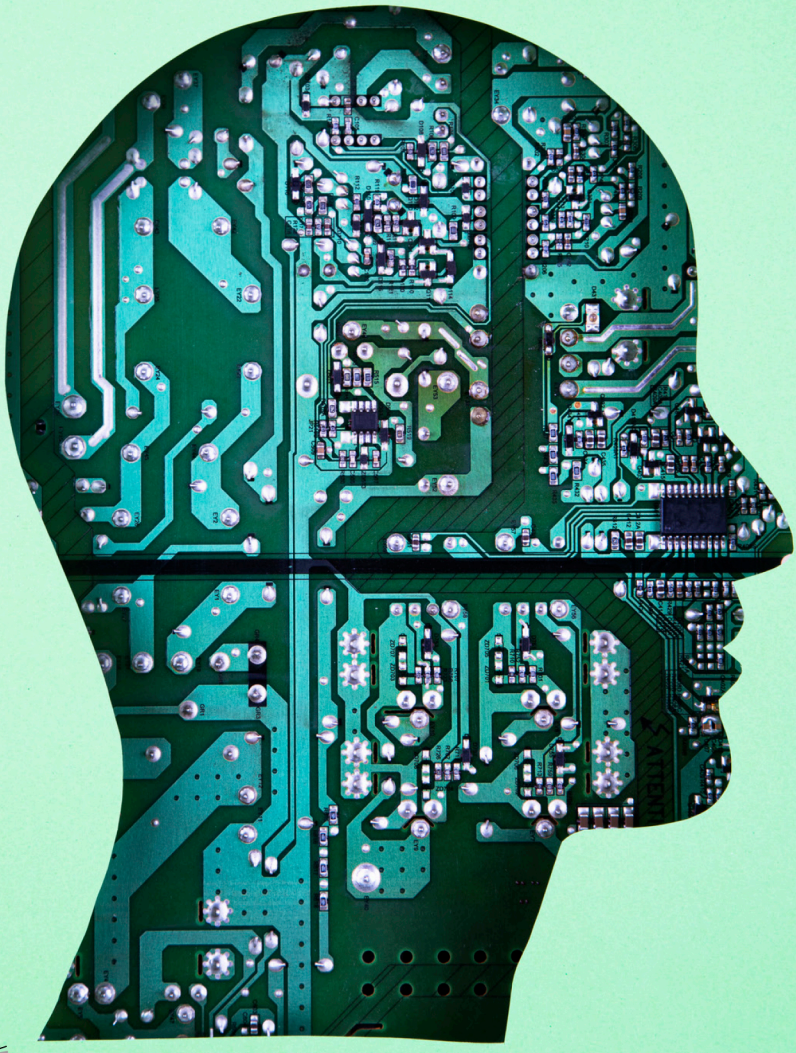


ДМК издательство

Хуттер Ф., Коттхофф Л., Ваншорен Х.

Введение в автоматизированное машинное обучение (AutoML)

Ошеломляющий успех коммерческих приложений машинного обучения (machine learning – ML) и быстрый рост этой отрасли создали высокий спрос на готовые методы ML, которые можно легко использовать без специальных знаний. Однако и сегодня успех практического применения в решающей степени зависит от экспертов – людей, которые вручную выбирают подходящие архитектуры и их гиперпараметры. Методы AutoML нацелены на устранение этого узкого места путем построения систем ML, способных к автоматической оптимизации и самонастройке независимо от типа входных данных.

В этой книге впервые представлен всеобъемлющий обзор базовых методов автоматизированного машинного обучения (AutoML). Издание послужит отправной точкой для изучения этой быстро развивающейся области; тем, кто уже использует AutoML в своей работе, книга пригодится в качестве справочника.

Среди рассматриваемых тем:

- оптимизация гиперпараметров;
- обучение модели на основе свойств задачи;
- обзор методов для NAS;
- системы и фреймворки AutoML;
- результаты проведения первых конкурсов в области AutoML;
- проблемы автоматизированного машинного обучения.

Разносторонний анализ всех аспектов AutoML

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК «Галактика»
e-mail: books@alians-kniga.ru

ДМК издательство
www.дмк.рф

ISBN 978-5-93700-196-2

Франк Хуттер
Ларс Коттхофф
Хоакин Ваншорен

Введение в автоматизированное машинное обучение (AutoML)

Frank Hutter • Lars Kotthoff • Joaquin Vanschoren

Automated Machine Learning

OPEN

Springer

Франк Хуттер • Ларс Коттхофф • Хоакин Ваншорен

Введение в автоматизированное машинное обучение (AutoML)

ДМК Пресс
издательство
Москва, 2023

УДК 004.4
ББК 32.972
Х98

Хуттер Ф., Коттхофф Л., Ваншорен Х.
Х98 Введение в автоматизированное машинное обучение (AutoML) / пер. с англ. В. С. Яценкова. – М.: ДМК Пресс, 2023. – 256 с.: ил.

ISBN 978-5-93700-196-2

Ошеломляющий успех коммерческих приложений машинного обучения (machine learning – ML) и быстрый рост этой отрасли создали высокий спрос на готовые методы ML, которые можно легко использовать без специальных знаний. Однако и сегодня успех практического применения в решающей степени зависит от экспертов – людей, которые вручную выбирают подходящие архитектуры и их гиперпараметры. Методы AutoML нацелены на устранение этого узкого места путем построения систем ML, способных к автоматической оптимизации и самонастройке независимо от типа входных данных. В этой книге впервые представлен всеобъемлющий обзор базовых методов автоматизированного машинного обучения (AutoML).

Издание послужит отправной точкой для изучения этой быстро развивающейся области; тем, кто уже использует AutoML в своей работе, книга пригодится в качестве справочника.

УДК 004.4
ББК 32.972

ISBN 978-1-80181-497-3 (англ.)

ISBN 978-5-93700-196-2 (рус.)

Содержание

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

«Я хотел бы использовать машинное обучение, но не могу уделять ему много времени». Мы слишком часто слышим это от самых разных людей, от инженеров до исследователей, работающих в других областях. Им нужны готовые решения для машинного обучения, не требующие трудоемкой подготовки и отладки. В ответ на возникший спрос появилась новая отрасль *автоматизированного машинного обучения* (automated machine learning, AutoML), и я рад, что у читателей будет первое исчерпывающее руководство в этой области.

Я сам очень увлечен автоматизацией машинного обучения с тех пор, как в 2014 г. стартовал наш проект Automatic Statistician. Я хочу, чтобы все исследователи AutoML стремились к амбициозной цели: мы должны попытаться автоматизировать все аспекты полного конвейера машинного обучения и анализа данных. Сюда входит автоматизация сбора данных и организации экспериментов; очистки данных и подстановки недостающих данных; выбора и преобразования признаков; исследования, критического оценивания и объяснения моделей; распределения вычислительных ресурсов; оптимизации гиперпараметров; вывода; мониторинга моделей и обнаружения аномалий. Это очень обширный список, и в идеале следует автоматизировать все, что в нем перечислено.

Конечно, здесь нужно сделать оговорку. Хотя полная автоматизация служит хорошей мотивацией для исследователей и долгосрочной целью для инженеров, на практике пользователи предпочитают *полуавтоматизацию* и постепенное выведение человека из цикла машинного обучения по мере необходимости. Но выгода от автоматизации заключается не только в исключении человеческого труда. Продвигаясь к полной автоматизации, мы попутно разрабатываем мощные инструменты, которые помогут сделать практику машинного обучения прежде всего более систематичной (поскольку в наши дни она очень ситуативна и хаотична), а также более эффективной.

Это достойные цели, даже если мы не преуспеем в полной автоматизации, но, как показывает эта книга, текущие методы AutoML уже превосходят человеческих экспертов машинного обучения в ряде задач. Эта тенденция, вероятно, будет только усиливаться по мере нашего прогресса и по мере того, как вычисления становятся все дешевле, и поэтому AutoML явно является одной из «горячих» тем. Сейчас самое время заняться AutoML, и данная книга – отличная отправная точка.

Книга содержит актуальные обзоры основных методов, необходимых в AutoML (оптимизация гиперпараметров, метаобучение и поиск нейронной архитектуры), подробное обсуждение существующих систем AutoML и де-

тальную оценку состояния дел в AutoML на основе серии конкурсов, которые проводятся с 2015 г. Поэтому я настоятельно рекомендую эту книгу каждому исследователю машинного обучения, желающему начать работу в этой области, и каждому практикующему специалисту, желающему понять методы, лежащие в основе всех существующих инструментов AutoML.

Зубин Гахрамани
Профессор Кембриджского университета
и главный научный сотрудник Uber
Сан-Франциско, США
Октябрь 2018 г.

Введение

В последнее десятилетие наблюдается бурный рост количества исследований и применений машинного обучения; в частности, методы глубокого обучения позволили добиться значительных успехов во многих прикладных областях, таких как компьютерное зрение, обработка речи и игры. Однако прикладные решения в области машинного обучения чрезвычайно чувствительны к многочисленным нюансам реализации проектов, что служит серьезным препятствием для новых пользователей. Это особенно актуально для бурно развивающейся области глубокого обучения, где разработчикам приложений приходится выбирать правильные нейронные архитектуры, процедуры обучения, методы регуляризации и гиперпараметры всех этих компонентов, чтобы заставить свои сети делать то, чего от них ждут, с достаточной производительностью[1]. Процесс подбора подходящей модели машинного обучения приходится повторять для каждого приложения. Даже опытные специалисты часто бывают вынуждены проходить через утомительную серию проб и ошибок, пока не найдут удачный вариант для конкретного набора данных.

Область автоматизированного машинного обучения нацелена на принятие конструкторских решений на основе данных объективным и автоматизированным способом: пользователь просто предоставляет данные, а система AutoML автоматически находит оптимальное решение для этого конкретного случая. Таким образом, AutoML делает машинное обучение доступным для тех, кто не имеет возможности детально изучать технологии, лежащие в его основе. Это можно рассматривать как *демократизацию* машинного обучения: с AutoML современное машинное обучение доступно каждому.

Как мы покажем в этой книге, подходы AutoML уже достаточно развиты, чтобы соперничать, а иногда и превосходить экспертов в области машинного обучения. Проще говоря, AutoML может привести к повышению производительности, экономя при этом значительное количество времени и денег, поскольку хороших экспертов в области машинного обучения трудно найти, и их услуги дорого стоят. В результате в последние годы интерес

---

[1] Термин *performance*, который в IT-литературе обычно переводят как «производительность», на самом деле очень многозначен. В машинном обучении в зависимости от контекста он может означать точность модели, ее быстродействие, стабильность работы, а иногда все одновременно, т. е. совокупный уровень качества. В этой книге мы тоже используем перевод «производительность», подразумевая под ним свойства модели, зависящие от контекста. – *Прим. перев.*

инвесторов к AutoML резко возрос, и несколько крупных технологических компаний сейчас разрабатывает собственные системы AutoML. Однако стоит отметить, что цели демократизации машинного обучения гораздо лучше служат системы AutoML с открытым исходным кодом, чем платные черные ящики.

Эта книга представляет собой обзор быстро развивающейся области AutoML. В связи с тем, что в настоящее время научное сообщество сосредоточено на глубоком обучении, некоторые исследователи ошибочно приравнивают AutoML к *поиску нейронной архитектуры* (neural architecture search, NAS); но вы наверняка знаете, что, хотя NAS является отличным примером AutoML, понятие AutoML намного шире, чем NAS. Цель этой книги – предоставить исходные данные и отправные точки для исследователей, заинтересованных в разработке собственных подходов к AutoML, рассказать о доступных системах практикам, которые хотят применить AutoML для решения своих задач, и предоставить обзор состояния дел исследователям, уже работающим в AutoML. Книга разделена на три части, посвященные этим различным аспектам AutoML.

Часть I представляет собой обзор методов AutoML. Она содержит масштабный обзор для новичков и может служить справочником для опытных исследователей AutoML.

В главе 1 рассмотрена задача оптимизации гиперпараметров – простейшая и наиболее распространенная задача, которую решает AutoML, – и описан широкий спектр различных подходов с особым акцентом на методы, которые в настоящее время являются наиболее эффективными.

В главе 2 показано, как *научить модель учиться*, т. е. как использовать полученный ранее опыт оценки моделей машинного обучения при решении новых задач обучения с новыми данными. Такие методы имитируют процесс развития навыков человека от новичка к эксперту и могут значительно сократить время, необходимое для получения хороших результатов при решении совершенно новых задач машинного обучения.

В главе 3 представлен полный обзор методов для NAS. Это одна из самых сложных задач в AutoML, поскольку пространство проектирования чрезвычайно велико, а одна оценка нейронной сети может занять очень много времени. Тем не менее в этой области ведутся активные исследования, и регулярно появляются новые интересные подходы к решению задачи NAS.

Часть II посвящена реальным системам AutoML, которые могут применять даже начинающие пользователи. Если вы заинтересованы в применении AutoML для решения задач машинного обучения, вам следует начать именно с этой части. Все главы этой части подробно описывают представленные в них системы, чтобы дать представление об их эффективности на практике.

