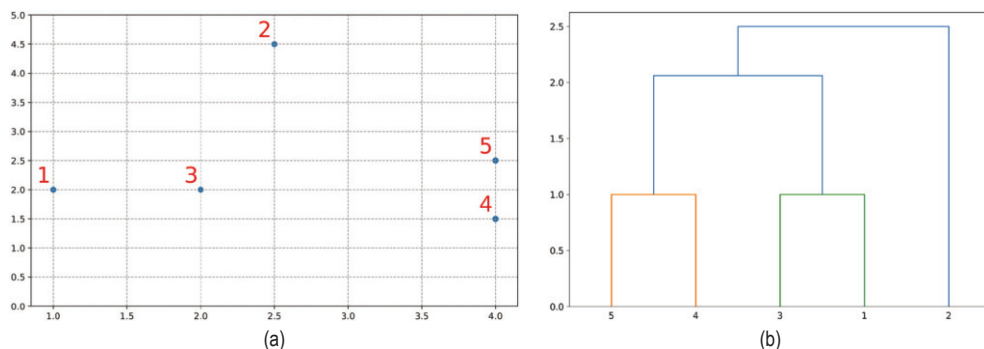


Рис. 21.2 ❖ (a) Пример кластеризации с одиночной связью с использованием манхэттенского расстояния. Элементы обеих пар (1,3) и (4,5) находятся на расстоянии 1 друг от друга, поэтому они объединяются первыми. (b) Результирующая дендрограмма. На основе рис 7.5 из работы [Alp04]. Построено программой по адресу figures.problml.ai/book1/21.2

21.2.1. Алгоритм

Агломеративная кластеризация начинается с образования N групп, каждая из которых содержит один объект. Затем на каждом шаге две самые похожие группы объединяются, и так до тех пор, пока не останется одна группа, содержащая все данные (см. псевдокод в алгоритме 11). Поскольку нахождение двух самых похожих кластеров занимает время $O(N^2)$, а всего шагов в алгоритме $O(N)$, полное время работы составляет $O(N^3)$. Но если использовать очередь с приоритетами, то это время можно уменьшить до $O(N^2 \log N)$ (детали см., например, в книге [MRS08, глава 17]).

Алгоритм 11. Агломеративная кластеризация

```

1  Инициализировать одиночные кластеры: for  $i \leftarrow 1$  to  $n$  do  $C_i \leftarrow \{i\}$ ;
2  ;
3  Инициализировать множество кластеров, доступных для объединения:
    $S \leftarrow \{1, \dots, ng\}$ , repeat
4  |   Выбрать два самых похожих кластера:  $(j, k) \leftarrow \operatorname{argmin}_{j,k \in S} d_{j,k}$ ;
5  |   Создать новый кластер  $C_l \leftarrow C_j \cup C_k$ ;
6  |   Пометить  $j$  и  $k$  как недоступные:  $S \leftarrow S \setminus \{j, k\}$ ;
7  |   if  $C_l \neq \{1, \dots, n\}$  then
8  |   |   Пометить  $l$  как доступный,  $S \leftarrow S \cup \{l\}$ ;
9  |   foreach  $i \in S$  do
10 |   |   Обновить матрицу различий  $d(i, l)$ ;
11 until не осталось кластеров, доступных для объединения;
```

На самом деле существует три варианта агломеративной кластеризации, в зависимости от того, как определяется несходство групп объектов. Детали приведены ниже.

21.2.1.1. Одиночная связь

При кластеризации с **одиночной связью**, называемой также **кластеризацией по ближайшему соседу**, расстояние между двумя группами G и H определяется как расстояние между двумя ближайшими представителями этих групп:

$$d_{SL}(G, H) = \min_{i \in G, i' \in H} d_{i,i'}. \quad (21.10)$$

Смотрите рис. 21.3а.

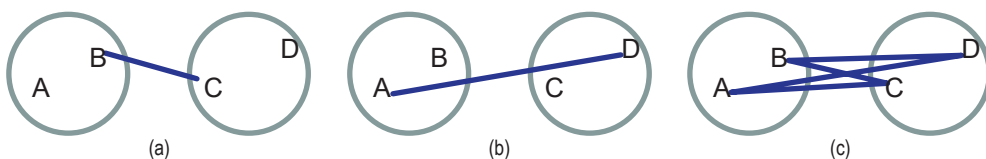


Рис. 21.3 ❖ (а) Одиночная связь. (б) Полная связь. (с) Средняя связь

Дерево, построенное в результате кластеризации с одиночной связью, – это минимальное остовное дерево данных, в котором все объекты соединены таким образом, что сумма весов ребер (расстояний) минимальна. Чтобы убедиться в этом, заметим, что при объединении двух кластеров мы соединяем их ближайших представителей; при этом добавляется ребро между соответствующими вершинами и гарантируется, что из всех ребер, соединяющих два кластера, оно имеет наименьший вес. А после того как два кластера объединены, они уже никогда не рассматриваются, поэтому циклов мы не создадим. Следовательно, для реализации кластеризации с одиночной связью, фактически нужно время $O(N^2)$, тогда как для других вариантов – время $O(N^3)$.

21.2.1.2. Полная связь

При кластеризации с **полной связью**, называемой также **кластеризацией по самому дальнему соседу**, расстояние между двумя группами определяется как расстояние между двумя самыми удаленными друг от друга представителями:

$$d_{CL}(G, H) = \max_{i \in G, i' \in H} d_{i,i'}. \quad (21.11)$$

Смотрите рис. 21.3б.

В случае одиночной связи для того чтобы две группы считались близкими, требуется лишь, чтобы была близка одна пара объектов, а похожи ли все остальные, неважно. Поэтому могут быть образованы кластеры, не обла-

дающие свойством **компактности**, которое означает, что все наблюдения в одной группе должны быть похожи. Именно, если определить **диаметр** группы как наибольшее несходство ее членов, $d_G = \max_{i \in G, i' \in G} d_{i,i'}$, то можно видеть, что одиночная связь может порождать кластеры с большим диаметром. Полная связь – это другая крайность: две группы считаются близкими, если относительно похожи все наблюдения в их объединении. Такая связь порождает кластеры с малым диаметром, т. е. компактные (ср. рис. 21.4а и б).

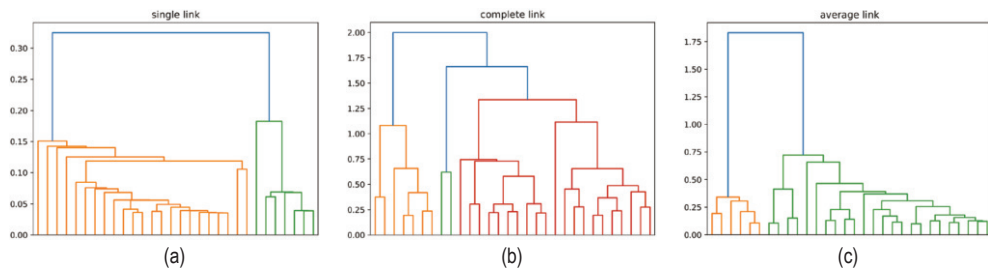


Рис. 21.4 ❖ Иерархическая кластеризация данных об экспрессии гена дрожжей. (а) Одиночная связь. (б) Полная связь. (с) Средняя связь. Построено программой по адресу figures.problml.ai/book1/21.4

21.2.1.3. Средняя связь

На практике предпочтительной является **кластеризация со средней связью**, при которой измеряется среднее расстояние между всеми парами:

$$d_{avg}(G, H) = \frac{1}{n_G n_H} \sum_{i \in G} \sum_{i' \in H} d_{i,i'}. \quad (21.12)$$

где n_G и n_H – число элементов в группах G и H (см. рис. 21.3с).

Кластеризация со средней связью – компромисс между одиночной и полной связью. Она порождает относительно компактные кластеры, расположенные достаточно далеко друг от друга (см. рис. 21.4с). Но, поскольку требуется вычислять среднее расстояний $d_{i,i'}$, любое изменение масштаба измерений может отразиться на результате. Напротив, одиночная и полная связь инвариантны относительно монотонных преобразований $d_{i,i'}$, так как не изменяют относительный порядок.

21.2.2. Пример

Пусть имеется временной ряд измерений уровня экспрессии для $N = 300$ генов в $T = 7$ точках. Это значит, что каждый пример является вектором $\mathbf{x}_n \in \mathbb{R}^7$ (см. визуализацию данных на рис. 21.5). Как видим, имеется несколько видов генов, в том числе такие, для которых уровень экспрессии монотонно возрастает (в ответ на заданный раздражитель), такие, для которых он монотонно убывает, и такие, для которых паттерн отклика более сложный.

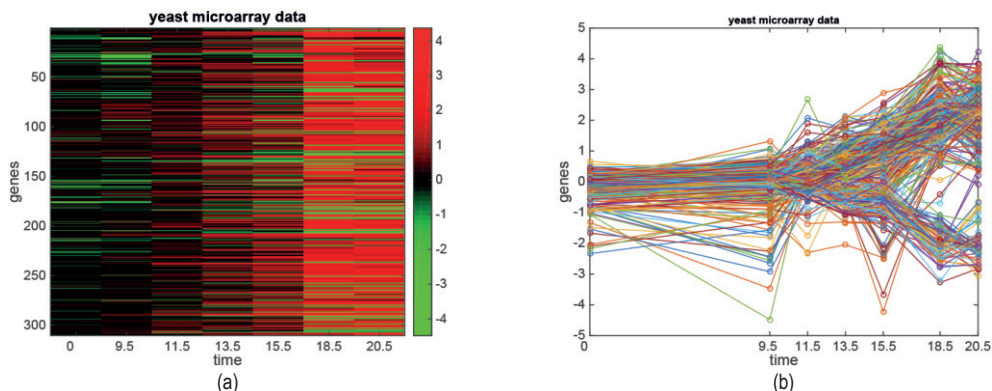


Рис. 21.5 ❖ (а) Данные об экспрессии генов дрожжей, представленные в виде тепловой карты. (b) Те же данные в виде временных рядов. Построено программой по адресу figures.problml.ai/book1/21.5

Допустим, что для вычисления матрицы попарных различий $\mathbf{D} \in \mathbb{R}^{300 \times 300}$ используется евклидово расстояние и применяется иерархическая кластеризация со средней связью. Тогда получается дендрограмма, показанная на рис. 21.6а. Если обрезать дерево на некоторой высоте, то получится 16 кластеров, показанных на рис. 21.6б. Временные ряды, отнесенные к каждому кластеру, действительно похожи.

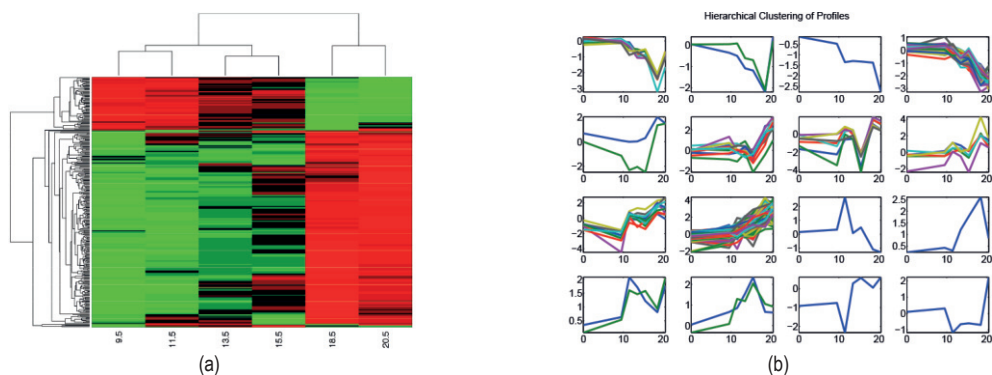


Рис. 21.6 ❖ Результаты применения иерархической кластеризации к данным об экспрессии генов. (а) Строки переставляются в соответствии со схемой иерархической кластеризации (со средней связью), чтобы похожие строки оказались ближе друг к другу. (b) 16 кластеров, образовавшихся в результате разрезания дерева со средней связью на некоторой высоте. Построено программой по адресу figures.problml.ai/book1/21.6

21.2.3. Расширения

Существует много расширения базового алгоритма НАС. Например, в работе [Mon+21] представлена лучше масштабируемая версия восходящего алгорит-

ма, который строит вложенные кластеры параллельно. А в работе [Мон+19] обсуждается онлайн-версия алгоритма, умеющая кластеризовать данные по мере поступления, пересматривая по ходу дела ранее принятые решения о кластеризации (а не просто принимая жадные решения). При некоторых предположениях можно доказать, что таким образом восстанавливается истинная структура данных. Это может быть полезно для кластеризации «упоминаний» о «сущностях» (например, людях или предметах) в потоке текстовых данных. (Эта задача называется **обнаружением сущностей**.)

21.3. КЛАСТЕРИЗАЦИЯ МЕТОДОМ К СРЕДНИХ

С иерархической агломеративной кластеризацией (раздел 21.2) связано несколько проблем. Во-первых, она требует времени порядка $O(N^3)$ (в случае метода средней связи), поэтому применять ее к большим наборам данных затруднительно. Во-вторых, предполагается, что матрица различий уже вычислена, тогда как на практике понятие «сходства» зачастую выражено нечетко и подлежит обучению. В-третьих, это всего лишь алгоритм, а не модель, поэтому трудно оценить, насколько он хорош. То есть не существует определенной целевой функции, подлежащей оптимизации.

В этом разделе мы обсудим **алгоритм К средних** [Mac67; Llo82], решающий эти проблемы. В частности, он работает за время $O(N)$, вычисляет сходство в терминах евклидова расстояния до центров обученных кластеров $\mu_k \in \mathbb{R}^D$ и оптимизирует четко определенную функцию стоимости.

21.3.1. Алгоритм

Мы предполагаем, что существует K центров кластеров $\mu_k \in \mathbb{R}^D$, поэтому можем кластеризовать данные, отнеся каждую точку $\mathbf{x}_n \in \mathbb{R}^D$ к ближайшему центру:

$$z_n^* = \operatorname{argmin}_k \|\mathbf{x}_n - \mu_k\|_2^2. \quad (21.13)$$

Разумеется, мы не знаем, где находятся центры кластеров, но можем оценить их, вычислив среднее значение всех отнесенных к ним точек:

$$\mu_k = \frac{1}{N_k} \sum_{n: z_n=k} \mathbf{x}_n. \quad (21.14)$$

Эти шаги повторяются до сходимости алгоритма.

Формально задачу можно рассматривать как нахождение локального минимума следующей функции стоимости, называемой **искажением**:

$$J(\mathbf{M}, \mathbf{Z}) = \sum_{n=1}^N \|\mathbf{x}_n - \mu_{z_n}\|^2 = \|\mathbf{X} - \mathbf{Z}\mathbf{M}^T\|_F^2, \quad (21.15)$$

где $\mathbf{X} \in \mathbb{R}^{N \times D}$, $\mathbf{Z} \in [0, 1]^{N \times K}$, а столбцы матрицы $\mathbf{M} \in \mathbb{R}^{D \times K}$ – центры кластеров μ_k . Алгоритм К средних оптимизирует эту функцию, применяя чередующуюся минимизацию. (Это тесно связано с EM-алгоритмом для моделей гауссовых смесей, описанным в разделе 21.4.1.1.)

21.3.2. Примеры

В этом разделе мы приведем несколько примеров кластеризации методом К средних.

21.3.2.1. Кластеризация точек на плоскости

На рис. 21.7 показано кластеризация точек на плоскости методом К средних. Как видим, этот метод порождает **диаграмму Вороного**. Результирующая кластеризация чувствительна к выбору начальных центров. И хорошо видно, что для правой кластеризации качество ниже, а искажение выше. По умолчанию в sklearn применяется 10 перезапусков со случайно выбранными центрами (в сочетании с инициализацией K-means++, описанной в разделе 21.3.4) и возвращается кластеризация с наименьшим искажением. (В sklearn искажение называется «инерцией».)

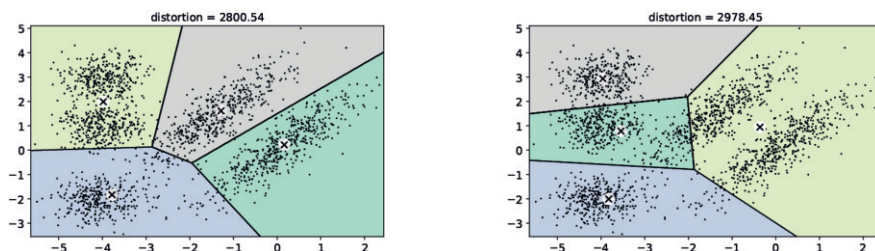


Рис. 21.7 ❖ Кластеризация методом К средних на плоскости. Показаны результаты при двух разных инициализациях. На основе рис. 9.5 из работы [Gér19]. Построено программой по адресу figures.problml.ai/book1/21.7

21.3.2.2. Кластеризация временных рядов экспрессии генов дрожжей

На рис. 21.8 показаны результаты применения метода К средних с $K = 16$ к матрице временных рядов дрожжей размера 300×7 , изображенной на рис. 21.5. Видно, что «похожие» временные ряды отнесены к одному кластеру. Также видно, что центроид каждого кластера является удовлетворительной сводкой всех точек, отнесенных к этому кластеру. Наконец, отметим, что группа 6 вообще не использовалась, так как к ней не отнесено ни одной точки. Однако это лишь случайный эффект процесса инициализации, и нет никакой гарантии, что мы получим ту же кластеризацию или хотя бы то же количество кластеров, если выполним алгоритм еще раз. (Мы обсудим хорошие способы инициализации в разделе 21.3.4, а способы выбора K в разделе 21.3.7.)

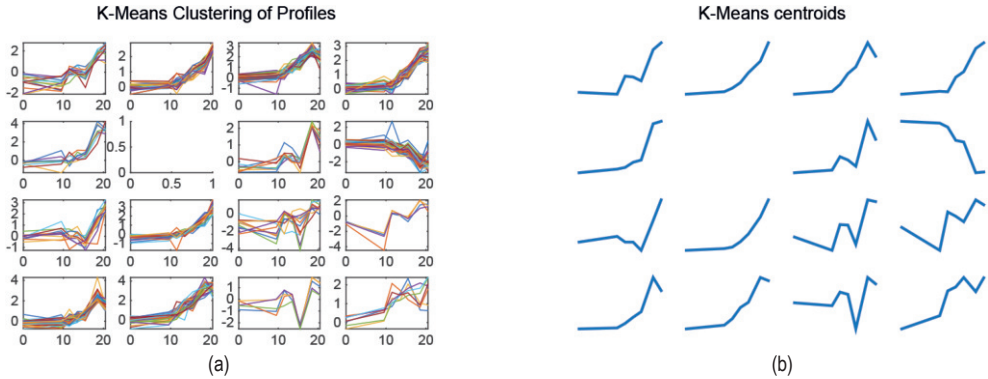


Рис. 21.8 ❖ Кластеризация данных об экспрессии генов дрожжей (рис. 21.5) методом К средних при $K = 16$. (a) Визуализация всех временных рядов, отнесенных к каждому кластеру. (b) Визуализация 16 центров кластеров, представленного типичным временным рядом. Построено программой по адресу figures.problml.ai/book1/21.8

21.3.3. Векторное квантование

Допустим, что мы хотим выполнить сжатие с потерей информации для некоторых вещественных векторов $\mathbf{x}_n \in \mathbb{R}^D$. Очень простой подход – использовать **векторное квантование** (vector quantization – **VQ**). Основная идея – заменить каждый вещественный вектор $\mathbf{x}_n \in \mathbb{R}^D$ дискретным символом $z_n \in \{1, \dots, K\}$, равным индексу в **кодовой книге**, содержащей K прототипов, $\boldsymbol{\mu}_k \in \mathbb{R}^D$. Каждый вектор кодируется индексом самого похожего прототипа, а сходство измеряется в терминах евклидова расстояния:

$$\text{encode}(\mathbf{x}_n) = \underset{k}{\operatorname{argmin}} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2. \quad (21.16)$$

Мы можем определить функцию стоимости, которая измеряет качество кодовой книги путем вычисления индуцированной **ошибки реконструкции**, или **искажения**:

$$J \triangleq \frac{1}{N} \sum_{n=1}^N \|\mathbf{x}_n - \text{decode}(\text{encode}(\mathbf{x}_n))\|^2 = \frac{1}{N} \sum_{n=1}^N \|\mathbf{x}_n - \boldsymbol{\mu}_{z_n}\|^2, \quad (21.17)$$

где $\text{decode}(k) = \boldsymbol{\mu}_k$. Это в точности та функция стоимости, которая минимизируется алгоритмом К средних.

Конечно, можно добиться нулевого искажения, сопоставив один прототип каждому вектору данных, т. е. положив $K = N$ и $\boldsymbol{\mu}_n = \mathbf{x}_n$. Однако при этом не будет никакого сжатия данных. Именно, потребуется $O(NDB)$ битов, где N – количество вещественных векторов длины D каждый, а B – количество битов, необходимых для представления вещественного скаляра (точность квантованного представления каждого \mathbf{x}_n).

Можно добиться большего, если обнаружить похожие векторы, создать для них прототипы, или центроиды, а затем представить данные отклонениями

от этих прототипов. Тогда потребление памяти снизится до $O(N \log_2 K + KDB)$ бит. Член $O(N \log_2 K)$ присутствует, потому что для каждого из N векторов необходимо указать, какое из K кодовых слов ему соответствует, а член $O(KDB)$ – потому что нужно хранить все записи кодовой книги, каждая из которых является D -мерным вектором. При больших N преобладает первый член, поэтому коэффициент схемы кодирования (число битов на один объект) имеет порядок $O(\log_2 K)$, что обычно гораздо меньше, чем $O(DB)$.

Одним из приложений векторного квантования является **сжатие изображений**. Возьмем изображение 200×320 пикселей на рис. 21.9; мы хотим рассматривать его как множество $N = 64\,000$ скаляров. Если для представления каждого пикселя используется один байт (яркость полутонового изображения от 0 до 255), то $B = 8$, поэтому понадобится $NB = 512\,000$ бит для представления несжатого изображения. Для представления сжатого изображения требуется $O(N \log_2 K)$ битов. Для $K = 4$ это приблизительно 128 Кб, т. е. коэффициент сжатия равен 4, но воспринимаемая потеря качества пренебрежимо мала (см. рис. 21.9b).

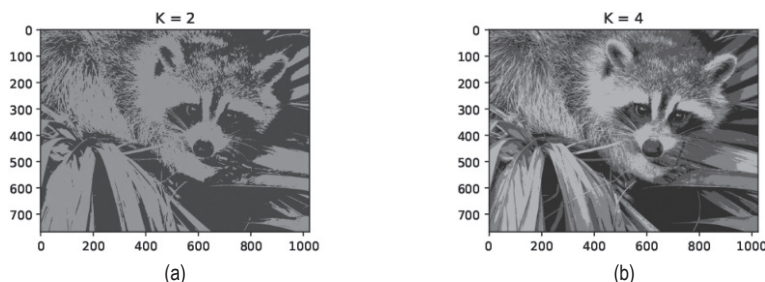


Рис. 21.9 ❖ Изображение, сжатое методом векторного квантования с кодовой книгой размера K . (a) $K = 2$. (b) $K = 4$. Построено программой по адресу figures.problml.ai/book1/21.9

Большей степени сжатия можно достичь, если моделировать пространственную корреляцию между пикселями, например кодировать блоки 5×5 (как в JPEG). Объясняется это тем, что невязки (отличия от предсказаний модели) будут меньше, а для кодирования потребуется меньше битов. Таким образом, существует глубокая связь между сжатием данных и оценкой плотности. Дополнительные сведения см. во втором томе этой книги, [Mur22].

21.3.4. Алгоритм K-means++

Алгоритм K средних оптимизирует невыпуклую целевую функцию, поэтому к его инициализации следует подходить с особой тщательностью. Самый простой подход – случайным образом выбрать K точек данных и сделать их начальными значениями μ_k . Результат можно улучшить, выполнив несколько перезапусков с различными начальными точками, а затем выбрав лучшее решение. Но это медленно.

Более перспективный подход – выбирать центры последовательно, стараясь «покрыть» данные. То есть первая точка выбирается случайно, а каждая последующая выбирается из оставшихся с вероятностью, пропорциональной квадрату расстояния до ближайшего к точке центра кластера. То есть на t -й итерации мы выбираем \mathbf{x}_n в качестве следующего центра кластера с вероятностью

$$p(\mu_t = \mathbf{x}_n) = \frac{D_{t-1}(\mathbf{x}_n)^2}{\sum_{n'=1}^N D_{t-1}(\mathbf{x}_{n'})^2}, \quad (21.18)$$

где

$$D_t(\mathbf{x}) = \min_{k=1}^{t-1} \|\mathbf{x} - \mu_k\|_2^2 \quad (21.19)$$

– расстояние от \mathbf{x} до ближайшего из уже выбранных центроидов. Таким образом, у точек, далеко отстоящих от центроида, больше шансов быть выбранными, вследствие чего искажение уменьшается. Этот алгоритм называется **кластеризацией с выбором самой дальней точки** (farthest point clustering) [Gon85], или **K-means++** [AV07; Bah+12; Bac+16; BLK17; LS19a]. Как ни странно, можно показать, что этот простой трюк гарантирует, что ошибка реконструкции не более чем в $O(\log K)$ раз хуже оптимальной [AV07].

21.3.5. Алгоритм К медоидов

В варианте метода К средних, называемом алгоритмом **К медоидов**, мы оцениваем центр каждого кластера μ_k , выбирая пример $\mathbf{x}_n \in \mathcal{X}$, для которого среднее несходство со всеми остальными примерами минимально; такая точка называется **медоидом**. Отметим, что в алгоритме К средних для вычисления центра берутся средние по точкам $\mathbf{x}_n \in \mathbb{R}^D$, отнесенным к кластерам. Алгоритм К медоидов может оказаться более робастным к выбросам (хотя эту проблему можно также решить, используя смеси распределений Стьюдента, а не гауссовых распределений). Важнее, однако, что алгоритм К медоидов можно применять к данным, не принадлежащим пространству \mathbb{R}^D , когда усреднение невозможно корректно определить. Входом для алгоритма К медоидов является матрица попарных расстояний $D(n, n')$ размера $N \times N$, а не матрица признаков размера $N \times D$.

Классический алгоритм нахождения К медоидов называется **разбиением вокруг медоидов** (partitioning around medoids – **РАМ**) [KR87]. На каждой итерации в цикле перебираются все К медоидов. Для каждого медоида t рассматривается каждая не являющаяся медоидом точка o ; t и o меняются местами, и пересчитывается функция стоимости (сумма расстояний от точек до их медоидов). Если стоимость уменьшилась, новая конфигурация медоидов сохраняется. Время работы этого алгоритма составляет $O(K(N - K)^2 I)$, где I – число итераций

Существует также более простой и быстрый **метод итераций Вороного**, описанный в работе [PJ09]. В нем на каждой итерации выполняется два шага,

как в алгоритме К средних. Сначала для каждого кластера k рассматриваются все отнесенные к нему точки, $S_k = \{n : z_n = k\}$, а затем m_k присваивается индекс медоида этого множества. (Для нахождения медоида нужно рассмотреть все $|S_k|$ точек-кандидатов и выбрать из них ту, для которой сумма расстояний до всех точек в S_k минимальна.) Затем каждая точка n относится к ближайшему к ней медоиду, $z_n = \operatorname{argmin}_k D(n, k)$. Псевдокод приведен в алгоритме 12. Время работы этого алгоритма составляет $O(NKI)$, что сопоставимо с временем работы алгоритма К средних и гораздо быстрее, чем в случае РАМ.

Алгоритм 12. Алгоритм К медоидов

```

1  Инициализировать  $m_{1:K}$  — случайное подмножество  $\{1, \dots, N\}$  размера  $K$ ;
2  repeat
3       $z_n = \operatorname{argmin}_k d(n, m_k)$  для  $n = 1 : N$ ;
4       $m_k \leftarrow \operatorname{argmin}_{n: z_n=k} \sum_{n': z_{n'}=k} d(n, n')$  для  $k = 1 : K$ ;
5  until не сошелся;
```

21.3.6. Способы ускорения

Кластеризация методом К средних занимает время $O(NKI)$, где I — число итераций, но величину постоянного множителя можно уменьшить с помощью различных приемов. Например, в работе [Elk03] показано, как воспользоваться неравенством треугольника, чтобы отслеживать нижнюю и верхнюю границы расстояний между входными данными и центроидами; это может быть полезно для исключения лишних вычислений. Можно также использовать аппроксимацию на мини-пакете, как описано в работе [Scu10]. Это может оказаться значительно быстрее, но иногда приводит к небольшому увеличению потери (см. рис. 21.10).

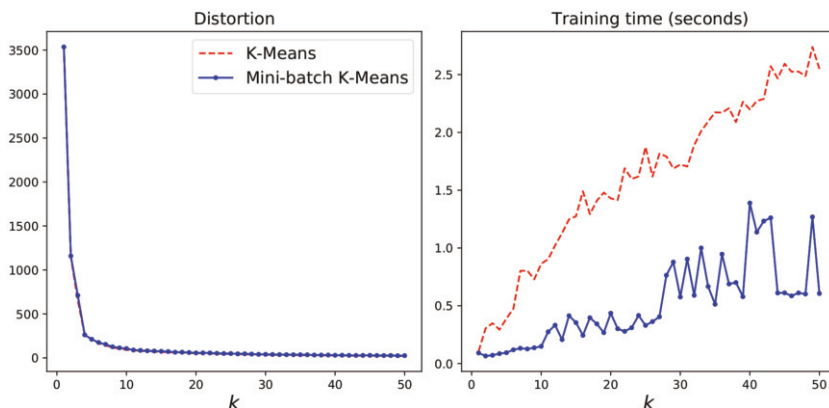


Рис. 21.10 ❖ Кластеризация двумерных данных на рис. 21.7 методом К средних на пакете и на мини-пакете. Слева: зависимость искажения от K . Справа: зависимость времени обучения от K . На основе рис. 9.6 из работы [Gér19]. Построено программой по адресу figures.problm.ai/book1/21.10

21.3.7. Выбор числа кластеров K

В этом разделе мы обсудим, как выбирать число кластеров K в алгоритме К средних и родственных ему методах.

21.3.7.1. Минимизация искажения

Исходя из нашего опыта обучения с учителем, естественно было бы выбрать для K значение, доставляющее минимум ошибке реконструкции на контрольном наборе:

$$\text{err}(\mathcal{D}_{\text{valid}}, K) = \frac{1}{|\mathcal{D}_{\text{valid}}|} \sum_{n \in \mathcal{D}_{\text{valid}}} \|\mathbf{x}_n - \hat{\mathbf{x}}_n\|_2^2, \quad (21.20)$$

где $\hat{\mathbf{x}}_n = \text{decode}(\text{encode}(\mathbf{x}_n))$ – реконструкция \mathbf{x}_n .

К сожалению, эта идея не работает. Действительно, как показано на рис. 21.11а, искажение монотонно убывает при росте K . Чтобы понять, почему это так, заметим, что модель K средних – это модель с вырожденной плотностью, состоящая из K пиков в центрах μ_k . С ростом K мы покрываем все большую часть пространства входов. Поэтому для любой входной точки повышается вероятность найти близкий прототип, точно представляющий ее, а значит, уменьшается ошибка реконструкции. Таким образом, в отличие от обучения с учителем, мы не можем использовать ошибку реконструкции на контрольном наборе как способ выбора наилучшей модели, обученной без учителя. (Это замечание относится также к выбору размерности в методе РСА, см. раздел 20.1.4.)

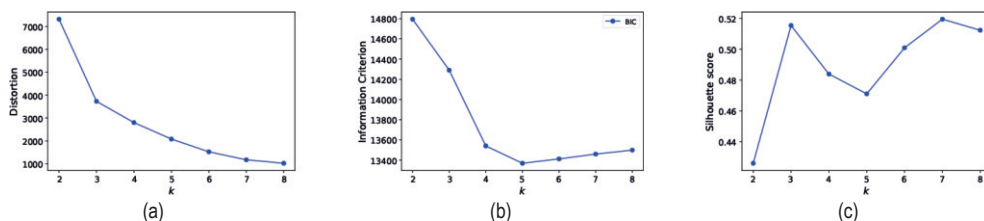


Рис. 21.11 ❖ Зависимость качества метода К средних и модели гауссовой смеси от K для двумерного набора данных на рис. 21.7. (а) Зависимость искажения на контрольном наборе от K . Построено программой по адресу figures.probl.ai/book1/21.11. (б) Зависимость байесовского информационного критерия от K . Построено программой по адресу figures.probl.ai/book1/21.11. (с) Зависимость силуэтной оценки от K . Построено программой по адресу figures.probl.ai/book1/21.11

21.3.7.2. Максимизация маргинального правдоподобия

А вот что действительно работает, так это использование подходящей вероятностной модели, например гауссовых смесей (GMM), который мы опишем в разделе 21.4.1. Для выбора модели можно использовать логарифмическое

маргинальное правдоподобие (log marginal likelihood – LML) данных.

Мы можем аппроксимировать LML, применяя оценку BIC, как было описано в разделе 5.2.5.1. Из формулы (5.59) имеем

$$\text{BIC}(K) = \log p(\mathcal{D}|\hat{\theta}_K) - \frac{D_K}{2} \log(N), \quad (21.21)$$

где D_K – количество параметров в модели с K кластерами, а $\hat{\theta}_K$ – оценка максимального правдоподобия. На рис. 21.11b видно, что имеет место типичная U-образная кривая, когда величина штрафа сначала убывает, а потом возрастает.

Это работает, потому что с каждым кластером ассоциирован не просто вырожденный пик, а гауссово распределение, заполняющее некоторый объем пространства входов. После того как кластеров оказывается достаточно для покрытия истинных мод распределения, вступает в игру байесовская бритва Оккама (раздел 5.2.3) и начинает штрафовать модель за излишнюю сложность.

Дальнейшее обсуждение байесовского выбора модели для смесовых моделей см. в разделе 21.4.1.3.