В главе 4 описана Auto-WEKA, одна из первых систем AutoML. Она основана на хорошо известном наборе инструментов машинного обучения WEKA и перебирает различные методы классификации и регрессии, настройки их гиперпараметров и методы предварительной обработки данных. Все это

доступно через графический пользовательский интерфейс WEKA одним нажатием кнопки, без необходимости написания кода.

В главе 5 представлен обзор Hyperopt-Sklearn – фреймворка AutoML, основанного на популярном фреймворке scikit-learn. Глава также содержит несколько практических примеров использования системы.

В главе 6 описан фреймворк Auto-sklearn, который также основан на scikit-learn. В нем применяются те же методы оптимизации, что и в Auto-WEKA, и добавлено несколько улучшений по сравнению с другими системами того времени, например метаобучение для теплого старта оптимизации и автоматическое ансамблирование. В главе сравнивается производительность Auto-sklearn с производительностью двух систем из предыдущих глав, Auto-WEKA и Hyperopt-Sklearn. К слову, Auto-sklearn – это система, которая победила в испытаниях, описанных в части III данной книги.

В главе 7 представлен обзор Auto-Net – системы автоматического глубокого обучения, которая выбирает архитектуру и гиперпараметры глубоких нейронных сетей. Даже начальная версия Auto-Net позволила создать первую автоматически настраиваемую нейронную сеть, которая победила в соревновании с экспертами-людьми.

В главе 8 описана система TPOT, которая автоматически строит и оптимизирует древовидные конвейеры машинного обучения. Эти конвейеры более гибкие, чем подходы, которые рассматривают только набор фиксированных компонентов машинного обучения, соединенных заранее определенными способами.

В главе 9 описана система Automatic Statistician, позволяющая автоматизировать науку о данных путем создания полностью автоматизированных отчетов, включающих анализ данных, а также прогностические модели и сравнение их эффективности. Уникальной особенностью Automatic Statistician является то, что она предоставляет описания результатов на естественном языке, подходящие для неспециалистов в области машинного обучения.

Наконец, в части III и главе 10 представлен обзор задач в области AutoML, над которыми исследователи работают с 2015 г. Назначение этого обзора – стимулировать разработку методов, которые хорошо справляются с практическими задачами, и определить лучший подход. В главе подробно описаны идеи и концепции, лежащие в основе текущих исследовательских задач, а также результаты предыдущих исследований.

Насколько нам известно, это первый разносторонний анализ всех аспектов AutoML: методов, лежащих в его основе, доступных систем, реализующих AutoML на практике, и задач для их оценки. Эта книга содержит справочную информацию, достаточную для начала разработки собственных систем AutoML, а также подробно описывает существующие системы, которые можно непосредственно применить для решения широкого круга задач машинного обучения. Область AutoML стремительно развивается, поэтому мы постарались объяснить и систематизировать последние достижения. Мы надеемся, что вам понравится эта книга и вы присоединитесь к растущему сообществу энтузиастов AutoML.

Благодарности

Мы хотим поблагодарить всех авторов глав, благодаря которым удалось издать эту книгу. Мы также благодарны программе исследований и инноваций Horizon 2020 Европейского союза за покрытие расходов на издание этой книги.

Франк Хуттер
Фрайбург, Германия

Ларс Коттхофф
Ларами, штат Вайоминг, США

Хоакин Ваншорен
Эйндховен, Нидерланды

Октябрь 2018 г.

Часть I

МЕТОДЫ AutoML

Глава 1

Оптимизация гиперпараметров

Маттиас Фойрер (✉), факультет компьютерных наук, Университет Фрайбурга, Фрайбург, Баден-Вюртемберг, Германия, e-mail: feurerm@informatik.uni-freiburg.de

Франк Хуттер, факультет компьютерных наук, Университет Фрайбурга, Фрайбург, Германия

Растущий интерес к сложным и вычислительно дорогим моделям машинного обучения с большим количеством гиперпараметров, таким как системы автоматического машинного обучения (AutoML) и глубокие нейронные сети, привел к возрождению исследований по *оптимизации гиперпараметров* (hyperparameter optimization, HPO). В этой главе мы представим обзор наиболее известных подходов к реализации HPO. Сначала мы обсудим способы оптимизации *функций черного ящика* (blackbox function), основанные на безмодельных методах и байесовской оптимизации. Поскольку высокие вычислительные требования многих современных приложений машинного обучения делают прямую оптимизацию методом черного ящика чрезвычайно дорогостоящей, далее мы сосредоточимся на современных методах *переменной точности* (multi-fidelity method), которые используют гораздо более дешевые в вычислительном плане аппроксимированные варианты функции черного ящика для приблизительной оценки качества настройки гиперпараметров. В завершение главы мы обсудим нерешенные проблемы и направления будущих исследований.

1.1. Введение

Каждая система машинного обучения имеет гиперпараметры, и самой основной задачей в автоматизированном машинном обучении является автоматическая настройка этих гиперпараметров для оптимизации производительности. Современные глубокие нейронные сети особенно сильно зависят

от широкого спектра гиперпараметров, определяющих архитектуру нейронной сети, регуляризацию и оптимизацию. Автоматизированная оптимизация гиперпараметров имеет несколько важных применений и позволяет:

- сократить человеческие трудозатраты, необходимые для применения машинного обучения. Это особенно важно в контексте AutoML;
- улучшить производительность алгоритмов машинного обучения (адаптируя их к конкретной задаче). Это привело к появлению новых важных эталонов машинного обучения, предложенных в нескольких исследованиях (например, [105, 140]);
- улучшить воспроизводимость и достоверность научных исследований. Автоматизированный процесс HPO явно более воспроизводим, чем ручной поиск. Он облегчает справедливое сравнение научных результатов, поскольку различные методы можно справедливо сравнивать только в том случае, если все они обладают одинаковым уровнем настройки для решения поставленной задачи [14, 133].

Задача HPO имеет долгую историю, восходящую к 1990-м гг. (например, [77, 82, 107, 126]), и уже тогда было установлено, что различные конфигурации гиперпараметров работают лучше для разных наборов данных [82]. В отличие от этого знания довольно новым является понимание, что методику HPO можно использовать для адаптации конвейеров общего назначения к конкретным областям применения [30]. В настоящее время также считается общепризнанным фактом, что настраиваемые гиперпараметры лучше, чем настройки по умолчанию, предоставляемые распространенными библиотеками машинного обучения [100, 116, 130, 149].

В связи с растущим использованием машинного обучения в компаниях HPO также представляет значительный коммерческий интерес и играет все большую роль в инструментах внутри компании [45] как часть облачных сервисов машинного обучения [6, 89] или как самостоятельная услуга [137].

Реализация HPO сталкивается с несколькими проблемами, которые затрудняют применение технологии на практике:

- оценка функций для больших моделей (например, в глубоком обучении), сложных конвейеров машинного обучения или больших наборов данных обходится чрезвычайно дорого;
- пространство конфигураций часто бывает сложным (состоящим из смеси непрерывных, категориальных и условных гиперпараметров) и многомерным. Кроме того, не всегда ясно, какие из гиперпараметров алгоритма необходимо оптимизировать и в каких диапазонах;
- обычно у нас нет доступа к градиенту функции потерь относительно гиперпараметров. Более того, другие свойства целевой функции, часто используемые в классической оптимизации, например выпуклость и гладкость, обычно не применимы к гиперпараметрам;
- невозможно напрямую оптимизировать эффективность обобщения, поскольку наборы обучающих данных имеют ограниченный размер.

Заинтересованных читателей, желающих глубже изучить данную проблематику, мы отсылаем к другим обзорам HPO [64, 94].

Данная глава построена следующим образом. Сначала мы формально определим задачу HPO и обсудим ее варианты (раздел 1.2). Затем мы об-

судим алгоритмы оптимизации черного ящика для решения задачи HPO (раздел 1.3). Далее мы сосредоточимся на современных методах переменной точности, которые позволяют использовать HPO для оптимизации даже очень дорогостоящих в вычислительном плане моделей благодаря приблизительным мерам производительности, которые дешевле, чем полные оценки модели (раздел 1.4). Затем мы представим обзор наиболее важных систем оптимизации гиперпараметров и приложений к AutoML (раздел 1.5) и завершим главу обсуждением нерешенных проблем (раздел 1.6).

1.2. Постановка задачи

Обозначим за $\mathcal{A}$ алгоритм машинного обучения с N гиперпараметрами. Обозначим область n-го гиперпараметра через Λ_n, а общее *пространство конфигураций гиперпараметров* как $\boldsymbol{\Lambda} = \Lambda_1 \times \Lambda_2 \times \ldots \Lambda_N$. Вектор гиперпараметров обозначим как $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$, а алгоритм $\mathcal{A}$ с его гиперпараметрами, представленными в $\boldsymbol{\lambda}$, обозначим как $\mathcal{A}_{\boldsymbol{\lambda}}$.

Область гиперпараметра может быть вещественной (например, скорость обучения), целочисленной (например, количество слоев), бинарной (например, использовать или нет раннюю остановку) или категориальной (например, выбор оптимизатора). Для целочисленных и вещественных гиперпараметров области в основном ограничены практическими соображениями, за некоторыми исключениями [12, 113, 136].

Кроме того, пространство конфигураций может обладать *условностью*, т. е. один гиперпараметр может иметь смысл только в том случае, если другой гиперпараметр (или некоторая комбинация гиперпараметров) принимает определенное значение. Условные пространства имеют форму направленных ациклических графов. Такие условные пространства возникают, например, при автоматической настройке конвейеров машинного обучения, когда выбор между различными алгоритмами предварительной обработки и машинного обучения моделируется как категориальный гиперпараметр – задача, известная как *полный выбор модели* (full model selection, FMS), или комбинированная задача *выбора алгоритма и оптимизации гиперпараметров* (combined algorithm selection and hyperparameter optimization, CASH) [30, 34, 83, 149]. Они также возникают при оптимизации архитектуры нейронной сети: например, число слоев может быть целочисленным гиперпараметром, а гиперпараметры каждого слоя i активны только в том случае, если глубина сети не меньше i [12, 14, 33].

При заданном наборе данных $\mathcal{D}$ наша цель состоит в том, чтобы найти

$$\boldsymbol{\lambda}^* = \underset{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}}{\operatorname{argmin}} \, \mathbb{E}_{(D_{train}, D_{valid}) \sim \mathcal{D}} \mathbf{V}(\mathcal{L}, \mathcal{A}_{\boldsymbol{\lambda}}, D_{train}, D_{valid}), \tag{1.1}$$

где $\mathbf{V}(\mathcal{L}, \mathcal{A}_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$ измеряет потери модели, созданной алгоритмом $\mathcal{A}$ с гиперпараметрами $\boldsymbol{\lambda}$ на обучающих данных D_{train} и оцененной на проверочных данных D_{valid}. На практике мы имеем доступ только к конечным данным $D \sim \mathcal{D}$ и поэтому должны аппроксимировать ожидание в уравнении (1.1).

Популярными вариантами протокола проверки $\mathbf{V}(\cdot, \cdot, \cdot, \cdot)$ являются выделение контрольных данных (holdout) и ошибка перекрестной проверки для заданной пользователем функции потерь (например, коэффициент неправильной классификации); обзор протоколов проверки представлен в статье Бишля и др. [16]. Существуют несколько стратегий для сокращения времени оценки: можно тестировать алгоритмы машинного обучения только на подмножестве сверток [149], только на подмножестве данных [78, 102, 147] или на небольшом количестве итераций; мы обсудим некоторые из этих стратегий более подробно в разделе 1.4. В публикациях по многозадачной [147] и многоисточниковой [121] оптимизации были предложены дополнительные вычислительно дешевые вспомогательные задачи, которые можно решать вместо уравнения (1.1). Они способны без особых вычислительных затрат предоставить информацию для поддержки процесса HPO, но не обязательно обучают модель на нужном наборе данных и поэтому не дают пригодную для использования модель в качестве побочного продукта.

1.2.1. Альтернативы оптимизации: ансамблирование и маргинализация

Решение уравнения (1.1) с помощью одного из методов, описанных в остальной части этой главы, обычно требует подбора алгоритма машинного обучения $\mathcal{A}$ с несколькими векторами гиперпараметров $\boldsymbol{\lambda}_t$. Вместо использования оператора argmin можно либо построить ансамбль (который стремится минимизировать потери для заданного протокола валидации), либо интегрировать все гиперпараметры (если рассматриваемая модель является вероятностной). Сравнение выбора моделей по частотному и байесовскому методу представлено в работе Гайона и др. [50] и ссылках на нее.

Использование только одной конфигурации гиперпараметров может быть расточительным, если процесс HPO нашел несколько потенциально хороших конфигураций, и их объединение в ансамбль может существенно улучшить производительность [109]. Это особенно полезно в системах AutoML с большим пространством конфигураций (например, в FMS или CASH), где хорошие конфигурации могут быть очень разнообразными, что увеличивает потенциальный выигрыш от их объединения [4, 19, 31, 34]. В целях дальнейшего повышения производительности метод Automatic Frankensteining [155] использует HPO для обучения стековой модели [156] на выходах моделей, найденных с помощью HPO; модели второго уровня затем объединяются с помощью традиционной стратегии ансамблирования.

В упомянутых методах ансамблирование применяют после процедуры HPO. Хотя это улучшает производительность на практике, базовые модели не оптимизированы для ансамблирования. Можно выполнить прямую оптимизацию этих моделей, что позволит максимально улучшить существующий ансамбль [97].

Наконец, при работе с байесовскими моделями часто удается интегрировать гиперпараметры алгоритма машинного обучения, например, используя

максимизацию доказательств [98], *усреднение байесовской модели* [56], *выборку по уровням* [111] или *эмпирический байесовский подход* [103].

1.2.2. Оптимизация по нескольким целям

На практике часто бывает необходимо найти компромисс между двумя или более целями, такими как производительность модели и потребление ресурсов [65] (см. также главу 3) или несколько функций потерь [57]. Возможные решения могут быть получены двумя способами.

Во-первых, если известно ограничение на вторичный показатель производительности (например, максимальное потребление памяти), то задачу можно сформулировать как задачу оптимизации с ограничениями. Мы обсудим обработку ограничений в байесовской оптимизации в разделе 1.3.2.4.

Во-вторых (и это более общий случай), можно применить многокритериальную оптимизацию для поиска *фронта Парето* – набора конфигураций, которые являются оптимальным компромиссом между целями в том смысле, что для каждой конфигурации на фронте Парето не существует другой конфигурации, которая работает лучше хотя бы для одной цели и хотя бы столь же хорошо для всех других целей. Затем пользователь может выбрать конфигурацию с фронта Парето. Мы отсылаем заинтересованного читателя к дополнительной литературе по этой теме [53, 57, 65, 134].

1.3. Оптимизация гиперпараметров методом черного ящика

В общем случае любой метод оптимизации черного ящика применим к HPO. Вследствие невыпуклой природы задачи алгоритмы глобальной оптимизации обычно предпочтительнее, но определенная локальность в процессе оптимизации полезна для того, чтобы добиться заметного прогресса за несколько оценок функции. Сначала мы обсудим методы HPO без моделей, а затем опишем методы байесовской оптимизации с черным ящиком.

1.3.1. Оптимизация методом черного ящика без моделей

Поиск по сетке (grid search) – это самый базовый метод HPO, также известный как *полное факторное планирование* (full factorial design) [110]. Пользователь задает конечное множество значений для каждого гиперпараметра, а поиск по сетке оценивает декартово произведение этих множеств. Такой подход страдает от проклятия размерности, поскольку требуемое количество оценок функций растет экспоненциально по мере увеличения размерности

пространства конфигураций. Дополнительный недостаток поиска по сетке заключается в том, что увеличение разрешения дискретизации (уменьшение шага гиперпараметров) существенно увеличивает требуемое число оценок функций.

Простой альтернативой поиску по сетке является *случайный поиск* (random search)[1] [13]. Как следует из названия, случайный поиск выбирает конфигурации случайным образом, пока не будет исчерпан определенный бюджет на поиск. Этот подход работает лучше, чем поиск по сетке, когда некоторые гиперпараметры намного важнее других (это свойство имеет место во многих случаях [13, 61]). Интуитивно понятно, что при фиксированном бюджете в B оценок функций количество различных значений, которые может позволить себе поиск по сетке для каждого из N гиперпараметров, составляет только $B^{1/N}$, в то время как случайный поиск будет исследовать B различных значений для каждого из них (рис. 1.1).

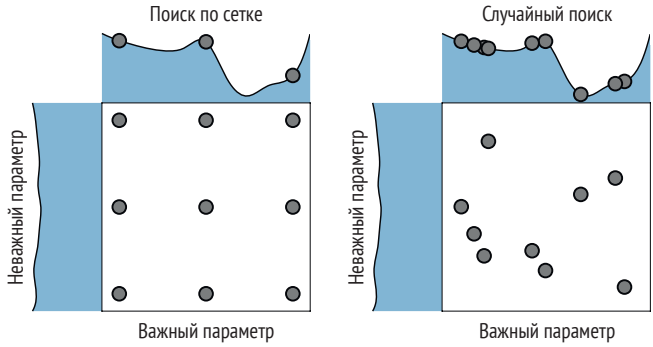


Рис. 1.1 ❖ Сравнение поиска по сетке и случайного поиска для минимизации функции с одним важным и одним неважным параметром. Этот рисунок основан на рис. 1 из работы Бергстры и Бенджио [13]

К дополнительным преимуществам случайного поиска в сравнении с поиском по сетке относятся более легкое распараллеливание (поскольку воркерам не нужно общаться друг с другом и отказавшие воркеры не оставляют дыр в структуре) и гибкое распределение ресурсов (поскольку можно добавить произвольное количество случайных точек к структуре случайного поиска, и все равно получится структура случайного поиска; для поиска по сетке это не так).

Случайный поиск является полезной базовой схемой, поскольку он не делает никаких предположений относительно оптимизируемого алгоритма машинного обучения и – при достаточном количестве ресурсов – стремится достичь производительности, произвольно близкой к оптимальной. Поэтому чередование случайного поиска с более сложными стратегиями оптимизации позволяет гарантировать минимальную скорость сходимости,

[1] В некоторых дисциплинах он также известен как *чисто случайный поиск* (pure random search) [158].

а также добавляет исследование, которое может улучшить поиск на основе модели [3, 59]. Случайный поиск также является полезным методом для инициализации процесса поиска, поскольку он исследует все пространство конфигурации и, таким образом, часто находит параметры с приемлемой производительностью. Однако он не является универсальным решением и часто требует гораздо больше времени, чем методы направленного поиска, чтобы определить одну из наиболее эффективных конфигураций гиперпараметров: например, при выборке без замены из пространства конфигураций с N булевыми гиперпараметрами с хорошими и плохими настройками и без эффектов взаимодействия для нахождения оптимума потребуется 2^{N-1} оценок функций, в то время как направленный поиск может найти оптимум за $N + 1$ оценок следующим образом: начиная с произвольной конфигурации, перебираем гиперпараметры и изменяем их по одному за раз, сохраняя полученную конфигурацию, если производительность улучшается, и отменяя изменения, если это не так. Соответственно, методы направленного поиска, которые мы рассматриваем в следующих разделах, обычно превосходят случайный поиск [12, 14, 33, 90, 153].

Популяционные методы, такие как *генетические алгоритмы*, *эволюционные алгоритмы*, *эволюционные стратегии* и *оптимизация роя частиц*, – это алгоритмы оптимизации, которые используют *популяцию*, т. е. набор конфигураций, и улучшают эту популяцию, применяя локальные возмущения (так называемые *мутации*) и комбинации различных членов (так называемый *кроссовер*) для получения нового поколения лучших конфигураций. Эти методы просты, могут работать с различными типами данных и, что удивительно, легко распараллеливаются [91], поскольку популяция из N членов может быть оценена параллельно на N машинах.

Одним из наиболее известных популяционных методов является эволюционная версия ковариационной матрицы (CMA-ES [51]). Эта простая эволюционная стратегия выбирает конфигурации из многомерного гауссова распределения, среднее и ковариация которого обновляются в каждом поколении на основе успеха особей, составляющих популяцию. CMA-ES является одним из наиболее конкурентоспособных алгоритмов оптимизации методом черного ящика и регулярно побеждает в соревновании Black-Box Optimization Benchmarking (BBOB) [11].

За более подробной информацией о популяционных методах мы отсылаем читателя к [28, 138]. Применение популяционных методов для оптимизации гиперпараметров будет рассмотрено в разделе 1.5, применение для поиска нейронных архитектур – в главе 3, и генетическое программирование для конвейеров AutoML – в главе 8.

1.3.2. Байесовская оптимизация

Байесовская оптимизация – это современный метод глобальной оптимизации вычислительно дорогостоящих функций черного ящика, который получил широкое распространение в области HPO благодаря достижению новых пере-

довых результатов в тонкой настройке глубоких нейронных сетей для классификации изображений [140, 141], распознавания речи [22] и нейронного моделирования языка [105], а также благодаря демонстрации широкой применимости к различным задачам. Для углубленного изучения байесовской оптимизации мы рекомендуем воспользоваться превосходными учебниками за авторством Шахриари и др. [135] и Брошу и др. [18].

В этом разделе мы сначала дадим краткое введение в байесовскую оптимизацию, представим альтернативные суррогатные модели, используемые в ней, опишем расширения на условные и ограниченные пространства конфигураций, а затем обсудим несколько важных приложений к гиперпараметрической оптимизации.

Многие современные наработки в области байесовской оптимизации больше не рассматривают HPO как черный ящик, например *многофакторная HPO* (см. раздел 1.4), *байесовская оптимизация с метаобучением* (глава 2) и *байесовская оптимизация, учитывающая структуру конвейера* [159, 160]. Кроме того, многие последние разработки в области байесовской оптимизации не направлены непосредственно на HPO, но часто могут быть легко применены к HPO. К ним относятся, например, новые *функции сбора* (acquisition function), новые модели и ядра, а также новые схемы распараллеливания.

1.3.2.1. Краткое введение в байесовскую оптимизацию

Байесовская оптимизация – это итерационный алгоритм с двумя ключевыми компонентами: *вероятностной суррогатной моделью* и *функцией сбора*, которая решает, какую точку оценивать следующей. На каждой итерации суррогатная модель подгоняется ко всем наблюдениям целевой функции, сделанным до сих пор. Затем функция сбора, которая использует прогнозируемое распределение вероятностной модели, определяет полезность различных точек-кандидатов, выбирая между исследованием и использованием. По сравнению с дорогостоящей оценкой функции черного ящика функция сбора вычислительно дешевле и поэтому может быть тщательно оптимизирована.

Хотя существует множество вариантов функции сбора, наиболее часто используют *ожидаемое улучшение* (expected improvement, EI) [72]:

$$\mathbb{E}[\mathbb{I}(\boldsymbol{\lambda})] = \mathbb{E}[\max(f_{\min} - y, 0)], \tag{1.2}$$

поскольку его можно найти аналитически, если прогноз модели y в конфигурации $\boldsymbol{\lambda}$ следует нормальному распределению:

$$\mathbb{E}[\mathbb{I}(\boldsymbol{\lambda})] = (f_{\min} - \mu(\boldsymbol{\lambda}))\Phi\left(\frac{f_{\min} - \mu(\boldsymbol{\lambda})}{\sigma}\right) + \sigma\varphi\left(\frac{f_{\min} - \mu(\boldsymbol{\lambda})}{\sigma}\right), \tag{1.3}$$

где $\varphi(\cdot)$ и $\Phi(\cdot)$ – стандартная нормальная плотность и стандартная нормальная функция распределения соответственно, а $f_{\min}$ – наилучшее наблюдаемое значение на данный момент.

На рис. 1.2 показана байесовская оптимизация некой условной функции, взятой в качестве примера.

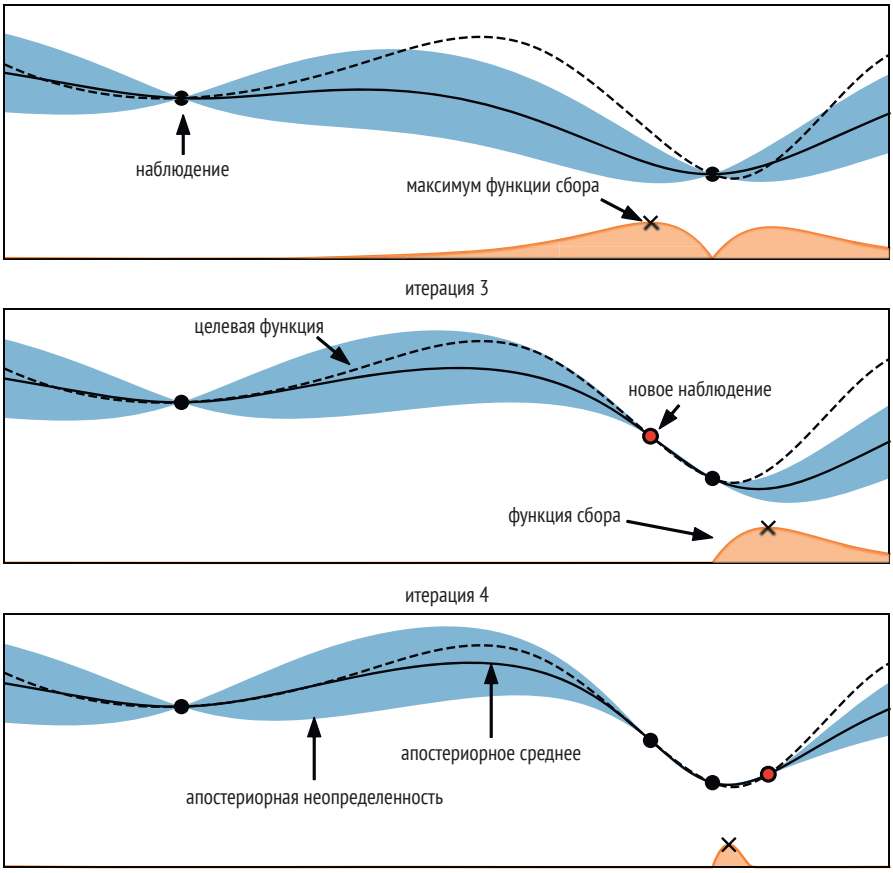


Рис. 1.2 ❖ Иллюстрация байесовской оптимизации одномерной функции. Мы стремимся минимизировать пунктирную линию, используя суррогат гауссова процесса (прогнозы показаны черной линией, а синяя область представляет неопределенность), максимизируя функцию сбора, представленную оранжевой кривой (верхняя часть). Значение функции сбора низкое вблизи наблюдений, и ее самое высокое значение находится в точке, где значение предсказанной функции низкое, а неопределенность предсказания относительно высокая (средняя часть). Хотя слева от нового наблюдения все еще существует большая дисперсия, предсказанное среднее значение справа намного ниже, и следующее наблюдение проводится там (нижняя часть). И хотя вокруг местоположения истинного максимума почти не осталось неопределенности, следующая оценка проводится там из-за ожидаемого улучшения по сравнению с лучшей точкой на данный момент