21.3.7.3. Силуэтный коэффициент

В этом разделе мы опишем распространенный эвристический метод выбора числа кластеров в методе K средних. Он предназначен для работы со сферическими (невтянутыми) кластерами. Сначала определим **силуэтный коэффициент** примера i как $sc(i) = (b_i - a_i) = \max(a_i, b_i)$, где a_i – среднее расстояние до других примеров в кластере $k_i = \text{argmin}_k \|\mu_k - \mathbf{x}_i\|$, а b_i – среднее расстояние до других примеров во втором по дальности кластере, $k'_i = \text{argmin}_{k \neq k_i} \|\mu_k - \mathbf{x}_i\|$. Таким образом, a_i – мера компактности i -го кластера, а b_i – мера расстояния между кластерами. Силуэтный коэффициент изменяется от -1 до $+1$. Значение $+1$ означает, что пример близок ко всем членам своего кластера и далек от остальных кластеров, значение 0 – что пример близок к границе кластера, а значение -1 – что пример, возможно, попал не в тот кластер. Определим **силуэтную оценку** кластеризации K как средний силуэтный коэффициент по всем примерам.

На рис. 21.11a показана зависимость искажения от K для данных на рис. 21.7. Как было объяснено выше, искажение монотонно убывает при росте K . При $K = 3$ имеет место небольшой **излом**, но его трудно обнаружить. На рис. 21.11c показана зависимость силуэтной оценки от K . Тут при $K = 3$ наблюдается куда более заметный пик, но при $K = 7$ ситуация почти выравнивается. На рис. 21.12 сравниваются некоторые из этих кластеризаций.

Поучительно взглянуть на отдельные силуэтные коэффициенты, а не только на усредненную оценку. Их можно представить на **силуэтной диаграмме**, показанной на рис. 21.13, где цветные области соответствуют разным кластерам. Вертикальная прямая показывает средний коэффициент. Для кластеров с большим количеством точек слева от этой прямой качество, вероятно, низкое. Силуэтную диаграмму можно также использовать для изучения размера кластеров, даже если данные не двумерные.

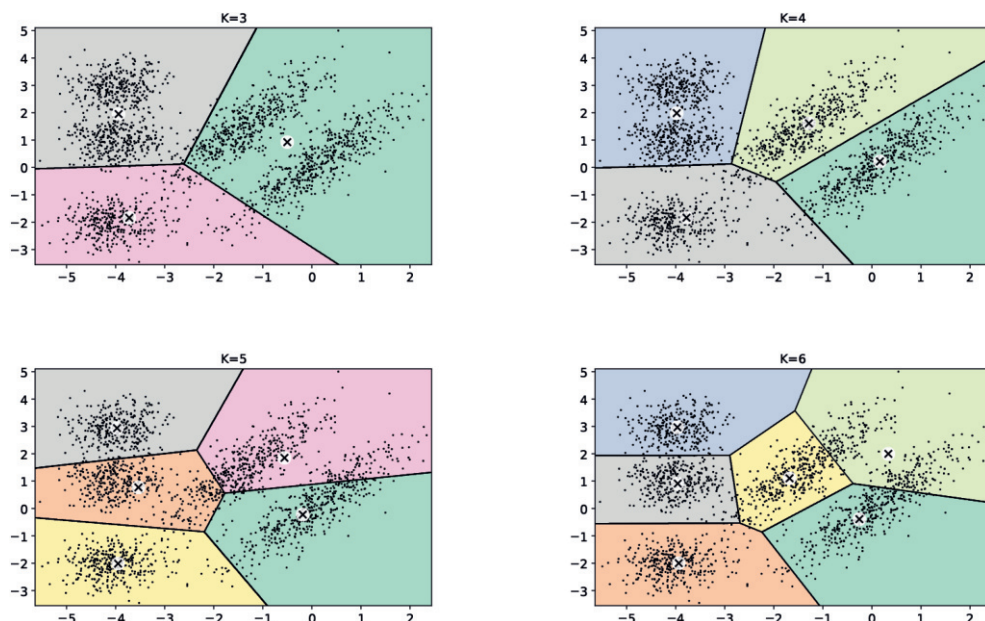


Рис. 21.12 ❖ Диаграммы Вороного метода К средних при различных К для двумерного набора данных на рис. 21.7. Построено программой по адресу figures.problml.ai/book1/21.12

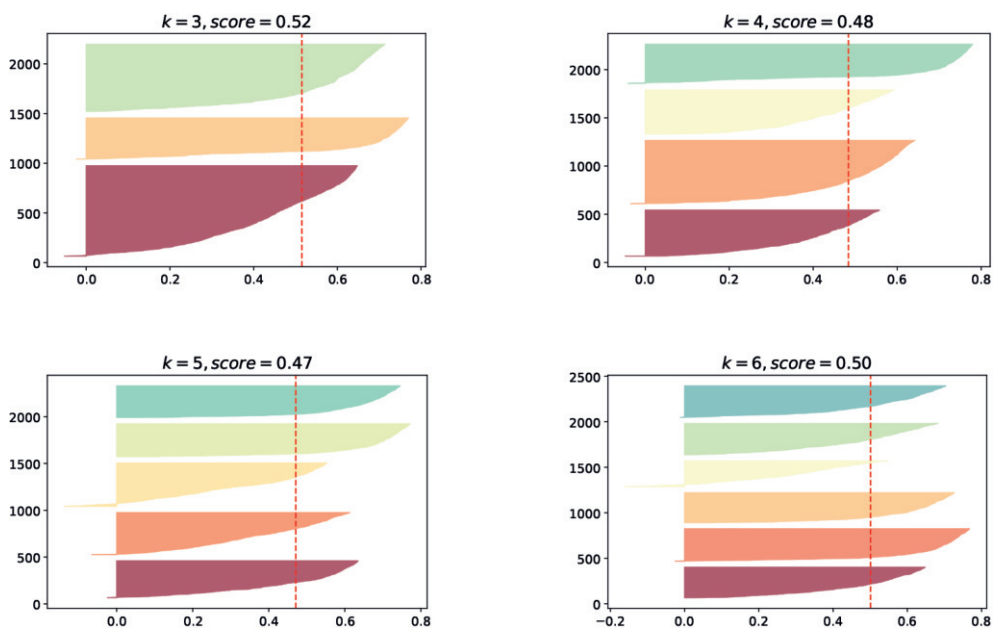


Рис. 21.13 ❖ Силуэтные диаграммы метода К средних при разных К для двумерного набора данных на рис. 21.7. Построено программой по адресу figures.problml.ai/book1/21.13

21.3.7.4. Инкрементное увеличение количества компонент смеси

Альтернативой поиску наилучшего значения K является инкрементный «рост» GMM. Можно начать с небольшого K , а после каждого раунда обучения попробовать разбить кластер с наибольшим весом смеси на два, выбрав в качестве новых центроидов случайные возмущения оригинального центроида, а в качестве новых оценок – уменьшенные вдвое старые оценки. Если для нового кластера оценка слишком мала или дисперсия слишком узкая, то он удаляется. Так продолжается, пока не будет получено желаемое число кластеров. Детали см. в работе [FJ02].

21.3.7.5. Методы разреженного оценивания

Еще один подход – выбрать большое значение K , а затем использовать какое-нибудь поощряющее разреженность априорное распределение или метод вывода, чтобы исключить ненужные компоненты смеси, например, вариационный байесовский вывод. Дополнительные сведения см. во втором томе этой книги, [Mur22].

21.4. КЛАСТЕРИЗАЦИЯ С ПОМОЩЬЮ СМЕСОВЫХ МОДЕЛЕЙ

Мы видели, что алгоритм K средних можно использовать для кластеризации векторов данных в \mathbb{R}^D . Однако этот метод предполагает, что все кластеры имеют одинаковую сферическую форму, а это слишком ограничительное предположение. Кроме того, в методе K средних предполагается, что все кластеры можно описать гауссовыми распределениями в пространстве входов, поэтому к дискретным данным он неприменим. Смесовые модели (раздел 3.5) позволяют преодолеть обе эти проблемы.

21.4.1. Смеси гауссовых распределений

Напомним (см. раздел 3.5.1), что модель гауссовой смеси (GMM) имеет вид:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \quad (21.22)$$

Если нам известны параметры модели $\boldsymbol{\theta} = (\boldsymbol{\pi}, \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\})$, то можно воспользоваться формулой Байеса для вычисления ответственности (апостериорной вероятности принадлежности) кластера k за точку \mathbf{x}_n :

$$r_{nk} \triangleq p(z_n = k|\mathbf{x}_n, \boldsymbol{\theta}) = \frac{p(z_n = k|\boldsymbol{\theta})p(\mathbf{x}_n|z_n = k, \boldsymbol{\theta})}{\sum_{k'=1}^K p(z_n = k'|\boldsymbol{\theta})p(\mathbf{x}_n|z_n = k', \boldsymbol{\theta})}. \quad (21.23)$$

Зная ответственности, мы можем вычислить наиболее вероятное назначение кластера следующим образом:

$$\hat{z}_n = \operatorname{argmax}_k r_{nk} = \operatorname{argmax}_k [\log p(\mathbf{x}_n | z_n = k, \boldsymbol{\theta}) + \log p(z_n = k | \boldsymbol{\theta})]. \quad (21.24)$$

Это называется **жесткой кластеризацией**.

21.4.1.1. Метод К средних – частный случай ЕМ-алгоритма

Мы можем оценить параметры GMM, применив ЕМ-алгоритм (раздел 8.7.3). Оказывается, что алгоритм К средних – это частный случай этого алгоритма, в котором делается две аппроксимации: фиксируется $\boldsymbol{\Sigma}_k = \mathbf{I}$ и $\pi_k = 1/K$ для всех кластеров (так что нужно лишь оценить средние $\boldsymbol{\mu}_k$), а Е-шаг аппроксимируется путем замены мягких ответственностей жесткими назначениями кластеров, т. е. мы вычисляем $z_n^* = \operatorname{argmax}_k r_{nk}$ и полагаем $r_{nk} \approx \mathbb{I}(k = z_n^*)$ вместо использования мягких ответственностей $r_{nk} = p(z_n = k | \mathbf{x}_n, \boldsymbol{\theta})$. При такой аппроксимации задача нахождения взвешенной MLE в формуле (8.165) М-шага сводится к формуле (21.14), т. е. мы возвращаемся к методу К средних.

Однако предположение о том, что все кластеры имеют одинаковую сферическую форму, чрезмерно ограничительно. Например, на рис. 21.14 показана маргинальная плотность и кластеризация, получающаяся при различных формах ковариационной матрицы для двумерных данных. Как видим, для моделирования этого конкретного набора данных необходимо, чтобы модель улавливала внедиагональную ковариацию для некоторых кластеров (верхний ряд).

21.4.1.2. Неидентифицируемость и переключение метки

Заметим, что мы можем переставлять метки в смесовой модели, не изменяя правдоподобия. Это называется **проблемой переключения метки** и является примером **неидентифицируемости** параметров. Это может приводить к проблемам, если мы хотим выполнить апостериорный вывод параметров (а не просто вычислить оценку MLE или MAP). Например, предположим, что требуется аппроксимировать моделью GMM с $K = 2$ компонентами данные на рис. 21.15, используя гамильтонов метод Монте-Карло (НМС). Апостериорное распределение средних, $p(\mu_1, \mu_2 | \mathcal{D})$, показано на рис. 21.16а. Мы видим, что апостериорное маргинальное распределение для каждой компоненты, $p(\mu_k | \mathcal{D})$, бимодально. Это отражает тот факт, что существует два одинаково хороших объяснения данных: либо $\mu_1 \approx 47$ и $\mu_2 \approx 57$, либо наоборот.

Чтобы нарушить эту симметрию, мы можем наложить **ограничение порядка** на центры: $\mu_1 < \mu_2$. Это можно сделать, прибавив штраф или потенциал к целевой функции, если ограничение нарушено. Точнее, логарифмическое совместное распределение со штрафом принимает вид:

$$\ell'(\boldsymbol{\theta}) = \log p(\mathcal{D} | \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) + \phi(\boldsymbol{\mu}), \quad (21.25)$$

где

$$\phi(\boldsymbol{\mu}) = \begin{cases} -\infty, & \text{если } \mu_1 < \mu_0 \\ 0 & \text{в противном случае} \end{cases}. \quad (21.26)$$

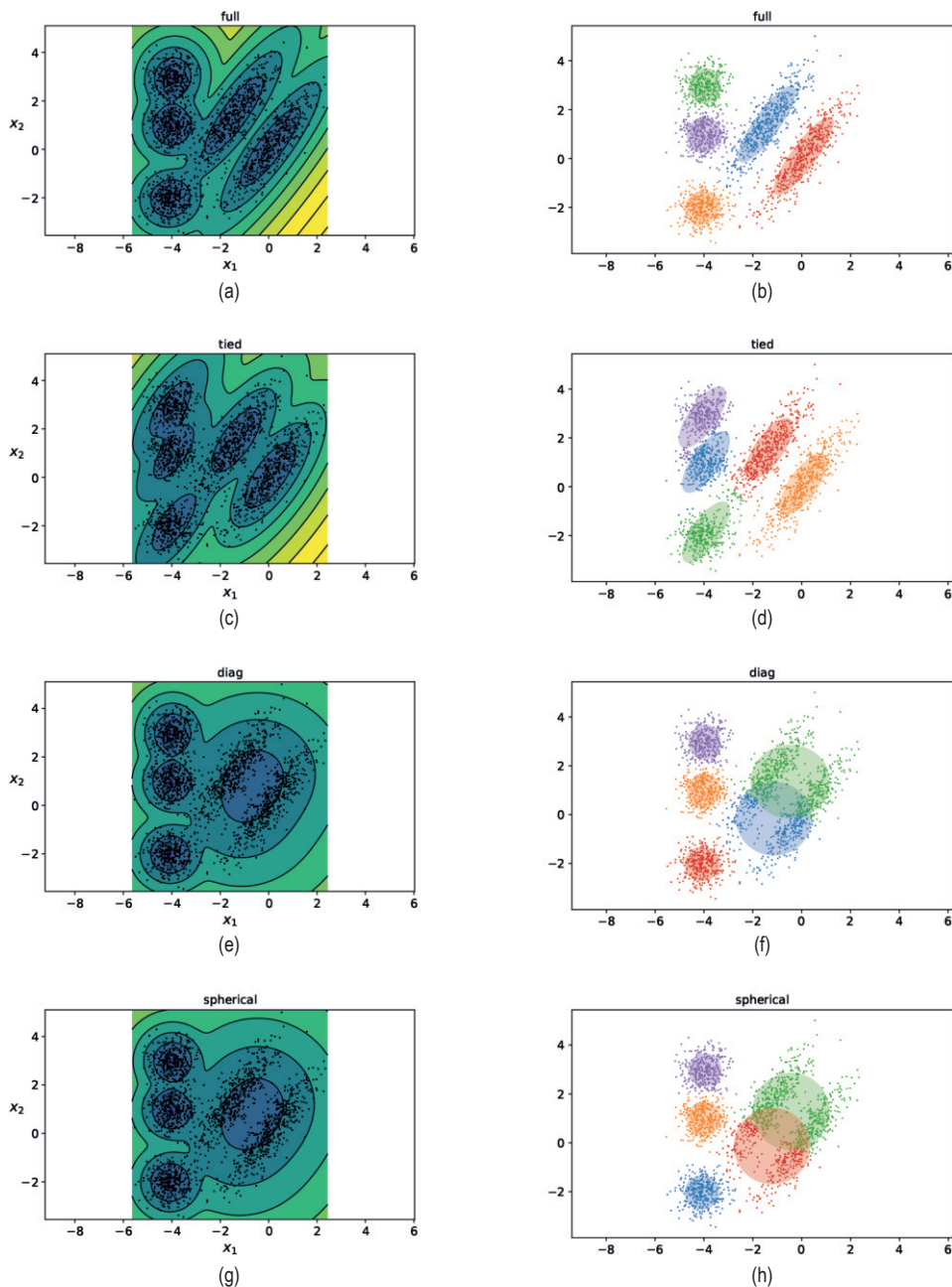


Рис. 21.14 ❖ Аппроксимация двумерных данных с помощью GMM с $K=5$ компонентами. Левый столбец: маргинальное распределение $p(x)$. Правый столбец: визуализация каждой смеси распределений и жесткое отнесение точек к наиболее вероятным кластерам. (a–b) Полная ковариационная матрица. (c–d) Связанная полная ковариационная матрица. (e–f) Диагональная ковариационная матрица. (g–h) Сферическая ковариационная матрица. Цветовое кодирование выбрано произвольно. Построено программой по адресу figures.probl.ai/book1/21.14

Это дает желаемый эффект, как показано на рис. 21.16b.

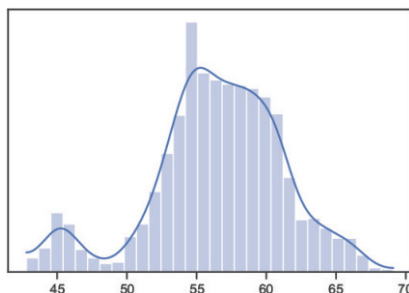


Рис. 21.15 ❖ Одномерные данные с наложенной оценкой плотности ядра. На основе рис. 6.2 из работы [Mar18]. Построено программой по адресу figures.problml.ai/book1/21.15

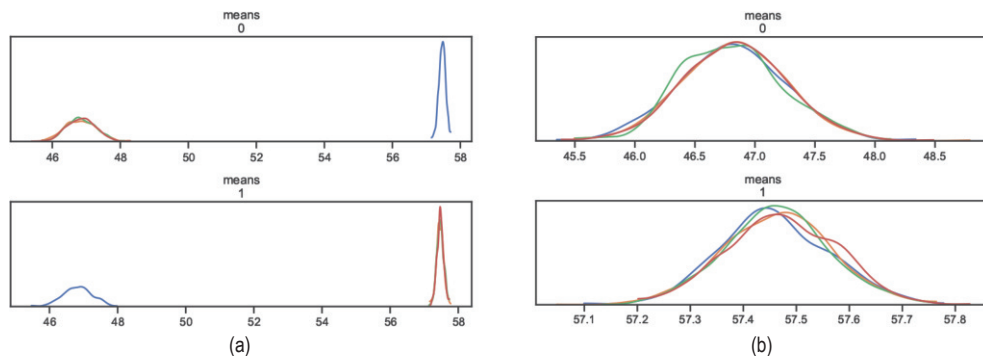


Рис. 21.16 ❖ Проблема переключения метки при выполнении апостериорного вывода параметров GMM. Приведена оценка KDE апостериорного маргинального распределения, вычисленная по 1000 примерам из 4 цепочек ММС. (a) Модель без ограничений. (b) Модель с ограничениями, добавлен штраф, обеспечивающий выполнение условия $\mu_0 < \mu_1$. На основе рис. 6.6–6.7 из работы [Mar18]. Построено программой по адресу figures.problml.ai/book1/21.16

Более общий подход – применить к параметрам преобразование, гарантирующее идентифицируемость. Именно, мы производим выборку параметров θ из вспомогательного распределения, а затем применяем к ним обратимое преобразование $\theta' = f(\theta)$, перед тем как вычислять логарифм совместного распределения, $\log p(\mathcal{D}, \theta')$. Чтобы учесть изменение переменных (раздел 2.8.3), мы прибавляем логарифм определителя якобиана. В случае одномерного упорядочивающего преобразования, которое просто сортирует свои входы, определитель якобиана равен 1, поэтому его логарифм обращается в 0.

К сожалению, этот подход не масштабируется на задачи размерности больше 1, потому что не существует очевидного способа ввести ограничение порядка на центры μ_k .

21.4.1.3. Байесовский выбор модели

Располагая надежным способом обеспечить идентифицируемость, мы можем воспользоваться байесовским выбором модели из раздела 5.2.2 для выбора числа кластеров K . На рис. 21.17 показаны результаты аппроксимации данных на рис. 21.15 моделью GMM с $K = 3$ –6 компонентами. Мы применяем упорядочивающее преобразование к средним и выполняем вывод с применением НМС. Результирующая модель GMM сравнивается с оценкой плотности ядра (раздел 16.3), которая часто приводит к чрезмерному сглаживанию данных. Мы видим явные признаки двух горбов, соответствующих двум разным подмножествам генеральной совокупности.

Эти модели можно сравнить количественно, вычислив оценки их WAIC (widely applicable information criterion – широко применимый информационный критерий) – аппроксимации логарифма маргинального правдоподобия (детали см. в работах [Wat10; Wat13; VGG17]). Результаты показаны на рис. 21.18. (Такой вид визуализации был предложен в работе [McE20, стр. 228].) Мы видим, что для модели с $K = 6$ оценка значительно выше, чем для остальных моделей, хотя следующая по качеству модель с $K = 5$ ненамного отстает. Это согласуется с графиком на рис. 21.17.

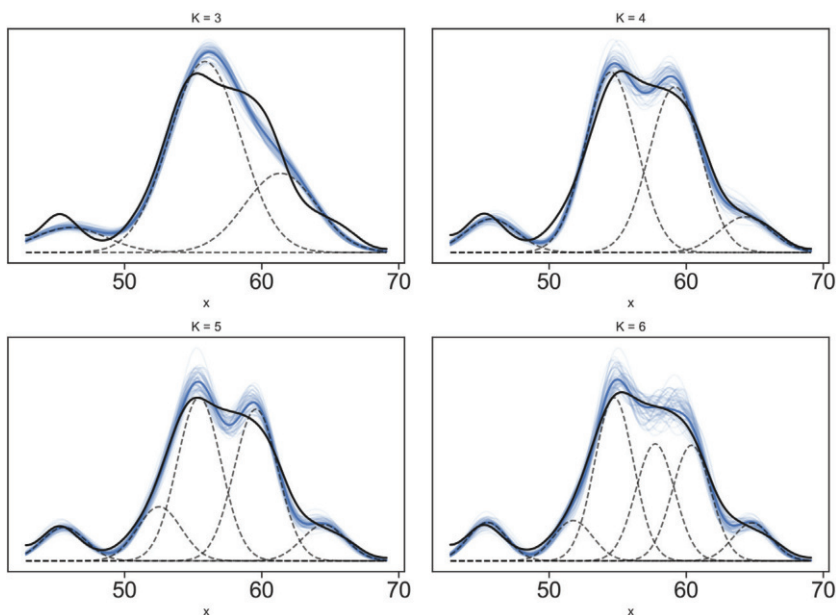


Рис. 21.17 ❖ Аппроксимация данных на рис. 21.15 моделями GMM с разным числом кластеров. Сплошная черная линия – аппроксимация ядерной оценки плотности. Сплошная синяя линия – апостериорное среднее; размытые голубые линии – выборка из апостериорного распределения. Пунктирные линии показывают отдельные компоненты гауссовой смеси, вычисленные путем подстановки их апостериорных средних. На основе рис. 6.8 из работы [Mar18]. Построено программой по адресу figures.probml.ai/book1/21.17

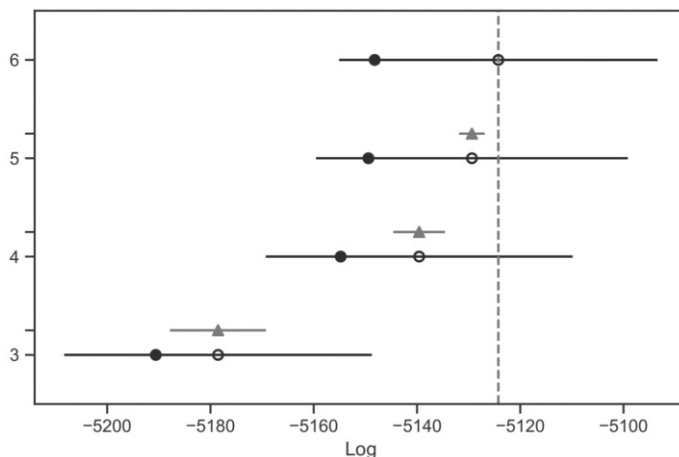


Рис. 21.18 ❖ Оценки WAIC для различных моделей GMM. Белый кружок обозначает оценку WAIC апостериорного среднего для каждой модели, а черные прямые – стандартную ошибку среднего. Черный кружок – внутривыборочное отклонение каждой модели, т. е. логарифмическое правдоподобие без штрафа. Пунктирная вертикальная прямая соответствует максимальному значению WAIC. Серым треугольником обозначена разность между оценкой WAIC для этой модели и для лучшей модели. На основе рис. 6.10 из работы [Mar18]. Построено программой по адресу figures.probml.ai/book1/21.18

21.4.2. Смеси распределений Бернулли

Как было сказано в разделе 3.5.2, мы можем использовать смеси распределений Бернулли для кластеризации бинарных данных. Модель имеет вид:

$$p(y|z = k, \theta) = \prod_{d=1}^D \text{Ber}(y_d | \mu_{dk}) = \prod_{d=1}^D \mu_{dk}^{y_d} (1 - \mu_{dk})^{1-y_d}. \quad (21.27)$$

Здесь μ_{dk} – вероятность, что бит d поднят в кластере k . Эту модель можно обучить с помощью ЕМ-алгоритма, СГС, метода Монте-Карло по схеме марковских цепей и т. д., см. пример на рис. 3.13, где кластеризации подвергнуты некоторые бинаризованные цифры из набора MNIST.

21.5. СПЕКТРАЛЬНАЯ КЛАСТЕРИЗАЦИЯ*

В этом разделе мы обсудим подход к кластеризации, основанный на анализе собственных значений матрицы попарных различий. В нем собственные векторы используются для нахождения векторов признаков для каждого примера, после чего эти векторы кластеризуются, например методом К средних (раздел 21.3). Этот подход называется **спектральной кластеризацией** [SM00; Lux07].

21.5.1. Нормализованные разрезы

Начнем с создания взвешенного неориентированного графа \mathbf{W} , вершинами которого являются примеры данных, а вес ребра $i - j$ определяет сходство. Обычно вершина соединяется только с самыми похожими на нее соседями – с целью сделать граф разреженным и ускорить вычисления.

Наша цель – найти K кластеров, содержащих похожие точки. То есть мы хотим найти такое **разбиение графа** на непересекающиеся множества вершин S_1, \dots, S_K , которое минимизирует некоторую функцию стоимости.

Первая попытка найти такую функцию заключается в том, чтобы вычислить суммарный вес ребер, соединяющих вершины внутри каждого кластера с вершинами вне него:

$$\text{cut}(S_1, \dots, S_K) \triangleq \frac{1}{2} \sum_{k=1}^K W(S_k, \bar{S}_k), \quad (21.28)$$

где $W(A, B) \triangleq \sum_{i \in A, j \in B} w_{ij}$ и $\bar{S}_k = V \setminus S_k$ – дополнение S_k , где $V = \{1, \dots, K\}$.

К сожалению, оптимальное решение этой задачи часто сводится к отщеплению одной вершины от всех остальных, поскольку так достигается минимальный вес разреза. Чтобы предотвратить это, мы можем поделить разрез на размер множества, получив в результате целевую функцию, называемую **нормализованным разрезом**:

$$\text{Ncut}(S_1, \dots, S_K) \triangleq \frac{1}{2} \sum_{k=1}^K \frac{\text{cut}(S_k, \bar{S}_k)}{\text{vol}(S_k)}, \quad (21.29)$$

где $\text{vol}(A) \triangleq \sum_{i \in A} d_i$ – полный вес множества A , а $d_i = \sum_{j=1}^N w_{ij}$ – взвешенная степень вершины i . Граф при этом разбивается на K кластеров таких, что вершины внутри каждого кластера похожи друг на друга и непохожи на вершины в других кластерах.

Задачу о нормализованном разрезе можно сформулировать как поиск бинарных векторов $\mathbf{c}_i \in \{0, 1\}^N$, доставляющих минимум приведенной выше целевой функции, где $c_{ik} = 1$ тогда и только тогда, когда точка i принадлежит кластеру k . К сожалению, эта задача NP-трудная [WW93]. Ниже мы обсудим ее ослабленную непрерывную постановку на основе собственных векторов, для которой найти решение проще.

21.5.2. Собственные векторы лапласиана графа кодируют кластеризацию

В разделе 20.4.9.2 мы обсуждали лапласиан графа, определенный как $\mathbf{L} \triangleq \mathbf{D} - \mathbf{W}$, где \mathbf{W} – симметричная матрица весов графа, а $\mathbf{D} = \text{diag}(d_i)$ – диагональная матрица, содержащая взвешенные степени вершин, $d_i = \sum_{j=1}^N w_{ij}$. Чтобы понять, почему \mathbf{L} может быть полезен для кластеризации графа, приведем следующий результат.

Теорема 21.5.1. Множество собственных векторов матрицы \mathbf{L} с собственным значением 0 натянуто на индикаторные векторы $\mathbf{1}_{S_1}, \dots, \mathbf{1}_{S_K}$, где S_k – K компонент связности графа.

Доказательство. Начнем со случая $K = 1$. Если \mathbf{f} – собственный вектор с собственным значением 0, то $0 = \sum_{ij} w_{ij} (f_i - f_j)^2$. Если две вершины соединены ребром, так что $w_{ij} > 0$, то должно быть $f_i = f_j$. Поэтому \mathbf{f} постоянный для всех вершин, соединенных путем в графе. Теперь предположим, что $K > 1$. В этом случае \mathbf{L} будет блочно-диагональной. Рассуждение, аналогичное приведенному выше, показывает, что мы будем иметь K индикаторных функций, которые «выбирают» компоненты связности. ■

Это приводит к следующему алгоритму кластеризации. Вычислим собственные векторы и собственные значения \mathbf{L} , и пусть \mathbf{U} – матрица размера $N \times K$, столбцами которой являются K собственных векторов с наименьшими собственными значениями. (Быстрые методы вычисления «нижних» собственных векторов обсуждаются в работе [YHJ09]). Обозначим $\mathbf{u}_i \in \mathbb{R}^K$ i -ю строку \mathbf{U} . Так как векторы \mathbf{u}_i кусочно-постоянны, мы можем применить к ним кластеризацию методом K средних (раздел 21.3) и найти компоненты связности. (Заметим, что векторы \mathbf{u}_i – те же самые, что получаются при вычислении лапласовых собственных отображений в разделе 20.4.9.)

В реальных данных может не быть такой строго блочной структуры, но, пользуясь результатами из теории возмущений, можно показать, что собственные векторы «возмущенного» лапласиана близки к этим идеальным индикаторным функциям [NJW01].

На практике важно нормировать лапласиан графа, чтобы учесть тот факт, что некоторые вершины имеют больше связей, чем остальные. Сделать это можно (следуя предложению из работы [NJW01]), создав симметричную матрицу:

$$\mathbf{L}_{\text{sym}} \triangleq \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}. \quad (21.30)$$

Теперь собственное подпространство, соответствующее собственному значению 0, натянуто на $\mathbf{D}^{1/2} \mathbf{1}_{S_K}$. Это приводит к следующему алгоритму: найти K собственных векторов \mathbf{L}_{sym} с наименьшими собственными значениями, собрать из них матрицу \mathbf{U} , нормировать каждую строку на единичную норму, вычислив $t_{ij} = u_{ij} / \sqrt{\sum_k u_{ik}^2}$, и таким образом получить матрицу \mathbf{T} . Затем кластеризовать строки \mathbf{T} методом K средних, после чего вывести разбиение исходных точек.

21.5.3. Пример

На рис. 21.19 показан этот метод в действии. На рис. 21.19а мы видим, что метод K средних плохо справляется с кластеризацией из-за неявного предположения о том, что каждый кластер описывается сферическим гауссовым распределением. Затем мы попробовали применить к тем же данным спектральную кластеризацию. Мы вычислили плотную матрицу сходства \mathbf{W} , вос-

пользовавшись гауссовым ядром, $W_{ij} = \exp\left(-\frac{1}{2\sigma^2}\|x_i - x_j\|_2^2\right)$. Затем вычислили первые два собственных вектора нормированного лапласиана L_{sym} . После этого произвели кластеризацию методом К средних с $K = 2$; результаты показаны на рис. 21.19б.

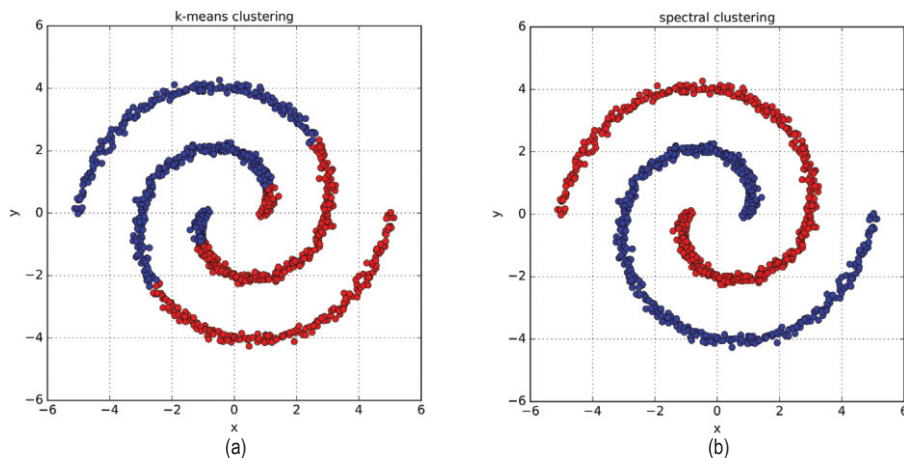


Рис. 21.19 ❖ Результаты кластеризации данных. (а) Методом К средних. (б) Методом спектральной кластеризации. Построено программой по адресу figures.probl.ai/book1/21.19

21.5.4. Связь с другими методами

Спектральная кластеризация тесно связана с несколькими другими методами обучения без учителя; некоторые из этих связей мы обсудим ниже.

21.5.4.1. Связь с kPCA

Спектральная кластеризация тесно связана с ядерным методом PCA (раздел 20.4.6). Именно, в kPCA используются наибольшие собственные векторы W , что эквивалентно наименьшим собственным векторам $I - W$. Это похоже на описанный выше метод, в котором вычисляются наименьшие собственные векторы $L = D - W$. Детали см. в работе [Ven+04a]. На практике спектральная кластеризация обычно дает лучшие результаты, чем kPCA.

21.5.4.2. Связь с анализом случайного блуждания

На практике можно улучшить результаты, вычислив собственные векторы нормированного лапласиана графа. Один из способов нормировки, используемый в работах [SM00; Mei01], заключается в том, чтобы определить

$$L_{rw} \triangleq D^{-1}L = I - D^{-1}W. \quad (21.31)$$

Можно показать, что для \mathbf{L}_{rw} , собственное подпространство, соответствующее собственному значению 0, тоже натянуто на индикаторные векторы $\mathbf{1}_{S_K}$ [Lux07], поэтому мы можем произвести кластеризацию непосредственно по K наименьшим собственным векторам \mathbf{U} .