1.3.2.2. Суррогатные модели

Традиционно в байесовской оптимизации для моделирования целевой функции используются гауссовы процессы [124] из-за их выразительности, гладких и хорошо откалиброванных оценок неопределенности и вычислимости прогнозируемого распределения в аналитической форме. Гауссов процесс $\mathcal{G}(m(\boldsymbol{\lambda}), k(\boldsymbol{\lambda}, \boldsymbol{\lambda}'))$ полностью определяется средним значением $m(\boldsymbol{\lambda})$ и ковариационной функцией $k(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, хотя в байесовской оптимизации

функция среднего обычно принимается постоянной. Предсказания среднего и дисперсии $\mu(\cdot)$ и $\sigma^2(\cdot)$ для случая без шума могут быть получены следующим образом:

$$\mu(\boldsymbol{\lambda}) = \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{y}, \; \sigma^2(\boldsymbol{\lambda}) = k(\boldsymbol{\lambda}, \boldsymbol{\lambda}) - \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{k}_*, \tag{1.4}$$

где $\mathbf{k}_*$ обозначает вектор ковариаций между $\boldsymbol{\lambda}$ и всеми предыдущими наблюдениями, $\mathbf{K}$ – ковариационная матрица всех ранее оцененных конфигураций, а $\mathbf{y}$ – наблюдаемые значения функции. Качество гауссова процесса зависит исключительно от ковариационной функции. Обычно выбирается ядро Матерна 5/2, гиперпараметры которого интегрируются с помощью марковской цепи Монте-Карло [140].

Одним из недостатков стандартных гауссовых процессов является кубическая зависимость от количества точек данных, что ограничивает их применимость, когда кто-то может позволить себе выполнять много оценок функций (например, при большом количестве параллельных воркеров или когда оценки обходятся дешево из-за использования более низкой точности). Кубического роста можно избежать с помощью масштабируемых аппроксимаций гауссовых процессов, таких как разреженные гауссовы процессы. Они аппроксимируют полный гауссов процесс, используя только подмножество исходного набора данных в качестве *индуцирующих точек* для построения матрицы ядра $\mathbf{K}$. Хотя этот подход позволил масштабировать байесовскую оптимизацию с применением графических процессоров до десятков тысяч точек данных для оптимизации параметров рандомизированного SAT-решателя [62], исследователи высказывают критические замечания в отношении калибровки их оценок неопределенности, а применимость к стандартным HPO не проверялась [104, 154].

Еще одним недостатком гауссовых процессов со стандартными ядрами является их плохая масштабируемость на большие размерности. В результате было предложено множество расширений для обслуживания пространств конфигураций с большим числом гиперпараметров, таких как применение случайных встраиваний [153], использующих гауссовы процессы на разделах пространства конфигурации [154], цилиндрических ядер [114] и аддитивных ядер [40, 75].

Поскольку есть модели машинного обучения, более масштабируемые и гибкие, чем гауссовы процессы, существует также большое количество исследований по адаптации этих моделей к байесовской оптимизации. Например, глубокие нейронные сети – это очень гибкие и масштабируемые модели. Самый простой способ их применения к байесовской оптимизации – использование в качестве экстрактора признаков для предварительной обработки входных данных, а затем использование выходов последнего скрытого слоя в качестве базисных функций для байесовской линейной регрессии [141]. Показано, что более сложная, полностью байесовская обработка весов сети возможна при использовании байесовской нейронной сети, обученной с помощью стохастического градиента по гамильтоновской версии метода Монте-Карло [144]. Нейронные сети, как правило, становятся быстрее гауссовых процессов для байесовской оптимизации после 250 оценок функций, что также позволяет использовать крупномасштабный параллелизм. Благодаря

гибкости глубокого обучения обычно удается проводить байесовскую оптимизацию на более сложных задачах. Например, вариационный автокодировщик можно применить для встраивания сложных входных данных (таких как структурированные конфигурации Automatic Statistician, глава 9) в вещественный вектор так, чтобы с ним мог справиться обычный гауссов процесс [92]. В случае многоисточниковой байесовской оптимизации нейросетевая архитектура, построенная на основе *факторизационных машин* (factorization machines) [125], может содержать информацию о предыдущих задачах [131]. Она была расширена для решения задачи CASH [132].

Другой альтернативной моделью для байесовской оптимизации являются случайные леса [59]. Хотя GP лучше, чем случайные леса, работают на небольших числовых пространствах конфигураций [29], случайные леса изначально обрабатывают большие категориальные и условные пространства конфигураций, где стандартные GP работают плохо [29, 70, 90]. Кроме того, вычислительная сложность случайных лесов гораздо лучше масштабируется для большого количества точек данных: в то время как вычислительная сложность подгонки и прогнозирования вариаций с помощью GP для n точек данных составляет $O(n^3)$ и $O(n^2)$ соответственно, для случайных лесов масштабирование по n составляет только $O(n \log n)$ и $O(\log n)$ соответственно. Благодаря этим преимуществам удалось разработать механизм SMAC для байесовской оптимизации со случайными лесами [59], позволяющий использовать известные механизмы AutoML Auto-WEKA [149] и Auto-sklearn [34] (они будут описаны в главах 4 и 6).

Вместо моделирования вероятности $p(\mathbf{y}|\boldsymbol{\lambda})$ наблюдений $\mathbf{y}$ с учетом конфигураций $\boldsymbol{\lambda}$ *древовидный оцениватель Парзена* (Tree Parzen Estimator, TPE [12, 14]) моделирует функции плотности $p(\boldsymbol{\lambda}|\mathbf{y} < \alpha)$ и $p(\boldsymbol{\lambda}|\mathbf{y} \geqslant \alpha)$. По отношению к перцентилю α (обычно задается 15 %) наблюдения делятся на хорошие и плохие, а для моделирования двух распределений используются простые одномерные окна Парзена. Отношение $\frac{p(\boldsymbol{\lambda}|\mathbf{y}<\alpha)}{p(\boldsymbol{\lambda}|\mathbf{y}\geq\alpha)}$ связано с функцией сбора и применяется для предложения новых конфигураций гиперпараметров. Метод TPE использует дерево оценивателя Парзена для подбора условных гиперпараметров. Он продемонстрировал хорошую производительность на структурированных задачах HPO [12, 14, 29, 33, 143, 149, 160], является концептуально простым и естественным образом распараллеливается [91]. Это рабочая лошадка, которая приводит в движение популярный фреймворк Hyperopt-sklearn [83], описанный в главе 5.

Наконец, отметим, что существуют также подходы на основе суррогатных моделей, которые не следуют парадигме байесовской оптимизации: Hord [67] использует детерминированный суррогат RBF, а Harmonica [52] использует метод *сжатого зондирования* (compressed sensing). Оба подхода предназначены для настройки гиперпараметров глубоких нейронных сетей.

1.3.2.3. Описание пространства конфигурации

Байесовская оптимизация изначально была разработана для оптимизации функций вещественных чисел с коробчатыми ограничениями (box-constrained). Однако для многих гиперпараметров машинного обучения, та-

ких как скорость обучения в нейронных сетях или регуляризация в методе опорных векторов, обычно оптимизируют показатель степени экспоненциального члена, чтобы показать, что изменение, например, с 0.001 до 0.01 будет иметь такое же большое влияние, как и изменение с 0.1 до 1. Прием, известный как *деформация входных данных* (input warping) [142], позволяет автоматически изучать такие преобразования в процессе оптимизации, заменяя каждое входное измерение двумя параметрами бета-распределения и оптимизируя их.

Один из очевидных недостатков коробчатых ограничений заключается в том, что пользователь должен определить их заранее. Чтобы избежать этого, можно динамически расширять пространство конфигураций [113, 136]. В качестве альтернативы модифицированный алгоритм TPE с оценкой по распределению [12] способен работать с бесконечными пространствами, на которые накладывается априорное распределение (обычно гауссово).

Целочисленные и категориальные гиперпараметры требуют особого подхода, но могут быть довольно легко интегрированы в обычную байесовскую оптимизацию путем небольшой адаптации ядра и процедуры оптимизации (см. раздел 12.1.2 в [58], а также [42]). Другие модели, такие как машины факторизации и случайные леса, также могут естественным образом работать с этими типами данных.

Условные гиперпараметры все еще являются активной областью исследований (в главах 5 и 6 пойдет речь о пространствах условных конфигураций в последних системах AutoML). Их можно обрабатывать с помощью древовидных методов, таких как случайные леса [59] и древовидные оценки Парзена (TPE) [12], но из-за многочисленных преимуществ гауссовых процессов перед другими моделями были также предложены разнообразные ядра для структурированных конфигурационных пространств [4, 12, 63, 70, 92, 96, 146].

1.3.2.4. Ограниченная байесовская оптимизация

В реальной жизни обычно необходимо удовлетворять ограничениям, таким как потребление памяти [139, 149], время обучения [149], время предсказания [41, 43], точность сжатой модели [41], потребление энергии [43], или просто не допустить сбоя во время процедуры обучения [43].

Ограничения могут быть *скрытыми*, когда доступно только бинарное наблюдение (успех или неудача) [88]. Типичными примерами в AutoML являются ограничения на память и время, позволяющие обучать алгоритмы в общей вычислительной системе с уверенностью, что единственная конфигурация медленного алгоритма не займет все процессорное время, доступное для HPO [34, 149] (см. также главы 4 и 6).

Ограничения также могут быть *неизвестными*, т. е. мы можем наблюдать и моделировать вспомогательную функцию ограничений, но узнаем о нарушении ограничений только после оценки целевой функции [46]. Примером может служить время предсказания по методу опорных векторов, которое становится известно только при обучении, поскольку оно зависит от количества опорных векторов, выбранных в процессе обучения.

Самый простой подход к моделированию нарушения ограничений заключается в определении штрафного значения (по крайней мере, такого же плохого, как наихудшее возможное наблюдаемое значение потерь) и использовании его в качестве наблюдения за неудачными прогонами [34, 45, 59, 149]. Более продвинутые подходы моделируют вероятность нарушения одного или нескольких ограничений и активно ищут конфигурации с низкими значениями потерь, которые вряд ли нарушат какое-либо из заданных ограничений [41, 43, 46, 88].

Байесовские механизмы оптимизации, использующие информационно-теоретические функции сбора, позволяют разделить оценку целевой функции и функции ограничений, чтобы динамически выбирать, какую из них оценивать следующей [43, 55]. Это становится выгодным, когда такие оценки требуют совершенно разного количества времени, например при оценке производительности глубокой нейронной сети и потребления памяти [43].

1.4. Методы оптимизации с переменной точностью

Увеличение размеров наборов данных и постоянное усложнение моделей являются основным препятствием для успешной оптимизации гиперпараметров, поскольку они делают оценку производительности черного ящика более дорогостоящей. Обучение одной конфигурации гиперпараметров на больших наборах данных в настоящее время превышает несколько часов и может занимать до нескольких дней [85].

Вследствие этого распространенным способом ускорения ручной настройки стало тестирование алгоритма/конфигурации гиперпараметров на небольшом подмножестве данных путем обучения в течение лишь нескольких итераций, запуска на подмножестве признаков, использования только одного или нескольких проходов перекрестной проверки или использования уменьшенных изображений в компьютерном зрении. Методы *переменной точности* (multi-fidelity) превращают упомянутые ручные эвристики в формальные алгоритмы, используя так называемые *неточные аппроксимации* (low-fidelity approximation) фактической функции потерь, которую нужно минимизировать. Эти приближения представляют собой компромисс между качеством оптимизации и временем выполнения, но на практике выгода от полученного ускорения часто перевешивает ошибку аппроксимации.

Сначала мы рассмотрим методы, которые моделируют кривую обучения алгоритма непосредственно во время обучения и могут остановить процедуру обучения, если прогнозируется, что добавление ресурсов не поможет. Далее мы обсудим простые методы выбора, которые выбирают только один вариант из конечного набора заданных алгоритмов/конфигураций. Наконец, мы обсудим методы переменной точности, которые могут активно решать, какая точность даст больше информации о нахождении оптимальных гиперпараметров. В этом разделе мы также ссылаемся на главу 2 (в которой

обсуждается, как методы переменной точности могут быть использованы для разных наборов данных) и главу 3 (в которой описано применение неточной аппроксимации для поиска нейронной архитектуры).

1.4.1. Прогнозирование на основе кривой обучения для ранней остановки

Мы начинаем этот раздел, посвященный методам переменной точности в HPO, с методов, которые оценивают и моделируют кривые обучения в ходе HPO [82, 123], а затем принимают решение о том, следует ли добавить дополнительные ресурсы или остановить процедуру обучения для данной конфигурации гиперпараметров. Примерами кривых обучения являются производительность одной и той же конфигурации, обученной на возрастающих подмножествах набора данных, или производительность итерационного алгоритма, измеряемая для каждой итерации (или каждой *i*-й итерации, если вычисление производительности является дорогостоящим).

В сценарии *предиктивного завершения* (predictive termination) [26] модель кривой обучения используется для экстраполяции частично наблюдаемой кривой обучения в определенной конфигурации, и процесс обучения останавливается, если прогнозируется, что данная конфигурация не сможет достичь производительности лучшей модели, обученной на данный момент в процессе оптимизации. Каждая кривая обучения моделируется как взвешенная комбинация 11 параметрических функций из различных научных областей. Параметры этих функций и их веса выбираются с помощью марковской цепи Монте-Карло, чтобы минимизировать потери при подгонке частично наблюдаемой кривой обучения. В результате получается прогностическое распределение, которое позволяет прекратить обучение на основе вероятности того, что не удастся победить лучшую известную модель. В сочетании с байесовской оптимизацией прогностический критерий прекращения обучения позволил добиться более низкого уровня ошибок, чем при простой байесовской оптимизации черного ящика. В среднем метод ускорил оптимизацию в два раза и смог найти самую современную (на тот момент) нейронную сеть для CIFAR-10 (без дополнения данных) [26].

Ограничением вышеупомянутого метода является отсутствие обмена информацией между различными конфигурациями гиперпараметров. Этот недостаток можно устранить, используя базисные функции в качестве выходного слоя байесовской нейронной сети [80]. В этом случае параметры и веса базисных функций, а значит и полная кривая обучения, могут быть предсказаны для произвольных конфигураций гиперпараметров. В качестве альтернативы можно использовать предыдущие кривые обучения в качестве экстраполяторов базисных функций [21]. Хотя экспериментальные результаты не дают однозначного ответа на вопрос, превосходит ли предложенный метод предварительно заданные параметрические функции, отсутствие необходимости определять их вручную является явным преимуществом.

Байесовская оптимизация методом *замораживания-размораживания* (freeze-thaw method) [148] – это полная интеграция кривых обучения в процесс моделирования и выбора в байесовской оптимизации. Вместо того чтобы завершать конфигурацию, модели машинного обучения обучаются итеративно в течение нескольких проходов, а затем *замораживаются*. Затем байесовская оптимизация может решить *разморозить* одну из замороженных моделей, т. е. продолжить ее обучение. Кроме того, метод может принять решение о запуске новой конфигурации. Данный метод моделирует производительность сходящегося алгоритма обычным гауссовым процессом и вводит специальную ковариационную функцию, соответствующую экспоненциально затухающим функциям, чтобы моделировать кривые обучения гауссовыми процессами в каждом отдельном случае.

1.4.2. Методы выбора алгоритма на основе приближений

В этом разделе мы описываем методы, которые пытаются определить лучший алгоритм из заданного конечного набора алгоритмов на основе приближений их производительности с низкой точностью; в конце мы также обсуждаем возможные комбинации со стратегиями адаптивной конфигурации. Мы сосредоточимся на стратегиях последовательного деления пополам и HyperBand, поскольку они показали высокую эффективность, особенно для оптимизации алгоритмов глубокого обучения. Строго говоря, некоторые из методов, которые мы обсудим в этом подразделе, также моделируют кривые обучения, но они не предоставляют средств для выбора новых конфигураций на основе этих моделей.

Однако сначала мы кратко опишем историческую эволюцию методов выбора алгоритмов с переменной точностью. В 2000 г. Петрак [120] отметил, что простое тестирование различных алгоритмов на небольшом подмножестве данных является мощным и дешевым механизмом выбора алгоритма. Более поздние подходы использовали итерационные схемы исключения алгоритмов для отсева конфигураций гиперпараметров, если они плохо работают на подмножествах данных [17], если они работают значительно хуже, чем группа наиболее эффективных конфигураций [86], если они работают хуже, чем лучшая конфигурация на заданном пользователем факторе [143], или если оптимистическая граница производительности алгоритма хуже, чем лучший известный алгоритм [128]. Аналогичным образом можно отказаться от конфигураций гиперпараметров, если они плохо работают на одной или нескольких подвыборках перекрестной проверки [149]. Наконец, Джеймисон и Талвалкар [69] предложили использовать для HPO алгоритм последовательного деления пополам, первоначально представленный Карнином и др. [76].

Последовательное деление пополам (successive halving) – это чрезвычайно простая, но мощная и поэтому популярная стратегия выбора алгоритмов с переменной точностью: для заданного начального бюджета опросить все

алгоритмы на этот бюджет; затем удалить половину, показавшую наихудшие результаты, удвоить бюджет[1] и последовательно повторять эти шаги, пока не останется только один алгоритм. Этот процесс показан на рис. 1.3. Джеймисон и Талвалкар [69] провели сравнительный анализ нескольких распространенных методов в контексте задачи об одноруком бандите и обнаружили, что последовательное деление пополам хорошо работает как по количеству необходимых итераций, так и по времени вычислений, что алгоритм теоретически превосходит стратегию равномерного распределения бюджета, если алгоритмы благоприятно сходятся, и что он предпочтительнее многих известных стратегий в задаче однорукого бандита, таких как UCB и EXP3.

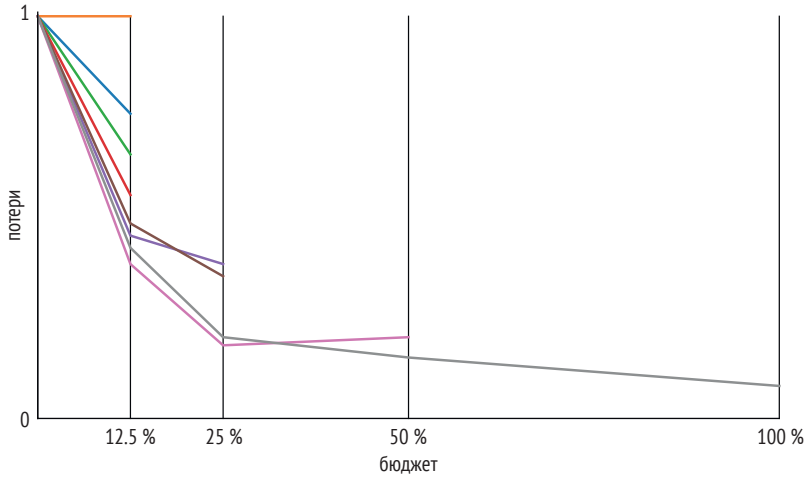


Рис. 1.3 ❖ Иллюстрация последовательного деления пополам для восьми алгоритмов/конфигураций. После оценки всех алгоритмов на 1 от общего бюджета половина из них отбрасывается, а бюджет, выделенный оставшимся алгоритмам, удваивается

Хотя метод последовательного деления пополам весьма эффективен, его недостатком является необходимость наличия компромисса между бюджетом и количеством конфигураций. Располагая определенным бюджетом, пользователь должен заранее решить, попробовать ли множество конфигураций и выделить каждой из них небольшой бюджет, или попробовать лишь несколько, но выделить им больший бюджет. Назначение слишком маленького бюджета может привести к преждевременному отказу от хороших конфигураций, в то время как назначение слишком большого бюджета может привести к тому, что плохие конфигурации будут проверяться слишком долго и, таким образом, вычислительные ресурсы будут расходоваться впустую.

[1] Точнее, отбрасывают худшую долю $(\eta - 1)/\eta$ алгоритмов и умножают бюджет для оставшихся алгоритмов на η, где η – гиперпараметр. Его значение по умолчанию было изменено с 2 на 3 с появлением HyperBand [90].

HyperBand [90] – это стратегия хеджирования, разработанная для решения проблемы компромисса путем выбора из случайно сформированного набора конфигураций. Она распределяет общий бюджет на несколько сочетаний количества конфигураций и индивидуальных бюджетов, а затем выполняет последовательное деление пополам в качестве подпрограммы на каждом наборе случайных конфигураций. Благодаря стратегии хеджирования, которая предусматривает проверку некоторых конфигураций только при максимальном бюджете, в худшем случае HyperBand занимает в несколько раз больше времени, чем базовый случайный поиск при максимальном бюджете. На практике благодаря использованию дешевых оценок низкой точности HyperBand стабильно оказывается лучше базового случайного поиска и байесовской оптимизации черного ящика для подмножеств данных, подмножеств признаков и итерационных алгоритмов, таких как стохастический градиентный спуск для глубоких нейронных сетей.

Несмотря на успех HyperBand при оптимизации глубоких нейронных сетей, его применение очень ограничено, поскольку он не применяет стратегию подбора конфигурации исходя из оценок функции. Чтобы преодолеть это ограничение, относительно новый подход BOHB [33] объединяет байесовскую оптимизацию и HyperBand, извлекая лучшее из двух миров: высокую производительность в начальный период (быстрые улучшения в начале за счет использования низкой точности в HyperBand) и высокую конечную точность (хорошая точность в долгосрочной перспективе за счет замены случайного поиска в HyperBand на байесовскую оптимизацию). BOHB также эффективно использует параллельные ресурсы и работает с задачами, содержащими от нескольких до многих десятков гиперпараметров. Компонент байесовской оптимизации BOHB похож на TPE [12], но отличается от него использованием многомерных ядерных оценок плотности. Он подгоняет модель только с наивысшей точностью, для которой было выполнено не менее $|\mathbf{\Lambda}| + 1$ оценок (число гиперпараметров плюс один). Поэтому первая модель BOHB подгоняется на самой низкой точности, и со временем модели, обученные на более высоких точностях, сменяют друг друга, но все еще используют более низкие точности при последовательном делении пополам. Эмпирически показано, что BOHB превосходит несколько современных методов HPO для настройки моделей опорных векторов, нейронных сетей и алгоритмов обучения с подкреплением, включая большинство методов, представленных в этом разделе [33]. Были предложены усовершенствованные подходы к объединению HyperBand и байесовской оптимизации [15, 151].

Оценки с переменной точностью можно сочетать с HPO и другими способами. Вместо того чтобы переключаться между низкой и самой высокой точностью, можно выполнить HPO на подмножестве исходных данных и извлечь конфигурации с наилучшими показателями, чтобы использовать их в качестве отправной точки для HPO на полном наборе данных [152]. Для ускорения решения задачи CASH можно также итеративно удалять из пространства конфигураций целые алгоритмы (и их гиперпараметры), показавшие низкую производительность на небольших подмножествах набора данных [159].

1.4.3. Адаптивный выбор точности

Все методы в предыдущем разделе строго соблюдают определенный порядок выбора точности. В качестве альтернативы можно было бы активно выбирать точность оценивания с учетом предыдущих наблюдений, чтобы предотвратить неправильное определение порядка.

Многозадачная байесовская оптимизация [147] использует многозадачный гауссов процесс для моделирования производительности связанных задач и автоматического изучения корреляции задач в процессе оптимизации. Этот метод может динамически переключаться между более дешевыми задачами с низкой точностью и дорогой целевой задачей с высокой точностью, используя информационно-теоретическую функцию сбора. На практике предложенный метод начинает исследование конфигурационного пространства на более дешевой задаче и переключается на более дорогое пространство конфигураций только на поздних этапах оптимизации, что примерно вдвое сокращает время, необходимое для HPO. Многозадачная байесовская оптимизация также может быть использована для передачи информации из предыдущих задач оптимизации. Более подробно об этом будет сказано в главе 2.

Многозадачная байесовская оптимизация (и методы, представленные в предыдущем подразделе) требует предварительного задания набора градаций точности. Здесь кроется источник потенциальной неоптимальности, поскольку они могут быть неправильно определены [74, 78] и поскольку число градаций, с которыми можно работать, невелико (обычно пять или меньше). Часто можно получить лучшие результаты, если рассматривать точность как непрерывную (и, например, выбрать непрерывный процент от полного набора данных для оценки конфигурации), соблюдая плавный компромисс между получением информации и временем, необходимым для оценки [78]. Чтобы использовать тот факт, что качество модели обычно улучшается с увеличением количества данных, но с уменьшением подвыборок, можно построить специальное ядро для подмножеств данных [78]. Это обобщение многозадачной байесовской оптимизации улучшает производительность и может достичь ускорения в 10–100 раз по сравнению с байесовской оптимизацией черного ящика.

Вместо использования информационно-теоретической функции сбора байесовская оптимизация с функцией сбора по методу *верхней доверительной границы* (upper confidence bound, UCB) также может быть расширена на метод переменной точности [73, 74]. Если первый такой подход, MF-GP-UCB [73], требовал предварительного определения точности, то более поздний алгоритм BOCA [74] обходится без этого требования. BOCA также применялся для оптимизации с более чем одной непрерывной точностью, и мы ожидаем, что алгоритм HPO для более чем одной непрерывной точности будет представлять дальнейший интерес в будущем.

Вообще говоря, методы, которые могут адаптивно выбирать точность, очень привлекательны и более мощны, чем концептуально более простые методы, рассмотренные в разделе 1.4.2, но, если речь идет о практическом

применении, мы предупреждаем, что для успешного выбора точности необходимы *сильные* модели. Когда модели не сильны (по причине нехватки данных или несоответствия данным/задаче), эти методы могут тратить слишком много времени на оценку более высоких точностей, а более надежные схемы с фиксированным бюджетом, обсуждаемые в разделе 1.4.2, могут дать лучшую производительность при фиксированном лимите времени.