Существует интересная связь между этим подходом и случайными блужданиями в графе. Сначала отметим, что $\mathbf{P} = \mathbf{D}^{-1}\mathbf{W} = \mathbf{I} - \mathbf{L}_{rw}$ – стохастическая матрица, а $p_{ij} = w_{ij}/d_i$ можно интерпретировать как вероятность перехода из i в j . Если граф связный и не двудольный, то он обладает единственным стационарным распределением $\boldsymbol{\pi} = (\pi_1, \dots, \pi_N)$, где $\pi_i = d_i/\text{vol}(V)$, а $\text{vol}(V) = \sum_i d_i$ – сумма степеней всех вершин. Можно также показать, что для разбиения на две части

$$\text{Ncut}(S, \bar{S}) = p(S|\bar{S}) + p(\bar{S}|S). \quad (21.32)$$

Это означает, что мы ищем такой разрез, для которого случайное блуждание чаще совершает переходы на похожие вершины и редко переходы из S в \bar{S} или обратно. Это анализ можно обобщить на случай $K > 2$; детали см. в работе [Mei01].

21.6. Бикластеризация*

Иногда имеется матрица данных $\mathcal{X} \in \mathbb{R}^{N \times D}$, и мы хотим кластеризовать строки и столбцы; это называется **бикластеризацией**, или **совместной кластеризацией**. Она широко используется в биоинформатике, где строки часто представляют гены, а столбцы – условия. Также она находит применения в коллаборативной фильтрации, где строки представляют пользователей, а столбцы – фильмы.

Были предложены различные ситуативные методы бикластеризации; см. обзор [MO04]. В разделе 21.6.1 мы представим простую вероятностную порождающую модель, которая сопоставляет идентификатор латентного кластера каждой строке и идентификатор другого латентного кластера каждому столбцу. В разделе 21.6.2 мы обобщим ее на случай, когда каждая строка может принадлежать нескольким кластерам в зависимости от того, какие группы признаков (столбцов) используются для определения различных групп объектов (строк).

21.6.1. Базовая бикластеризация

В этом разделе описана простая вероятностная порождающая модель бикластеризации, основанная на работе [Kem+06] (см. также родственный подход в работе [SMM03]). Идея заключается в том, чтобы ассоциировать с каждой строкой и с каждым столбцом латентный индикатор $u_i \in \{1, \dots, N_u\}$, $v_j \in \{1, \dots, N_v\}$, где N_u – число кластеров строк, а N_v – число кластеров столбцов. Мы используем следующую порождающую модель:

$$p(\mathbf{U}) = \prod_{i=1}^{N_r} \text{Unif}(u_i | \{1, \dots, N_u\}); \quad (21.33)$$

$$p(\mathbf{V}) = \prod_{j=1}^{N_c} \text{Cat}(v_j | \{1, \dots, N_v\}); \quad (21.34)$$

$$p(\mathbf{X} | \mathbf{U}, \mathbf{V}, \boldsymbol{\theta}) = \prod_{i=1}^{N_r} \prod_{j=1}^{N_c} p(X_{ij} | \boldsymbol{\theta}_{u_i, v_j}), \quad (21.35)$$

где $\boldsymbol{\theta}_{a,b}$ – параметры кластера строк a и кластера столбцов b .

На рис. 21.20 показан простой пример. Данные имеют вид $X_{ij} = 1$ тогда и только тогда, когда животное i имеет признак j , где $i = 1 \dots 50, j = 1 \dots 85$. Среди животных имеются киты, медведи, лошади и т. д. Признаками являются свойства среды обитания (джунгли, дерево, прибрежная линия), анатомические свойства (имеет зубы, четвероногое), поведенческие свойства (плавает, питается мясом) и т. д. Метод выявил 12 кластеров животных и 33 кластера признаков. (В работе [Kem+06] для вывода числа кластеров используется байесовский непараметрический метод). Например, кластер O2 имеет вид { антилопа, лошадь, жираф, зебра, олень } и характеризуется кластерами признаков F2 = { копыта, длинная шея, рога } и F6 = { ходит, четвероногое, наземное }, тогда как кластер O4 имеет вид { гиппопотам, слон, носорог } и характеризуется кластерами признаков F4 { приземистое тело, медленно двигается, неактивно } и F6.

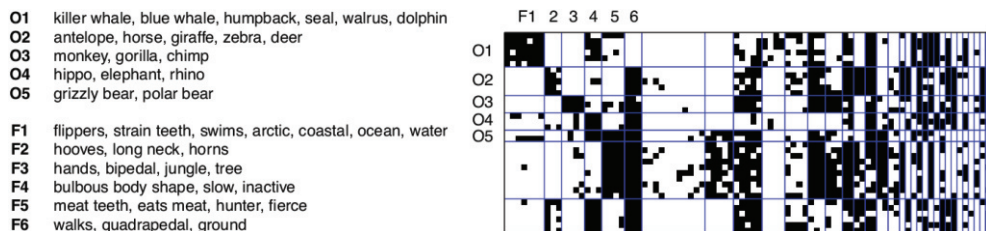


Рис. 21.20 ❖ Бикластеризация. Показаны 5 из 12 кластеров организмов и 6 из 33 кластеров признаков. Показана также исходная матрица данных, разбитая на части в соответствии с выявленными кластерами. На основе рис. 3 из работы [Kem+06]. Печатается с разрешения Чарльза Кемпа

21.6.2. Модели вложенного разбиения (Crosscat)

Проблема базовой бикластеризации (раздел 21.6.1) состоит в том, что каждый объект (строка) может принадлежать только одному кластеру. Интуитивно понятно, что у объекта может быть несколько ролей, так что его можно отнести к разным кластерам в зависимости от того, какое подмножество признаков используется. Например, в наборе данных о животных мы иногда хотим группировать их на основе анатомических признаков (например,

млекопитающие теплокровные, а рептилии нет), а иногда на основе поведенческих признаков (например, хищник или добыча).

Далее мы опишем модель, способную уловить это явление. Проиллюстрируем метод на примере. Пусть имеется матрица 6×6 , содержащая $N_u = 2$ кластеров строк и $N_v = 3$ кластеров столбцов. Предположим далее, что латентные группировки столбцов таковы: $\mathbf{v} = [1, 1, 2, 3, 3, 3]$. Это означает, что столбцы 1 и 2 входят в группу 1, столбец 3 в группу 2, а столбцы 4–6 в группу 3. Для столбцов, отнесенных к кластеру 1, строки кластеризуются следующим образом: $\mathbf{u}_{:,1} = [1, 1, 1, 2, 2, 2]$. Для столбцов, отнесенных к кластеру 2, строки кластеризуются следующим образом: $\mathbf{u}_{:,2} = [1, 1, 2, 2, 2, 2]$, а для столбцов, отнесенных к кластеру 3, строки кластеризуются следующим образом: $\mathbf{u}_{:,3} = [1, 1, 1, 1, 1, 2]$. Получающееся разбиение показано на рис. 21.21b. Мы видим, что кластеризация строк зависит от того, какая группа столбцов нам интересна в данный момент.

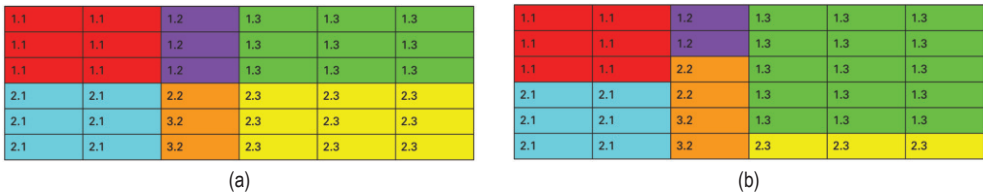


Рис. 21.21 ❖ (a) Пример бикластеризации. Каждая строка и каждый столбец отнесены к единственному кластеру. (b) Пример мультикластеризации с использованием модели вложенного разбиения. Строки могут принадлежать разным кластерам в зависимости от того, какое подмножество признаков столбцов нам интересно

Формально модель можно определить следующим образом:

$$p(\mathbf{U}) = \prod_{i=1}^{N_r} \prod_{l=1}^{N_c} \text{Unif}(u_{il} | \{1, \dots, N_u\}); \quad (21.36)$$

$$p(\mathbf{V}) = \prod_{j=1}^{N_c} \text{Unif}(v_j | \{1, \dots, N_v\}); \quad (21.37)$$

$$p(\mathbf{Z} | \mathbf{U}, \mathbf{V}) = \prod_{i=1}^{N_r} \prod_{j=1}^{N_c} \mathbb{I}(Z_{ij} = (u_{i,v_j}, v_j)); \quad (21.38)$$

$$p(\mathbf{X} | \mathbf{Z}, \boldsymbol{\theta}) = \prod_{i=1}^{N_r} \prod_{j=1}^{N_c} p(X_{ij} | \boldsymbol{\theta}_{z_{ij}}), \quad (21.39)$$

где $\boldsymbol{\theta}_{k,l}$ – параметры совместного кластера $k \in \{1, \dots, N_u\}$, $l \in \{1, \dots, N_v\}$.

Эта модель была независимо предложена в работах [Sha+06; Man+16], где называется **crosscat** (от cross-categorization – перекрестная категоризация), в работах [Gua+10; CFD10], где названа **мультикластеризацией** (multi-clust), и в работе [RG11], где названа **вложенным разбиением** (nested partitioning).

Во всех этих статьях авторы предлагают использовать процессы Дирихле, чтобы не оценивать число кластеров. Здесь мы предполагаем, что число кластеров известно и для простоты обозначений показываем параметры явно.

На рис. 21.22 приведены результаты применения модели к бинарным данным, содержащим 22 животных и 106 признаков. Показано (приближенное) разбиение MAP. Первая группа столбцов содержит таксономические признаки, например: «имеет кости», «теплокровное», «откладывает яйца» и т. д. По ним животные разбиваются на птиц, рептилий и амфибий, млекопитающих и беспозвоночных. Вторая группа столбцов содержит признаки, трактуемые как шум и не имеющие очевидной структуры (кроме единственной строки с меткой «лягушка»). Третья группа столбцов содержит экологические признаки: «опасное», «плотоядное», «живет в воде» и т. д. По ним животные разбиваются на добычу, наземных хищников, морских хищников и воздушных хищников. Таким образом, каждое животное (строка) может принадлежать разным кластерам в зависимости от того, какая группа признаков рассматривается.

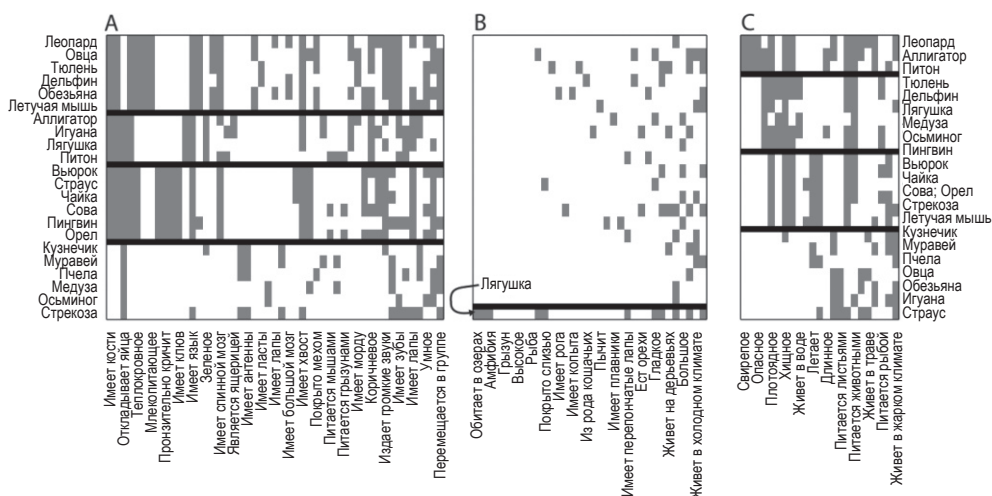


Рис. 21.22 ❖ Оценка MAP, порожденная моделью crosscat применительно к матрице бинарной категоризации животных (строки) по признакам (столбцы). Детали см. в тексте. На основе рис. 7 из работы [Sha+06]. Печатается с разрешения Викаша Мансикха

Глава 22

Рекомендательные системы

Рекомендательные системы предназначены для рекомендации чего-то (фильмов, книг, рекламных объявлений) пользователям на основе информации о прошлых просмотрах или потребительском поведении (например, каким фильмам они ставили высокие или низкие оценки, по каким объявлениям щелкали и т. п.), а также факультативной «побочной информации», например демографических данных или данных о собственных свойствах предмета (названии, жанре или цене). Подобные системы широко используются различными интернет-компаниями, в том числе Facebook, Amazon, Netflix, Google и т. д. В этой главе мы дадим краткий обзор темы. Дополнительные сведения можно найти, например, в работах [DKK12; Pat12; Yan+14; AC16; Agg16; Zha+19b].

22.1. ЯВНАЯ ОБРАТНАЯ СВЯЗЬ

В этом разделе мы рассмотрим простейшую постановку, когда пользователь предоставляет системе **явную обратную связь** в виде **рейтинга**, например $+1$ или -1 (нравится / не нравится) либо оценки от 1 до 5. Обозначим $Y_{ui} \in \mathbb{R}$ оценку, поставленную пользователем u предмету i . Мы можем представить рейтинги матрицей $M \times N$, где M – число пользователей, а N – число предметов. Как правило, эта матрица очень велика, но разрежена, так как большинство пользователей не высказывается по поводу большинства предметов (см. пример на рис. 22.1а). Эту разреженную матрицу можно также рассматривать как двудольный граф, в котором ребру $u - i$ назначен вес Y_{ui} . Это отражает тот факт, что мы имеем дело с **реляционными данными**, т. е. у значений u и i нет никакого внутреннего смысла (это просто произвольные индексы), а важно лишь, что u и i соединены.

Если Y_{ui} отсутствует, то причиной может быть одно из двух: либо пользователь u вообще ничего не знает о предмете i , либо знает, что он ему не понравился бы, поэтому даже не стал с ним связываться. В первом случае отсутствие данных – случайный факт, а во втором отсутствие несет информацию об истинном значении Y_{ui} . (Дальнейшее обсуждение этого момента

см., например, в работе [Mar+11].) Для простоты будем предполагать, что отсутствие данных ни о чем не говорит.

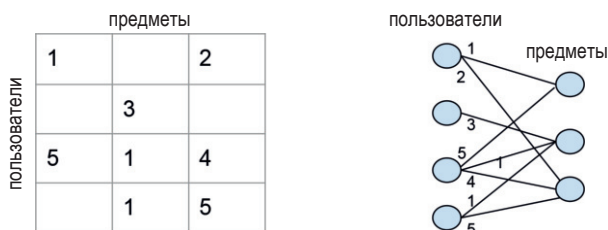


Рис. 22.1 ❖ Пример реляционного набора данных, представленных разреженной матрицей (слева) или разреженным двудольным графом (справа). Значения, соответствующие пустым ячейкам (отсутствующим ребрам), неизвестны. Строки 3 и 4 похожи и означают, что у пользователей 3 и 4 схожие предпочтения, поэтому данные от пользователя 3 можно использовать для предсказания предпочтений пользователя 4. Однако предпочтения пользователя 1 совершенно иные; он, похоже, низко оценивает все предметы. Для пользователя 2 данных очень мало, поэтому делать надежные предсказания трудно

22.1.1. Наборы данных

Знаменитый пример матрицы явных оценок раскрыла для публики стриминговая компания Netflix. В 2006 году она опубликовала большой набор данных, содержащий 100 480 507 оценок (от 1 до 5), поставленных 480 189 пользователями 17 770 фильмам. Несмотря на большой размер обучающего набора, матрица оценок на 99 % разрежена (оценки не проставлены). Опубликовав данные, компания предложили награду в 100 000 долл., получившую известность как **Netflix Prize**, любой группе, которая сможет предсказать истинные оценки для тестового набора пар (пользователь, фильм) точнее, чем собственная система Netflix. 21 сентября 2009 года награду получила группа «Pragmatic Chaos». Они применили целый ансамбль различных методов, как описано в работах [Kor09; BK07; FHK12]. Однако важнейшим компонентом ансамбля был метод, описанный в разделе 22.1.3.

К сожалению, данные Netflix больше недоступны по соображениям конфиденциальности. Но группа MovieLens из Миннесотского университета опубликовала анонимизированный набор данных с оценками фильмов по шкале от 1 до 5, который можно использовать для исследований [HK15]. Имеются также и другие наборы данных с явными оценками, например набор анекдотов **Jester** [Gol+01] и набор данных **BookCrossing** об обмене книгами [Zie+05].

22.1.2. Коллаборативная фильтрация

Оригинальный подход к проблеме рекомендаций называется **коллаборативной фильтрацией** [Gol+92]. Идея заключается в том, что, рекомендуя предметы, пользователи сотрудничают, поскольку делятся своими оценками

с другими пользователями; если u хочет что-то узнать о i , он может посмотреть, какие оценки ставили i другие пользователи u' , и вычислить взвешенное среднее:

$$\hat{Y}_{ui} = \sum_{u': Y_{u',i} \neq ?} \text{sim}(u, u') Y_{u',i}, \quad (22.1)$$

где предполагается, что $Y_{u',i} = ?$, если элемент неизвестен. При традиционном подходе сходство двух пользователей определяется путем сравнения множеств $S_u = \{Y_{u,i} \neq ? : i \in \mathcal{I}\}$ и $S_{u'} = \{Y_{u',i} \neq ? : i \in \mathcal{I}\}$, где \mathcal{I} – множество предметов. Однако разреженность данных может стать препятствием. В разделе 22.1.3 мы обсудим подход, основанный на обучении плотных векторов погружения для каждого предмета и каждого пользователя, который позволяет вычислять сходство в пространстве признаков низкой размерности.

22.1.3. Матричная факторизация

Мы можем рассматривать задачу о рекомендациях как задачу о **дополнении матрицы**, в которой мы хотим предсказать все отсутствующие элементы \mathbf{Y} . Ее можно сформулировать как следующую задачу оптимизации:

$$\mathcal{L}(\mathbf{Z}) = \sum_{ij: Y_{ij} \neq ?} (Z_{ij} - Y_{ij})^2 = \|\mathbf{Z} - \mathbf{Y}\|_F^2. \quad (22.2)$$

Однако это недоопределенная задача, потому что способов заполнить отсутствующие элементы \mathbf{Z} бесконечно много.

Необходимо добавить какие-то ограничения. Предположим, к примеру, что матрица \mathbf{Y} низкого ранга. Это можно записать в виде $\mathbf{Z} = \mathbf{U}\mathbf{V}^T \approx \mathbf{Y}$, где \mathbf{U} – матрица $M \times K$, \mathbf{V} – матрица $N \times K$, K – ранг матрицы, M – число пользователей, а N – число предметов. Это соответствует предсказанию вида

$$\hat{y}_{ui} = \mathbf{u}_u^T \mathbf{v}_i. \quad (22.3)$$

Это называется **матричной факторизацией**.

Если наблюдаемы все элементы Y_{ij} , то оптимальную \mathbf{Z} можно найти методом СГС (раздел 7.5). Но если некоторые элементы \mathbf{Y} отсутствуют, то соответствующая целевая функция уже не выпукла и не имеет единственного оптимума [SJ03]. Мы можем найти приближенное решение методом **чередующихся наименьших квадратов** (alternating least squares – **ALS**), когда сначала оценивается \mathbf{U} при известной \mathbf{V} , а затем \mathbf{V} при известной \mathbf{U} (детали см., например, в [KBV09]). Или же можно просто использовать СГС.

На практике важно также учесть базовые уровни для пользователей и предметов, записав ограничение:

$$\hat{y}_{ui} = \mu + b_u + c_i + \mathbf{u}_u^T \mathbf{v}_i. \quad (22.4)$$

Оно улавливает тот факт, что есть пользователи, которые всегда ставят низкие оценки, а также такие, которые ставят высокие оценки; кроме того,

у некоторых предметов (например, очень популярных фильмов) могут быть необычно высокие оценки.

Дополнительно можно добавить ℓ_2 -регуляризацию параметров, и в результате получается целевая функция:

$$\mathcal{L}(\theta) = \sum_{ij: Y_{ij} \neq ?} (y_{ij} - \hat{y}_{ij})^2 + \lambda(b_u^2 + c_i^2 + \|\mathbf{u}_u\|^2 + \|\mathbf{v}_i\|^2). \quad (22.5)$$

Ее можно оптимизировать методом СГС, произведя случайный выбор элемента (u, i) из множества наблюдаемых значений и выполнив следующие обновления:

$$b_u = b_u + \rho(e_{ui} - \lambda b_u); \quad (22.6)$$

$$c_i = c_i + \rho(e_{ui} - \lambda b_i); \quad (22.7)$$

$$\mathbf{u}_u = \mathbf{u}_u + \rho(e_{ui}\mathbf{u}_u - \lambda \mathbf{u}_u); \quad (22.7)$$

$$\mathbf{v}_i = \mathbf{v}_i + \rho(e_{ui}\mathbf{v}_i - \lambda \mathbf{v}_i), \quad (22.9)$$

где $e_{ui} = y_{ui} - \hat{y}_{ui}$ – ошибка, а $\rho \geq 0$ – скорость обучения. Этот подход, предложенный Саймоном Фанком, был одним из первых, показавших хорошие результаты на раннем этапе объявленного Netflix конкурса¹.

22.1.3.1. Вероятностная матричная факторизация

Матричную факторизацию можно преобразовать в вероятностную модель, положив

$$p(y_{ui} = y) = \mathcal{N}(y | \mu + b_u + c_i + \mathbf{u}_u^\top \mathbf{v}_i, \sigma^2). \quad (22.10)$$

Это называется **вероятностной матричной факторизацией** (probabilistic matrix factorization – **PMF**) [SM08]. Отрицательное логарифмическое правдоподобие этой модели эквивалентно целевой функции матричной факторизации (22.2). Однако вероятностный взгляд упрощает обобщение модели. Например, мы можем отразить тот факт, что оценки – целые, а не вещественные числа (чаще всего нули), воспользовавшись распределением Пуассона или отрицательным биномиальным распределением (см., например, [GOF18]). Это похоже на метод PCA экспоненциального семейства (раздел 20.2.7) с тем отличием, что строки и столбцы обрабатываются симметрично.

22.1.3.2. Пример: Netflix

Допустим, что мы применяем PMF к набору данных Netflix, используя $K = 2$ латентных фактора. На рис. 22.2 показаны обученные векторы погружения \mathbf{u}_i для нескольких фильмов. В левой части диаграммы находятся фильмы ужа-

¹ <https://sifter.org/~simon/journal/20061211.html>.

сов и низкопробный юмор («Непропеченный», «Фредди против Джейсона»), а в правой части – более серьезные драмы («Софи делает выбор», «Во власти Луны»). В верхней части находятся тепло принятые критиками независимые фильмы («Любовь, сбивающая с ног», «Взломщики сердец»), а в нижней части – мейнстримовское кино («Армагеддон», «Сбежавшая невеста»). «Волшебник страны Оз» расположен точно посередине, поскольку это в некотором смысле «средний фильм».

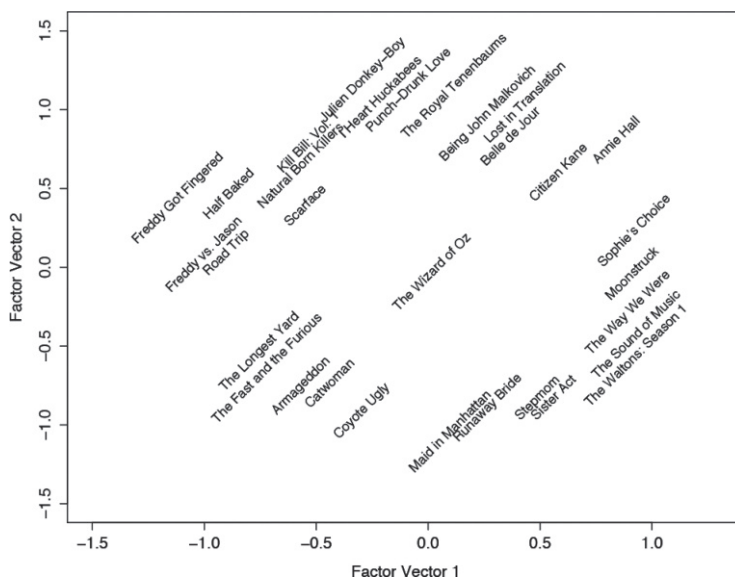


Рис. 22.2 ❖ Первые два латентных фактора фильмов, оцененные по данным из конкурса Netflix. Каждый фильм j показан в позиции, определяемой v_j . Детали см. в тексте. На основе рис. 3 из работы [KBV09]. Печатается с разрешения Иегуды Корена

Пользователи погружены в то же пространство, что и фильмы. Раз так, то мы можем предсказать оценку для любой пары (пользователь, фильм), используя близость в латентном пространстве погружения.

22.1.3.3. Пример: MovieLens

Теперь применим PMF к набору данных MovieLens-1M, содержащему 6040 пользователей, 3706 фильмов и 1 000 209 оценок. Будем использовать $K = 50$ факторов. Для простоты обучим модель, применив сингулярное разложение к плотной матрице оценок, в которой отсутствующие элементы заменены 0. (Это всего лишь аппроксимация, призванная упростить демонстрационный код.) На рис. 22.3 показан фрагмент истинной и предсказанной матрицы оценок. (Мы «обрубили» предсказания, так чтобы они попали в диапазон [1,5].) Как видим, модель не особенно точна, но улавливает некоторые структурные особенности данных.

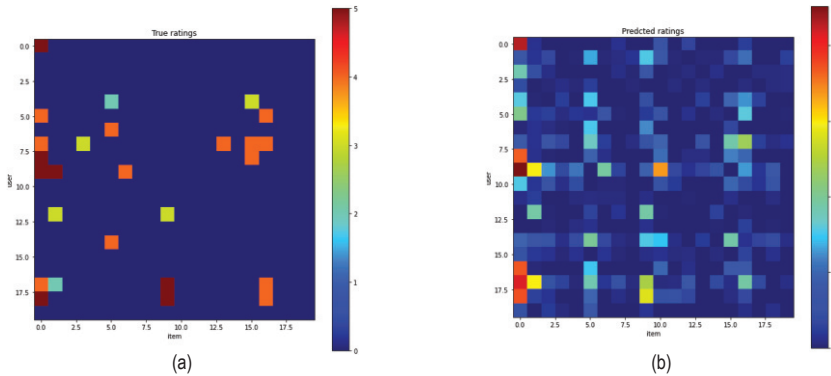


Рис. 22.3 ❖ (а) Фрагмент наблюдаемой матрицы оценок для набора данных MovieLens-1M. (б) Предсказания, полученные в результате сингулярного разложения с 50 латентными компонентами. Построено программой по адресу figures.probl.ai/book1/22.3

Кроме того, качественно она ведет себя вполне разумно. Например, на рис. 22.4 показаны 10 лучших фильмов, по мнению данного пользователя, а также 10 лучших предсказаний для фильмов, которые он не смотрел. Похоже, модель «вычислила» предпочтения этого пользователя. Например, мы видим, что многие предсказанные фильмы относятся к жанру боевиков или фильмов-нуар, и оба эти жанра присутствуют в собственном списке лучших фильмов пользователя, хотя на этапе обучения модели информация о жанре явно не указывалась.

22.1.4. Автокодировщики

Матричная факторизация – (би)линейная модель. Нелинейную версию можно получить, воспользовавшись автокодировщиками. Обозначим $\mathbf{y}_{:,i} \in \mathbb{R}^M$ – i -й столбец матрицы оценок, в которой неизвестные оценки заменены нулями. Этот вектор оценок можно предсказать с помощью автокодировщика вида

$$f(\mathbf{y}_{:,i}; \boldsymbol{\theta}) = \mathbf{W}^T \varphi(\mathbf{V} \mathbf{y}_{:,i} + \boldsymbol{\mu}) + \mathbf{b}, \quad (22.11)$$

где $\mathbf{V} \in \mathbb{R}^{KM}$ отображает оценки в пространство погружения, $\mathbf{W} \in \mathbb{R}^{KM}$ отображает пространство погружения в распределение оценок, $\boldsymbol{\mu} \in \mathbb{R}^K$ – смещения скрытых блоков, $\mathbf{b} \in \mathbb{R}^M$ – смещения выходных блоков. Это называется (предметной) версией модели **AutoRec** [Sed+15]. Она имеет $2MK + M + K$ параметров. Существует также пользовательская версия, которую можно вывести аналогично и которая имеет $2NK + N + K$ параметров. (Для наборов данных MovieLens и Netflix авторы обнаружили, что предметная модель работает лучше.)

Мы можем обучить модель, обновляя только параметры, ассоциированные с наблюдаемыми элементами $\mathbf{y}_{:,i}$. Кроме того, можно прибавить ℓ_2 -регуляризатор к матрице весов и получить целевую функцию:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^N \sum_{u: y_{ui} \neq ?} (y_{u,i} - f(\mathbf{y}_{:,i}; \boldsymbol{\theta})_u)^2 + \frac{\lambda}{2} (\|\mathbf{W}\|_F^2 + \|\mathbf{V}\|_F^2). \quad (22.12)$$

	MovieID	Title	Genres
36	858	Godfather, The (1972)	Action Crime Drama
35	1387	Jaws (1975)	Action Horror
65	2028	Saving Private Ryan (1998)	Action Drama War
63	1221	Godfather: Part II, The (1974)	Action Crime Drama
11	913	Maltese Falcon, The (1941)	Film-Noir Mystery
20	3417	Crimson Pirate, The (1952)	Adventure Comedy Sci-Fi
34	2186	Strangers on a Train (1951)	Film-Noir Thriller
55	2791	Airplane! (1980)	Comedy
31	1188	Strictly Ballroom (1992)	Comedy Romance
28	1304	Butch Cassidy and the Sundance Kid (1969)	Action Comedy Western

(a)

	MovieID	Title	Genres
516	527	Schindler's List (1993)	Drama War
1848	1953	French Connection, The (1971)	Action Crime Drama Thriller
596	608	Fargo (1996)	Crime Drama Thriller
1235	1284	Big Sleep, The (1946)	Film-Noir Mystery
2085	2194	Untouchables, The (1987)	Action Crime Drama
1188	1230	Annie Hall (1977)	Comedy Romance
1198	1242	Glory (1989)	Action Drama War
897	922	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	Film-Noir
1849	1954	Rocky (1976)	Action Drama
581	593	Silence of the Lambs, The (1991)	Drama Thriller

(b)

Рис. 22.4 ❖ (a) 10 лучших фильмов (из 69), которым пользователь «837» поставил высокие оценки. (b) 10 лучших предсказаний (из 3637), сделанных алгоритмом. Построено программой по адресу figures.problm.ai/book1/22.4

Несмотря на простоту метода, авторы обнаружили, что он работает лучше более сложных методов, например ограниченных машин Больцмана (RBM, [SMH07]) и локальной аппроксимации матрицей низкого ранга (LLORMA, [Lee+13]).

22.2. НЕЯВНАЯ ОБРАТНАЯ СВЯЗЬ

До сих пор мы предполагали, что пользователь явно проставляет оценки всем предметам, с которыми имел дело. Это очень ограничительное предположение. В общем случае мы хотели бы обучиться **неявной обратной связи**, которую пользователи оставляют, просто взаимодействуя с системой. Например, можно рассматривать фильмы, просмотренные пользователем u , как положительные отзывы, а все остальные – как отрицательные. Тогда мы получим разреженную матрицу, состоящую только из положительных оценок.

Или можно считать тот факт, что пользователь смотрел фильм i , но не смотрел фильм j , неявным сигналом о предпочтении, отдаваемом i перед j .

Получающиеся данные можно представить в виде множества кортежей вида $y_n = (u, i, j)$, где (u, i) – положительная пара, а (u, j) – отрицательная (или непомеченная) пара.

22.2.1. Байесовское персонализированное ранжирование

Чтобы обучить модель на данных вида (u, i, j) , мы должны использовать **потерю ранжирования**, так чтобы модель ранжировала i перед j для пользователя u . Простой способ добиться этого – взять модель Бернулли вида:

$$p(y_n = (u, i, j) | \theta) = \sigma(f(u, i; \theta) - f(u, j; \theta)). \quad (22.13)$$

Если объединить ее с гауссовым априорным распределением θ , то получится следующая задача нахождения оценки максимальной апостериорной вероятности:

$$\mathcal{L}(\theta) = \sum_{(u, i, j) \in \mathcal{D}} \log \sigma(f(u, i; \theta) - f(u, j; \theta)) - \lambda \|\theta\|^2, \quad (22.14)$$

где $\mathcal{D} = \{(u, i, j) : i \in \mathcal{I}_u^+, j \in \mathcal{J} \setminus \mathcal{I}_u^+, \mathcal{I}_u^+ \text{ – множество всех предметов, выбранных пользователем } u, \text{ а } \mathcal{J} \setminus \mathcal{I}_u^+ \text{ – множество всех остальных предметов (которые пользователю не понравились или он их просто не видел). Это называется байесовским персонализированным ранжированием (Bayesian personalized ranking – BPR) [Ren+09].}\}$

Рассмотрим пример из книги [Zha+20, раздел 16.5]. Всего имеется 4 предмета, $\mathcal{J} = \{i_1, i_2, i_3, i_4\}$, и пользователь u решил взаимодействовать с $\mathcal{I}_u^+ = \{i_2, i_3\}$. В этом случае неявная матрица предпочтений одного предмета перед другим для пользователя u имеет вид:

$$\mathbf{Y}_u = \begin{pmatrix} . & + & + & ? \\ - & . & ? & - \\ - & ? & . & - \\ ? & + & + & . \end{pmatrix}, \quad (22.15)$$

где $Y_{u,i,i'} = +$ означает, что u отдает предпочтение i' перед i , $Y_{u,i,i'} = -$ – что пользователь предпочитает i , а не i' , а $Y_{u,i,i'} = ?$ – что мы не знаем, чему отдает предпочтение пользователь. Например, из второго столбца ясно, что этот пользователь оценивает i_2 выше, чем i_1 и i_4 , потому что он выбрал i_2 , а не i_1 и не i_4 ; однако мы не можем сказать, предпочитает ли он i_2 или i_3 .

Если множество возможных предметов велико, то количество отрицательных оценок в $\mathcal{J} \setminus \mathcal{I}_u^+$ может быть очень большим. По счастью, потерю можно аппроксимировать, произведя подвыборку отрицательных оценок.

Заметим, что альтернативой логарифмической потере (22.14) может служить кусочно-линейная потеря, похожая на принятую в методе опорных векторов (раздел 17.3). Она имеет вид:

$$\begin{aligned}\mathcal{L}(y_n = (u, i, j), f) &= \max(m - (f(u, i) - f(u, j)), 0) \\ &= \max(m - f(u, i) + f(u, j), 0),\end{aligned}\quad (22.16)$$

где $m \geq 0$ – безопасный зазор. Здесь мы пытаемся гарантировать, что отрицательные предметы j никогда не получают оценку, более чем на m превышающую оценку положительных предметов i .

22.2.2. Машины факторизации

Подход AutoRec из раздела 22.1.4 нелинеен, но пользователи и предметы рассматриваются асимметрично. В этом разделе мы обсудим более симметричную дискриминантную модель. Начнем с линейной версии. Основная идея – предсказать выход (например, оценку) для любой заданной пары (пользователь, предмет), $\mathbf{x} = [\text{one-hot}(u), \text{one-hot}(i)]$, с помощью функции:

$$f(\mathbf{x}) = \mu + \sum_{i=1}^D w_i x_i + \sum_{i=1}^D \sum_{j=i+1}^D (\mathbf{v}_i^T \mathbf{v}_j) x_i x_j, \quad (22.17)$$

где $\mathbf{x} \in \mathbb{R}^D$, $D = (M + N)$ – число входов, $\mathbf{V} \in \mathbb{R}^{D \times K}$ – матрица весов, $\mathbf{w} \in \mathbb{R}^D$ – вектор весов, а $\mu \in \mathbb{R}$ – глобальное смещение. Эта модель называется **машиной факторизации** (factorization machine – FM) [Ren12].

Член $(\mathbf{v}_i^T \mathbf{v}_j) x_i x_j$ измеряет взаимодействие между входными признаками i и j . Это обобщение модели матричной факторизации (22.4), поскольку допускается обработка других видов информации, содержащейся во входе \mathbf{x} , а не только о пользователе и предмете, как в разделе 22.3.

Для вычисления функции (22.17) требуется время $O(KD^2)$, поскольку рассматриваются все возможные попарные комбинации пользователя и предмета. Но выражение можно переписать, так что время вычислений сократится до $O(KD)$:

$$\sum_{i=1}^D \sum_{j=i+1}^D (\mathbf{v}_i^T \mathbf{v}_j) x_i x_j = \frac{1}{2} \sum_{i=1}^D \sum_{j=1}^D (\mathbf{v}_i^T \mathbf{v}_j) x_i x_j - \frac{1}{2} \sum_{i=1}^D (\mathbf{v}_i^T \mathbf{v}_i) x_i x_i \quad (22.18)$$

$$= -\frac{1}{2} \left(\sum_{i=1}^D \sum_{j=1}^D \sum_{k=1}^K v_{ik} v_{jk} x_i x_j - \sum_{i=1}^D \sum_{k=1}^K v_{ik} v_{ik} x_i x_i \right) \quad (22.19)$$

$$= -\frac{1}{2} \sum_{k=1}^K \left(\left(\sum_{i=1}^D v_{ik} x_i \right)^2 - \sum_{i=1}^D v_{ik}^2 x_i^2 \right). \quad (22.20)$$

Для разреженных векторов общая сложность линейно зависит от числа ненулевых компонентов. Поэтому если для идентификаторов пользователя и предмета применяется унитарное кодирование, то сложность составляет всего $O(K)$, как и в оригинальной целевой функции матричной факторизации (22.4).

Мы можем обучить эту модель минимизировать любую желаемую потерю. Например, если присутствует явная обратная связь, то можно выбрать потерю СКО, а в случае неявной обратной связи – потерю ранжирования. В работе [Guo+17] предложена модель **глубоких машин факторизации**, сочетающая

описанный выше метод с МСП применительно к конкатенации, а не скалярному произведению векторов погружения. Точнее, модель имеет вид:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \sigma(\text{FM}(\mathbf{x}) + \text{MLP}(\mathbf{x})). \quad (22.21)$$

Она тесно связано с **широкой и глубокой** моделью, предложенной в работе [Che+16]. Идея в том, что билинейная модель FM улавливает явные взаимодействия между конкретными пользователями и предметами (вариант запоминания), тогда как МСП улавливает неявные взаимодействия между признаками пользователей и признаками предметов, что открывает перед моделью возможность обобщения.

22.2.3. Нейронная матричная факторизация

В этом разделе мы опишем модель **нейронной матричной факторизации** из работы [He+17]. Это еще один способ объединить билинейные модели с глубокими нейронными сетями. Билинейная часть используется для того, чтобы определить обобщенную матричную факторизацию (generalized matrix factorization – GMF), при которой вычисляется следующий вектор признаков для пользователя u и предмета i :

$$\mathbf{z}_{ui}^1 = \mathbf{P}_{u,:} \odot \mathbf{Q}_{i,:}, \quad (22.22)$$

где $\mathbf{P} \in \mathbb{R}^{MK}$ – матрица погружений пользователей, а $\mathbf{Q} \in \mathbb{R}^{NK}$ – матрица погружений предметов. Нейронная часть сводится к применению МСП к конкатенации векторов погружения (с использованием других матриц погружения):

$$\mathbf{z}_{ui}^2 = \text{MLP}([\tilde{\mathbf{U}}_{u,:}, \tilde{\mathbf{V}}_{i,:}]). \quad (22.23)$$

Наконец, обе часть объединяются и получается

$$f(u, i; \boldsymbol{\theta}) = \sigma(\mathbf{w}^T[\mathbf{z}_{ui}^1, \mathbf{z}_{ui}^2]). \quad (22.24)$$

Смотрите иллюстрацию на рис. 22.5.

В работе [He+17] модель обучается на неявной обратной связи, когда $y_{ui} = 1$, если наблюдается взаимодействие пользователя u с предметом i , и $y_{ui} = 0$ в противном случае. Однако ее можно также обучить для минимизации потери BPR.

22.3. ИСПОЛЬЗОВАНИЕ ПОБОЧНОЙ ИНФОРМАЦИИ

До сих пор мы предполагали, что предиктору доступны только целые идентификаторы пользователей и предметов. Это сильно обедненное представление, которое к тому же не будет работать, если встретится новый пользователь или новый предмет (так называемая проблема **холодного старта**). Чтобы справиться с этим, нужно использовать «**побочную информацию**», а не только идентификаторы.

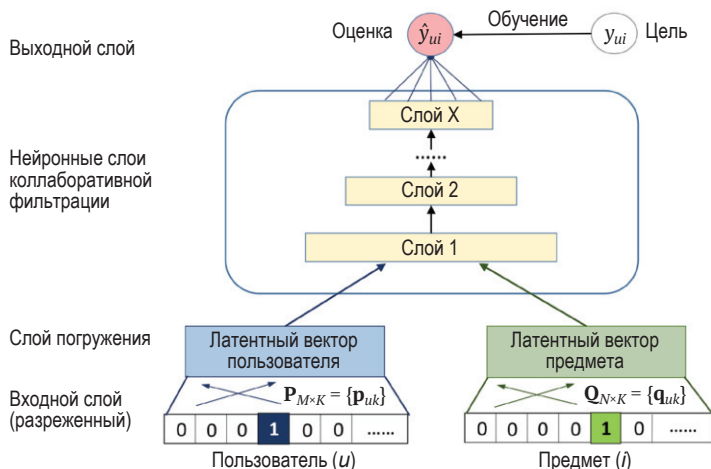


Рис. 22.5 ❖ Модель нейронной матричной факторизации.

На основе рис. 2 из работы [He+17].

Печатается с разрешения Сяньнан Хе

Существует много видов побочной информации. Для предметов часто используются подробные метаданные, как то: текст (например, название), изображения (например, обложка), многомерные категориальные переменные (например, местоположение) или просто скаляры (например, цена). Для пользователей состав доступной побочной информации зависит от вида интерактивной системы. В поисковых системах это список отправленных пользователем запросов и (для зарегистрированных пользователей) информация, собранная на сайтах, которые он посещал (отслеживается с помощью куков). На сайтах интернет-магазинов это список поисковых запросов плюс информация о прошлых просмотрах и покупках. На сайтах социальных сетей для каждого пользователя имеется информация о графе его друзей.

Эту побочную информацию очень легко собрать в машинах факторизации, если не ограничиваться в определении x унитарными векторами, а обобщить его, как показано на рис. 22.6. Такую же схему кодирования входов можно конечно использовать и для других видов моделей, например deepFM или neuralMF.

Помимо признаков пользователей и предметов, могут существовать и другие контекстуальные признаки, например время взаимодействия (скажем, день или вечер). Порядок (последовательность) недавно просмотренных предметов тоже может служить полезным сигналом. «Рекомендация на основе сверточного погружения последовательности» (Convolutional Sequence Embedding Recommendation), или модель **Caser**, предложенная в работе [TW18], улавливает это путем погружения последних M предметов, после чего вход размера $M \times K$ рассматривается как изображение, для чего в модель включается сверточный слой.

Для рекомендательных систем разработано также много других нейронных моделей (см., например, обзор [Zha+19b]).

		Feature vector x																Target y					
		User				Movie				Other Movies rated				Time	Last Movie rated								
		A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW		ST	...	TI	NH	SW	ST	...		
x_1		1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5	y_1
x_2		1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3	y_2
x_3		1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1	y_3
x_4		0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4	y_4
x_5		0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5	y_5
x_6		0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1	y_6
x_7		0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5	y_7

Рис. 22.6 ❖ Матрица плана для системы рекомендации фильмов. Показаны идентификаторы пользователя и фильма, а также побочная информация. На основе рис. 1 из работы [Ren12]. Печатается с разрешения Стефена Рендла

22.4. Компромисс между исследованием и использованием

Интересный поворот, присущий только рекомендательным системам и не характерный для других задач предсказания, – тот факт, что данные, на которых обучается система, представляют собой следствие рекомендаций, сделанных ее предыдущими версиями. Таким образом, образуется петля обратной связи [Bot+13]. Например, рассмотрим рекомендательную систему видео на YouTube [CAS16]. На сайте размещены миллионы видеороликов, поэтому система должна предложить пользователю короткий список роликов, помогающий найти то, что он хочет (см., например, [Ie+19]). Если пользователь решил просмотреть какой-то из них, то система может считать это положительной обратной связью, подтверждающей, что она дала хорошую рекомендацию, и соответственно обновить параметры модели. Но, быть может, есть какой-то ролик, который пользователю понравился бы еще больше? На этот вопрос невозможно ответить, если система не попытается предложить пользователю какие-то ролики, реакция на которые заранее неизвестна. Это пример **компромисса между исследованием и использованием**.

В дополнение к исследованию системе, возможно, придется довольно долго ждать, прежде чем станет ясно, были ли полезны предложенные изменения в рекомендациях. Для обучения политики, оптимизирующей долгосрочное вознаграждение, обычно используют **обучение с подкреплением**. Детали см. во втором томе этой книги, [Mur22].

Глава 23

Погружения графов*

Эта глава написана в соавторстве с Брайаном Пероцци, Сами Абу Эль Хайджа и Инес Чами.

23.1. ВВЕДЕНИЕ

Теперь переключим внимание на данные, в которых имеются семантические связи между обучающими примерами $\{\mathbf{x}_n\}_{n=1}^N$. Связи (называемые ребрами) соединяют обучающие примеры (вершины) с зависящей от приложения семантикой (обычно определяющей сходство). Теория графов предлагает математический аппарат для рассуждения о таких связях.

Графы – это универсальные структуры данных, которые позволяют представить сложные реляционные данные (состоящие из вершин и ребер). Они встречаются в таких разных предметных областях, как социальные сети, вычислительная химия [Gil+17], биология [Sta+06], рекомендательные системы [KSJ09], обучение с частичным привлечением учителя [GB18] и др.

Обозначим $\mathbf{A} \in \{0, 1\}^{N \times N}$ матрицу смежности, где N – число узлов, а $\mathbf{W} \in \mathbb{R}^{N \times N}$ – ее взвешенная версия. В обсуждаемых ниже методах иногда считается, что $\mathbf{W} = \mathbf{A}$, а иногда – что \mathbf{W} – некоторое преобразование \mathbf{A} , например построчная нормировка. Наконец, пусть $\mathbf{X} \in \mathbb{R}^{N \times D}$ – матрица вершинных признаков.

При проектировании и обучении модели нейронной сети на графовых данных мы хотим, чтобы метод был применим к вершинам, участвующим в различных графовых контекстах (например, имеющих разные связи и структуру сообщества). Сравните с моделью нейронной сети, проектируемой для изображений, где каждый пиксель (вершина) имеет одну и ту же структуру соседства. Тогда как в произвольном графе нет никакого регулярного выстраивания вершин, и структура соседства для разных вершин может радикально отличаться. Такое сравнение проведено на рис. 23.1. Следовательно, операции типа свертки в евклидовом пространстве просто так неприменимы к нерегулярным графам: евклидова свертка опирается на априорные геометрические предположения (например, инвариантность относительно сдвигов), которые не обобщаются на неевклидовы домены.

Эти проблемы привели к развитию исследований по **геометрическому глубокому обучению** (Geometric Deep Learning – GDL) [Bro+17b], нацелен-

ных на применение методов глубокого обучения к неевклидовым данным. В частности, ввиду широкого присутствия графов в реальных приложениях был отмечен всплеск интереса к применению методов машинного обучения к данным, имеющим графовую структуру. Так, методы **обучения графовых представлений** (Graph Representation Learning – GRL) [Cha+20] имеют целью обучение непрерывных низкоразмерных векторных представлений графовых данных; такие представления называются погружениями.

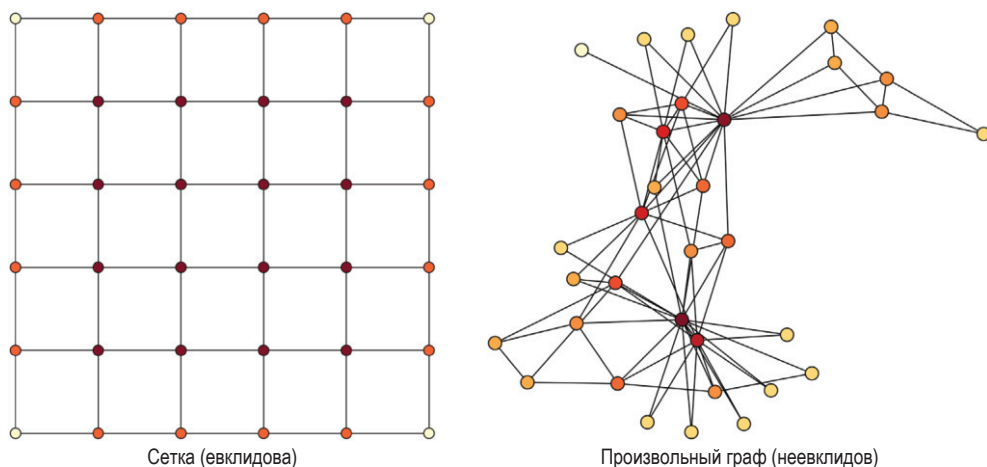


Рис. 23.1 ❖ Евклидовы и неевклидовы графы. Из работы [Cha+20]

Мы разделим задачи GRL на два класса: **без учителя** и **с учителем** (или с частичным привлечением учителя). В первом случае целью является обучение низкоразмерных евклидовых представлений, оптимизирующих некоторую целевую функцию, например сохраняющих структуру входного графа. Во втором случае тоже требуется обучить низкоразмерные евклидовы представления, но в расчете на конкретную последующую задачу предсказания, например классификацию вершин или графов. Кроме того, структура графа может быть фиксирована на протяжении обучения и тестирования – это называется **трандуктивным обучением** (например, предсказать свойства пользователя в большой социальной сети), или же ожидается, что модель будет отвечать на вопросы о графах, которые не предъявлялись во время обучения, – это называется **индуктивным обучением** (например, классификация молекулярных структур). Наконец, хотя большинство методов с учителем и без учителя обучают представления в евклидовых векторных пространствах, недавно обнаружился интерес к **обучению неевклидовых представлений**, например в пространствах погружений с гиперболической или сферической геометрией. Цель таких работ – использовать непрерывное пространство погружения, структура которого имеет сходство с дискретной структурой вкладываемых входных данных (например, гиперболическое пространство – это непрерывная версия деревьев [Sar11]).

23.2. ПОГРУЖЕНИЕ ГРАФА КАК ЗАДАЧА О КОДИРОВЩИКЕ И ДЕКОДЕРЕ

Существует много подходов к GRL, но в основном все методы следуют общему образцу. Сначала вход сети (вершинные признаки $\mathbf{X} \in \mathbb{R}^{N \times D}$ и ребра графа из множества \mathbf{A} или $\mathbf{W} \in \mathbb{R}^{N \times N}$) кодируются, т. е. переводятся из дискретного домена графа в непрерывное представление (погружение), $\mathbf{Z} \in \mathbb{R}^{N \times L}$. Затем обученное представление \mathbf{Z} используется для оптимизации некоторой целевой функции (например, реконструкции связей в графе). В этом разделе мы будем использовать модель кодировщика-декодера графа (GraphEDM), предложенную в работе Chami et al. [Cha+20], для анализа популярных семейств методов GRL.

Схема GraphEDM (рис. 23.2, [Cha+20]) предлагает общий подход, включающий широкий спектр методов погружения графов с учителем и без учителя, в том числе такие, где граф используется как регуляризатор (например, [ZG02]), позиционные погружения (например, [PARS14]) и графовые нейронные сети, например основанные на передаче сообщений [Gil+17; Sca+09] или свертке графов [Bru+14; KW16a]).

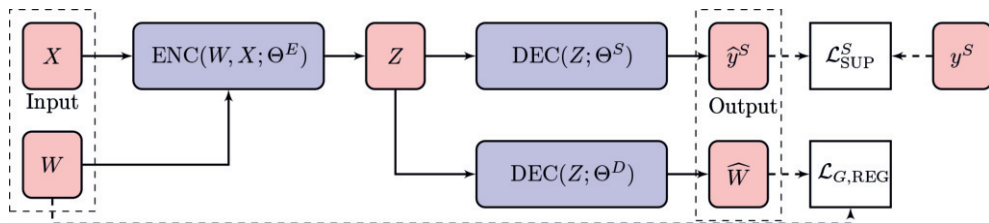


Рис. 23.2 ❖ Схема GraphEDM из работы Chami et al. [Cha+20]. В зависимости от имеющейся информации, предоставленной учителем, используются все или некоторые ветви. В частности, в методах без учителя для обучения не используется декодирование меток, а только оптимизируется декодер сходства (нижняя ветвь). С другой стороны, в методах с учителем и частичным привлечением учителя дополнительная информация используется для обучения параметров модели (верхняя ветвь). Печатается с разрешения авторов

Схема GraphEDM принимает на входе взвешенный граф $\mathbf{W} \in \mathbb{R}^{N \times N}$ и факультативные вершинные признаки $\mathbf{X} \in \mathbb{R}^{N \times D}$. В постановке с полным или частичным привлечением учителя предполагается, что имеются обучающие метки для вершин (обозначены N), ребер (обозначены E) и (или) для всего графа (обозначены G). Будем обозначать сигнал учителя $S \in \{N, E, G\}$.

Саму модель GraphEDM можно разложить на следующие компоненты:

- **сеть кодировщика графа** $\text{ENC}_{\theta^E} : \mathbb{R}^{N \times N} \times \mathbb{R}^{N \times D} \rightarrow \mathbb{R}^{N \times L}$ с параметрами θ^E , которая объединяет структуру графа с факультативными вершинными признаками для порождения матрицы погружения вершин $\mathbf{Z} \in \mathbb{R}^{N \times L}$ следующим образом:

$$\mathbf{Z} = \text{ENC}(\mathbf{W}, \mathbf{X}; \Theta^E). \quad (23.1)$$

Ниже мы увидим, что эта матрица погружения может улавливать различные свойства графа в зависимости от информации, предоставленной учителем на этапе обучения;

- **сеть декодера графа** $\text{DEC}_{\Theta^D} : \mathbb{R}^{N \times L} \rightarrow \mathbb{R}^{N \times N}$ с параметрами Θ^D , которая использует погружения вершин \mathbf{Z} для вычисления оценок сходства для всех пар вершин в матрице $\mathbf{W} \in \mathbb{R}^{N \times N}$ следующим образом:

$$\hat{\mathbf{W}} = \text{DEC}(\mathbf{Z}; \Theta^D). \quad (23.2)$$

- **сеть классификации** $\text{DEC}_{\Theta^S} : \mathbb{R}^{N \times L} \rightarrow \mathbb{R}^{N \times |\mathcal{Y}|}$, где \mathcal{Y} – пространство меток. Эта сеть используется при обучении с полным или частичным привлечением учителя и параметризована Θ^S . Выходом является распределение меток $\hat{\mathbf{y}}^S$, в котором используются погружения вершин:

$$\hat{\mathbf{y}}^S = \text{DEC}(\mathbf{Z}; \Theta^S). \quad (23.3)$$

Конкретный выбор сетей (кодировщика и декодера) позволяет GraphEDM реализовать различные методы погружения графов, как будет объяснено в следующих разделах.

Выходом модели, согласно схеме GraphEDM, является реконструированная матрица сходства графов $\hat{\mathbf{W}}$ (которая часто используется для обучения алгоритмов погружения без учителя) и (или) метки $\hat{\mathbf{y}}^S$ для обучения с учителем. Выходное пространство меток \mathcal{Y} зависит от приложения. Например, при классификации на уровне вершин $\hat{\mathbf{y}}^N \in \mathcal{Y}^N$, где \mathcal{Y} представляет пространство меток вершин. А при пометке на уровне ребер $\hat{\mathbf{y}}^E \in \mathcal{Y}^{N \times N}$, где \mathcal{Y} представляет пространство меток ребер. Отметим также, что возможны и другие виды пометки, например пометка на уровне графов (тогда должно быть $\hat{\mathbf{y}}^G \in \mathcal{Y}$, где \mathcal{Y} представляет пространство меток графов).

Наконец, необходимо задать потерю. Ее можно использовать для оптимизации параметров $\Theta = \{\Theta^E, \Theta^D, \Theta^S\}$. Модели на основе GraphEDM можно оптимизировать, применяя сочетание трех разных членов. Во-первых, член потери обучения с учителем, $\mathcal{L}_{\text{SUP}}^S$ сравнивает предсказанные метки $\hat{\mathbf{y}}^S$ с истинными метками \mathbf{y}^S . Во-вторых, член потери реконструкции графа, $\mathcal{L}_{G, \text{RECON}}$, может использовать структуру графа, чтобы наложить ограничения регуляризации на параметры модели. Наконец, член потери регуляризации, \mathcal{L}_{REG} , позволяет представлять априорные предположения о подлежащих обучению параметрах модели, чтобы предотвратить переобучение. Модели, основанные на схеме GraphEDM, обучаются путем минимизации следующей полной потери \mathcal{L} :

$$\mathcal{L} = \alpha(\mathbf{y}^S, \hat{\mathbf{y}}^S; \Theta) + \beta \mathcal{L}_{G, \text{RECON}}(\mathbf{W}, \hat{\mathbf{W}}; \Theta) + \gamma \mathcal{L}_{\text{REG}}(\Theta), \quad (23.4)$$

где α , β и γ – гиперпараметры, которые можно настроить или обнулить. Заметим, что методы погружения графов можно обучать с учителем ($\alpha \neq 0$) или без учителя ($\alpha = 0$). При обучении с учителем используется дополнительный источник информации для обучения погружений, например метки вершин или графов. С другой стороны, методы обучения без учителя опираются только на структуру графа и так обучают погружения вершин.

23.3. ПОВЕРХНОСТНЫЕ ПОГРУЖЕНИЯ ГРАФОВ

Методы поверхностного погружения являются трансдуктивными, а функция кодировщика отображает категориальные идентификаторы вершин в евклидово пространство с помощью матрицы погружения. Каждой вершине $v_i \in V$ соответствует обучаемый вектор погружения низкой размерности $\mathbf{Z}_i \in \mathbb{R}^L$, а функция поверхностного кодировщика имеет вид:

$$\mathbf{Z} = \text{ENC}(\theta^E) \triangleq \theta^E, \text{ где } \theta^E \in \mathbb{R}^{N \times L}. \quad (23.5)$$

Важно, что словарь погружения \mathbf{Z} обучается непосредственно как параметры модели. Если учителя нет, то погружения \mathbf{Z} оптимизируются с целью восстановления какой-то информации о входном графе (например, матрицы смежности \mathbf{W} или какого-то ее преобразования). Это напоминает методы понижения размерности типа PCA (раздел 20.1), но для графовых структур данных. При наличии учителя погружения оптимизируются с целью предсказания каких-то меток – для вершин, ребер и (или) графа в целом.

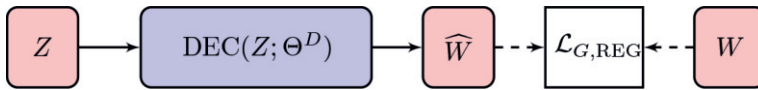


Рис. 23.3 ❖ Методы поверхностного погружения. Кодировщик представляет собой простой поиск в погружении, а графовая структура используется только в функции потерь. Печатается с разрешения авторов работы [Cha+20]

23.3.1. Обучение погружений без учителя

В случае обучения без учителя мы рассмотрим два основных типа методов поверхностного погружения графов: на основе расстояния и на основе внешнего произведения. Методы на основе расстояния оптимизируют словарь погружения $\mathbf{Z} = \theta^E \in \mathbb{R}^{N \times L}$, так что вершины i и j , близкие в графе (в смысле некоторой функции расстояния на графе), погружаются в \mathbf{Z} таким образом, что $d_2(\mathbf{Z}_i, \mathbf{Z}_j)$ мало, где $d_2(\cdot, \cdot)$ – функция расстояния между векторами погружения. Функцию $d_2(\cdot, \cdot)$ можно настраивать, получая евклидовы (раздел 23.3.2) или неевклидовы (раздел 23.3.3) погружения. Декодер выводит матрицу $\hat{\mathbf{W}} = \text{DEC}(\mathbf{Z}, \theta_D)$, где $\hat{W}_{ij} = d_2(\mathbf{Z}_i, \mathbf{Z}_j)$.

С другой стороны, некоторые методы опираются на попарные скалярные произведения при вычислении сходства вершин. Сеть декодера можно записать в виде $\hat{\mathbf{W}} = \text{DEC}(\mathbf{Z}, \theta_D) = \mathbf{Z}\mathbf{Z}^T$.

В обоих случаях погружения для методов, основанных на расстоянии и на произведении, обучаются без учителя посредством минимизации потери регуляризации графа:

$$\mathcal{L}_{G, \text{RECON}}(\mathbf{W}, \hat{\mathbf{W}}; \theta) = d_1(s(\mathbf{W}), \hat{\mathbf{W}}), \quad (23.6)$$

где $s(\mathbf{W})$ – факультативное преобразование матрицы смежности \mathbf{W} , а d_1 – функция расстояния между матрицами, которая не обязана совпадать с d_2 .

Как мы увидим, s , d_1 , d_2 можно выбрать многими разумными способами. Например, s может быть самой матрицей смежности, $s(\mathbf{W}) = \mathbf{W}$, или ее степенью, $s(\mathbf{W}) = \mathbf{W}^2$. Если входом является взвешенная бинарная матрица $\mathbf{W} = \mathbf{A}$, то можно положить $s(\mathbf{W}) = 1 - \mathbf{W}$, так что связанные вершины, для которых $A_{ij} = 1$, имеют вес (расстояние) 0.

23.3.2. На основе расстояния: евклидовы методы

Методы на основе расстояния минимизируют евклидовы расстояния между похожими (связанными) вершинами. Ниже будет приведено несколько примеров.

Многомерное шкалирование (ММШ, раздел 20.4.4) эквивалентно определению $s(\mathbf{W})$ как некоторой матрицы расстояний, измеряющей сходство между вершинами (например, пропорционально кратчайшим попарным расстояниям), и последующему определению:

$$d_1(s(\mathbf{W}), \hat{\mathbf{W}}) = \sum_{i,j} (s(\mathbf{W})_{ij} - \hat{W}_{ij})^2 = \|\mathbf{s}(\mathbf{W}) - \hat{\mathbf{W}}\|_F^2, \quad (23.7)$$

где $\hat{W}_{ij} = d_2(\mathbf{Z}_i, \mathbf{Z}_j) = \|\mathbf{Z}_i - \mathbf{Z}_j\|$ (хотя допустимы и другие метрики).

Лапласовы собственные отображения (раздел 20.4.9) обучают погружения путем решения обобщенной задачи на собственные значения:

$$\min_{\mathbf{Z} \in \mathbb{R}^{M \times d}} \mathbf{Z}^T \mathbf{L} \mathbf{Z} \text{ при условии } \mathbf{Z}^T \mathbf{D} \mathbf{Z} = \mathbf{I} \text{ и } \mathbf{Z}^T \mathbf{D} \mathbf{1} = 0, \quad (23.8)$$

где $\mathbf{L} = \mathbf{D} - \mathbf{W}$ – лапласиан графа (раздел 20.4.9.2), а \mathbf{D} – диагональная матрица, содержащая суммы по столбцам для каждой строки. Первое ограничение убирает произвольный масштабный коэффициент из погружения, а второе предотвращает тривиальные решения, соответствующие постоянному собственному вектору (с нулевым собственным значением для связных графов). Кроме того, заметим, что $\mathbf{Z}^T \mathbf{L} \mathbf{Z} = \frac{1}{2} \sum_{i,j} W_{ij} \|\mathbf{Z}_i - \mathbf{Z}_j\|_2^2$, и потому целевую функцию минимизации можно эквивалентно записать как член реконструкции графа:

$$d_1(s(\mathbf{W}), \hat{\mathbf{W}}) = \sum_{i,j} \mathbf{W}_{ij} \times \hat{\mathbf{W}}_{ij}; \quad (23.9)$$

$$\hat{\mathbf{W}}_{ij} = d_2(\mathbf{Z}_i, \mathbf{Z}_j) = \|\mathbf{Z}_i - \mathbf{Z}_j\|_2^2, \quad (23.10)$$

где $s(\mathbf{W}) = \mathbf{W}$.

23.3.3. На основе расстояния: неевклидовы методы

До сих пор мы обсуждали методы, в которых предполагается, что погружения находятся в евклидовом пространстве. Однако в недавних работах рассматривается пространство погружения графов с гиперболической геометрией.

Конкретно, гиперболические погружения идеальны для деревьев и предлагают захватывающую альтернативу евклидовой геометрии для графов с иерархической структурой. Ниже приведено несколько примеров.

В работе [NK17] производится обучение иерархических графов с использованием **модели Пуанкаре** гиперболического пространства. В нашей нотации представить ее просто, достаточно лишь заменить $d_2(\mathbf{Z}_i, \mathbf{Z}_j)$ функцией расстояния Пуанкаре:

$$d_2(\mathbf{Z}_i, \mathbf{Z}_j) = d_{\text{Poincaré}}(\mathbf{Z}_i, \mathbf{Z}_j) = \text{arcosh} \left(1 + 2 \frac{\|\mathbf{Z}_i - \mathbf{Z}_j\|_2^2}{(1 - \|\mathbf{Z}_i\|_2^2)(1 - \|\mathbf{Z}_j\|_2^2)} \right). \quad (23.11)$$

Тогда в процессе оптимизации будут обучены погружения, доставляющие минимум расстояниям между связанными вершинами и максимум расстояниям между несвязанными вершинами:

$$d_1(\mathbf{W}, \hat{\mathbf{W}}) = \sum_{i,j} \mathbf{W}_{ij} \log \frac{e^{-\hat{\mathbf{W}}_{ij}}}{\sum_{k|\mathbf{W}_{ik}=0} e^{-\hat{\mathbf{W}}_{ik}}}, \quad (23.12)$$

где знаменатель аппроксимируется с помощью отрицательной выборки. Заметим, что, поскольку гиперболическое пространство имеет структуру многообразия, нужно обращать внимание на то, чтобы погружения оставались на многообразии (применяя методы римановой оптимизации [Bon13]).

Предлагались и другие варианты этих методов. В работе [NK18] изучается **лоренцева модель** гиперболического пространства и показано, что она численно устойчивее, чем модель Пуанкаре. Еще в одной серии работ неевклидовы погружения обобщаются на представления смешанной кривизны в произведениях пространств [Gu+18], что дает больше гибкости для других типов графов (например, кольца деревьев). Наконец, в работе [CCD17] погружения Пуанкаре обобщаются с помощью скипграммной потери с гиперболическими скалярными произведениями.

23.3.4. На основе внешнего произведения: методы матричной факторизации

Подходы на основе матричной факторизации обучают погружения, которые приводят к низкоранговому представлению некоторой матрицы сходства $s(\mathbf{W})$, где $s: \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{N \times N}$. Часто выбирают следующие преобразования: $s(\mathbf{W}) = \mathbf{W}$, $s(\mathbf{W}) = \mathbf{L}$ (лапласиан графа) или еще какую-нибудь меру близости, например индекс центральности Каца, индекс общих соседей или индекс Адамик-Адара. Функцией декодера в методах матричной факторизации является просто скалярное произведение:

$$\hat{\mathbf{W}} = \text{DEC}(\mathbf{Z}; \Theta^D) = \mathbf{Z}\mathbf{Z}^\top. \quad (23.13)$$

Методы матричной факторизации обучают \mathbf{Z} посредством минимизации потери регуляризации $\mathcal{L}_{G;\text{RECON}}(\mathbf{W}, \hat{\mathbf{W}}; \Theta) = \|s(\mathbf{W}) - \hat{\mathbf{W}}\|_F^2$.

Метод **факторизации графов** (GF) из работы [Ahm+13] обучается низкоранговой факторизации графа посредством минимизации потери регуляризации графа $\mathcal{L}_{G;\text{RECON}}(\mathbf{W}, \hat{\mathbf{W}}; \Theta) = \sum_{(v_i, v_j) \in E} \|\mathbf{W}_{ij} - \hat{\mathbf{W}}_{ij}\|^2$.