1.5. Применение оптимизации гиперпараметров в AutoML

В этом разделе представлен исторический обзор наиболее важных систем оптимизации гиперпараметров и их применения в автоматизированном машинном обучении.

Поиск по сетке используется для оптимизации гиперпараметров с 1990-х гг. [71, 107] и уже в 2002 г. поддерживался инструментами машинного обучения [35]. Первыми адаптивными методами оптимизации, примененными к HPO, были жадный поиск в глубину [82] и поиск по образцу [109]; оба улучшили конфигурации гиперпараметров по умолчанию, а поиск по образцу улучшил и поиск по сетке. Генетические алгоритмы были впервые применены для настройки двух гиперпараметров C и γ модели RBF-SVM в 2004 г. [119] и привели к улучшению классификации за меньшее время, чем поиск по сетке. В том же году эволюционный алгоритм был использован для обучения композиции из трех различных ядер для SVM, гиперпараметров ядра и совместного выбора подмножества признаков; обученная комбинация ядер смогла превзойти каждое отдельное оптимизированное ядро. В 2004 г. аналогичным образом генетический алгоритм был использован для выбора используемых признаков и гиперпараметров как SVM, так и нейронной сети [129].

Впервые алгоритм CMA-ES был использован для оптимизации гиперпараметров в 2005 г. [38]. Тогда он был задействован в оптимизации гиперпараметров SVM C и γ, масштаба ядра l_i для каждой размерности входных данных, а также полной матрицы вращения и масштабирования. Недавно было продемонстрировано, что CMA-ES является отличным выбором для параллельного HPO, превосходя инструменты байесовской оптимизации при подборе 19 гиперпараметров глубокой нейронной сети на 30 GPU параллельно [91].

В 2009 г. Эскаланте и др. [30] расширили задачу HPO до задачи *полного выбора модели* (full model selection), которая включает выбор алгоритма предварительной обработки, алгоритма выбора признаков, классификатора и всех их гиперпараметров. Получив возможность построить конвейер машинного обучения из нескольких готовых алгоритмов машинного обучения с помощью HPO, авторы эмпирически обнаружили, что могут применять свой метод к любому набору данных, поскольку для этого не требуется знание предметной области, и продемонстрировали применимость своего подхода к различным областям [32, 49]. Предложенный ими метод – *выбор модели роя частиц* (particle swarm model selection, PSMS) – использует модифици-

рованный оптимизатор роя частиц для обработки пространства условных конфигураций. Чтобы избежать переобучения, метод PSMS был расширен с помощью специальной стратегии ансамблирования, которая объединяет лучшие решения из нескольких поколений [31]. Поскольку оптимизация методом роя частиц изначально была создана для работы с непрерывными пространствами конфигураций, позже PSMS расширили с целью использования генетического алгоритма для оптимизации структуры конвейера и применения оптимизации по методу роя частиц только для оптимизации гиперпараметров каждого конвейера [145].

Насколько нам известно, первое применение байесовской оптимизации для HPO относится к 2005 г., когда Фролих и Зелл [39] использовали онлайновый гауссов процесс вместе с EI для оптимизации гиперпараметров SVM, добившись ускорения в 10 раз (классификация, 2 гиперпараметра) и в 100 раз (регрессия, 3 гиперпараметра) по сравнению с поиском по сетке. В методе Tuned Data Mining [84] было предложено настраивать гиперпараметры полного конвейера машинного обучения с помощью байесовской оптимизации; в частности, использовался один фиксированный конвейер и настраивались гиперпараметры классификатора, а также порог классификации для каждого класса и веса классов.

В 2011 г. Бергстра и др. [12] первыми применили байесовскую оптимизацию для настройки гиперпараметров глубокой нейронной сети, превзойдя ручной и случайный поиск. Более того, они продемонстрировали, что TPE дает лучшие результаты, чем подход на основе гауссова процесса. TPE, как и байесовская оптимизация со случайными лесами, также хорошо проявили себя при совместном поиске нейронной архитектуры и оптимизации гиперпараметров [14, 106].

Еще один важный шаг в применении байесовской оптимизации к HPO был сделан Сноеком и др. в 2012 г. в работе [140], в которой описаны несколько приемов применения оптимизации в HPO на основе гауссовых процессов, реализованных в системе Spearmint, и получен новый результат оптимизации гиперпараметров глубоких нейронных сетей.

Независимо от парадигмы полного выбора модели, в методе Auto-WEKA [149] (см. также главу 4) была представлен подход *комбинированного выбора алгоритма и оптимизации гиперпараметров* (combined algorithm selection and hyperparameter optimization, CASH), в соответствии с которым выбор алгоритма классификации моделируется как категориальная переменная, гиперпараметры алгоритма моделируются как условные гиперпараметры, а система байесовской оптимизации SMAC [59] на основе случайного леса применяется для совместной оптимизации в полученном 786-мерном пространстве конфигураций.

В последние годы методы оптимизации с переменной точностью стали очень популярны, особенно в глубоком обучении. Сначала, используя низкоточные аппроксимации на основе подмножеств данных, подмножеств признаков и коротких прогонов итерационных алгоритмов, метод HyperBand [90] продемонстрировал превосходство над методами байесовской оптимизации черного ящика, которые не использовали прогоны с низкой точностью. Наконец, в работе, опубликованной в 2018 г., Фолкнер и др. [33] пред-

ставили надежную, гибкую и распараллеливаемую комбинацию байесовской оптимизации и HyperBand, которая существенно превосходит как HyperBand, так и байесовскую оптимизацию черного ящика на широком круге задач, включая настройку моделей SVM, различных типов нейронных сетей и алгоритмов обучения с подкреплением.

На момент написания этой книги мы можем дать следующие рекомендации по выбору инструментов для использования в практических приложениях HPO:

- если применима переменная точность (т. е. если удается определить существенно более дешевые в вычислительном отношении версии интересующей нас целевой функции, так что производительность для них примерно коррелирует с производительностью для полной интересующей нас целевой функции), мы рекомендуем BOHB [33] как надежный, эффективный, универсальный и параллелизуемый метод оптимизации гиперпараметров по умолчанию;
- если переменная точность неприменима:
  - если все гиперпараметры являются вещественными и можно позволить себе лишь несколько десятков оценок функций, мы рекомендуем использовать инструмент байесовской оптимизации на основе гауссова процесса, такой как Spearmint [140];
  - для обширных и условных пространств конфигураций мы предлагаем использовать SMAC [59] или TPE [14] на основе случайного леса, поскольку они доказали свою высокую эффективность в таких задачах [29];
  - для чисто вещественных пространств и относительно дешевых целевых функций, для которых можно позволить себе более сотни оценок, мы рекомендуем CMA-ES [51].

1.6. Проблемы и перспективные направления исследований

Мы завершаем эту главу обсуждением открытых проблем, текущих исследований и перспективных направлений, которые, как мы ожидаем, окажут влияние на HPO в будущем. Заметим, что, несмотря на актуальность, мы не обсуждаем в этой главе важность гиперпараметров и определение пространства конфигураций, поскольку эти вопросы относятся к метаобучению и будут рассмотрены в главе 2.

1.6.1. Бенчмарки и сопоставимость результатов

Учитывая широкий выбор существующих методов HPO, будет естественно задаться вопросом, каковы сильные и слабые стороны каждого из них. Для того чтобы обеспечить справедливое сравнение между различными под-

ходами HPO, сообществу машинного обучения необходимо разработать и согласовать общий набор эталонов, который будет расширяться со временем, по мере появления новых вариантов HPO, таких как оптимизация с переменной точностью. В качестве конкретного примера того, как это может выглядеть, мы хотели бы привести платформу COCO (сокращение от comparing continuous optimizers), которая предоставляет бенчмарки и инструменты анализа для непрерывной оптимизации и используется в качестве рабочей задачи для ежегодного конкурса Black-Box Optimization Benchmarking (BBOB) [11]. Аналогичные усилия в области HPO уже привели к созданию библиотеки гиперпараметрической оптимизации (HPOlib [29]) и коллекции эталонов специально для методов байесовской оптимизации [25]. Однако ни один из них не получил такого распространения, как платформа COCO.

Кроме того, сообщество нуждается в четко определенных метриках, но в настоящее время в разных работах используются разные метрики. Одним из важных аспектов, по которому различаются оценки, является то, какой набор они используют для определения производительности – валидационный, который использован для оптимизации, или отдельный тестовый набор. Первый вариант помогает изучить силу оптимизатора в изоляции, без шума, который добавляется в оценку при переходе от валидационного к тестовому набору; с другой стороны, одни оптимизаторы могут привести к большему переобучению, чем другие, что можно обнаружить только с помощью тестового набора. Другим важным аспектом, по которому различаются оценки, является то, сообщают ли они о производительности после определенного количества оценок функции или по истечении определенного количества времени. Второй критерий учитывает разницу во времени между оценкой различных конфигураций гиперпараметров и включает накладные расходы на оптимизацию, поэтому лучше отражает то, что требуется на практике; однако первый более удобен и способствует воспроизводимости, поскольку дает одинаковые результаты независимо от используемого оборудования. Поэтому для обеспечения воспроизводимости, особенно в исследованиях, использующих критерий времени, следует выпускать реализацию бенчмарка, ориентированную на количество оценок.

При использовании новых бенчмарков важно иметь возможность соотносить их с общепринятыми базовыми уровнями, что является еще одной причиной, по которой методы HPO должны быть опубликованы вместе с сопутствующей реализацией оценки. К сожалению, не существует общей программной библиотеки, в которой были бы реализованы все основные структурные блоки [2, 117]. В качестве простого, но эффективного базового уровня, который может быть тривиально включен в эмпирические исследования, Джеймисон и Рехт [68] предлагают сравнивать исследуемые алгоритмы с различными уровнями распараллеливания случайного поиска, чтобы продемонстрировать ускорение по сравнению с обычным случайным поиском. При сравнении с другими методами оптимизации важно проводить сопоставление с надежной реализацией, так как, например, было показано, что более простые версии байесовской оптимизации имеют более низкую производительность [79, 140, 142].

1.6.2. Оптимизация на основе градиента

В некоторых случаях (например, метод опорных векторов с использованием наименьших квадратов и нейронные сети) можно получить градиент критерия выбора модели относительно некоторых гиперпараметров модели. В отличие от оптимизации гиперпараметров методом черного ящика, в таких случаях каждая оценка целевой функции дает целый вектор гиперградиента вместо одного плавающего значения, что позволяет ускорить процесс HPO.

Маклаурин и др. [99] описали процедуру вычисления точных градиентов производительности при валидации относительно всех непрерывных гиперпараметров нейронной сети путем обратного распространения через всю процедуру обучения стохастического градиентного спуска с импульсом (используя новый алгоритм, экономящий память). Возможность эффективной обработки многих гиперпараметров с помощью градиентных методов позволяет использовать новую парадигму гиперпараметризации модели для получения гибкости при выборе классов моделей, регуляризации и методов обучения. Маклаурин и др. продемонстрировали применимость градиентного HPO для решения многих задач HPO высокой размерности, таких как оптимизация скорости обучения нейронной сети для каждой итерации и слоя в отдельности, оптимизация гиперпараметра инициализации веса для каждого слоя нейронной сети, оптимизация штрафа l_2 для каждого отдельного параметра в логистической регрессии и обучение на совершенно новых наборах данных. Небольшим недостатком является то, что за обратное распространение по всей процедуре обучения приходится платить удвоением временной сложности процедуры обучения. Описанный метод также может быть обобщен для работы с другими алгоритмами обновления параметров [36]. Чтобы преодолеть необходимость обратного распространения через всю процедуру обучения, в более новых алгоритмах допускается выполнять обновление гиперпараметров относительно отдельного валидационного набора в чередовании с процессом обучения [5, 10, 36, 37, 93].

Некоторые примеры градиентной оптимизации гиперпараметров простой модели [118] и нейросетевых структур (глава 3) демонстрируют многообещающие результаты, превосходящие модели байесовской оптимизации. Несмотря на применимость градиентной оптимизации гиперпараметров только к определенному типу моделей, возможность настройки нескольких сотен гиперпараметров позволяет рассчитывать на существенное улучшение HPO.

1.6.3. Масштабируемость

Несмотря на недавние успехи в области оптимизации с переменной точностью, остаются проблемы машинного обучения, которые не были напрямую решены с помощью HPO из-за их масштаба и которые могут потребовать новых подходов. Здесь под масштабом может подразумеваться как размер пространства конфигураций, так и затраты на оценку отдельных моделей. Например, на момент написания этой главы не было опубликовано никаких

работ по HPO для глубоких нейронных сетей на наборе данных ImageNet [127] в основном из-за высокой стоимости обучения даже простой нейронной сети на этом наборе данных. Будет интересно посмотреть, позволят ли методы, выходящие за рамки представления черного ящика из раздела 1.3, такие как методы переменной точности, описанные в разделе 1.4, градиентные методы или методы метаобучения (описанные в главе 2), решить такие проблемы. В главе 3 описаны первые успехи в обучении структурных блоков нейронных сетей на небольших наборах данных и их применении к ImageNet, однако гиперпараметры процедуры обучения по-прежнему задаются вручную.