Заметим, что если \mathbf{A} – бинарная матрица смежности ($\mathbf{A}_{ij} = 1$ тогда и только тогда, когда $(v_i, v_j) \in E$ и $\mathbf{A}_{ij} = 0$ в противном случае), то потерю регуляризации графа можно выразить в терминах нормы Фробениуса:

$$\mathcal{L}_{G;\text{RECON}}(\mathbf{W}, \hat{\mathbf{W}}; \Theta) = \|\mathbf{A} \odot (\mathbf{W} - \hat{\mathbf{W}})\|_F^2, \quad (23.14)$$

где \odot – оператор поэлементного умножения матриц. Поэтому GF обучается также низкоранговой факторизации матрицы смежности \mathbf{W} , измеряемой нормой Фробениуса. Заметим, что это разреженная операция (суммирование производится только по присутствующим в графе ребрам), поэтому вычислительная сложность метода составляет $O(M)$.

Все описанные до сих пор методы симметричны, т. е. предполагается, что $\mathbf{W}_{ij} = \mathbf{W}_{ji}$. Такое предположение неприемлемо при работе с ориентированными графами, поскольку в этом случае для некоторых связей нет обратных. Метод **GraRep** из работы [CLX15] преодолевает это ограничение, обучая два погружения на каждую вершину: исходное \mathbf{Z}^s и целевое \mathbf{Z}^t , которые улавливают асимметричную близость в ориентированных сетях. Помимо асимметрии, GraRep обучается погружениям, которые сохраняют соседство с расстоянием в k переходов; для этого вычисляются степени матрицы смежности и минимизируется потеря реконструкции графа:

$$\hat{\mathbf{W}}^{(k)} = \mathbf{Z}^{(k),s} \hat{\mathbf{Z}}^{(k),t^T}; \quad (23.15)$$

$$\mathcal{L}_{G;\text{RECON}}(\mathbf{W}, \hat{\mathbf{W}}^{(k)}; \Theta) = \|\mathbf{D}^{-k} \mathbf{W}^k - \hat{\mathbf{W}}^{(k)}\|_F^2 \quad (23.16)$$

для каждого $1 \leq k \leq K$. GraRep конкатенирует все представления для получения исходных погружений $\mathbf{Z}^s = [\mathbf{Z}^{(1),s} \mid \dots \mid \mathbf{Z}^{(K),s}]$ и целевых погружений $\mathbf{Z}^t = [\mathbf{Z}^{(1),t} \mid \dots \mid \mathbf{Z}^{(K),t}]$. К сожалению, GraRep не очень хорошо масштабируется, потому что возведение матрицы $\mathbf{D}^{-1} \mathbf{W}$ в степень увеличивает плотность. Это ограничение можно обойти, воспользовавшись неявной матричной факторизацией [Per+17], как описывается ниже.

23.3.5. На основе внешнего произведения: скипграммные методы

Скипграммные модели погружения графов возникли из исследований в области обработки естественного языка, посвященных моделированию дистрибутивного поведения слов [Mik+13c; PSM14b]. Скипграммные погружения слов оптимизируются, так чтобы предсказывать слова в контексте (набор окружающих слов) для каждого целевого слова в предложении. Если дана последовательность слов (w_1, \dots, w_T) , то скипграммная модель оптимизирует целевую функцию:

$$\mathcal{L} = - \sum_{-K \leq i \leq K, i \neq 0} \log \mathbb{P}(w_{k-i} | w_k)$$

для каждого целевого слова w_k . Эти условные вероятности можно эффективно оценить с использованием нейронных сетей. Детали см. в разделе 20.5.2.2.

Эта идея была применена к погружениям графов в системе DeepWalk [PARS14]. Авторы обосновали это, эмпирически показав, что частотная статистика, индуцированная случайными блужданиями в реальных графах, имеет примерно такое же распределение, как у слов в естественном языке. В терминах GraphEDM методы скипграммного погружения графов используют внешнее произведение (23.13) в качестве функции декодера, а член реконструкции графа вычисляется по случайным блужданиям на графе.

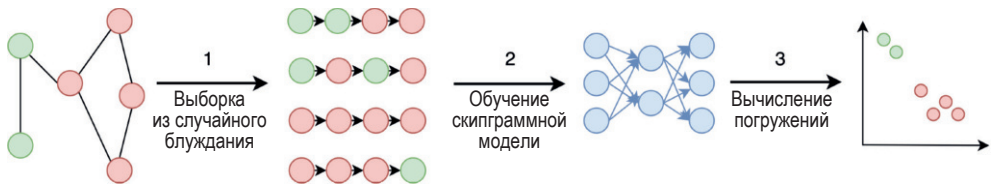


Рис. 23.4 ❖ Конвейер методов погружения с применением случайного блуждания на графе. Печатается с разрешения авторов работы [God18]

Чуть подробнее: **DeepWalk** обучает погружения вершин максимизировать вероятность предсказания контекстных вершин для каждой центральной вершины. Контекстными называются вершины, которые оказываются смежными с центральной при эмулированных случайных блужданиях на графе A . Для обучения погружений DeepWalk генерирует последовательности вершин, применяя усеченное несмещенное случайное блуждание на графе – их можно сравнить с предложениями в естественно-языковых моделях, – а затем максимизирует их логарифмическое правдоподобие. Каждое случайное блуждание начинается с вершины $v_{i_1} \in V$ и итеративно выбирает следующую вершину случайным образом: $v_{i_{j+1}} \in \{v \in V \mid (v_j, v) \in E\}$. Длина блуждания – гиперпараметр. Затем все сгенерированные случайные блуждания можно закодировать с помощью последовательностной модели. Эта двухшаговая парадигма, впервые предложенная в работе [PARS14], была затем развита во многих работах, например в модели **node2vec** [GL16].

Отметим, что в конкретных реализациях обычно используют два разных представления для каждой вершины: одна для случая, когда вершина является центром усеченного случайного блуждания, а другая для случая, когда она принадлежит контексту. Следствия такого подхода к моделированию изучаются в работе [АЕНPAR17].

Чтобы представить DeepWalk в терминах схемы GraphEDM, мы можем положить:

$$s(\mathbf{W}) = \mathbb{E}_q[(\mathbf{D}^{-1}\mathbf{W})^q] \text{ при } q \sim P(Q) = \text{Categorical}([1, 2, \dots, T_{\max}]), \quad (23.17)$$

где $P(Q = q) = (T_{\max} - 1 + q)/T_{\max}$ (см. вывод в работе [АЕН+18]).

Обучение DeepWalk эквивалентно минимизации:

$$\mathcal{L}_{G, \text{RECON}}(W, \hat{W}; \Theta) = \log Z(\mathbf{Z}) - \sum_{v_i \in V, v_j \in V} s(\mathbf{W})_{ij} \hat{W}_{ij}, \quad (23.18)$$

где $\hat{\mathbf{W}} = \mathbf{Z}\mathbf{Z}^T$, а функция разбиения, имеющая вид $Z(\mathbf{Z}) = \prod_i \sum_j \exp(\hat{\mathbf{W}}_{ij})$, может быть аппроксимирована за время $O(N)$ иерархической softmax (см. раздел 20.5.2). (Также часто берут $\mathbf{W} = \mathbf{Z}_{\text{out}}\mathbf{Z}_{\text{in}}^T$ для ориентированных графов с использованием словарей погружения $\mathbf{Z}_{\text{out}}, \mathbf{Z}_{\text{in}} \in \mathbb{R}^{N \times L}$.)

Как отмечено в работе [LG14], скипграммные методы можно рассматривать как неявную матричную факторизацию, а обсуждаемые здесь методы связаны с методами матричной факторизации (см. раздел 23.3.4). Эта связь более глубоко обсуждается в работе [Qiu+18], где предложена общая схема матричной факторизации, **NetMF**, в которой используется та же исходная информация о близости в графе, что в DeepWalk, LINE [Tan+15] и node2vec [GL16]. При переформулировании задачи о погружении вершин как задачи матричной факторизации можно заодно получить все преимущества эффективных операций над разреженными матрицами [Qiu+19b].

23.3.6. Обучение погружений с учителем

Во многих приложениях имеются не только вершинные признаки и структура графа, но и помеченные данные. Задачу обучения с учителем можно попытаться решить, сначала обучив представления без учителя, а затем используя их как признаки во вторичной модели, но такой путь не идеален. Обученные без учителя погружения вершин могут не сохранять важные свойства графов (например, окрестности или атрибуты вершин), которые были бы очень полезны для последующего обучения с учителем.

Ввиду этого ограничения было предложено несколько методов, объединяющих оба шага: обучение погружений и предсказание меток вершин или графов. Сейчас мы обсудим простые поверхностные методы. А к глубоким нелинейным погружениям перейдем позже.

23.3.6.1. Распространение меток

Распространение меток (label propagation – LP) [ZG02] – очень популярный графовый алгоритм классификации вершин с частичным привлечением учителя. Кодировщиком является поверхностная модель, представленная справочной таблицей \mathbf{Z} . В LP пространство меток используется для прямого представления погружений вершин (т. е. декодер – просто тождественная функция):

$$\hat{\mathbf{y}}^N = \text{DEC}(\mathbf{Z}; \Theta^C) = \mathbf{Z}.$$

Точнее, структура графа используется в LP, чтобы сгладить распределение меток в графе путем прибавления члена регуляризации к функции потерь, исходя из предположения, что соседние вершины должны иметь похожие метки (т. е. метки связанных вершин как-то согласованы).

Чтобы гарантировать гладкость, при регуляризации используются лапласовы собственные отображения:

$$\mathcal{L}_{G, \text{RECON}}(\mathbf{W}, \hat{\mathbf{W}}; \Theta) = \sum_{i,j} \mathbf{w}_{ij} \|y_i^N - \hat{y}_j^N\|_2^2. \quad (23.19)$$

LP минимизирует эту функцию энергии на пространстве функций, которые принимают фиксированные значения на помеченных вершинах (т. е. $\hat{y}_i^N = y_i^N \forall i | v_i \in V_L$). Для этого используется итеративный алгоритм, который обновляет распределение меток в непомеченной вершине с помощью взвешенного среднего меток ее соседей.

Размазывание меток (label spreading – LS) [Zho+04] – это вариант распространения меток, в котором минимизируется следующая функция энергии:

$$\mathcal{L}_{G, \text{RECON}}(\mathbf{W}, \hat{\mathbf{W}}; \Theta) = \sum_{i,j} \mathbf{W}_{ij} \left\| \frac{\hat{y}_i^N}{\sqrt{D_i}} - \frac{\hat{y}_j^N}{\sqrt{D_j}} \right\|_2^2, \quad (23.20)$$

где $D_i = \sum_j W_{ij}$ – степень вершины v_i .

В обоих методах потеря равна просто сумме расстояний между предсказанными и истинными метками (унитарными векторами):

$$\mathcal{L}_{\text{SUP}}^N(y^N, \hat{y}^N; \Theta) = \sum_{i | v_i \in V_L} \|y_i^N - \hat{y}_i^N\|_2^2. \quad (23.21)$$

Заметим, что, хотя член регуляризации вычисляется по всем вершинам графа, потеря обучения с учителем вычисляется только по помеченным вершинам. Ожидается, что эти методы должны хорошо работать с согласованными графами, т. е. такими, для которых имеется положительная корреляция между близостью вершин и сходством меток.

23.4. ГРАФОВЫЕ НЕЙРОННЫЕ СЕТИ

Большой интерес у исследователей вызывает определение сверток над графовыми данными. В обозначениях работы [Cha+20] такие методы агрегирования соседства с полным или частичным привлечением учителя можно представить кодировщиком вида $\mathbf{Z} = \text{ENC}(\mathbf{X}, \mathbf{W}; \Theta^E)$ и декодерами вида $\hat{\mathbf{W}} = \text{DEC}(\mathbf{Z}; \Theta^D)$ и (или) $\hat{y}^S = \text{DEC}(\mathbf{Z}; \Theta^S)$. В этом семействе много моделей, некоторые из них мы обсудим ниже.

23.4.1. Графовые нейронные сети передачи сообщений

Оригинальная модель **графовой нейронной сети** (graph neural network – GNN) [GMS05; Sca+09] была первой формулировкой методов глубокого обучения применительно к данным с графовой структурой. В ней задача погружения графа с учителем рассматривается как механизм диффузии информации, когда вершины отправляют информацию своим соседям, пока не будет достигнуто некое равновесное состояние. Формально если даны случайно инициализированные погружения вершин, \mathbf{Z}^0 , то применяется следующее рекуррентная формула:

$$\mathbf{Z}^{t+1} = \text{ENC}(\mathbf{X}, \mathbf{W}, \mathbf{Z}^t; \Theta^E), \quad (23.22)$$

где параметры Θ^E повторно используются на каждой итерации. После достижения сходимости ($t = T$) погружения вершин \mathbf{Z}^T используются, чтобы предсказать финальный выход, например метки вершин или графов:

$$\hat{\mathbf{y}}^S = \text{DEC}(\mathbf{X}, \mathbf{Z}^T; \Theta^S). \quad (23.23)$$

Этот процесс повторяется несколько раз, и параметры GNN Θ^E и Θ^D обучаются с помощью обратного распространения с применением алгоритма Альмейды–Пинеды [Alm87; Pin88]. В силу теоремы Банаха о неподвижной точке эта процедура гарантированно сходится к однозначно определенному решению, если рекуррентное соотношение является сжатием. В свете этого в работе [Sca+09] исследуются отображения, которые можно выразить с помощью сетей передачи сообщений:

$$\mathbf{Z}_i^{t+1} = \sum_{j|(v_i, v_j) \in E} f(\mathbf{X}_i, \mathbf{X}_j, \mathbf{Z}_j^t; \Theta^E), \quad (23.24)$$

где $f(\cdot)$ – многослойный перцептрон (МСП), на который наложено ограничение сжимающего отображения. Однако на функцию декодера никаких ограничений не накладывается, она может быть произвольным МСП.

В работе [Li+15] предложены **вентильные графовые последовательностные нейронные сети** (Gated Graph Sequence Neural Network – GGSNN), в которых с GNN снято требование сжимающего отображения. В GGSNN рекурсивный алгоритм (23.22) ослаблен путем применения отображающих функций на протяжении фиксированного числа шагов, а каждая такая функция является вентильным рекуррентным блоком [Cho+14a] с параметрами, разделяемыми на всех итерациях. Модель GGSNN выводит предсказания на каждом шаге, поэтому она особенно полезна для задач с последовательной структурой (например, временных графов).

В работе [Gil+17] предложена общая схема графовых нейронных сетей, названная **нейронными сетями передачи сообщений** (message passing neural network – MPNN), которая инкапсулирует многие недавние модели. В отличие от модели GNN, в которой количество итераций неопределенное, MPNN предоставляет абстракцию для современных подходов, состоящую из многослойных нейронных сетей с фиксированным числом слоев. На каждом слое ℓ функции сообщений $f^\ell(\cdot)$ принимают сообщения от соседей (основанные на скрытом состоянии соседа), которые затем передаются функциям агрегирования $h^\ell(\cdot)$:

$$\mathbf{m}_i^{\ell+1} = \sum_{j|(v_i, v_j) \in E} f^\ell(\mathbf{H}_i^\ell, \mathbf{H}_j^\ell); \quad (23.25)$$

$$\mathbf{H}_i^{\ell+1} = h^\ell(\mathbf{H}_i^\ell, \mathbf{m}_i^{\ell+1}), \quad (23.26)$$

где $\mathbf{H}^0 = \mathbf{X}$. После ℓ слоев передачи сообщений в скрытых представлениях вершин оказывается закодирована информация, содержащаяся в окрестности радиусом ℓ переходов.

В работе [Bat+18] предложена схема **GraphNet**, которая обобщает MPNN на обучение представлений ребер, вершин и всего графа с помощью функций передачи сообщений. Явное добавление представлений ребер и графов придает дополнительную выразительность модели MPNN и позволяет распространить графовые модели на новые предметные области.

23.4.2. Спектральные свертки графов

Спектральные методы определяют свертки графов с помощью спектральной области лапласиана графа. Эти методы можно отнести к двум крупным категориям: *спектрально-зависимые методы* явно вычисляют спектральное разложение лапласиана (например, **спектральные СНС** [Bru+14]), а *спектрально-независимые* методы, хотя и опираются на спектральную теорию графов, но спектральное разложение не вычисляют (например, **графовые сверточные сети**, или **GCN** [KW16a]).

Основной недостаток спектрально-зависимых методов в том, что они полагаются на спектр лапласиана графа и потому доменно-зависимы (т. е. не обобщаются на новые графы). К тому же спектральное разложение лапласиана дорого обходится с вычислительной точки зрения. Спектрально-независимые методы обходят эти ограничения с помощью аппроксимаций спектральных фильтров. Однако спектрально-независимые методы вынуждены использовать граф W целиком, поэтому плохо масштабируются.

Дополнительные сведения о спектральных подходах см., например, в работах [Bro+17b; Cha+20].

23.4.3. Пространственные свертки графов

Спектрально-зависимые методы принципиально зависят от домена, что ограничивает применение модели, обученной на одном графе, к новому набору данных. С другой стороны, *спектрально-независимые* методы (например, GCN) должны использовать весь граф A , так что по мере роста его размера быстро становятся практически бесполезными.

Для преодоления этих ограничений было разработано другое семейство графовых сверток (пространственные методы), заимствующее идеи из стандартных СНС, – применять свертку в пространственной области, определяемой топологией графа. Например, в компьютерном зрении сверточные фильтры пространственно локализованы, потому что используются фиксированные прямоугольные патчи вокруг каждого пикселя. С учетом естественного порядка пикселей в изображениях (верх, лево, низ, право) мы можем повторно использовать веса фильтров в каждой точке. Этот процесс значительно уменьшает общее число параметров модели. Хотя такие пространственные свертки напрямую неприменимы к графовым доменам, но сам подход нашел отражение в пространственных графовых свертках. Основная идея – использовать выборку из окрестности и механизмы внимания, чтобы создать графовые патчи фиксированного размера, компенсируя тем самым нерегулярность графов.

23.4.3.1. Выборочные пространственные методы

Чтобы преодолеть зависимость от домена и ограничения на память, присущие GCN, в работе [HYL17] предложен каркас **GraphSAGE** для индуктивного обучения погружений вершин. Вместо усреднения сигналов от всех соседей на расстоянии одного перехода (путем умножения на лапласиан), SAGE производит выборку из фиксированных окрестностей (радиуса q) для каждой вершины. Это устраняет сильную зависимость от фиксированной структуры графа и открывает возможность обобщения на новые графы. В каждом слое SAGE вершина агрегирует информацию от вершин, выбранных из ее окрестности (см. рис. 23.5). В обозначениях GraphEDM правило распространения можно записать в виде:

$$\mathbf{H}_{:,i}^{\ell+1} = \sigma(\Theta_1^{\ell} \mathbf{H}_{:,i}^{\ell} + \Theta_2^{\ell} \text{AGG}(\{\mathbf{H}_{:,j}^{\ell} \mid v_j \in \text{Sample}(\text{nbr}(v_i), q)\})), \quad (23.27)$$

где $\text{AGG}(\cdot)$ – функция агрегирования. Эта функция агрегирования может быть любым инвариантным относительно перестановок оператором, например усреднения (SAGE-mean) или max-пулинга (SAGE-pool). Поскольку SAGE работает с окрестностями фиксированного размера (а не со всей матрицей смежности), он также уменьшает вычислительную сложность обучения GCN.

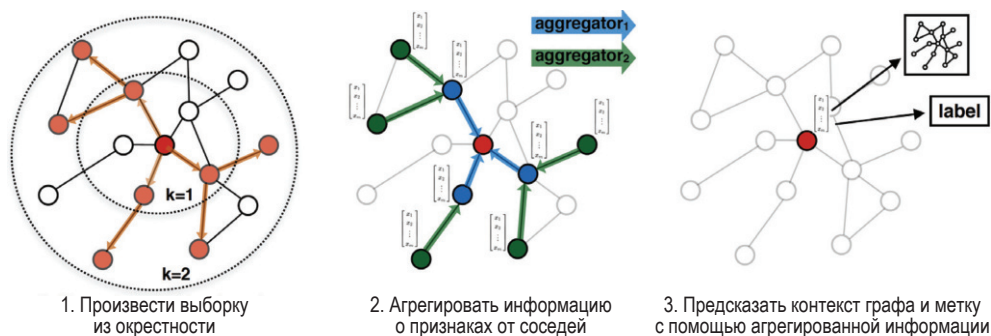


Рис. 23.5 ❖ Модель GraphSAGE.

Печатается с разрешения авторов работы [HYL17]

23.4.3.2. Пространственные методы на основе механизма внимания

Механизмы внимания (раздел 15.4) успешно применялись в языковых моделях, где они, например, позволяют модели идентифицировать релевантные части длинных входных последовательностей. Воодушевленные успехом на уровне естественных языков, ученые предложили использовать аналогичные идеи для графовых сверточных сетей. Такие графовые модели внимания обучаются фокусировать внимание на важных соседях на шаге передачи сообщений посредством параметрических патчей, которые обучаются на основе вершинных признаков. Это дает больше гибкости при индуктивном обучении по сравнению с методами, опирающимися на фиксированные веса, такими как GCN.

Модель **графовой сети внимания** (graph attention network – GAT) из работы [Vel+18] является основанным на внимании вариантом GCN. В каждом слое GAT уделяет внимание окрестности каждой вершины и обучается выбирать вершины, которые приводят к наилучшему качеству на последующей задаче. Идея опирается на те же интуитивные соображения, что и [HYL17], что делает GAT пригодной для индуктивных и трансдуктивных задач. Но, в отличие от SAGE, которая ограничивает шаг свертки окрестностями фиксированного размера, GAT позволяет каждой вершине уделять внимание всем ее соседям – назначая каждому различные веса. Параметры механизма внимания обучаются с помощью обратного распространения, после чего оценки внимания нормируются по строкам функцией активации softmax.

23.4.3.3. Геометрические пространственные методы

В работе [Mon+17] предложен общий каркас **MoNet**, который особенно хорошо работает, когда вершинные признаки лежат в геометрическом пространстве, например в трехмерном облаке точек или на сетке. MoNet обучает патчи внимания с помощью параметрических функций в предопределенном пространственном домене (например, пространственных координат), а затем применяет сверточные фильтры к результирующему графовому домену.

MoNet обобщает пространственные подходы, которые вводят конструкции сверток на многообразиях, например геодезические СНС (GCNN) [Mas+15] и анизотропные СНС (ACNN) [Bos+16]. И в GCNN, и в ACNN используются фиксированные патчи, определенные в конкретной системе координат, поэтому они не обобщаются на данные со структурой графа. Но каркас MoNet более общий; патчи могут описываться в терминах любых псевдокоординат (т. е. вершинных признаков). Формально если \mathbf{U}^s – псевдокоординаты, а \mathbf{H}^ℓ – признаки из другого домена, то слой MoNet в наших обозначениях можно записать в виде:

$$\mathbf{H}^{\ell+1} = \sigma \left(\sum_{k=1}^K (\mathbf{W} \odot g_k(\mathbf{U}^s)) \mathbf{H}^\ell \Theta_k^\ell \right), \quad (23.28)$$

где $g_k(\mathbf{U}^s)$ – обученные параметрические патчи, представляющие собой матрицы $N \times N$. На практике в MoNet для обучения патчей используются гауссовы ядра, т. е

$$g_k(\mathbf{U}^s) = \exp \left(-\frac{1}{2} (\mathbf{U}^s - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{U}^s - \boldsymbol{\mu}_k) \right), \quad (23.29)$$

где $\boldsymbol{\mu}_k$ и $\boldsymbol{\Sigma}_k$ – обученные параметры и $\boldsymbol{\Sigma}_k$ должна быть диагональной.

23.4.4. Неевклидовы графовые свертки

В разделе 23.3.3 мы говорили, что пространство с гиперболической геометрией позволяет обучать поверхностные погружения иерархических графов, дающие меньшее искажение, чем евклидовы погружения. Но у поверхностных погружений есть серьезный недостаток – они плохо обобщаются (если

вообще обобщаются) с одних графов на другие. С другой стороны, графовые нейронные сети, в которых используются вершинные признаки, показали хорошие результаты на многих задачах индуктивного погружения графов.

Поэтому естественно, что был проявлен интерес к обобщению графовых нейронных сетей на обучение неевклидовых погружений. При этом главная проблема – сама природа свертки. Как выполнить свертку в неевклидовом пространстве, где такие стандартные операции, как скалярное произведение и умножение матриц, не определены?

Модели гиперболических графовых сверточных сетей (Hyperbolic Graph Convolution Networks – HGCN) [Cha+19a] и гиперболических графовых нейронных сетей (Hyperbolic Graph Neural Networks – HGNN) [LNK19] применяют графовы свертки в гиперболическом пространстве, пользуясь евклидовым касательным пространством, которое дает приближение первого порядка к гиперболическому многообразию в точке. На каждом шаге графовой свертки погружения вершин отображаются в евклидово касательное пространство в начале координат, там свертки применяются, а затем производится обратное отображение в гиперболическое пространство. Такие подходы дают значительное улучшение на графах с иерархической структурой (рис. 23.6).



Рис. 23.6 ❖ Евклидовы (слева) и гиперболические (справа) погружения дерева. Гиперболические погружения обучаются естественным иерархиям в пространстве погружения (глубина индицируется цветом). Перепечатано работы из [Cha+19a] с разрешения авторов

23.5. ГЛУБОКИЕ ПОГРУЖЕНИЯ ГРАФОВ

В этом разделе мы будем использовать нейронные сети для реализации погружения графов при обучении без учителя и с частичным привлечением учителя.

23.5.1. Обучение погружений без учителя.

В этом разделе мы обсудим потери GNN, обучаемых без учителя (рис. 23.7).

23.5.1.1. Структурное погружение с помощью глубокой сети

В методе **структурного погружения с помощью глубокой сети** (structural deep network embedding – SDNE) из работы [WCZ16] используются автокоди-

ровщики, которые сохраняют близость вершин первого и второго порядка. Кодировщик SDNE принимает на входе строку матрицы смежности (полагает $s(\mathbf{W}) = \mathbf{W}$) и порождает погружение вершин $\mathbf{Z} = \text{ENC}(\mathbf{W}, \Theta^E)$. (Заметим, что при этом все вершинные признаки игнорируются.) Декодер SDNE возвращает $\hat{\mathbf{W}} = \text{DEC}(\mathbf{Z}, \Theta^D)$, реконструкцию, обученную восстанавливать оригинальную матрицу смежности графа. SDNE сохраняет близость вершин второго порядка благодаря минимизации следующей функции потерь:

$$\|s(\mathbf{W}) - \hat{\mathbf{W}}\|_F^2 + \alpha_{\text{SDNE}} \sum_{ij} s(\mathbf{W})_{ij} \|\mathbf{Z}_i - \mathbf{Z}_j\|_2^2. \quad (23.30)$$

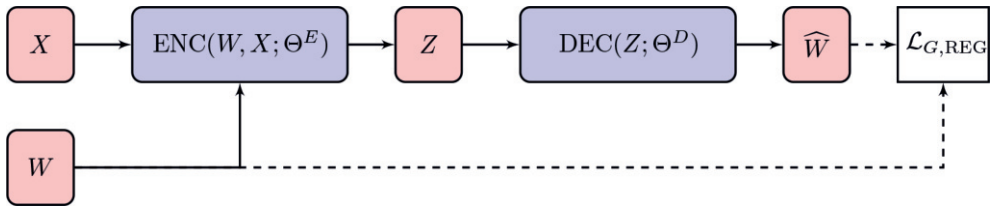


Рис. 23.7 ❖ Графовые нейронные сети, обучаемые без учителя. Структура графа и входные признаки отображаются в погружения низкой размерности с помощью графового нейронного кодировщика. Затем погружения декодируются для вычисления потери регуляризации на графе (без учителя). Перепечатано работы из [Cha+20] с разрешения авторов

Первый член похож на регуляризацию в целевой функции матричной факторизации с тем отличием, что при вычислении $\hat{\mathbf{W}}$ не используются внешние произведения. Второй член используется в методах поверхностного погружения на основе расстояния.

23.5.1.2. Вариационные графовые автокодировщики

В работе [KW16b] графовые свертки (раздел 23.4.2) используются для обучения погружений вершин $\mathbf{Z} = \text{GCN}(\mathbf{W}, \mathbf{X}; \Theta^E)$. Декодер представляет собой внешнее произведение: $\text{DEC}(\mathbf{Z}; \Theta^D) = \mathbf{Z}\mathbf{Z}^T$. В роли члена реконструкции графа выступает сигмоидная перекрестная энтропия между истинной матрицей смежности и предсказанными оценками сходства ребер:

$$\mathcal{L}_{G, \text{RECON}}(\mathbf{W}, \hat{\mathbf{W}}; \Theta) = - \left(\sum_{i,j} (1 - \mathbf{W}_{ij}) \log(1 - \sigma(\hat{\mathbf{W}}_{ij})) + \mathbf{W}_{ij} \log \sigma(\hat{\mathbf{W}}_{ij}) \right). \quad (23.31)$$

Вычисление члена регуляризации по всем парам вершин на практике оказывается трудной задачей, поэтому для ее решения в модели графовых автокодировщиков (Graph Auto Encoders – GAE) используется отрицательная выборка.

GAE – детерминированная модель, и в дополнение к ней авторы предлагают модель вариационных графовых автокодировщиков (variational graph auto-encoder – VGAE), которая опирается на вариационные автокодировщи-

ки (см. раздел 20.3.5) для кодирования и декодирования структуры графа. В VGAE погружение \mathbf{Z} моделируется в виде латентной переменной со стандартным многомерным нормальным априорным распределением $p(\mathbf{Z}) = \mathcal{N}(\mathbf{Z}|\mathbf{0}, \mathbf{I})$, а в качестве амортизированной сети вывода, $q_\phi(\mathbf{Z}|\mathbf{W}, \mathbf{X})$, используется графовая свертка. Модель обучается минимизировать соответствующую отрицательную вариационную нижнюю границу:

$$\text{NELBO}(\mathbf{W}, \mathbf{X}; \Theta) = -\mathbb{E}_{q_\phi(\mathbf{Z}|\mathbf{W}, \mathbf{X})}[\log p(\mathbf{W}|\mathbf{Z})] + \text{KL}(q_\phi(\mathbf{Z}|\mathbf{W}, \mathbf{X})||p(\mathbf{Z})) \quad (23.32)$$

$$= \mathcal{L}_{G, \text{RECON}}(\mathbf{W}, \hat{\mathbf{W}}; \Theta) + \text{KL}(q_\phi(\mathbf{Z}|\mathbf{W}, \mathbf{X})||p(\mathbf{Z})). \quad (23.33)$$

23.5.1.3. Итеративное порождающее моделирование графов (Graphite)

Модель **Graphite** из работы [GZE19] обобщает GAE и VGAE благодаря более сложному декодеру. Этот декодер поочередно применяет функцию декодирования и графовую свертку:

$$\hat{\mathbf{W}}^{(k)} = \frac{\mathbf{Z}^{(k)}\mathbf{Z}^{(k)\top}}{\|\mathbf{Z}^{(k)}\|_2^2} + \frac{\mathbf{1}\mathbf{1}^\top}{N};$$

$$\mathbf{Z}^{(k+1)} = \text{GCN}(\hat{\mathbf{W}}^{(k)}, \mathbf{Z}^{(k)}),$$

где $\mathbf{Z}^{(0)}$ инициализируется выходом сети кодировщика. В результате Graphite удастся обучить более выразительные декодеры. Наконец, как и в случае GAE, модель Graphite может быть детерминированной или вариационной.

23.5.1.4. Методы на основе сопоставительных потерь

Метод **deep graph infomax** (DGI) из работы [Vel+19] похож на порождающую состязательную сеть (GAN), но предназначен для создания погружений на уровне графов. Получив один или несколько *настоящих* (положительных) графов с матрицей смежности $\mathbf{W} \in \mathbb{R}^{N \times N}$ и вершинными признаками $\mathbf{X} \in \mathbb{R}^{N \times D}$, этот метод создает *фальшивые* (отрицательные) матрицы смежности $\mathbf{W}^- \in \mathbb{R}^{N^- \times N^-}$ и признаки $\mathbf{X}^- \in \mathbb{R}^{N^- \times D}$. Он обучает (i) кодировщик, который обрабатывает как настоящие, так и фальшивые примеры, порождая, соответственно, $\mathbf{Z} = \text{ENC}(\mathbf{X}, \mathbf{W}; \Theta^E) \in \mathbb{R}^{N \times L}$ и $\mathbf{Z}^- = \text{ENC}(\mathbf{X}^-, \mathbf{W}^-; \Theta^E) \in \mathbb{R}^{N^- \times L}$; (ii) графовую пулингую функцию $\mathcal{R} : \mathbb{R}^{N \times L} \rightarrow \mathbb{R}^L$ и (iii) дискриминантную функцию $\mathcal{D} : \mathbb{R}^L \times \mathbb{R}^L \rightarrow [0, 1]$, которая обучается выводить, соответственно, $\mathcal{D}(\mathbf{Z}_i, \mathcal{R}(\mathbf{Z})) \approx 1$ и $\mathcal{D}(\mathbf{Z}_j^-, \mathcal{R}(\mathbf{Z}^-)) \approx 0$ для вершин, соответствующих заданному графу $i \in V$ и фальшивому графу $j \in V^-$. Точнее, DGI оптимизирует следующую функцию:

$$\min_{\Theta} - \mathbb{E}_{\mathbf{x}, \mathbf{w}} \sum_{i=1}^N \log \mathcal{D}(\mathbf{Z}_i, \mathcal{R}(\mathbf{Z})) - \mathbb{E}_{\mathbf{x}^-, \mathbf{w}^-} \sum_{i=1}^{N^-} \log(1 - \mathcal{D}(\mathbf{Z}_j^-, \mathcal{R}(\mathbf{Z}^-))), \quad (23.34)$$

где Θ содержит Θ^E параметры \mathcal{R} , \mathcal{D} . В первом математическом ожидании DGI производит выборку из настоящих (положительных) графов. Если задан только один граф, то метод мог бы выбрать из него некоторые подграфы

(например, компоненты связности). Во втором математическом ожидании производится выборка из фальшивых (отрицательных) графов. В DGI в фальшивой выборке используется настоящая матрица смежности $\mathbf{W}^- := \mathbf{W}$, но строки фальшивой матрицы признаков \mathbf{X}^- получены перестановкой элементов соответствующих строк настоящей \mathbf{X} . Функция ENC в DGI – это графовая сверточная сеть, хотя можно использовать любую GNN. Функция \mathcal{R} сводит весь граф (переменного размера) к одному вектору (фиксированного размера). В работе [Vel+19] в качестве \mathcal{R} используется построчное среднее, но можно использовать и другие виды графового пулинга, например учитывающие смежность.

В работе [Vel+19] показано, что оптимизация функции (23.34) доставляет максимум нижней границе взаимной информации между выходами кодировщика и функцией графового пулинга, т. е. между представлениями отдельных вершин и представлением графа.

В работе [Pen+20] представлен вариант модели, названный **графовой взаимной информацией** (Graphical Mutual Information – GMI). Вместо того чтобы максимизировать взаимную информацию между вершинами и всем графом, GMI максимизирует взаимную информацию между представлением вершины и ее соседней.

23.5.2. Обучение погружений с частичным привлечением учителя

В этом разделе мы обсудим потери GNN, обучаемых с частичным привлечением учителя. Мы рассмотрим простой частный случай, когда используется нелинейный кодировщик вершинных признаков, но игнорируется структура графа, т. е. $\mathbf{Z} = \text{ENC}(\mathbf{X}, \theta^E)$.

23.5.2.1. SemiEmb

В работе [WRC08] предложен подход, названный **погружениями с частичным привлечением учителя** (semi-supervised embeddings – SemiEmb). В качестве кодировщика \mathbf{X} используется МСП. В качестве декодера можно взять графовый декодер на основе расстояния: $\hat{\mathbf{W}}_{ij} = \text{DEC}(\mathbf{Z}, \theta^D)_{ij} = \|\mathbf{Z}_i - \mathbf{Z}_j\|^2$, где $\|\cdot\|$ может быть нормой L2 или L1.

SemiEmb регуляризует промежуточные или вспомогательные слои сети, применяя тот же регуляризатор, что в функции потерь распространения меток (23.19). Для предсказания меток по промежуточным погружениям используется сеть прямого распространения, затем эти метки сравниваются с истинными с применением кусочно-линейной потери.

23.5.2.2. Planetoid

Скипграммные методы без учителя типа DeepWalk и node2vec обучают погружения в конвейере, где сначала генерируются случайные блуждания по графу, а затем они используются для обучения погружений. Скорее всего, эти

погружения неоптимальны для последующих задач классификации. Метод **Planetoid** из работы [YCS16] обобщает такие методы на основе случайного блуждания, задействуя информацию о метках вершин в алгоритме погружения.

Сначала Planetoid отображает вершины на погружения $\mathbf{Z} = [\mathbf{Z}^c || \mathbf{Z}^F] = \text{ENC}(\mathbf{X}, \Theta^E)$, используя нейронную сеть (и снова игнорируя структуру графа). Погружения вершин \mathbf{Z}^c улавливают структурную информацию, а погружения вершин \mathbf{Z}^F – информацию о признаках. Существует два варианта: трансдуктивный, когда \mathbf{Z}^c обучается напрямую (как поиск в погружении), и индуктивный, когда \mathbf{Z}^c вычисляется с помощью параметрических отображений, воздействующих на входные признаки \mathbf{X} . Целевая функция Planetoid содержит как потерю обучения с учителем, так и потерю регуляризации графа. Последняя измеряет способность предсказывать контекст с помощью погружений вершин:

$$\mathcal{L}_{G, \text{RECON}}(\mathbf{W}, \hat{\mathbf{W}}; \Theta) = -\mathbb{E}_{(i,j,\gamma)} \log \sigma(\gamma \hat{\mathbf{W}}_{ij}), \quad (23.35)$$

где $\hat{\mathbf{W}}_{ij} = \mathbf{Z}_i^T \mathbf{Z}_j$ и $\gamma \in \{-1, 1\}$, причем $\gamma = 1$, если $(v_i, v_j) \in E$ – положительная пара, и $\gamma = -1$, если (v_i, v_j) – отрицательная пара. Распределение под знаком математического ожидания определено самим процессом выборки.

Потеря обучения с учителем в Planetoid – отрицательное логарифмическое правдоподобие предсказания правильных меток:

$$\mathcal{L}_{\text{SUP}}^N(y^N, \hat{y}^N; \Theta) = -\frac{1}{|V_L|} \sum_{i|v_i \in V_L} \sum_{1 \leq k \leq C} y_{ik}^N \log \hat{y}_{ik}^N, \quad (23.36)$$

где i – индекс вершины, k обозначает метки классов, а \hat{y}_i^N вычисляются с помощью нейронной сети, за которой следует активация softmax, отображающая \mathbf{Z}_i в предсказанные метки.

23.6. Приложения

У погружений графов как с учителем, так и без учителя много приложений. В следующих разделах мы приведем несколько примеров.

23.6.1. Приложения без учителя

В этом разделе мы обсудим типичные приложения погружений графов без учителя.

23.6.1.1. Реконструкция графа

Популярным приложением является реконструкция графа. В этом случае цель состоит в том, чтобы обучить функции (параметрические или нет), отображающие вершины в многообразие, которые позволят реконструировать граф. Эта задача считается относящейся к категории «без учителя» в том смысле, что структуре графа никто специально не обучает. Модель можно

обучить, стараясь минимизировать ошибку реконструкции, т. е. ошибку при восстановлении оригинального графа по обученным погружениям. Специально для этой задачи было разработано несколько алгоритмов, и в разделах 23.3.1 и 23.5.1 мы приведем примеры целевых функций реконструкции. На верхнем уровне реконструкция графа напоминает задачу понижения размерности в том смысле, что основная цель – построить по входным данным обобщающее погружение низкой размерности. Но вместо сжатия векторов высокой размерности для получения векторов более низкой размерности, как в стандартных методах понижения размерности (например, РСА), целью моделей реконструкции графа является сжатие определенных на графе данных и представление их в виде векторов низкой размерности.

23.6.1.2. Предсказание связей

В этом случае целью является предсказание отсутствующих или ненаблюдаемых связей (например, связей, которые могут появиться в будущем в динамических или темпоральных сетях). Предсказание связей также помогает идентифицировать и устранять случайные связи. Это основное практическое применение моделей обучения графов, а типичные примеры – предсказание друзей в **социальных сетях**, предсказание взаимодействий между пользователями и предметами в **рекомендательных системах**, предсказание подозрительных связей в **системах обнаружения мошеннических действий** (см. рис. 23.8) и предсказание отсутствующих связей между сущностями в **графе знаний** (см., например, работу [Nic+15]).

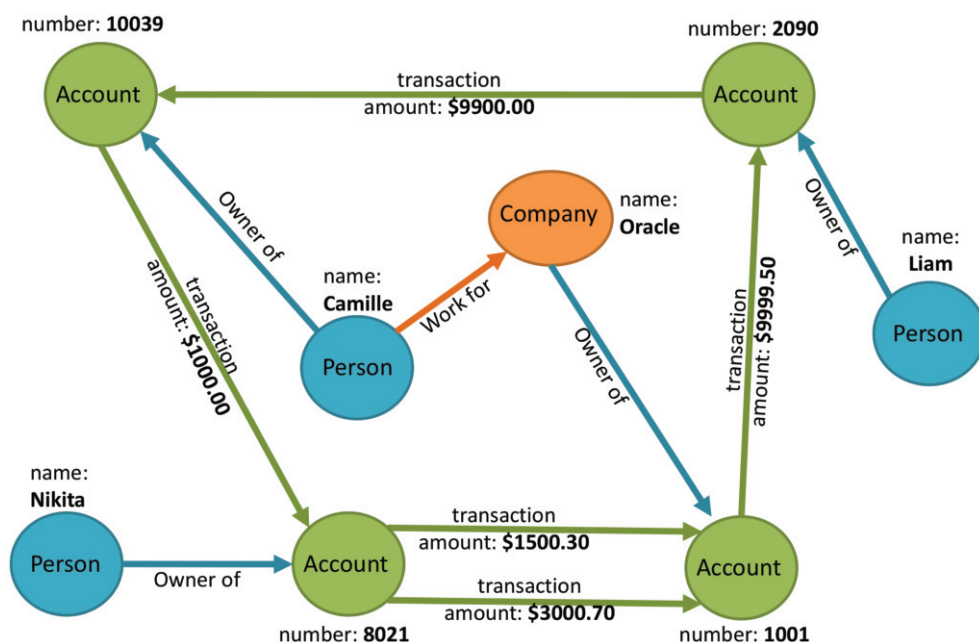


Рис. 23.8 ❖ Представление графа финансовых транзакций.
Из статьи <http://pgql-lang.org/spec/1.2/>

Типичный подход к обучению моделей предсказания связей – замаскировать некоторые ребра графа (положительные и отрицательные), обучить модель на оставшихся ребрах, а затем протестировать ее на множестве замаскированных ребер. Отметим, что предсказание связей отличается от реконструкции графа. В первом случае мы хотим предсказывать связи, не наблюдаемые в исходном графе, а во втором – только вычислить погружения, сохраняющие структуру графа, посредством минимизации ошибки реконструкции.

Наконец, хотя задача предсказания связей похожа на задачи обучения с учителем в том смысле, что для ребер имеются метки (положительное, отрицательное, ненаблюдаемое), мы относим ее к категории приложений без учителя, потому что метки ребер обычно не используются во время обучения, а служат только для измерения прогностического качества погружений.

23.6.1.3. Кластеризация

Кластеризация особенно полезна для выявления сообществ и имеет много практических приложений. Например, кластеры существуют в биологических сетях (скажем, группы протеинов с похожими свойствами) и в социальных сетях (группы людей с похожими интересами).

Методы обучения без учителя, рассмотренные в этой главе, можно использовать для решения таких задач путем применения алгоритма кластеризации (например, K средних) к погружениям, построенным кодировщиком. Кроме того, кластеризацию можно сочетать с алгоритмом обучения при обучении поверхностной [Roz+19] или графовой сверточной [Chi+19a; CEL19] модели погружений.

23.6.1.4. Визуализация

Существует много готовых инструментов для отображения вершин графа на двумерные многообразия с целью визуализации. Визуализация позволяет исследователю качественно оценить свойства графа, понять связи между вершинами или увидеть кластеры вершин. Из популярных инструментов упомянем основанные на *силовом алгоритме визуализации*, который имеет различные реализации на Javascript для веб-приложений.

Методы погружения графов без учителя также используются для визуализации: сначала обучается модель кодировщик-декодер (соответствующая поверхностному погружению или графовой сверточной сети), а затем представления вершин отображаются в двумерное пространство методом t-SNE (раздел 20.4.10) или PCA (раздел 20.1). Такой процесс (погружение → понижение размерности) часто применяется для качественной оценки алгоритмов обучения на графах. Если у вершин имеются атрибуты, то их можно использовать для раскраски вершин на диаграммах двумерной визуализации. Хорошие алгоритмы погружения размещают вершины со схожими атрибутами близко друг к другу в пространстве погружения, как демонстрируется на примерах визуализации различных методов в работах [PARS14; KW16a; АЕН+18]. Наконец, помимо отображения вершин на двумерную плоскость,

методы, отображающие граф в его представление [ARZP19], также можно спроецировать на плоскость, чтобы наглядно показать и качественно проанализировать свойства графа в целом.

23.6.2. Приложения с учителем

В этом разделе мы обсудим типичные приложения погружений графов с учителем.

23.6.2.1. Классификация вершин

Классификация вершин – важное приложение графов, в котором целью является обучение представлений вершин, способных точно предсказывать метки вершин. (Иногда это называют **статистическим реляционным обучением** [GT07].) Например, метками вершин могут быть научные дисциплины в сетях цитирования или пол и другие атрибуты в социальных сетях.

Поскольку пометка больших графов занимает много времени и стоит дорого, особенно популярны методы классификации вершин с частичным привлечением учителя. В такой постановке помечается только часть вершин, а цель состоит в том, чтобы использовать информацию о связях между вершинами для предсказания атрибутов непомеченных вершин. Эта постановка трансдуктивна, потому что существует только один частично помеченный фиксированный граф. Можно также выполнить индуктивную классификацию вершин, что соответствует задаче классификации вершин в нескольких графах.

Отметим, что наличие вершинных признаков может значительно повысить качество классификации вершин, если они как-то описывают целевую метку. Действительно, недавно разработанные методы, такие как GCN (раздел 23.4.2) и GraphSAGE (раздел 23.4.3.1), показывают отличное качество в нескольких тестах классификации вершин благодаря своей способности объединять структурную информацию с семантикой, заключенной в признаках. С другой стороны, такие методы, как случайное блуждание, не позволяют использовать информацию из признаков и потому хуже работают на подобных задачах.

23.6.2.2. Классификация графов

Классификация графов – это приложение, обучаемое с учителем, в котором целью является предсказание меток графов. Это индуктивные задачи, типичным примером является классификация химических соединений (например, предсказание токсичности или запаха по структуре молекулы, рис. 23.9).

Для классификации графов необходимо какое-то понятие пулинга, чтобы агрегировать информацию с уровня вершин на уровень графа. Выше уже отмечалось, что обобщить понятие пулинга на произвольные графы – нетривиальная задача из-за отсутствия регулярности в структуре графа; поэтому графовый пулинг – область активных исследований. Помимо рассмотренных

выше методов обучения с учителем, был предложен ряд методов обучения представлений графов без учителя [Tsi+18; ARZP19; TMP20].

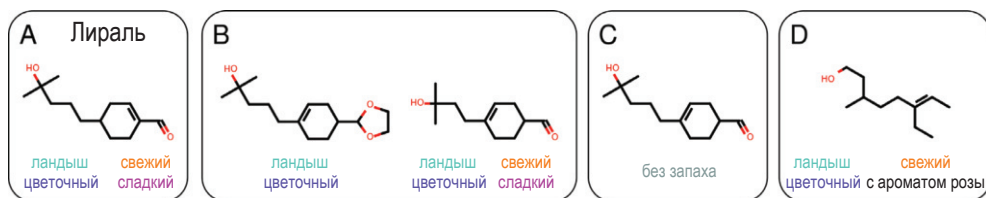


Рис. 23.9 ❖ Структурно схожие молекулы необязательно имеют схожие дескрипторы запаха. (A) Лираль, референтная молекула. (B) Молекулы с похожей структурой могут иметь похожие дескрипторы запаха. (C) Однако небольшое изменение структуры может лишить молекулу запаха. (D) С другой стороны, существенное изменение структуры может оставить запах без изменения. На основе рис. 1 из работы [SL+19], оригинал в работе [OPK12]. Печатается с разрешения Бенджамина Санчеса-Ленгелинга

Приложение A

Обозначения

А.1. ВВЕДЕНИЕ

Очень трудно выработать единую непротиворечивую систему обозначений, покрывающую все разнообразие данных, моделей и алгоритмов, обсуждаемых в этой книге. Кроме того, различные соглашения применяются не только в разных дисциплинах (машинном обучении, статистике и оптимизации), но и в разных книгах и статьях в рамках одной дисциплины. Тем не менее мы старались придерживаться единой системы. Ниже кратко перечислены основные обозначения, встречающиеся в книге, хотя в отдельных разделах может применяться другая нотация. Заметим также, что один и тот же символ может иметь разный смысл в зависимости от контекста, хотя мы старались по возможности избегать таких разночтений.

А.2. ОБЩЕПРИНЯТЫЕ МАТЕМАТИЧЕСКИЕ СИМВОЛЫ

Ниже перечислены некоторые общепринятые символы.

Символ	Значение
∞	Бесконечность
\rightarrow	Стремится, например $n \rightarrow \infty$
\propto	Пропорционально, т. е. $y = ax$ можно записать как $y \propto x$
\triangleq	По определению равно
$O(\cdot)$	О большое; приближенный порядок величины
\mathbb{Z}_+	Множество целых положительных чисел
\mathbb{R}	Множество вещественных чисел
\mathbb{R}_+	Множество положительных вещественных чисел
\mathcal{S}_K	K -мерный вероятностный симплекс
\mathcal{S}_{++}^D	Конус положительно определенных матриц размера $D \times D$
\approx	Приближенно равно
$\{1, \dots, N\}$	Конечное множество $\{1, 2, \dots, N\}$
$1 : N$	Конечное множество $\{1, 2, \dots, N\}$
$[\ell, u]$	Непрерывный интервал $\{\ell \leq x \leq u\}$

А.3. Функции

Функция вообще обозначается буквой f (иногда также g или h). В книге встречается много конкретных функций, например $\tanh(x)$ или $\sigma(x)$. Когда скалярная функция применяется к вектору, предполагается, что она применяется к каждому элементу, например: $\mathbf{x}^2 = [x_1^2, \dots, x_D^2]$. Функционалы (функции от функции) записываются ажурным шрифтом, например $\mathbb{H}(p)$ обозначает энтропию распределения p . Функция с фиксированными параметрами θ обозначается $f(\mathbf{x}; \theta)$, а иногда $f_\theta(\mathbf{x})$. Ниже перечислено несколько часто встречающихся функций (без свободных параметров).

А.3.1. Функции с одним аргументом

Символ	Значение
$\lfloor x \rfloor$	Пол x , т. е. результат округления до ближайшего меньшего целого
$\lceil x \rceil$	Потолок x , т. е. результат округления до ближайшего большего целого
$\neg a$	Логическое НЕ
$\mathbb{I}(x)$	Индикаторная функция: $\mathbb{I}(x) = 1$, если x истинно, иначе $\mathbb{I}(x) = 0$
$\delta(x)$	Дельта-функция Дирака: $\delta(x) = \infty$, если $x = 0$, иначе $\delta(x) = 0$
$ x $	Абсолютная величина
$ S $	Размер (мощность) множества
$n!$	Факториал
$\log(x)$	Натуральный логарифм x
$\exp(x)$	Экспоненциальная функция e^x
$\Gamma(x)$	Гамма-функция: $\Gamma(x) = \int_0^\infty u^{x-1} e^{-u} du$
$\Psi(x)$	Дигамма-функция: $\Psi(x) = (d/dx) \log \Gamma(x)$
$\sigma(x)$	Сигмоидная (логистическая) функция, $1/(1 + e^{-x})$

А.3.2. Функции двух аргументов

Символ	Значение
$a \wedge b$	Логическое И
$a \vee b$	Логическое ИЛИ
$B(a, b)$	Бета-функция: $B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$
$\binom{n}{k}$	Число сочетаний из n по k , $n!/(k!(n-k)!)$
δ_{ij}	Символ Кронекера, $\mathbb{I}(i=j)$
$\mathbf{u} \odot \mathbf{v}$	Поэлементное произведение двух векторов
$\mathbf{u} \otimes \mathbf{v}$	Свертка двух векторов

А.3.3. Функции более двух аргументов

Символ	Значение
$B(\mathbf{x})$	Многомерная бета-функция $\frac{\prod_k \Gamma(x_k)}{\Gamma(\sum_k x_k)}$
$\Gamma(\mathbf{x})$	Многомерная гамма-функция $\pi^{D(D-1)/4} \prod_{d=1}^D \Gamma(x + (1-d)/2)$
$S(\mathbf{x})$	Функция softmax $\left[\frac{e^{x_c}}{\sum_{c'=1}^C e^{x_{c'}}} \right]_{c=1}^C$

А.4. ЛИНЕЙНАЯ АЛГЕБРА

В этом разделе описываются обозначения, применяемые в линейной алгебре (см. главу 7).

А.4.1. Общие обозначения

Векторы обозначаются строчными полужирными буквами, например: \mathbf{x} , \mathbf{w} . Матрицы обозначаются прописными полужирными буквами, например: \mathbf{X} , \mathbf{W} . Скаляры обозначаются строчными обычными буквами. Создание вектора из списка N скаляров обозначается $\mathbf{x} = [x_1, \dots, x_N]$; в зависимости от контекста это может быть вектор-строка или вектор-столбец. (Если явно не оговорено противное, то предполагаются векторы-столбцы). Создание матрицы $M \times N$ из списка векторов обозначается $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$, если векторы образуют столбцы матрицы, и $\mathbf{X} = [\mathbf{x}_1; \dots; \mathbf{x}_M]$, если векторы образуют строки матрицы.

А.4.2. Векторы

Приведем стандартные обозначения для векторов. (Предполагается, что \mathbf{u} и \mathbf{v} – N -мерные векторы.)

Символ	Значение
$\mathbf{u}^T \mathbf{v}$	Скалярное (внутреннее) произведение, $\sum_{i=1}^N u_i v_i$
$\mathbf{u} \mathbf{v}^T$	Внешнее произведение (матрица $N \times N$)
$\mathbf{u} \odot \mathbf{v}$	Поэлементное произведение, $[u_1 v_1, \dots, u_N v_N]$
\mathbf{v}^T	Транспонирование \mathbf{v}
$\dim(\mathbf{v})$	Размерность \mathbf{v}
$\text{diag}(\mathbf{v})$	Диагональная матрица $N \times N$, построенная по вектору \mathbf{v}
$\mathbf{1}$ или $\mathbf{1}_N$	Вектор, состоящий из одних единиц (длины N)
$\mathbf{0}$ или $\mathbf{0}_N$	Вектор, состоящий из одних нулей (длины N)

Символ	Значение
$\ \mathbf{v}\ = \ \mathbf{v}\ _2$	Евклидова ℓ_2 -норма $\sqrt{\sum_{i=1}^N v_i^2}$
$\ \mathbf{v}\ _1$	ℓ_1 -норма $\sum_{i=1}^N v_i $

A.4.3. Матрицы

Ниже приведена стандартная нотация для матриц. (Предполагается, что \mathbf{S} – квадратная матрица размера $N \times N$, \mathbf{X} и \mathbf{Y} имеют размер $M \times N$, а \mathbf{Z} – размер $M' \times N'$.)

Символ	Значение
$\mathbf{X}_{:,j}$	j -й столбец матрицы
$\mathbf{X}_{i,:}$	i -я строка матрицы (рассматриваемая как вектор-столбец)
X_{ij}	Элемент (i, j) матрицы
$\mathbf{S} > 0$	Истина тогда и только тогда, когда \mathbf{S} – положительно определенная матрица
$\text{tr}(\mathbf{S})$	След квадратной матрицы
$\det(\mathbf{S})$	Определитель квадратной матрицы
$ \mathbf{S} $	Определитель квадратной матрицы
\mathbf{S}^{-1}	Обращение квадратной матрицы
\mathbf{X}^\dagger	Псевдообращение матрицы
\mathbf{X}^\top	Транспонирование матрицы
$\text{diag}(\mathbf{S})$	Диагональный вектор квадратной матрицы
\mathbf{I} или \mathbf{I}_N	Единичная матрица размера $N \times N$
$\mathbf{X} \odot \mathbf{Y}$	Поэлементное произведение
$\mathbf{X} \otimes \mathbf{Y}$	Произведение Кронекера (см. раздел 7.2.5)

A.4.4. Матричное исчисление

В этом разделе описываются обозначения, используемые нами в матричном исчислении (см. раздел 7.8). Пусть $\boldsymbol{\theta} \in \mathbb{R}^N$ – вектор, а $f: \mathbb{R}^N \rightarrow \mathbb{R}$ – скалярная функция. Производная f по ее аргументу обозначается следующим образом:

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \triangleq \nabla f(\boldsymbol{\theta}) \triangleq \nabla f \triangleq \left(\frac{\partial f}{\partial \theta_1} \quad \dots \quad \frac{\partial f}{\partial \theta_N} \right). \quad (\text{A.1})$$

Градиент – это вектор, который вычисляется в точке пространства. Чтобы подчеркнуть этот факт, мы иногда пишем

$$\mathbf{g}_t \triangleq \mathbf{g}(\boldsymbol{\theta}_t) \triangleq \nabla f(\boldsymbol{\theta})|_{\boldsymbol{\theta}_t}. \quad (\text{A.2})$$

Можно также вычислить (симметричную) матрицу $N \times N$ вторых частных производных, гессиан:

$$\nabla^2 f \triangleq \begin{pmatrix} \frac{\partial^2 f}{\partial \theta_1^2} & \dots & \frac{\partial^2 f}{\partial \theta_1 \partial \theta_N} \\ \vdots & & \\ \frac{\partial^2 f}{\partial \theta_N \partial \theta_1} & \dots & \frac{\partial^2 f}{\partial \theta_N^2} \end{pmatrix}. \quad (\text{A.3})$$

Гессиан – это матрица, которая вычисляется в точке пространства. Чтобы подчеркнуть этот факт, мы иногда пишем

$$\mathbf{H}_t \triangleq \mathbf{H}(\theta_t) \triangleq \nabla^2 f(\theta)|_{\theta_t}. \quad (\text{A.4})$$

А.5. ОПТИМИЗАЦИЯ

В этом разделе описываются обозначения, используемые нами, когда речь идет о методах оптимизации (см. главу 8).

Мы часто обозначаем целевую функцию, или функцию стоимости, подлежащую минимизации, $\mathcal{L}(\theta)$, где θ – переменные, по которым производится оптимизация (часто это параметры статистической модели). Значение параметра, при котором достигается минимум, обозначается $\theta_* = \operatorname{argmin}_{\theta \in \Theta} \mathcal{L}(\theta)$, где Θ – множество, на котором оптимизируется целевая функция. (Отметим, что таких оптимальных значений может быть несколько, поэтому правильнее было бы писать $\theta_* \in \operatorname{argmin}_{\theta \in \Theta} \mathcal{L}(\theta)$.)

При выполнении итеративной оптимизации номер итерации обозначается t . Буквой ρ обозначается размер шага (скорость обучения). Таким образом, алгоритм градиентного спуска (раздел 8.4) можно записать в виде: $\theta_{t+1} = \theta_t - \rho_t \mathbf{g}_t$.

Мы часто используем символ крышки для обозначения оценки или предсказания (например, $\hat{\theta}$, \hat{y}), верхнюю или нижнюю звездочку для обозначения истинного (но обычно неизвестного) значения (например, θ_* или θ^*), подчеркивание для обозначения среднего значения (например, $\bar{\theta}$).

А.6. ВЕРОЯТНОСТЬ

В этом разделе описываются обозначения, используемые нами в контексте теории вероятностей (см. главу 2).

Мы обозначаем функцию плотности вероятности (ФПВ) или функцию вероятности (ФВ) буквой p , функцию распределения буквой P , а вероятность бинарного события – \Pr . Распределение случайной величины X обозначается $p(X)$, а распределение случайной величины Y – $p(Y)$; эти обозначения относятся к разным распределениям, хотя в обоих случаях используется один и тот же символ p . (Если возможна неоднозначность, то мы пишем $p_X(\cdot)$ и $p_Y(\cdot)$.) Аппроксимации распределения p часто обозначаются q , а иногда \hat{p} .

В некоторых случаях мы различаем саму случайную величину (v) и принимаемое ей значение. Тогда случайная величина (скалярная) обозначается прописной буквой (например, X), а ее значение – строчной буквой (x). Но часто это различие игнорируется. Например, иногда мы пишем $p(x)$ для обозначения как скалярного значения (распределения, вычисленного в точке), так и самого распределения – в зависимости от того, является X наблюдаемой или нет.

Запись $X \sim p$ означает, что X имеет распределение p . Запись $X \perp Y | Z$ означает, что X условно независима от Y при условии Z . Если $X \sim p$, то мы обозначаем математическое ожидание $f(X)$ следующим образом:

$$\mathbb{E}[f(X)] = \mathbb{E}_{p(X)}[f(X)] = \mathbb{E}_X[f(X)] = \int_x f(x)p(x)dx. \quad (\text{A.5})$$

Если f – тождественная функция, то мы пишем $\bar{X} \triangleq \mathbb{E}[X]$. Аналогично дисперсия обозначается

$$\mathbb{V}[f(X)] = \mathbb{V}_{p(X)}[f(X)] = \mathbb{V}_X[f(X)] = \int_x (f(x) - \mathbb{E}[f(X)])^2 p(x)dx. \quad (\text{A.6})$$

Если \mathbf{x} – случайный вектор, то ковариационная матрица обозначается

$$\text{Cov}[\mathbf{x}] = \mathbb{E}[(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^\top]. \quad (\text{A.7})$$

Если $X \sim p$, то мода распределения обозначается

$$\hat{x} = \text{mode}[p] = \underset{x}{\operatorname{argmax}} p(x). \quad (\text{A.8})$$

Параметрические распределения обозначаются $p(\mathbf{x}|\boldsymbol{\theta})$, где \mathbf{x} – случайные величины, $\boldsymbol{\theta}$ – параметры, а p – функция плотности вероятности или функция вероятности. Например, $\mathcal{N}(x|\mu, \sigma^2)$ – гауссово (нормальное) распределение со средним μ и стандартным отклонением σ .

A.7. ТЕОРИЯ ИНФОРМАЦИИ

В этом разделе описываются обозначения, используемые нами в контексте теории информации (см. главу 6).

Если $X \sim p$, то мы обозначаем (дифференциальную) энтропию $\mathbb{H}(X)$ или $\mathbb{H}(p)$. Если $Y \sim q$, то расхождение Кульбака–Лейблера между распределениями p и q обозначается $\mathbb{KL}(p||q)$. Если $(X, Y) \sim p$, то взаимная информация между X и Y обозначается $\mathbb{I}(X, Y)$.

А.8. СТАТИСТИКА И МАШИННОЕ ОБУЧЕНИЕ

В этом разделе описываются обозначения, которыми мы пользуемся при обсуждении статистического обучения.

А.8.1. Обучение с учителем

В случае обучения с учителем наблюдаемые признаки (называемые также входами или **ковариатами**) обозначаются $\mathbf{x} \in \mathcal{X}$. Часто $\mathcal{X} = \mathbb{R}^D$, т. е. признаки вещественные. (Заметим, что сюда относится и случай дискретных признаков, которые можно представить унитарными векторами.) Иногда мы вычисляем заданные вручную признаки входных данных, они обозначаются $\phi(\mathbf{x})$. Имеются также выходы (**цели** или **переменные отклика**) $y \in \mathcal{Y}$, которые мы хотим предсказать. Наша задача – обучить условное распределение вероятностей $p(y|\mathbf{x}, \theta)$, где θ – параметры модели. Если $\mathcal{Y} = \{1, \dots, C\}$, то такая задача называется **классификацией**. Если $\mathcal{Y} = \mathbb{R}^C$, то задача называется **регрессией** (часто $C = 1$, т. е. предсказывается скалярный выход).

Параметры θ оцениваются по **обучающим данным**, обозначаемым $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n) : n \in \{1, \dots, N\}\}$ (т. е. N – число обучающих примеров). Если $\mathcal{X} = \mathbb{R}^D$, то обучающие входы можно сохранить в **матрице плана** размера $N \times D$, которая обозначается \mathbf{X} . Если $\mathcal{Y} = \mathbb{R}^C$, то обучающие выходы можно сохранить в матрице \mathbf{Y} размера $N \times C$. Если $\mathcal{Y} = \{1, \dots, C\}$, то каждую метку класса можно представить C -мерным битовым вектором, в котором ровно один элемент равен 1 (это называется **унитарным кодированием**), поэтому все обучающие выходы можно сохранить в бинарной матрице \mathbf{Y} размера $N \times C$.

А.8.2. Обучение без учителя и порождающие модели

Обучение без учителя обычно формализуется как задача оценивания безусловной плотности, т. е. моделирования $p(\mathbf{x}|\theta)$. В некоторых случаях мы хотим оценивать условную плотность; значения, по которым производится обусловливание, обозначаются \mathbf{u} , т. е. модель принимает вид $p(\mathbf{x}|\mathbf{u}, \theta)$. Это похоже на обучение с учителем с тем отличием, что размерность \mathbf{x} обычно велика (например, изображение), а размерность \mathbf{u} мала (например, метка класса или текстовое описание).

В некоторых моделях имеются **латентные переменные**, не наблюдаемые в обучающих данных. Такие модели называются **моделями с латентными переменными** (latent variable models – LVM). Латентные переменные в примере n обозначаются $z_n \in \mathcal{Z}$. Иногда латентные переменные называются **скрытыми** и обозначаются h_n . **Видимые же переменные** обозначаются \mathbf{v}_n . Как правило, латентные переменные являются непрерывными или дискретными, т. е. $\mathcal{Z} = \mathbb{R}^L$ или $\mathcal{Z} = \{1, \dots, K\}$.

Большинство LVM имеют вид $p(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\theta})$; такие модели используются для обучения без учителя. Однако LVM можно использовать и для обучения с учителем. Именно, можно создать либо порождающую (безусловную) модель вида $p(\mathbf{x}_n, \mathbf{y}_n, \mathbf{z}_n | \boldsymbol{\theta})$, либо дискриминантную (условную) модель вида $p(\mathbf{y}_n, \mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\theta})$.

А.8.3. Байесовский вывод

При работе с байесовским выводом мы записываем априорное распределение параметров в виде $p(\boldsymbol{\theta} | \boldsymbol{\phi})$, где $\boldsymbol{\phi}$ – гиперпараметры. Для сопряженных моделей апостериорное распределение имеет такую же форму, как априорное (по определению). Поэтому мы можем просто обновить гиперпараметры, перейдя от априорного значения $\boldsymbol{\phi}$ к апостериорному $\hat{\boldsymbol{\phi}}$.

В вариационном выводе (раздел 4.6.8.3) мы используем v для представления параметров вариационного апостериорного распределения, т. е. $p(\boldsymbol{\theta} | \mathcal{D}) \approx q(\boldsymbol{\theta} | v)$. Мы оптимизируем вариационную нижнюю границу относительно v , чтобы это приближение было хорошим.

При выполнении выборки методом Монте-Карло мы используем нижний или верхний индекс s для обозначения выборки (например, $\boldsymbol{\theta}_s$ или $\boldsymbol{\theta}^s$).

А.9. АББРЕВИАТУРЫ

Ниже приведен список некоторых встречающихся в книге аббревиатур.