Учитывая важность параллельных вычислений, мы с нетерпением ждем появления новых методов, в полной мере использующих возможности крупномасштабных вычислительных кластеров. Хотя существует много работ по параллельной байесовской оптимизации [12, 24, 33, 44, 54, 60, 135, 140], за исключением нейронных сетей, описанных в разделе 1.3.2.2 [141], до сих пор ни один метод не продемонстрировал масштабируемость до сотен воркеров. Несмотря на свою перспективность, популяционные подходы еще не показали свою применимость для оптимизации гиперпараметров на наборах, превышающих несколько тысяч точек данных, за единственным исключением HPO, примененного к глубоким нейронным сетям [91].[1]

В целом мы ожидаем, что по мере дальнейшего роста количества гиперпараметров моделей, предназначенных для решения интересных задач, потребуются более сложные и специализированные методы, оставляющие далеко позади подход черного ящика.

1.6.4. Переобучение и обобщение

Переобучение остается актуальной проблемой в области HPO. Как отмечалось в постановке задачи (раздел 1.2), мы обычно имеем только конечное число точек данных, доступных для вычисления оптимизируемых потерь при валидации, и, таким образом, не обязательно оптимизируем модель для обобщения на незнакомые тестовые точки данных. Аналогично переобучению модели машинного обучения на обучающем наборе проблема HPO заключается в переобучении гиперпараметров на ограниченном валидационном наборе; это явление было продемонстрировано экспериментально [20, 81].

Простой стратегией борьбы с переобучением является использование разной перестановки обучающего и валидационного разбиения для каждой оценки функции; было показано, что это улучшает эффективность обобщения при настройке SVM как при стратегии отложенной выборки, так и при стратегии перекрестной проверки [95]. Выбор окончательной конфигурации может быть более надежен, если использовать не наименьшее наблюдаемое значение, а наименьшее среднее прогнозируемое значение модели гауссова процесса, используемой в байесовской оптимизации [95].

---

[1] Смотрите также главу 3, где популяционные методы применяются к задаче поиска нейронной архитектуры.

Разновидностью этого подхода является использование отдельной отложенной выборки для оценки конфигураций, найденных HPO, чтобы избежать смещения в сторону стандартного валидационного набора [108, 159]. Различные аппроксимации показателей обобщения могут привести к различным показателям тестирования [108], и есть сообщения о том, что использование разных стратегий повторной выборки может привести к измеримым различиям в производительности HPO для моделей на основе метода опорных векторов [150].

Другой подход к борьбе с переобучением может заключаться в поиске *стабильных оптимумов* вместо *резких оптимумов* целевой функции [112]. Идея заключается в том, что значение функции вблизи стабильного оптимума не меняется при незначительных возмущениях гиперпараметров, в то время как для острых оптимумов оно меняется ощутимо. Стабильные оптимумы приводят к лучшему обобщению при применении найденных гиперпараметров к новому, незнакомому набору точек данных (т. е. тестовому набору). Было показано, что при использовании принципа стабильного оптимума наблюдается лишь незначительное переобучение функции сбора в процессе оптимизации гиперпараметров модели опорных векторов, в то время как обычная байесовская оптимизация демонстрирует сильное переобучение [112].

Другими подходами к борьбе с переобучением являются методы ансамблей и байесовские методы, представленные в разделе 1.2.1. В силу разнообразия методов не существует общепринятой методики борьбы с переобучением, и пользователю остается выяснить, какая стратегия лучше всего подходит для его конкретной задачи HPO.

1.6.5. Построение конвейера произвольного размера

Все методы HPO, которые мы обсуждали до сих пор, предполагают конечный набор компонентов для конвейеров машинного обучения или конечное максимальное число слоев в нейронных сетях. Для конвейеров машинного обучения (системы AutoML, рассмотренные в части II этой книги) может оказаться полезным использовать более одного алгоритма предварительной обработки признаков и динамически добавлять их, если это необходимо для решения задачи, расширяя пространство поиска гиперпараметром выбора подходящего алгоритма предварительной обработки и его собственных гиперпараметров. Хотя в пространство поиска для стандартных инструментов оптимизации методом черного ящика можно легко добавить несколько таких дополнительных предобработчиков (и их собственных гиперпараметров) в качестве условных гиперпараметров, труднее реализовать поддержку произвольного числа гиперпараметров.

Один из методов удобной работы с конвейерами произвольного размера – это *инструментарий оптимизации конвейеров с древовидной структурой* (treestructured pipeline optimization toolkit, TPOT [115], см. также

главу 8), который использует генетическое программирование и описывает возможные конвейеры с помощью грамматики. TPOT применяет многоцелевую оптимизацию для достижения компромисса между сложностью конвейера и производительностью во избежание генерации излишне сложных конвейеров.

К другой парадигме создания конвейеров относится использование иерархического планирования; алгоритм ML-Plan [101, 108] использует иерархические сети задач и показывает конкурентоспособную производительность по сравнению с Auto-WEKA [149] и Auto-sklearn [34].

В настоящее время системы AutoML для контейнеров переменной длины не превосходят системы с фиксированной длиной конвейера, но на длинных конвейерах можно ожидать более заметное улучшение. Аналогично поиск нейронной архитектуры порождает сложные пространства конфигураций, и в главе 3 будут описаны методы решения этой задачи.

Благодарности: авторы благодарят Луку Франчески, Рагху Раджана, Стефана Фолкнера и Арлинда Кадру за ценные замечания к этой главе.

1.7. Литература

1. Proceedings of the International Conference on Learning Representations (ICLR'18) (2018), онлайн-публикация: https://iclr.cc.
2. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/.
3. Ahmed, M., Shahriari, B., Schmidt, M.: Do we need "harmless" Bayesian optimization and "first-order" Bayesian optimization. In: NeurIPS Workshop on Bayesian Optimization (BayesOpt'16) (2016).
4. Alaa, A., van der Schaar, M.: AutoPrognosis: Automated Clinical Prognostic Modeling via Bayesian Optimization with Structured Kernel Learning. In: Dy and Krause [27], pp. 139–148.
5. Almeida, L.B., Langlois, T., Amaral, J.D., Plakhov, A.: Parameter Adaptation in Stochastic Optimization, p. 111–134. Cambridge University Press (1999).
6. Amazon: Automatic model tuning (2018), https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning.html.
7. Bach, F., Blei, D. (eds.): Proceedings of the 32nd International Conference on Machine Learning (ICML'15), vol. 37. Omnipress (2015).
8. Balcan, M., Weinberger, K. (eds.): Proceedings of the 33rd International Conference on Machine Learning (ICML'17), vol. 48. Proceedings of Machine Learning Research (2016).

9. Bartlett, P., Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.): Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12) (2012).
10. Baydin, A.G., Cornish, R., Rubio, D.M., Schmidt, M., Wood, F.: Online Learning Rate Adaption with Hypergradient Descent. In: Proceedings of the International Conference on Learning Representations (ICLR'18) [1], published online: iclr.cc.
11. BBOBies: Black-box Optimization Benchmarking (BBOB) workshop series (2018), http://numbbo.github.io/workshops/index.html.
12. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K. (eds.) Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NeurIPS'11). pp. 2546–2554 (2011).
13. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research 13, 281–305 (2012).
14. Bergstra, J., Yamins, D., Cox, D.: Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: Dasgupta and McAllester [23], pp. 115–123.
15. Bertrand, H., Ardon, R., Perrot, M., Bloch, I.: Hyperparameter optimization of deep neural networks: Combining hyperband with Bayesian model selection. In: Conférence sur l'Apprentissage Automatique (2017).
16. Bischl, B., Mersmann, O., Trautmann, H., Weihs, C.: Resampling methods for meta-model validation with recommendations for evolutionary computation. Evolutionary Computation 20(2), 249–275 (2012).
17. Van den Bosch, A.: Wrapped progressive sampling search for optimizing learning algorithm parameters. In: Proceedings of the sixteenth Belgian-Dutch Conference on Artificial Intelligence. pp. 219–226 (2004).
18. Brochu, E., Cora, V., de Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv:1012.2599v1 [cs.LG] (2010).
19. Bürger, F., Pauli, J.: A Holistic Classification Optimization Framework with Feature Selec- tion, Preprocessing, Manifold Learning and Classifiers., pp. 52–68. Springer (2015).
20. Cawley, G., Talbot, N.: On Overfitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. Journal of Machine Learning Research 11 (2010).
21. Chandrashekaran, A., Lane, I.: Speeding up Hyperparameter Optimization by Extrapolation of Learning Curves using Previous Builds. In: Ceci, M., Hollmen, J., Todorovski, L., Vens, C., Džeroski, S. (eds.) Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'17). Lecture Notes in Computer Science, vol. 10534. Springer (2017).
22. Dahl, G., Sainath, T., Hinton, G.: Improving deep neural networks for LVCSR using rectified linear units and dropout. In: Adams, M., Zhao, V. (eds.) International Conference on Acoustics, Speech and Signal Processing (ICASSP'13). pp. 8609–8613. IEEE Computer Society Press (2013).
23. Dasgupta, S., McAllester, D. (eds.): Proceedings of the 30th International Conference on Machine Learning (ICML'13). Omnipress (2014).

24. Desautels, T., Krause, A., Burdick, J.: Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. Journal of Machine Learning Research 15, 4053–4103 (2014).
25. Dewancker, I., McCourt, M., Clark, S., Hayes, P., Johnson, A., Ke, G.: A stratified analysis of Bayesian optimization methods. arXiv:1603.09441v1 [cs.LG] (2016).
26. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Yang, Q., Wooldridge, M. (eds.) Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'15). pp. 3460–3468 (2015).
27. Dy, J., Krause, A. (eds.): Proceedings of the 35th International Conference on Machine Learning (ICML'18), vol. 80. Proceedings of Machine Learning Research (2018).
28. Eberhart, R., Shi, Y.: Comparison between genetic algorithms and particle swarm optimization. In: Porto, V., Saravanan, N., Waagen, D., Eiben, A. (eds.) 7th International conference on evolutionary programming. pp. 611–616. Springer (1998).
29. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., Leyton-Brown, K.: Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In: NeurIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'13) (2013).
30. Escalante, H., Montes, M., Sucar, E.: Particle Swarm Model Selection. Journal of Machine Learning Research 10, 405–440 (2009).
31. Escalante, H., Montes, M., Sucar, E.: Ensemble particle swarm model selection. In: Proceedings of the 2010 IEEE International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE Computer Society Press (2010).
32. Escalante, H., Montes, M., Villaseñor, L.: Particle swarm model selection for authorship verification. In: Bayro-Corrochano, E., Eklundh, J.O. (eds.) Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. pp. 563–570 (2009).
33. Falkner, S., Klein, A., Hutter, F.: BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In: Dy and Krause [27], pp. 1437–1446.
34. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J.T., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'15). pp. 2962–2970 (2015).
35. Fischer, S., Klinkenberg, R., Mierswa, I., Ritthoff, O.: Yale: Yet another learning environment – tutorial. Tech. rep., University of Dortmund (2002).
36. Franceschi, L., Donini, M., Frasconi, P., Pontil, M.: Forward and Reverse Gradient-Based Hyperparameter Optimization. In: Precup and Teh [122], pp. 1165–1173.
37. Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., Pontil, M.: Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In: Dy and Krause [27], pp. 1568–1577
38. Friedrichs, F., Igel, C.: Evolutionary tuning of multiple SVM parameters. Neurocomputing 64, 107–117 (2005).

39. Frohlich, H., Zell, A.: Efficient parameter selection for support vector machines in classification and regression via model-based global optimization. In: Prokhorov, D., Levine, D., Ham, F., Howell, W. (eds.) Proceedings of the 2005 IEEE International Joint Conference on Neural Networks (IJCNN). pp. 1431–1436. IEEE Computer Society Press (2005).
40. Gardner, J., Guo, C., Weinberger, K., Garnett, R., Grosse, R.: Discovering and Exploiting Additive Structure for Bayesian Optimization. In: Singh, A., Zhu, J. (eds.) Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS). vol. 54, pp. 1311–1319. Proceedings of Machine Learning Research (2017).
41. Gardner, J., Kusner, M., Xu, Z., Weinberger, K., Cunningham, J.: Bayesian Optimization with Inequality Constraints. In: Xing and Jebara [157], pp. 937–945.
42. Garrido-Merchán, E., Hernández-Lobato, D.: Dealing with integer-valued variables in Bayesian optimization with Gaussian processes. arXiv:1706.03673v2 [stats.ML] (2017).
43. Gelbart, M., Snoek, J., Adams, R.: Bayesian optimization with unknown constraints. In: Zhang, N., Tian, J. (eds.) Proceedings of the 30th conference on Uncertainty in Artificial Intelligence (UAI'14). AUAI Press (2014).
44. Ginsbourger, D., Le Riche, R., Carraro, L.: Kriging Is Well-Suited to Parallelize Optimization. In: Computational Intelligence in Expensive Optimization Problems, pp. 131–162. Springer (2010).
45. Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., Sculley, D.: Google Vizier: A service for black-box optimization. In: Matwin, S., Yu, S., Farooq, F. (eds.) Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). pp. 1487–1495. ACM Press (2017).
46. Gramacy, R., Lee, H.: Optimization under unknown constraints. Bayesian Statistics 9(9), 229–246 (2011).
47. Gretton, A., Robert, C. (eds.): Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS), vol. 51. Proceedings of Machine Learning Research (2016).
48. Guyon, I., von Luxburg, U., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.): Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'17) (2017).
49. Guyon, I., Saffari, A., Dror, G., Cawley, G.: Analysis of the IJCNN 2007 agnostic learning vs. prior knowledge challenge. Neural Networks 21(2), 544–550 (2008).
50. Guyon, I., Saffari, A., Dror, G., Cawley, G.: Model Selection: Beyond the Bayesian/Frequentist Divide. Journal of Machine Learning Research 11, 61–87 (2010).
51. Hansen, N.: The CMA evolution strategy: A tutorial. arXiv:1604.00772v1 [cs.LG] (2016).
52. Hazan, E., Klivans, A., Yuan, Y.: Hyperparameter optimization: A spectral approach. In: Proceedings of the International Conference on Learning Representations (ICLR'18) [1], published online: iclr.cc.
53. Hernandez-Lobato, D., Hernandez-Lobato, J., Shah, A., Adams, R.: Predictive Entropy Search for Multi-objective Bayesian Optimization. In: Balcan and Weinberger [8], pp. 1492– 1501.