Аббревиатура	Расшифровка
ГНС	Глубокая нейронная сеть
КЛ	Расхождение Кульбака–Лейблера
МСП	Многослойный перцептрон
РНС	Рекуррентная нейронная сеть
св	Случайная величина
СГС	Стохастический градиентный спуск
СКО	Среднеквадратическая ошибка
СНС	Сверточная нейронная сеть
ФВ	Функция вероятности
ФПВ	Функция плотности вероятности
cdf	Функция распределения
DAG	Ориентированный ациклический граф
DML	Глубокое обучение метрики
ЕВ	Эмпирический байесовский
ЕМ	Алгоритм математического ожидания-максимизации
GLM	Обобщенная линейная модель
GMM	Модель гауссовой смеси
HMC	Гамильтонов метод Монте-Карло
HMM	Скрытая марковская модель

Аббревиатура	Расшифровка
KDE	Оценка ядерной плотности
KNN	К ближайших соседей
LSTM	Долгая краткосрочная память (вид РНС)
LVM	Модель латентных переменных
MAP	Оценка апостериорного максимума
MCMC	Метод Монте-Карло по схеме марковской цепи
MLE	Оценка максимального правдоподобия
NLL	Отрицательное логарифмическое правдоподобие
OLS	Метод обыкновенных наименьших квадратов
psd	Положительно определенная (матрица)
PNLL	NLL со штрафом
PGM	Вероятностная графовая модель
RSS	Сумма квадратов невязок
RVM	Метод релевантных векторов
SVI	Стохастический вариационный вывод
SVM	Метод опорных векторов
VB	Вариационный байесовский

Предметный указатель

Символы

1×1 свертка, 569
 ν -SVM классификатор, 702

А

AdaBoostClassifier, 731
 Adaboost.M1, 730
 AdaDelta, 375
 AdaGrad, 375
 Adam, 376, 539
 AlexNet, 577
 ALS, 875
 AMSGrad, 376
 ARD, 502, 679, 711
 à trous алгоритм, 582
 AUC, 233
 AugMix, 741
 AutoAugment, 741
 Auto-ML, 582
 AutoRec, 878

В

В-сплайны, 488
 BALD, 770
 BatchBALD, 770
 BBO, 399
 BERT, 644
 BFGS, 364
 BFGS с ограниченной памятью, 364, 436
 BIC, 247
 BinomialBoost, 733
 BIO, 648
 BiT, 637
 Blue Brain проект, 528
 BMA, 179
 BMM, 145
 BN, 572
 BNN, 551
 BookCrossing, 874

С

C4, 650
 C4.5, 718

CART, 716, 718
 Caser, 883
 CatBoost, 734
 CatPCA, 798
 CBOW, 835
 CCA, 802
 CIFAR, 56
 CLIP, 751
 cloze, 645, 747
 CoLA, 650
 COVID-19, 84
 CPD, 147
 CPT, 147
 crosscat, 871
 CV, 174, 257

Д

DAG, 146, 536
 DeepDream, 596
 deep graph infomax, 902
 DeepWalk, 893
 DenseNets, 581
 DFO, 399

Е

Е-шаг, 391, 392
 EB, 198
 ECM, 791
 EER, 233
 EfficientNetv2, 582
 ELBO, 208, 392, 809
 ELM, 696
 ELMo, 643
 ELU, 542
 EM, 389
 EMA, 167
 EMNIST, 55
 ERM, 163, 254
 EWMA, 167, 375

Ф

F-мера, 235
 FaceNet, 665

FAISS, 657
 Fashion-MNIST, 56
 FFNN, 515
 FIM, 210
 FISTA, 486
 FLDA, 409

G

GAM, 490
 GAN, 753
 GCN, 897
 GDA, 402
 GELU, 541, 543
 GLM, 505
 glmnet, 485
 GloVe, 837
 GMM, 143
 GMRES, 460
 GNN, 515, 895
 GoogLeNet, 578
 GPT, 649
 GPT-2, 649
 GPT-3, 649
 GPU, 526, 547
 GPyTorch, 695
 Graphite, 902
 GraphNet, 897
 GraphSAGE, 898
 GraRep, 892
 GRU, 612
 Gshard, 638

H

HAC, 846
 HDI, 200
 HMC, 208
 HPD, 199

I

I-проекция, 280
 ID3, 718
 ILSVRC, 56
 ImageNet, 56, 526, 562, 577
 ImageNet-21k, 637
 IMDB, 58, 177
 inceptionism, 596
 InfoNCE, 664
 Iris набор данных, 36
 IRLS, 429
 isomap, 820
 ISTA, 486

J

Jeopardy, 229
 Jester, 874
 JFT, 637
 JPEG, 854
 JVP, 342

K

К ближайших соседей метод, 653
 К медоидов алгоритм, 855
 К средних алгоритм, 145, 851
 k-d дерево, 657
 KDE, 669, 671
 K-means++, 855
 KNN, 653

L

L0-норма, 471
 L1-потеря, 237
 L1-регуляризация, 472
 L1VM, 711
 L2-потеря, 236
 L2-регуляризация, 172, 430, 467
 L2VM, 711
 LAR, 487
 LARS, 487
 lasso, 471
 L-BFGS, 364
 LCA, 660
 LeNet, 571, 575
 LightGBM, 734
 Linformer, 641
 LMNN, 659
 LMS, 368
 LOESS, 675
 logit функция, 92
 logitBoost, 731
 log-sum-exp трюк, 97
 LOWESS, 675
 lse функция, 97
 LSH, 657
 LSTM, 613

M

M1, 764
 M2, 764
 M-проекция, 279
 M-шаг, 391
 MALA, 592
 MAML, 771, 772

mAP, 235
 MAP, 169
 MAP оценка, 170, 228, 255
 MART, 734
 max-пулинг, 570
 McKernel, 696
 MCMC, 208
 MDL, 248
 MDN, 558
 MICE, 286
 mixmatch, 756
 MLE, 42, 154
 MLP-mixer, 515
 MM-алгоритмы, 389
 MMSE, 237
 MNIST, 55, 575
 MobileNet, 585
 MoCo, 750
 modus tollens, 262
 MoE, 557
 MoG, 143
 MOM, 165
 MoNet, 899
 MovieLens, 874
 MVN, 126

N

N-парная потеря, 664
 NAS, 582
 NBC, 412
 NCA, 660
 NCM, 407
 NEF, 139
 Netflix Prize, 874
 NetMF, 894
 NeurIPS, 54
 NHST, 260
 NHWC, 569
 NIPS, 54
 NLL, 154
 NLP, 57
 node2vec, 893
 NT-Xent, 664

O

Olivetti набор данных о лицах, 778
 OLS, 162, 338, 457
 OOD, 658
 OpenPose, 591
 opt-einsum библиотека, 315

P

p-значение, 240, 261
 PAC-обучаемость, 257
 Padam, 376
 PageRank, 324
 PAM, 855
 PCA, 50, 776
 PCA-отбеливание, 324
 PCA с учителем, 799
 PCA экспоненциального семейства, 797
 Performer, 641
 PersonLab, 591
 PGM, 146
 Planetoid, 904
 PLS, 801
 PMF, 876
 PMI, 834
 POS, 644
 PPCA, 789
 PreResnet, 580
 ProxQuant, 389

Q

QALY, 226
 QP, 382

R

RAND-WALK, 839
 RANSAC, 494
 RBF, 670
 RBF-сеть, 554
 RBF-ядро, 554, 677
 Reformer, 640
 ReLU, 519, 542
 ReLU с утечкой, 541, 542
 ResNet, 544, 579
 ResNet-18, 580
 resnet с предактивацией, 580
 RFF, 695
 RMSE, 162
 ROC-кривая, 231
 RSS, 162
 RVM, 711

S

SAGA, 373, 427
 SAMME, 730
 SARS-CoV-2, 84
 SELU, 543
 seq2seq, 59, 607

seq2vec, 606
 SGNS, 836
 Shampoo, 377
 SiLU, 543
 SimCLR, 748
 SMACOF, 818
 SMO, 700
 SNLI, 627
 softmax, 41, 94
 softplus, 101, 541
 Stanford Natural Language Inference, 627
 STSB, 650
 STS Benchmark, 629
 SVM, 697
 SVM-перспекция, 710
 SVRG, 372
 SWA, 372
 swish, 541, 543

T

T5, 650
 TF-IDF, 62
 TlCe, 286
 TinyImages, 56
 TL\;DR, 649
 TPU, 528
 t-SNE, 827

U

U-образная кривая, 48
 U-сеть, 589
 UMAP, 831
 UNK, 64

V

VAE, 808
 varimax, 795
 VC-размерность, 258
 vec2seq, 603
 VI, 207
 ViT, 637
 VJP, 342
 VQ, 853

W

WAIC, 864
 Watson система, 229
 wavenet, 619
 WebText, 649
 WMT, набор данных, 59

word2vec, 835
 WSD, 644

X

XGBoost, 734
 XOR задача, 516

Y

Yogi, 376
 YOLO, 588

Z

ZCA, 324

A

Автоковариационная матрица, 122
 Автокодировщик, 802
 Автокорреляционная матрица, 122
 Автоматическое дифференцирование, 530
 Автоматическое определение релевантности, 502, 679, 711
 Авторегрессионная модель, 148
 Агент, 53, 225
 Агрегированный градиент, 373
 Адамик-Адара индекс, 891
 Адаптация домена, 743, 752
 Адаптеры, 744
 Адаптивная индивидуальная нормировка, 601
 Адаптивная скорость обучения, 374, 376
 Адаптивные базисные функции, 726
 Аддитивная модель, 726
 Аддитивное внимание, 623
 Адьюнкт, 537
 Аисты, 123
 Акаике информационный критерий, 247
 Аксон, 527
 Активное множество, 487
 Активное обучение, 497, 768
 Активное обучение с пулом, 768
 Алгоритм Ланжевена
 без поправки, 592
 с поправкой Метрополиса, 592
 Алгоритм
 обучения перцептрона, 427
 стрельбы, 485
 Алеаторная неопределенность, 41, 70
 Альтернативная гипотеза, 240, 258
 Амортизированный вывод, 809
 Анализ
 компонент соседства, 660

латентных совпадений, 660
 эмоциональной окраски, 58
 Аннотирование изображений, 433, 586
 Ансамблевое обучение, 721
 Ансамбль, 371, 550
 Антисвертка, 584
 Апостериорная медиана, 237
 Апостериорная ожидаемая потеря, 226
 Апостериорное прогнозное распределение, 179, 184, 201, 449, 495
 Апостериорное распределение, 84, 178
 Апостериорное среднее, 237
 Апостериорный вывод, 84
 Аппаратные ускорители, 528
 Аппроксимация кривой, 48
 Априорное распределение, 83, 169, 179
 Априорное распределение по умолчанию, 197
 Армихо метод поиска с возвратом, 358
 Архитектура кодировщик-декодер, 608
 Асимптотическая нормальность, 210
 Асинхронное обучение, 548
 Ассоциативность, 307
 Атомная бомба, 116
 Аффинная функция, 41

Б

Базис, 296
 Базисные векторы, 308
 Базовая мера, 139
 Байесовская интерпретация бритвы Оккама, 245
 Байесовская нейронная сеть, 551
 Байесовская оптимизация, 768
 Байесовская оценка, 226, 250
 Байесовская ошибка, 177, 654
 Байесовская сеть, 146
 Байесовская статистика, 178, 208
 Байесовская теория принятия решений, 225
 Байесовская факторная регрессия, 800
 Байесовский выбор модели, 246
 Байесовский вывод, 84
 Байесовский доверительный интервал, 195, 198, 214
 Байесовский информационный критерий, 247
 Байесовский риск, 250
 Байесовское активное обучение по несогласию, 770
 Байесовское глубокое обучение, 551
 Байесовское машинное обучение, 200

Байесовское персонализированное ранжирование, 880
 Байесовское усреднение модели, 179
 Баллонная оценка ядерной плотности, 673
 Бариецентрические координаты, 823
 Барнса-Хата алгоритм, 831
 Безразличие к риску, 227
 Безусловная независимость, 76
 Безусловная оптимизация, 349
 Берксона парадокс, 149
 Бернулли распределение, 89
 Бесконечно широкая RBF-сеть, 694
 Бесселя функция, 680
 Бета-биномиальное распределение, 186
 Бета-распределение, 105, 170, 181
 Бета-функция, 105
 Биграмная модель, 149, 273
 Биекция, 109
 Бикластеризация, 869
 Бинарная классификация, 35, 84
 Бинарная логистическая регрессия, 420
 Бинарная перекрестная энтропия, 423
 Бинарная связь, 389
 Бинарная функция потерь, 40, 228
 Биномиальная регрессия, 506
 Биномиальное распределение, 89
 Биномиальный коэффициент, 90
 Биты, 268
 Близорукая стратегия, 770
 Блоки, 150
 Блок линейной ректификации, 519, 542
 Блочная диагональная матрица, 303
 Бриера оценка, 240, 762
 Броуновское движение, 680
 Булева логика, 70
 Бустинг, 726
 Бустинг наименьших квадратов, 487
 Бутстрэп, 211
 Быстрая адаптация, 772
 Быстрое преобразование Адамара, 696
 Бэггинг, 723

В

Важность признака, 736
 Вальда интервал, 215
 Вариационная нижняя оценка, 208, 392, 809
 Вариационная РНС, 604
 Вариационное исчисление, 142
 Вариационный автокодировщик, 51, 763, 795, 808
 Вариационный вывод, 207

- Вариационный ЕМ-алгоритм, 393
 Вариация информации, 846
 Вектор, 292
 Векторное квантование, 853
 Векторное поле, 340, 805
 Векторное пространство, 295
 Вектор-столбец, 292
 Вентиль
 забывания, 614
 обновления, 613
 сброса, 613
 Вентильные графовые
 последовательностные нейронные
 сети, 896
 Верификация лиц, 658
 Вероятная приближенная корректность,
 257
 Вероятностная матричная
 факторизация, 876
 Вероятностная точка зрения, 34
 Вероятностное предсказание, 238
 Вероятностный анализ главных
 компонент, 789
 Вероятностный вывод, 84
 Вероятностный прогноз, 238
 Вероятностный симплекс, 189
 Верхнетреугольная матрица, 304
 Веса, 42
 Веса смеси, 188
 Вес внимания, 621
 Взаимная информация, 122, 280
 Взаимная ковариация, 121
 Взаимная независимость, 117
 Взаимозаменяемость, 150
 Взвешенная линейная регрессия, 461
 Взвешенных наименьших квадратов
 метод, 461
 Видимые переменные, 915
 Витроу–Хоффа правило, 368
 Визуальные n-граммы, 752
 Виртуальное состязательное обучение, 761
 Витерби декодирование, 616
 Вишарта нормальное обратное
 распределение, 397
 Включающее расхождение КЛ, 279
 Вложенная оптимизация, 256
 Вложенное разбиение, 871
 Внесловарные слова, 61, 64
 Внешнее произведение, 307
 Внимание, 624
 Внимание в форме масштабированного
 скалярного произведения, 623
 Внутреннее произведение, 307
 Внутренний дрейф ковариат, 573
 Внутренняя размерность, 815
 Вознаграждение, 54
 Вопросно-ответная система, 59, 648
 Вороного диаграмма, 654, 852
 Вороного метод итераций, 855
 Восполнение, 65
 Всемирная организация
 здравоохранения, 286
 Всеобъемлющая сегментация, 589
 Вспомогательные точки, 694
 Вульфа условия, 364
 Входной вентиль, 613
 Выбор ведущего элемента, 333
 Выборка
 по правилу квадратного корня, 443
 со сбалансированными классами, 442
 со сбалансированными экземплярами,
 442
 максимальной энтропии, 770
 Выбор модели, 242
 Выборочная дисперсия, 195
 Выборочная эффективность, 52
 Выборочное распределение, 209
 Выбор переменных, 479
 Выбросы, 102, 237, 443, 491
 Вывод, 153, 178
 Выделение
 именных групп, 648
 признаков, 46
 Выпуклая комбинация, 183
 Выпуклая оптимизация, 349
 Выпуклая функция, 350, 426
 Выпуклое множество, 349
 Выпуклое ослабление, 473
 Выходной вентиль, 613
 Выявление признаков, 565
- Г**
 Галерея, 587, 657
 Гамильтонов метод Монте-Карло, 208
 Гамма-распределение, 106
 Гауссов дискриминантный анализ, 402
 Гауссово распределение, 97
 Гауссово ядро, 553, 641, 670, 677
 Гауссов процесс, 554, 683
 Гельмгольца машина, 809
 Генератор случайных чисел, 116
 Геометрическая прогрессия, 168, 360
 Геометрическое глубокое обучение, 885
 Гессиан, 342, 426, 912
 Гетероскедастическая регрессия, 100, 461

- Гипероним, 440
 Гиперпараметры, 181, 197, 255, 399
 Гиперплоскость, 421
 Гиперстолбец, 569
 Гипотеза, 627
 Гипотеза многообразия, 814
 Гладкая оптимизация, 353
 Глобальная оптимизация, 346
 Глобально сходящийся алгоритм, 347
 Глобальный оптимум, 346, 426
 Глорота инициализация, 546
 Глубокая смесь экспертов, 558
 Глубокие машины факторизации, 881
 Глубокие нейронные сети, 47, 515
 Глубокий CCA, 802
 Глубокое обучение метрики, 659, 661
 ГНС, 47, 514
 Год жизни с поправкой на качество, 226
 Голова, 523
 Гомоскедастическая регрессия, 100
 Гомотопии метод, 487
 Градиент, 340, 356
 Градиентный бустинг, 732
 Градиентный бустинг деревьев, 734
 Грама матрица, 305, 312, 600, 677
 Грама-Шмидта ортогонализация, 306
 Граница объединения, 258
 Граничная оптимизация, 389, 436
 Граф
 вычислений, 536
 знаний, 905
 График
 каменистой осыпи, 785
 невязок, 465
 парных отношений, 37
 частичной зависимости, 738
 Графические процессоры, 526, 546
 Графовая вероятностная модель, 146
 Графовая взаимная информация, 903
 Графовая сеть внимания, 899
 Графовые модели, 77
 Графовые нейронные сети, 515, 895
 Графовые сверточные сети, 897
 Гребневая регрессия, 172, 218, 467, 548
 Группирующий эффект, 484
 Групповая нормировка, 574
 Групповая разреженность, 481
 Групповой lasso, 482
 Группы, 174, 257
 Гумбеля шум, 617
- Д**
 Дальнее управление, 775
 Датазаврова дюжина, 81
 Дважды смягченная логистическая регрессия, 446
 Двойное центрирование, 312
 Двойное экспоненциальное распределение, 105
 Двойственная задача, 699
 Двойственная форма, 700
 Двойственные переменные, 707
 Двумерное гауссово распределение, 126
 Двухнаправленная РНС, 607
 Двухуровневая оптимизация, 256
 Девиация, 719
 Дедукция, 262
 Действия, 225
 Декартовы координаты, 111
 Декодер, 802, 811
 Декодирование, 776
 Дельта-правило, 368
 Дельта-функция Дирака, 102, 201
 Дендриты, 527
 Дендрограмма, 846
 Деревья
 классификации и регрессии, 716
 регрессии с градиентным бустингом, 734
 Детальная классификация, 57, 773
 Деформация, 816
 Джини индекс, 719
 Диагонализируемая матрица, 321
 Диагональная ковариационная матрица, 127
 Диагональная матрица, 303
 Диагональное преобладание, 305
 Диаметр, 849
 Дикое обучение, 548
 Динамический граф, 537
 Динамическое программирование, 616
 Дирихле
 распределение, 189, 415
 энергия, 826
 Дискретизация, 275, 284
 Дискретная оптимизация, 346
 Дискретная случайная величина, 72
 Дискретный AdaBoost, 730
 Дискриминантная функция, 402
 Дискриминантный классификатор, 401, 417
 Дисперсия, 78, 99
 Дистилляция знаний, 767
 Дистрибутивная гипотеза, 832
 Дистрибутивность, 307
 Дифференциальная энтропия, 274
 Дифференцирование, 339

в обратном режиме, 531
 в прямом режиме, 531
 под знаком интеграла, 113
 Дифференцируемое
 программирование, 536
 Диффузное априорное распределение, 197
 Длинный хвост, 205, 441
 Добыча данных, 65
 Доверительный интервал, 198, 212, 214
 Долгая краткосрочная память, 613
 Доля объясненной дисперсии, 785
 Дообучение, 643, 742
 Дополнение
 матрицы, 875
 нулями, 566
 Дополнительный двойной логарифм, 508
 Допустимая оценка, 252
 Допустимое множество, 348
 Допустимость, 381
 Допустимость для двойственной
 задачи, 381
 Допущение
 замкнутости мира, 657
 многообразия, 759
 открытого мира, 658
 Достаточная статистика, 139, 157, 159, 180,
 288
 Дырявая свертка, 582, 589

Е

Единичная матрица, 303
 Единичный вектор, 93, 292
 Епанечникова ядро, 670
 Естественное экспоненциальное
 семейство, 139
 Естественно-языковой вывод, 627
 Естественные параметры, 139
 Естественный параллелизм, 547

Ж

Жадное декодирование, 615
 Жадный прямой выбор, 487
 Жесткая кластеризация, 144, 861
 Жесткий порог, 475, 479
 Жесткое внимание, 629

З

Зависимая переменная, 455
 Задача
 оптимизации, 346
 подстановки, 747

с конечной суммой, 367
 существования, 349
 Зазор, 164, 697, 728
 Замаскированная языковая модель, 645
 Замаскированное внимание, 621
 Замена переменных, 110
 Замещающие задачи, 748
 Замороженные параметры, 743
 Записи из электронной медицинской
 карты, 626
 Заполнение пропусков, 645, 747
 Запрос, 620
 Зарезервированный набор, 256
 Зашумленные метки, 443, 775
 Зигзагообразная траектория, 359
 Значения, 620
 Значимость, 260

И

Идентифицируемость, 252, 438
 Иерархическая агломеративная
 кластеризация, 846
 Иерархическая байесовская модель, 197
 Иерархическая softmax-модель, 441
 Иерархическая смесь экспертов, 559
 Иерархическое определение, 525
 Иерархия, 440
 Избегание
 нулей, 279
 риск, 227
 Извлечение имплицитных знаний
 из текста, 627
 Излом, 858
 Изотропная ковариационная матрица, 127
 ИИ, 66
 безопасность, 66
 этика, 66
 Имитация отжига, 81
 Импульс, 360
 Импульсное сопоставительное
 обучение, 750
 Инвариантность относительно сдвига, 561
 Индивидуальная нормировка, 574
 Индикаторная функция, 40, 73
 Индикаторное кодирование, 60, 93
 Индуктивное обучение, 760
 Индуктивное смещение, 48, 515, 636
 Индукция, 170, 262
 Индуцированная норма, 299
 Инкрементное обучение, 658
 ИНС, 526
 Инстаграм, 586

Интегральный риск, 250
 Интервал наивысшей плотности, 200
 Интерполированная точность, 234
 Интерполятор, 684
 Интерполяции изображений, 812
 Интерполяция, 45
 Интерпретируемость, 52
 Информационная диаграмма, 281
 Информационная проекция, 280
 Информационный выигрыш, 275, 769
 Информационный критерий, 246
 Информационный поиск, 233
 Искажение, 776, 851, 853
 Исключающее или, 555
 Исключающее расхождение КЛ, 279
 Исключение путем интегрирования, 179
 Исключение систематической ошибки, 476
 Исключение стоп-слов, 61
 Искривленное экспоненциальное семейство, 139
 Искусственные нейронные сети, 526
 Искусственный интеллект, 66
 Исходный домен, 752
 Ичисление, 339
 Итеративное усреднение, 371
 Итеративный алгоритм мягкого порога, 486

Й

Йенсена неравенство, 277, 392

К

Каждый против каждого, 706
 Калибровочная кривая, 511
 Калмана фильтр, 137
 Каналы, 561, 568
 Каноническая форма, 139
 Каноническая функция связи, 508
 Канонические параметры, 139
 Канонический корреляционный анализ, 802
 Карта признаков, 565
 Каруша–Куна–Такера условия, 381
 Касательное пространство, 813
 Категориальное распределение, 93
 Категориальный PCA, 798
 Каузальная свертка, 619
 Каузальная СНС, 618
 Каца индекс центральности, 891
 Квадратическая ошибка, 236
 Квадратичная аппроксимация, 206

Квадратичная потеря, 43, 236
 Квадратичная форма, 304, 323
 Квадратичное программирование, 382, 473
 Квадратичное ядро, 678
 Квадратично-экспоненциальное ядро, 677
 Квадратичный дискриминантный анализ, 402
 Квадратная матрица, 293
 Квадратный корень из матрицы, 299, 310, 334
 Квазиньютоновские методы, 364, 436
 Квазиреферирование, 649
 Квантиль, 75, 98
 Квантильная функция, 75
 Квантование, 275, 284, 388
 Квартиль, 75, 98
 ККТ, 381
 Классификатор
 максимальной энтропии, 439
 по ближайшему среднему классов, 407, 443
 по ближайшему центроиду, 407
 с широким зазором, 697
 Классификация, 35, 915
 без примеров, 752
 изображений, 35
 по малому числу примеров, 658
 с C классами и N примерами, 772
 спама, 58
 Классическая статистика, 208
 Классическое ММШ, 816
 Классы, 35
 Кластеризация, 143, 843
 по ближайшему соседу, 848
 по самому дальнему соседу, 848
 с выбором самой дальней точки, 855
 с одиночной связью, 848
 со средней связью, 849
 с полной связью, 848
 Кластерное допущение, 756
 Кластеры, 49
 Ключи, 620
 Ковариаты, 35, 455, 915
 Ковариационная матрица, 120, 126
 Ковариация, 120
 Кодирование, 776
 Кодирование пар байтов, 64
 Кодировщик, 802, 811
 Кодовая книга, 853
 Коллаборативная фильтрация, 869, 874
 Коммутативность, 307
 Компактность, 849
 Композиция, 525

- Компромисс
 между исследованием
 и использованием, 884
 между смещением и дисперсией, 217
- Компьютерная графика, 591
- Конгруэнтная свертка, 566
- Конечноразностная аппроксимация
 производной, 339
- Консенсусная последовательность, 270
- Конструирование
 описаний, 752
 признаков, 45
- Контекстуальные погружения слов, 63, 643, 840
- Контрольный набор, 48, 256
- Конформер, 638
- Корень из среднеквадратической
 ошибки, 162
- Корпус, 832
- Корректная свертка, 566
- Корреляционная матрица, 122, 161
- Косинусное ядро, 681
- Коши распределение, 104
- Коэффициент
 Байеса, 240, 258
 ветвления, 273
 детерминации, 466
 дырявости, 582
 корреляции, 121, 126
 равных ошибок, 233
 усадки, 476, 733
- Коэффициенты регрессии, 42, 455
- Кривая
 обучения, 177
 точность-полнота, 233
- Критериальная статистика, 259
- Критерий отношения правдоподобия, 259
- Критическая точка, 379
- Кронекера произведение, 313
- Крыловского типа методы, 694
- Ксавье инициализация, 546
- Кубические сплайны, 488
- Кульбака–Лейблера расхождение, 155, 238, 275, 393
- Кусочно-линейная потеря, 164, 399, 704, 880
- Л**
- Лагранжа
 множители, 142, 158, 379
 нотация, 340
- Лагранжиан, 142, 326, 349, 379, 473
- Ланцоша алгоритм, 792
- Лапласа
 аппроксимация, 206, 447
 метод векторов, 711
 правило следования, 185
 распределение, 105, 472
 сглаживание, 415
- Лапласиан графа, 825, 866
- Лапласовы собственные отображения, 824, 867, 890
- Латентная интерполяция, 812
- Латентно-семантический анализ, 833
- Латентно-семантическое
 индексирование, 833
- Латентные переменные, 143, 915
- Латентные факторы, 50, 51, 779
- Латентный вектор, 779
- ЛДА, 401
- Лейбница нотация, 340
- Лексема, 61
- Лекуна инициализация, 546
- Лемма
 об обращении матрицы, 317, 707
 об определителе матрицы, 318
- Ленточная матрица, 303
- Линейная алгебра, 292
- Линейная гауссова система, 132
- Линейная зависимость, 296
- Линейная комбинация, 308
- Линейная независимость, 296
- Линейная оболочка, 296
- Линейная пороговая функция, 516
- Линейная разделимость, 421
- Линейная регрессия, 100, 455, 506, 514
- Линейная скорость, 359
- Линейная функция, 42
- Линейное отображение, 296
- Линейное подпространство, 308
- Линейное преобразование, 296
- Линейное программирование, 492
- Линейное ядро, 692
- Линейность математического
 ожидания, 78
- Линейный автокодировщик, 802
- Линейный дискриминантный анализ, 401, 403
- Линейный оператор, 460
- Линейный поиск, 358
- Линии уровня, 127
- Липшица постоянная, 353
- Логарифмическая потеря, 239, 704
- Логарифмическая функция разбиения, 139
- Логарифмически-билинейная языковая
 модель, 839

Логарифмическое правдоподобие, 154
 Логарифм отношения шансов, 92
 Логистическая регрессия, 42, 92, 202, 420, 505
 Логистическая функция, 92, 202
 Логит, 41, 94, 420, 432
 Логотип последовательности, 270
 Ложная корреляция, 123
 Локальная минимизация риска, 742
 Локально линейная регрессия, 675
 Локально линейное погружение, 823
 Локально-чувствительное
 хеширование, 657
 Локальный максимум, 347
 Локальный минимум, 347
 Локальный оптимум, 347, 426
 Лоренца распределение, 104
 Лоренцева модель, 891
 ЛСА, 833
 ЛСИ, 833
 Лучевой поиск, 617
 с повышенным разнообразием, 617

М

Мажорирование, 252, 261
 Мажорирование-минимизация, 389
 Мак–Каллока–Питтса модель, 527
 Максимальная энтропия, 101, 269
 Максимальное правдоподобие типа II, 198
 Максимальный коэффициент
 информации, 285
 Максимальный риск, 251
 Манипулирование вознаграждением, 66
 Манхэттенское расстояние, 846
 Маргинализация, 179, 201
 Маргинальное правдоподобие, 84, 179, 187, 197, 242
 Маргинальное распределение, 76
 Марковская модель, 148
 порядка M , 149
 Марковская цепь, 148
 Марковское условие первого порядка, 148
 Марковское ядро, 148
 Масштабирование Платта, 703
 Математическое ожидание, 78, 99
 Матерна ядро, 680
 Матрица, 293
 весов позиций, 270
 неточностей классификации, 231
 переходных вероятностей, 148
 плана, 37, 310, 515, 915
 поворота, 305
 предобуславливания, 374
 проекции, 459
 рассеяния, 160, 312
 с блочной структурой, 316
 сумм квадратов, 311
 точности, 129, 159
 факторных нагрузок, 787
 Матричная факторизация, 875
 Махаланобиса
 отбеливание, 324
 расстояние, 128, 654
 Машина
 неустойчивых состояний, 612
 факторизации, 881
 экстремального обучения, 696
 Машинное обучение, 34
 Машинный перевод, 59, 609
 Медиана, 75, 98
 Медианное абсолютное отклонение, 672
 Медоид, 855
 Межквартильный диапазон, 82
 Мерсера
 теорема, 678
 ядро, 677
 Мертвый ReLU, 542
 Мета-обучение, 770, 774
 Метка, 35
 Метода главных компонент, 50, 776
 Метод
 ближайших соседей с большим
 зазором, 659
 внутренней точки, 382
 второго порядка, 362, 428
 комитета, 722
 моментов, 165
 Монте-Карло по схеме марковских
 цепей, 208
 наименьших квадратов с итеративным
 пересчетом весов, 429
 опорных векторов, 697
 пробуждения-засыпания, 809
 разреженных векторов, 711
 релевантных векторов, 711
 частичных наименьших квадратов, 801
 Метрика, 275
 Метрическое ММШ, 817
 Механизм отсутствия данных, 64
 Мешок, 775
 погружений слов, 63
 слов, 61, 522
 Минимаксная оценка, 251
 Минимаксное шкалирование, 432
 Минимальная достаточная статистика, 288

- Минимальное остовное дерево, 848
 - Минимальное представление, 140
 - Минимально информативное априорное распределение, 197
 - Минимальный уровень шума, 177
 - Минимизация
 - структурного риска, 256
 - с учетом остроты, 552
 - эмпирического риска, 40, 163, 254, 367
 - энтропия, 756
 - Мини-пакет, 367
 - ММШ, 816
 - Многозначная классификация, 433, 440
 - Многозначные функции, 555
 - Многоклассовая классификация, 433
 - Многоклассовая логистическая регрессия, 420
 - Многомерная бернуллиева наивная байесовская модель, 413
 - Многомерная линейная регрессия, 456
 - Многомерное гауссово распределение, 126
 - Многомерное нормальное распределение, 126
 - Многомерное шкалирование, 816, 890
 - Многомодальное распределение, 79
 - Многообразие, 813
 - Многопутевое внимание, 632
 - Многослойный перцептрон, 515, 516
 - Многоуровневая модель, 197
 - Многофакторная линейная регрессия, 44, 456
 - Многоцелевое сопровождение, 658
 - Множественная подстановка, 132
 - Мода, 79, 228
 - Моделенезависимое мета-обучение, 772
 - Модель
 - бернуллиевой смеси, 145
 - векторного пространства, 62
 - гауссовой смеси, 143
 - максимальной энтропии, 142
 - на основе энергии, 749
 - на основе эталонов, 653
 - с латентными переменными, 915
 - стохастической волатильности, 523
 - Модификация первого ранга, 318
 - Монте-Карло
 - аппроксимация, 116, 208, 451
 - прореживание, 551
 - Монти Холла парадокс, 86
 - Мотив последовательности ДНК, 269
 - Мощность критерия, 260
 - МСП, 515, 516
 - Мультикластеризация, 871
 - Мультиномиальная логистическая регрессия, 95, 420, 432
 - Мультиномиальная функция logit, 94
 - Мультиномиальное распределение, 94
 - Мультиномиальный коэффициент, 94
 - Мура-Пенроуза псевдообратная матрица, 329
 - Мягкая кластеризация, 144
 - Мягкая триплетная потеря, 666
 - Мягкий порог, 475, 478
- ## Н
- На всех парах, 706
 - Надарая-Ватсона модель, 674
 - Надграфик, 350
 - Наиболее мощный критерий, 261
 - Наивное байесовское предположение, 406, 412
 - Наивный байесовский классификатор, 412
 - Наивысшая апостериорная плотность, 199
 - Наименее выгодное априорное распределение, 251
 - Наименьших средних квадратов метод, 368, 463
 - Наименьших углов метод, 487
 - Нанизывание меток, 440
 - Направление
 - скорейшего спуска, 356
 - спуска, 356
 - Настройка с возвращениями, 490
 - Насыщение, 518
 - Насыщенная модель, 511
 - Нат, 269
 - Наука о данных, 65
 - Начальный блок, 578
 - Невошедшие в набор экземпляры, 723
 - Невязка, 43, 162, 465
 - Негладкая оптимизация, 354
 - Недообученность, 48, 173, 176
 - Недоопределенная система, 335
 - Независимые и одинаково распределенные величины, 115, 154, 180
 - Независимые переменные, 455
 - Независимые случайные величины, 76
 - Не зависящая от времени модель, 148
 - Неидентифицируемость, 438, 501, 861
 - Неинформативное априорное распределение, 182, 197
 - Неймана-Пирсона теорема, 261
 - Нейронная матричная факторизация, 882
 - Нейронная сеть прямого распространения, 515

- Нейронная языковая модель, 149
 Нейронные сети передачи сообщений, 896
 Нейронный машинный перевод, 609
 Нейронный перенос стиля, 597
 Нейтральная связь, 627
 Некорректно поставленная задача, 88
 Нелинейное понижение размерности, 813
 Нелинейный факторный анализ, 795
 Неметрическое ММШ, 818
 Неокогнитрон, 571
 Неопределенная матрица, 304
 Неопределенность, 69
 данных, 41, 70
 модели, 41, 70
 Непараметрическая модель, 554
 Непараметрические методы, 676, 813
 Непараметрические модели, 653
 Непараметрический бутстрэп, 211
 Неполного ранга, 301
 Непрерывная оптимизация, 346
 Непрерывная случайная величина, 73
 Непрерывное обучение, 658
 Неравенство обработки данных, 287
 Неравенство треугольника, 275
 Неразрешимая задача, 381
 Несбалансированность классов, 233, 441, 705
 Несмещенная оценка, 215
 Нестерова ускоренный метод градиентов, 361
 Неструктурированные данные, 515
 Неустойчивость, 720
 Неявная обратная связь, 879
 Неявная регуляризация, 552
 Нижнетреугольная матрица, 304
 Норма, 298, 302
 Нормализованный разрез, 866
 Нормализующие потоки, 766
 Нормальное распределение, 43, 97
 Нормальные уравнения, 338, 457
 Нормированная взаимная информация, 284, 846
 Нормированный вектор, 305
 Нормировка по слою, 574
 Нулевая гипотеза, 240, 258
 Нулевое пространство, 297
 Нулевой счетчик, 170
 Ньютона метод, 362, 428
- посторонних, 658
 сущностей, 851
 Обобщаемость, 41, 169
 Обобщение на вневыворочные примеры, 813
 Обобщенная аддитивная модель, 490
 Обобщенная пробит-аппроксимация, 452
 Обобщенная проблема собственных значений, 410
 Обобщенные линейные модели, 505
 Обобщенные низкоранговые модели, 797
 Обобщенный лагранжиан, 380, 699
 Обобщенный ССА, 802
 Обработка естественного языка, 57, 439
 Образ, 297
 Обратная вероятность, 88
 Обратная матрица, 315
 Обратная функция распределения, 75
 Обратная частота документа, 62
 Обратное гамма-распределение, 107
 Обратное обучение с подкреплением, 66
 Обратное распределение Уишарта, 171
 Обратное распространение, 518
 алгоритм, 529
 во времени, 610
 Обратное расхождение КЛ, 279
 Обратные задачи, 89
 Обратный ход, 336, 460
 Обращение знака градиента, 753
 Обрезание градиента, 540
 Обрезка, 719
 Обучающие данные, 915
 Обучающий набор, 35
 Обучение
 без примеров, 772
 без учителя, 49
 втроем, 759
 графовых представлений, 886
 метрики ближайшего среднего классов, 407
 многообразия, 813
 модели, 40, 153
 на малом числе примеров, 772
 на нескольких экземплярах, 775
 на одном примере, 407, 772
 на основе запоминания, 653
 на примерах, 653
 на протяжении всей жизни, 658
 представлений, 748
 с критиком, 54
 со слабым учителем, 774
 с подкреплением, 53, 884
 с учителем, 35
- О**
 Обнаружение новизны, 657

- с частичным привлечением учителя, 418, 753
- Общая скрытая причина, 123
- Объединенная пакетная нормировка, 573
- Объективное априорное распределение, 197
- Объясняющие переменные, 455
- Обыкновенных наименьших квадратов метод, 162, 338, 457
- Ограничение порядка, 861
- Ограничения, 348
 - в виде неравенств, 348, 377
 - в виде равенств, 348, 377
- Ограничения мягкого зазора, 701
- Ограничивающие прямоугольники, 586
- Один против всех, 705
- Один против остальных, 705
- Однопроходный детектор, 588
- Однородная модель, 148
- Однофазный план скоростей обучения, 370
- Ожидаемое полное логарифмическое правдоподобие данных, 393
- Ожидаемые достаточные статистики, 393
- Ожидание-максимизация, 389
- Оккама
 - бритва, 245
 - коэффициент, 247
- Оконечные устройства, 388, 585
- Онлайновое обучение, 167, 389, 658, 727
- ОП, 53
- Оператор
 - мягкого порога, 388
 - обратной косой черты, 336
- Описание изображений, 605
- Опорные векторы, 697, 701, 710
- Оправдание, 149
- Оправдывающие задачи, 748
- Определение
 - лиц, 587
 - объектов, 586
- Определитель, 300
- Оптимальная политика, 226
- Оптимизация
 - без использования производных, 399
 - в доверительной области, 365
 - черного ящика, 399
- Оптимистичность ошибки на обучающем наборе, 256
- Ориентированный ациклический граф, 146, 536
- Орнштейна–Уленбека процесс, 680
- Ортогональная проекция, 459
- Ортогональность, 305
- Ортогональные случайные признаки, 696
- Ортодоксальная статистика, 208
- Ортонормированность, 305, 322
- Осепараллельное разбиение, 716
- Осепараллельный эллипс, 128
- Основная задача, 699
- Основной эффект, 60
- Основные переменные, 707
- Основополагающие переменные, 795
- Остаточная сеть, 544
- Остаточный блок, 544, 579
- Острые минимумы, 551
- Отбеливание данных, 324
- Отбор признаков, 288, 387, 471
- Ответственность, 144, 394, 558
- Отклик, 35
- Отклонение, 229
- Открытый класс, 64
- Открытый мир, 586
- Относительная энтропия, 275
- Отношение
 - правдоподобия, 240, 262
 - сигнал-шум, 136
- Отрицательное логарифмическое правдоподобие, 42, 154
- Отрицательно определенная матрица, 304
- Отрицательно полуопределенная матрица, 304
- Отсутствие
 - вполне случайное, 64
 - не случайное, 64
 - случайное, 64
- Отсутствующие данные, 64, 391
- Оценивание плотности, 52
- Оценивание позы человека, 590
- Оцениватель, 209
- Оценка
 - BIC, 247, 858
 - KSG, 284
 - апостериорного максимума, 169, 228
 - внимания, 621
 - максимального правдоподобия, 42, 154
 - минимума среднеквадратической ошибки, 237
 - усадки, 171
- Ошибка
 - аппроксимации, 255
 - зазора, 702
 - защитника, 118
 - обобщения, 255, 257
 - оценивания, 255
 - прокурора, 118
 - реконструкции, 776, 779, 853

П

- Пакетная нормировка, 572
- Пакетная перенормировка, 573
- Пакетное обучение, 167
- Парадокс черного лебедя, 170
- Параметрические модели, 653
- Параметрический бутстрэп, 211
- Параметрический ReLU, 542
- Параметр
 - рассеяния, 505
 - регуляризации, 169
- Параметры, 39
- Парцена оконная оценка плотности, 671
- Патчи изображения, 562
- Перекрестная корреляция, 563
- Перекрестная проверка, 174, 257
- Перекрестная проверка с исключением по одному, 175, 257
- Перекрестная энтропия, 239, 271, 276, 278
- Перекрестные произведения признаков, 60
- Перекрестный риск, 175, 257
- Переменная метрика, 364
- Переменные
 - невязки, 701, 709
 - отклика, 915
- Перенос
 - обучения, 643, 668, 742
 - обучения без примеров, 649
- Переобучение, 47, 169, 184
- Переопределенная система, 335, 458
- Перепараметризованная модель, 96
- Переходная функция, 148
- Переходное ядро, 148
- Периодическое ядро, 680
- Перплексия, 273, 604
- Перцептрон, 427
- Пифагора теорема, 337
- План изменения скорости обучения, 356, 369
- Планирование эксперимента, 768
- Плановая выборка, 610
- План по закону квадратного корня, 370
- Плоские минимумы, 551
- Плоский локальный минимум, 347
- Плоское априорное распределение, 197
- Плотная разметка
 - последовательности, 607
- Плотное предсказание, 589
- Плохо обусловленная матрица, 161, 302
- Площадь под кривой, 233
- Победитель забирает всё, 95
- Побочная информация, 882
- Поверхностный синтаксический анализ, 648
- Повторная идентификация человека, 658
- Погружение, 776
 - слов, 63, 455, 832
 - с частичным привлечением учителя, 903
- Подгонка модели, 40
- Подозрительное совпадение, 241
- Подсказка, 649
- Подслова, 64
- Подстановка отсутствующих значений, 131
- Подстановка среднего значения, 65
- Подстановочная аппроксимация, 184, 201, 450
- Позиционное погружение, 632
- Поиск
 - архитектуры нейронной сети, 582
 - документов, 833
 - мод, 280
 - на сетке, 174, 399
- Покоординатный спуск, 485
- Покрытие мод, 279
- Полиномиальная регрессия, 45, 172, 456
- Политика, 53
- Политоп, 381
- Полная вариация, 594
- Полная ковариационная матрица, 127
- Полная производная, 341
- Полная редукция, 547
- Полного ранга, 301
- Полноматричный Adagrad, 377
- Полнота, 231, 234
- Полный дифференциал, 341
- Половинное распределение Коши, 104
- Положительная PMI, 834
- Положительно определенная матрица, 304
- Положительно определенное ядро, 677
- Положительно полуопределенная матрица, 304
- Полоса пропускания, 554, 670, 677
- Полуинварианты, 141
- Полуопределенное погружение, 822
- Полуопределенное
 - программирование, 660, 823
- Полупространство, 421
- Полутвердо отрицательные примеры, 665
- Поляка–Рупперта усреднение, 371
- Полярные координаты, 111
- Понижение
 - порядка матрицы, 326, 841
 - размерности, 37, 776
- Понимание
 - визуальной сцены, 88
 - естественного языка, 88

- Попарная независимость, 117
 Поппер Карл, 170
 Порождающая изображения модель, 591
 Порождающие состязательные сети, 753
 Порождающий классификатор, 401, 417
 Порядковые статистики, 166
 Порядок
 по столбцам, 295
 тензора, 294
 Последовательная минимальная оптимизация, 700
 Последовательно-послойная инициализация с единичной дисперсией, 546
 Посылка, 627
 Потенциал действия, 527
 Потеря
 BIC, 247
 ранжирования, 662, 880
 Поточковое активное обучение, 768
 Поточечная взаимная информация, 834
 Поточечная свертка, 569
 Правдоподобие, 84
 Правдоподобие профиля, 786
 Правило
 верной оценки, 240
 дифференцирования сложной функции, 343
 одной стандартной ошибки, 175
 повторного математического ожидания, 151
 сложения вероятностей, 76
 умножения вероятностей, 76, 83
 Преактивация, 420, 518
 Предварительно обученная матрица погружений, 63
 Предвзятость подтверждения, 755
 Предиктор, 35
 Предобработка признаков, 45
 Предобусловленный SGC, 374
 Предобусловливатель, 374
 Предобучение, 643, 742
 без учителя, 746
 Предпочтения, 226
 Предсказание
 глубины, 589
 нормали к поверхности, 589
 связей, 905
 следующего предложения, 645
 Представители, 665
 Преобразование изображения в изображение, 589
 Приближенный апостериорный вывод, 205
 Привратник, 557
 Признаки, 35
 Принуждение со стороны учителя, 610
 Принцип
 максимальной ожидаемой полезности, 227
 минимальной длины описания, 248
 правдоподобия, 264
 Принятие решений в условиях неопределенности, 34
 Приращение данных, 279, 740
 Пробит-аппроксимация, 451
 Пробит-функция, 98, 451
 Пробит-функция связи, 508
 Проблема
 аналогичных слов, 838
 взрывного градиента, 539
 исчезающего градиента, 518, 539
 переключения меток, 398, 861
 скученности, 829
 Проверка
 гипотез, 240
 значимости нулевой гипотезы, 260
 Прогнозная аналитика, 65
 Прогрев скорости обучения, 370
 Продолжения метод, 487
 Проекция, 297
 момента, 279
 Произведение
 вектора на якобиан, 342
 якобиана на вектор, 342
 Производная, 339
 по направлению, 340
 Проклятие размерности, 655
 Проксимальный градиентный спуск, 385, 486
 Проксимальный оператор, 385
 Прореживание, 549
 Простая гипотеза, 260
 Пространственная раздельная свертка, 584
 Пространство
 весов, 687
 гипотез, 254
 параметров, 346
 состояний, 72
 столбцов, 297
 функций, 688
 элементарных событий, 72
 Противоречие, 627
 Процентильная функция, 75
 Прямая модель, 89
 Прямое поэтапное аддитивное моделирование, 726

Прямое расхождение КЛ, 279
 Прямоугольное ядро, 669, 671
 Прямые связи, 579
 Псевдовходы, 694
 Псевдонорма, 298
 Псевдообратная матрица, 338, 457
 Псевдопометка, 754
 Псевдоправдоподобие, 645
 Псевдосчетчики, 181, 415
 Пуанкаре модель, 891
 Пуассона регрессия, 507
 Пулинг
 глобальным усреднением, 522, 571
 усреднением, 570
 Путь регуляризации, 471, 476, 486

Р

Рабочая характеристика, 429
 приемника, 231
 Равноценность, 286
 Радиально-базисная функция, 554, 641, 670
 Разбиение
 вокруг медоидов, 855
 графа, 866
 Разведочный анализ данных, 37
 Развертывание с максимальной дисперсией, 823
 Разветвление
 по входу, 545
 по выходу, 545
 Разворачивание модели, 150
 Разложение по базисным функциям, 514
 Размазывание меток, 895
 Размер
 выборки, 35, 157, 180
 шага, 356
 Размерность объемлющего пространства, 815
 Разметка семантических ролей, 439
 Разреженная линейная регрессия, 384
 Разреженная ядерная машина, 554, 656
 Разреженное байесовское обучение, 502
 Разреженность, 190, 471
 Разреженный ГП, 694
 Разреженный факторный анализ, 795
 Разрешение
 кореференции, 631
 лексической неоднозначности, 644
 сущностей, 648, 658
 Разрыв обобщения, 47, 255
 Разумный РСА, 789
 Ранг, 294, 301

Ранняя остановка, 176, 548
 Распараллеливание
 модели, 547
 по данным, 547
 Распознавание
 именованных сущностей, 648
 лиц, 587
 образов, 35
 открытого множества, 657
 Распознающая сеть, 808
 Распределение
 Больцмана, 94
 вероятностей, 34
 наблюдаемых значений, 84
 Распространение меток, 759, 894
 Распространенность, 85, 235
 Распрямление, 520
 Расширяющее представление, 802
 Расщепление переменной, 486
 Регрессия, 915
 главных компонент, 470
 опорных векторов, 710
 Регуляризатор, стимулирующий
 разреженность, 387
 Регуляризация, 169
 активности, 806
 по согласованности, 760
 Регуляризованный дискриминантный анализ, 406
 Регуляризованный эмпирический риск, 255
 Редуцированное QR-разложение, 333
 Резервуарные вычисления, 612
 Рейтинг, 873
 Рекомендательные системы, 873, 905
 Рекуррентные нейронные сети, 47, 515, 602
 Рекурсивное вычисление MLE, 462
 Рекурсивное обновление, 167
 Реляционные данные, 873
 Реферирование, 649
 документов, 59
 Рецептивное поле, 568, 582
 Речевой синтез, 619
 Решатель, 346
 Решающая граница, 38, 92, 202, 421
 Решающая поверхность, 39
 Решающее дерево, 39, 716
 Решающее правило, 38
 Риманова метрика, 813
 Риманово многообразие, 813
 Риск, 226, 248
 на генеральной совокупности, 47, 174, 254

на контрольном наборе, 174, 256
 на тестовом наборе, 47
 РНС, 515, 602
 Робастная линейная регрессия, 382
 Робастная логистическая регрессия, 443
 Робастность, 102, 237, 491
 Роббинса–Монро условия, 369
 Рулет с кремом, 816
 Рэлея отношение, 325
 Рэнда индекс, 845

С

Саммона отображение, 819
 Самовнимание, 631
 Самозванцы, 659
 Самообучение, 754, 767
 Самостоятельное обучение, 51, 746
 Сбор информации, 41
 Свертка, 113, 563
 с дырками, 582
 с шагом, 568
 Сверточная марковская модель, 618
 Сверточная нейронная сеть, 36, 58
 Сверхполное представление, 140
 Свидетельство, 242
 Свидетельствующая нижняя граница, 392
 Свободный член, 42
 Свойство
 самонормировки, 839
 следа, 300
 циклической перестановки, 300
 Связи-глазки, 615
 Связывание
 параметров, 148
 сущностей, 658
 Сглаживание
 локально-взвешенной диаграммы
 рассеяния, 675
 меток, 767, 775
 прибавлением единицы, 170, 182, 415
 Сглаживающие сплайны, 490
 СГС, 366
 Сегментация экземпляров, 588
 Седловая точка, 348, 352
 Семантическая сегментация, 589
 Семантическое сходство текстов, 629
 Сети без нормировки, 575
 Сеточная аппроксимация, 206
 Сеть
 вывода, 808
 на основе смеси моделей разной
 плотности, 558
 Сжатие, 806
 данных, 52, 511, 854
 изображений, 854
 модели, 548
 с потерей информации, 853
 Сжимающий автокодировщик, 806
 Сиамская сеть, 663, 748
 Сигмоидная функция, 91, 92, 202, 404
 Силуэтная диаграмма, 858
 Силуэтная оценка, 858
 Силуэтный коэффициент, 858
 Сильная выпуклость, 352
 Сильный обучаемый, 726
 Симметричная матрица, 294
 Симметричное SNE, 829
 Симплекс-метод, 382
 Симпсона парадокс, 123
 Синаптическая связь, 527
 Сингулярная матрица, 315
 Сингулярное разложение, 328
 Сингулярные векторы, 327
 Сингулярные числа, 302, 327
 Синтаксический сахар, 150
 Синтез запросов, 768
 Синхронное обучение, 547
 Система обнаружения мошеннических
 действий, 905
 Системы линейных уравнений, 335
 Скалярное поле, 340
 Скалярное произведение, 307
 Сквозной оценщик, 388
 Скелет, 523
 Скипграмма с отрицательной
 выборкой, 836
 Скипграммная модель, 836
 СКО, 43, 162
 Скользящее среднее, 167
 Скорость
 обучения, 356
 сходимости, 359
 Скорректированный индекс Рэнда, 845
 Скрипичная диаграмма, 82
 Скрытая случайная величина, 83
 Скрытые блоки, 517
 Скрытые переменные, 149, 391, 915
 Слабый обучаемый, 726
 След, 300
 Следование, 627
 Следовая норма, 299
 Слияние показаний датчиков, 138
 Сложная гипотеза, 260
 Слои нормировки, 571
 Случайная величина, 72

- Случайные конечные множества, 658
 Случайный лес, 724
 Смесовая модель, 142
 Смесь
 бета-распределений, 188
 гауссиан, 143
 распределений Бернулли, 145
 факторных анализаторов, 796
 экспертов, 557, 638
 Смешанно-целочисленное
 программирование, 384
 Смещение, 42, 44, 215, 456
 Смягченная перекрестная энтропия, 444
 Смягченная softmax, 445
 СНС, 36, 515, 562
 Собеля детектор границ, 594
 Собственная информация, 268
 Собственное значение, 320
 Собственные лица, 778
 Собственный вектор, 320
 Событие, 70
 Совместная адаптация, 550
 Совместная вероятность, 71
 Совместная кластеризация, 869
 Совместное обучение, 758
 Совместное распределение, 75
 Согласованность с выбором модели, 479
 Сопоставительная потеря, 663
 Сопоставительные задачи, 748
 Сопоставление с шаблоном, 562, 565
 Сопоставляющая сеть, 773
 Сопряженная функция, 350
 Сопряженное априорное
 распределение, 133, 179
 Сопряженный градиент, 359, 460
 Составная целевая функция, 354
 Состояние природы, 225
 Состоятельная оценка, 252, 497
 Состоятельное обучение доменов, 753
 Сохранения массы вероятности закон, 245
 Социальные сети, 905
 Спектральная кластеризация, 865
 Спектральная теория графов, 826
 Спектральное погружение, 824
 Спектральное разложение, 320
 Спектральные СНС, 897
 Спектральный радиус, 539
 Спектр собственных значений, 171
 Специфичность, 85
 Спроецированный градиентный
 спуск, 386, 486
 Среднее, 78, 99
 Среднее гармоническое, 235
 Среднеквадратическая ошибка, 43, 162
 Средняя усредненная точность, 234
 Стандартизация, 311, 323, 431, 461
 Стандартная ошибка, 183
 среднего, 195, 175
 Стандартная форма, 381
 Стандартное нормальное
 распределение, 98
 Стандартное отклонение, 78, 100
 Стандартный базис, 296
 Старый Служака, 396
 Статистика, 65
 Статистическая значимость, 261
 Статистическая теория обучения, 257
 Статистический машинный перевод, 609
 Статистическое реляционное
 обучение, 907
 Статический граф, 537
 Стационарная модель, 148
 Стационарная точка, 348
 Стационарное ядро, 678
 Стековое обобщение, 722
 Стемминг, 61
 Степень нормальности, 103
 Степень свободы, 48, 103, 470
 Степень согласия, 465
 СТО, 257
 Столбцовый ранг, 301
 Стохастическая матрица, 148
 Стохастическая оптимизация, 366
 Стохастический градиентный бустинг, 734
 Стохастический градиентный спуск, 366, 427
 с теплым перезапуском, 371
 Стохастический градиент с уменьшенной
 регрессией, 372
 Стохастический лучевой поиск, 617
 Стохастический усредненный
 ускоренный градиент, 373
 Стохастическое погружение соседей, 827
 Стохастическое усреднение весов, 372, 762
 Строгий локальный минимум, 347
 Строго выпуклая функция, 350
 Строковое ядро, 683
 Строковый ранг, 301
 Структурированные данные, 515
 Структурное погружение с помощью
 глубокой сети, 900
 Ступенчатая функция, 108
 Ступенчатое затухание, 370
 Стьюдента t-распределение, 103
 Субградиент, 354
 Субдифференциал, 355

Субдифференцируемая функция, 355
 Субмодулярная функция, 770
 Субпроизводная, 535
 Сужающее представление, 802
 Сужение, 802
 Сумма
 квадратов невязок, 162, 457
 с нарастающим итогом, 167
 Суррогатная функция, 390
 потеря, 163
 Суррогатные разделения, 720
 Сферическая ковариационная матрица, 127
 Сферическое ограничение на погружение, 668
 Сходство, 653
 СЦПП, 384

Т

Таблица условных вероятностей, 147
 Табличные данные, 37, 515
 Таксономия, 440
 Твердо отрицательные примеры, 665
 Тейлора ряд, 206, 290
 Температура, 94
 Тензор, 294, 568
 Тензорный процессор, 528
 Теорема
 об отсутствии бесплатных завтраков, 48
 о голландской книге, 263
 о полном классе, 252
 о ранге и дефекте, 331
 о свертке, 113
 Теория
 возмущений, 867
 информации, 239, 268
 Теплый запуск, 471, 486
 Терм-документная матрица, 62, 832
 Тестовый набор, 47
 Тихонова регуляризация, 366
 Топологическая неустойчивость, 820
 Топологический порядок, 146
 Точечная оценка, 153
 Точная визуальная классификация, 742
 Точность, 98, 193, 234
 Точность на K элементах, 234
 Точный линейный поиск, 358
 Тощий SVD, 327
 Трансдуктивное обучение, 760
 Транспонирование, 293
 Транспонированная свертка, 583, 589
 Трансформер, 515, 629

Трассировка, 537
 Трехкубовое ядро, 670
 Триграммная модель, 149
 Тридиагональная матрица, 303
 Триpletная потеря, 663
 Трубка, 709
 Трюк перепараметризации, 810
 Тьюринга машина, 604
 Тяжелого шарика метод, 360
 Тяжелые хвосты, 103, 491

У

Угловой коэффициент, 44
 Уменьшение
 весов, 172, 430, 467, 548
 при выходе на плато, 370
 Умножение матрицы на вектор, 694
 Универсальный аппроксиматор функций, 524
 Уникальность, 789
 Унитарная матрица, 305
 Унитарное кодирование, 60, 434, 915
 Унитарный вектор, 93, 292
 Упорядоченное марковское свойство, 146
 Управляемый рекуррентный блок, 612
 Управляющая переменная, 372
 Усадка, 136, 194, 470
 Усеченное сингулярное разложение, 331
 Условие дополняющей нежесткости, 381
 Условная вероятность, 71
 Условная взаимная информация, 282
 Условная индивидуальная нормировка, 600
 Условная независимость, 72, 77, 146
 Условная оптимизация, 348, 377, 492
 Условная плотность класса, 401
 Условная смесовая модель, 556
 Условная энтропия, 272
 Условное вычисление, 557
 Условное распределение, 76
 Условное распределение вероятностей, 41, 90, 147
 Усредненная точность, 234

Ф

Факторизации графов, 892
 Факторный анализ, 50, 787
 Факторный анализ экспоненциального семейства, 797
 Фано неравенство, 288
 Фастфуд, 696

Фильтр, 562, 563
 Фильтрующее свойство, 102, 201
 Фичеризация, 37
 Фишера информационная матрица, 210, 429
 Фишера критерий, 429
 Фишера линейный дискриминантный анализ, 409
 Флуктуация, 555
 Форма миски, 426
 Формула Байеса, 83
 для гауссовых распределений, 133
 обращения клеточных матриц, 316
 полного математического ожидания, 80
 полной вероятности, 76
 полной дисперсии, 80
 условной дисперсии, 80
 Форсирование нулей, 280
 ФПР, 74
 Фрагментация данных, 718
 Фробениуса норма, 299
 Функция
 вклада, 210, 805
 вознаграждения, 346
 двоичной энтропии, 269
 напряжения, 817
 оценки, 346
 ошибок, 98
 плотности распределения, 74
 полезности, 226
 потерь, 40, 225, 346
 правдоподобия, 179
 разбиения, 96, 139, 749
 распределения, 73, 98
 связи, 505, 508
 среднего, 505
 стоимости, 346
 энергии, 206
 Фурье случайные признаки, 695

Х

Характеристическая матрица, 285
 Характеристический линейный масштаб, 679
 Характеристическое уравнение, 321
 Хатчинсона оценка следа, 299
 Хаффмана кодирование, 441
 Хевисайда ступенчатая функция, 91, 427, 516, 535
 Хе инициализация, 546
 Хёфдинга неравенство, 258
 Хи-квадрат распределение, 107

Хинтона диаграмма, 131
 Холески разложение, 468
 Холодный старт, 882
 Хорошо обусловленная матрица, 302
 Хьюбера функция потерь, 237, 493, 733

Ц

Целевая функция, 154, 346
 метода наименьших квадратов, 338
 Целевой домен, 752
 Целевой набор данных, 742
 Целочисленное линейное программирование, 384
 Цель, 35, 455, 915
 Ценность информации, 769
 Центральная предельная теорема, 101, 115
 Центральным интервал, 198
 Центрирующая матрица, 160, 312, 821
 Центроиды, 553
 Цепное правило
 для вероятностей, 76
 для взаимной информации, 282
 для энтропии, 272
 Циклический план скоростей
 обучения, 371
 ЦЛП, 384

Ч

Частеречная разметка, 644, 648
 Части слов, 64
 Частично наблюдаемое состояние, 225
 Частная производная, 340
 Частный коэффициент регрессии, 462
 Частота
 истинно-отрицательных результатов, 85
 истинно-положительных
 результатов, 84, 231, 232
 ложноотрицательных результатов, 85
 ложноположительных результатов, 85,
 231, 232
 ложных тревог, 231
 неправильной классификации, 39, 163
 ошибок первого рода, 231, 260
 попаданий, 231
 Частотная статистика, 208
 Частотная теория принятий решений, 248
 Частотный подход, 69
 Чередующихся наименьших квадратов
 метод, 875
 Черный ящик, 399
 Число обусловленности, 172, 301, 359

Чистота кластера, 844
 Чистый узел, 719
 Чувствительность, 84, 232
 к риску, 227

Ш

Шаттена р-норма, 299
 Шеннона теорема о кодировании
 источника, 271
 Шермана–Моррисона–Вудбери
 формула, 317
 Шермана–Моррисона формула, 318
 Широкая resnet, 581
 Широкий формат, 61
 Шкала силы свидетельства, 240
 Штейна парадокс, 252
 Штраф за сложность, 169
 Штрафующий член, 385
 Шумоподавляющий автокодировщик, 804
 Шура дополнение, 129, 316

Э

Эвристики, 529
 Эйнштейна суммирование, 314
 Эквивалентный размер выборки, 181
 Эквивариантность, 569
 Экзогенные переменные, 35
 Экономное QR-разложение, 333
 Экономное SVD, 327
 Экспоненциальная перекрестная
 энтропия, 273
 Экспоненциальная потеря, 728
 Экспоненциально взвешенное скользящее
 среднее, 167
 Экспоненциальное распределение, 107
 Экспоненциальное семейство, 139, 142,
 196
 Экспоненциальное семейство
 рассеяния, 505
 Экспоненциальное скользящее
 среднее, 167
 Экспоненциально-квадратичное ядро, 677
 Экспоненциальный линейный блок, 541
 Экстрактор признаков, 456

Эластичная сеть, 478, 484
 Эластичное погружение, 828
 Эмпирический байесовский подход, 198,
 243, 501
 Эмпирический риск, 40, 254
 Эмпирическое распределение, 108, 116,
 155, 278
 Эндогенные переменные, 35
 Энскомба квартет, 81
 Энтропийный СГС, 552
 Энтропия, 239, 268, 392, 719
 Эпистемическая неопределенность, 41, 70
 Эпистемология, 70
 Эпоха, 367
 Эпсилон-нечувствительная функцией
 потерь, 708
 Эффекты взаимодействия, 60
 Эхо-сеть, 612

Я

Явная обратная связь, 873
 Ядерная гребневая регрессии, 686, 708
 Ядерная норма, 299
 Ядерная оценка плотности, 669, 671
 Ядерная регрессия, 622, 674, 686
 Ядерная функция, 553, 676
 Ядерное сглаживание, 674
 Ядерный трюк, 702
 Ядерный PCA, 821, 868
 Ядро, 563, 669
 ARD, 678
 внимания, 773
 матрицы, 297
 плотности, 622, 669
 случайного блуждания, 683
 Языковая модель, 148
 Языковое моделирование, 59, 604
 Языковые модели, 273, 643
 Якобиан, 111, 341, 434
 Якоби нотация, 341
 Якорные рамки, 587
 Якорь, 663
 Ячейка памяти, 613
 Ящик с усами, вид диаграммы, 83

Книги издательства «ДМК ПРЕСС»
можно купить оптом и в розницу
в книготорговой компании «Галактика»
(представляет интересы издательств
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).

Адрес: г. Москва, пр. Андропова, 38, оф. 10;
тел.: **(499) 782-38-89**, электронная почта: **books@aliants-kniga.ru**.

При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.

Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **<http://www.galaktika-dmk.com/>**.

Кэвин П. Мэрфи

Вероятностное машинное обучение Введение

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com

Зам. главного редактора *Сенченкова Е. А.*

Перевод *Снастин А. В.*

Корректор *Абросимова Л. А.*

Верстка *Чаннова А. А.*

Дизайн обложки *Мовчан А. Г.*

Гарнитура РТ Serif. Печать цифровая.

Усл. печ. л. 76,38. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**

Данный классический труд содержит обстоятельное современное введение в машинное обучение (включая глубокое обучение), рассматриваемое сквозь объединяющую призму вероятностного моделирования и байесовской теории принятия решений. Включен базовый математический аппарат (в т. ч. элементы линейной алгебры и теории оптимизации), основы обучения с учителем (включая линейную и логистическую регрессию и глубокие нейронные сети), а также более сложные темы (в т. ч. перенос обучения и обучение без учителя). Упражнения в конце глав помогут читателям применить полученные знания, а в приложении имеется сводка используемых обозначений.

В основу издания легла вышедшая в 2012 году книга Кэвина Мэрфи «Machine Learning: A Probabilistic Perspective». Однако это совершенно новая работа, отражающая многие достижения, случившиеся в этой области за последние 10 лет.

«Революция, которую принесло нам глубокое обучение, изменила весь ландшафт машинного обучения на протяжении последнего десятилетия. Источникам вдохновения для этой дисциплины стали попытки имитировать то, как обучается мозг человека, но основана она на принципах статистики, теории информации, теории принятия решений и оптимизации. Автор книги прекрасно справился с задачей объяснения этих принципов. В результате получилась стройная система, позволяющая понять связи и компромиссы между разными подходами к машинному обучению, как старыми, так и новыми».

Джеффри Хинтон, член инженерного совета Google, заслуженный профессор информатики, Торонтский университет



Кэвин Патрик Мэрфи получил степень бакалавра в Кэмбридже, Англия, и продолжил образование в США (магистр технических наук в Пенсильванском университете, доктор в Калифорнийском университете в Беркли, постдокторантура в МТИ). В 2004 году занял должность профессора информатики и статистики в Университете Британской Колумбии в Ванкувере. Работает в отделении Google в Маунтин-Вью, где занимается искусственным интеллектом, машинным обучением, компьютерным зрением и пониманием текстов на естественном языке.

Интернет-магазин:
www.dmkpress.com
Оптовая продажа:
КТК «Галактика»
books@alians-kniga.ru

 The MIT Press


www.dmk.pф

ISBN 978-5-93700-119-1



9 785937 001191 >