54. Hernández-Lobato, J., Requeima, J., Pyzer-Knapp, E., Aspuru-Guzik, A.: Parallel and distributed Thompson sampling for large-scale accelerated exploration of chemical space. In: Precup and Teh [122], pp. 1470–1479.
55. Hernández-Lobato, J., Gelbart, M., Adams, R., Hoffman, M., Ghahramani, Z.: A general framework for constrained Bayesian optimization using information-based search. The Journal of Machine Learning Research 17(1), 5549–5601 (2016).
56. Hoeting, J., Madigan, D., Raftery, A., Volinsky, C.: Bayesian model averaging: a tutorial. Statistical science pp. 382–401 (1999).
57. Horn, D., Bischl, B.: Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In: Likas, A. (ed.) 2016 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1–8. IEEE Computer Society Press (2016).
58. Hutter, F.: Automated Configuration of Algorithms for Solving Hard Computational Problems. Ph.D. thesis, University of British Columbia, Department of Computer Science, Vancouver, Canada (2009).
59. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C. (ed.) Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11). Lecture Notes in Computer Science, vol. 6683, pp. 507–523. Springer (2011).
60. Hutter, F., Hoos, H., Leyton-Brown, K.: Parallel algorithm configuration. In: Hamadi, Y., Schoenauer, M. (eds.) Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION'12). Lecture Notes in Computer Science, vol. 7219, pp. 55–70. Springer (2012).
61. Hutter, F., Hoos, H., Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance. In: Xing and Jebara [157], pp. 754–762.
62. Hutter, F., Hoos, H., Leyton-Brown, K., Murphy, K.: Time-bounded sequential parameter optimization. In: Blum, C. (ed.) Proceedings of the Fourth International Conference on Learning and Intelligent Optimization (LION'10). Lecture Notes in Computer Science, vol. 6073, pp. 281–298. Springer (2010).
63. Hutter, F., Osborne, M.: A kernel for hierarchical parameter spaces. arXiv:1310.5738v1 [stats.ML] (2013).
64. Hutter, F., Lücke, J., Schmidt-Thieme, L.: Beyond Manual Tuning of Hyperparameters. KI - Künstliche Intelligenz 29(4), 329–337 (2015).
65. Igel, C.: Multi-objective Model Selection for Support Vector Machines. In: Coello, C., Aguirre, A., Zitzler, E. (eds.) Evolutionary Multi-Criterion Optimization. pp. 534–546. Springer (2005).
66. Ihler, A., Janzing, D. (eds.): Proceedings of the 32nd conference on Uncertainty in Artificial Intelligence (UAI'16). AUAI Press (2016).
67. Ilievski, I., Akhtar, T., Feng, J., Shoemaker, C.: Efficient Hyperparameter Optimization for Deep Learning Algorithms Using Deterministic RBF Surrogates. In: Sierra, C. (ed.) Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'17) (2017).
68. Jamieson, K., Recht, B.: The news on auto-tuning (2016), http://www.argmin.net/2016/06/20/hypertuning/.
69. Jamieson, K., Talwalkar, A.: Non-stochastic best arm identification and hyperparameter optimization. In: Gretton and Robert [47], pp. 240–248.

70. Jenatton, R., Archambeau, C., González, J., Seeger, M.: Bayesian Optimization with Tree-structured Dependencies. In: Precup and Teh [122], pp. 1655–1664.
71. John, G.: Cross-Validated C4.5: Using Error Estimation for Automatic Parameter Selection. Tech. Rep. STAN-CS-TN-94-12, Stanford University, Stanford University (1994).
72. Jones, D., Schonlau, M., Welch, W.: Efficient global optimization of expensive black box functions. Journal of Global Optimization 13, 455–492 (1998).
73. Kandasamy, K., Dasarathy, G., Oliva, J., Schneider, J., Póczos, B.: Gaussian Process Bandit Optimisation with Multi-fidelity Evaluations. In: Lee et al. [87], pp. 992–1000.
74. Kandasamy, K., Dasarathy, G., Schneider, J., Póczos, B.: Multi-fidelity Bayesian Optimisation with Continuous Approximations. In: Precup and Teh [122], pp. 1799–1808.
75. Kandasamy, K., Schneider, J., Póczos, B.: High Dimensional Bayesian Optimisation and Bandits via Additive Models. In: Bach and Blei [7], pp. 295–304.
76. Karnin, Z., Koren, T., Somekh, O.: Almost optimal exploration in multi-armed bandits. In: Dasgupta and McAllester [23], pp. 1238–1246.
77. King, R., Feng, C., Sutherland, A.: Statlog: comparison of classification algorithms on large real-world problems. Applied Artificial Intelligence an International Journal 9(3), 289–333 (1995).
78. Klein, A., Falkner, S., Bartels, S., Hennig, P., Hutter, F.: Fast bayesian hyperparameter optimization on large datasets. In: Electronic Journal of Statistics. vol. 11 (2017).
79. Klein, A., Falkner, S., Mansur, N., Hutter, F.: RoBO: A flexible and robust Bayesian optimization framework in Python. In: NeurIPS workshop on Bayesian Optimization (BayesOpt'17) (2017).
80. Klein, A., Falkner, S., Springenberg, J.T., Hutter, F.: Learning curve prediction with Bayesian neural networks. In: Proceedings of the International Conference on Learning Representations (ICLR'17) (2017), published online: https://iclr.cc.
81. Koch, P., Konen, W., Flasch, O., Bartz-Beielstein, T.: Optimizing support vector machines for stormwater prediction. Tech. Rep. TR10-2-007, Technische Universität Dortmund (2010).
82. Kohavi, R., John, G.: Automatic Parameter Selection by Minimizing Estimated Error. In: Prieditis, A., Russell, S. (eds.) Proceedings of the Twelfth International Conference on Machine Learning, pp. 304–312. Morgan Kaufmann Publishers (1995).
83. Komer, B., Bergstra, J., Eliasmith, C.: Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In: Hutter, F., Caruana, R., Bardenet, R., Bilenko, M., Guyon, I., Kégl, B., Larochelle, H. (eds.) ICML workshop on Automated Machine Learning (AutoML workshop 2014) (2014).
84. Konen, W., Koch, P., Flasch, O., Bartz-Beielstein, T., Friese, M., Naujoks, B.: Tuned data mining: a benchmark study on different tuners. In: Krasnogor, N. (ed.) Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO'11). pp. 1995– 2002. ACM (2011).
85. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In: Bartlett et al. [9], pp. 1097–1105.

86. Krueger, T., Panknin, D., Braun, M.: Fast cross-validation via sequential testing. Journal of Machine Learning Research (2015).
87. Lee, D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.): Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16) (2016).
88. Lee, H., Gramacy, R.: Optimization Subject to Hidden Constraints via Statistical Emulation. Pacific Journal of Optimization 7(3), 467–478 (2011).
89. Li, F.F., Li, J.: Cloud AutoML: Making AI accessible to every business (2018), https://www.blog.google/products/google-cloud/cloud-AutoML-making-ai-accessible-every-business/.
90. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research 18(185), 1–52 (2018).
91. Loshchilov, I., Hutter, F.: CMA-ES for hyperparameter optimization of deep neural networks. In: International Conference on Learning Representations Workshop track (2016), published online: https://iclr.cc.
92. Lu, X., Gonzalez, J., Dai, Z., Lawrence, N.: Structured Variationally Auto-encoded Optimization. In: Dy and Krause [27], pp. 3273–3281.
93. Luketina, J., Berglund, M., Greff, K., Raiko, T.: Scalable Gradient-Based Tuning of Continuous Regularization Hyperparameters. In: Balcan and Weinberger [8], pp. 2952–2960.
94. Luo, G.: A review of automatic selection methods for machine learning algorithms and hyperparameter values. Network Modeling Analysis in Health Informatics and Bioinformatics 5(1) (2016).
95. Lévesque, J.C.: Bayesian Hyperparameter Optimization: Overfitting, Ensembles and Conditional Spaces. Ph.D. thesis, Université Laval (2018).
96. Lévesque, J.C., Durand, A., Gagné, C., Sabourin, R.: Bayesian optimization for conditional hyperparameter spaces. In: Howell, B. (ed.) 2017 International Joint Conference on Neural Networks (IJCNN). pp. 286–293. IEEE (2017).
97. Lévesque, J.C., Gagné, C., Sabourin, R.: Bayesian Hyperparameter Optimization for Ensemble Learning. In: Ihler and Janzing [66], pp. 437–446.
98. MacKay, D.: Hyperparameters: Optimize, or Integrate Out?, pp. 43–59. Springer (1996).
99. Maclaurin, D., Duvenaud, D., Adams, R.: Gradient-based Hyperparameter Optimization through Reversible Learning. In: Bach and Blei [7], pp. 2113–2122.
100. Mantovani, R., Horvath, T., Cerri, R., Vanschoren, J., Carvalho, A.: Hyper-Parameter Tuning of a Decision Tree Induction Algorithm. In: 2016 5th Brazilian Conference on Intelligent Systems (BRACIS). pp. 37–42. IEEE Computer Society Press (2016).
101. Marcel Wever, F.M., Hüllermeier, E.: ML-Plan for unlimited-length machine learning pipelines. In: Garnett, R., Vanschoren, F.H.J., Brazdil, P., Caruana, R., Giraud-Carrier, C., Guyon, I., Kégl, B. (eds.) ICML workshop on Automated Machine Learning (AutoML workshop 2018) (2018).
102. Maron, O., Moore, A.: The racing algorithm: Model selection for lazy learners. Artificial Intelligence Review 11(1–5), 193–225 (1997).
103. McInerney, J.: An Empirical Bayes Approach to Optimizing Machine Learning Algorithms. In: Guyon et al. [48], pp. 2712–2721.

104. McIntire, M., Ratner, D., Ermon, S.: Sparse Gaussian Processes for Bayesian Optimization. In: Ihler and Janzing [66].
105. Melis, G., Dyer, C., Blunsom, P.: On the state of the art of evaluation in neural language models. In: Proceedings of the International Conference on Learning Representations (ICLR'18) [1], published online: iclr.cc.
106. Mendoza, H., Klein, A., Feurer, M., Springenberg, J., Hutter, F.: Towards automatically-tuned neural networks. In: ICML 2016 AutoML Workshop (2016).
107. Michie, D., Spiegelhalter, D., Taylor, C., Campbell, J. (eds.): Machine Learning, Neural and Statistical Classification. Ellis Horwood (1994).
108. Mohr, F., Wever, M., Höllermeier, E.: ML-Plan: Automated machine learning via hierarchical planning. Machine Learning 107(8–10), 1495–1515 (2018).
109. Momma, M., Bennett, K.: A Pattern Search Method for Model Selection of Support Vector Regression. In: Proceedings of the 2002 SIAM International Conference on Data Mining, pp. 261–274 (2002).
110. Montgomery, D.: Design and analysis of experiments. John Wiley & Sons, Inc, eighth edn. (2013).
111. Murray, I., Adams, R.: Slice sampling covariance hyperparameters of latent Gaussian models. In: Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., Culotta, A. (eds.) Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NeurIPS'10). pp. 1732–1740 (2010).
112. Nguyen, T., Gupta, S., Rana, S., Venkatesh, S.: Stable Bayesian Optimization. In: Kim, J., Shim, K., Cao, L., Lee, J.G., Lin, X., Moon, Y.S. (eds.) Advances in Knowledge Discovery and Data Mining (PAKDD'17). Lecture Notes in Artificial Intelligence, vol. 10235, pp. 578– 591 (2017).
113. Nguyen, V., Gupta, S., Rana, S., Li, C., Venkatesh, S.: Filtering Bayesian optimization approach in weakly specified search space. Knowledge and Information Systems (2018).
114. Oh, C., Gavves, E., Welling, M.: BOCK: Bayesian Optimization with Cylindrical Kernels. In: Dy and Krause [27], pp. 3865–3874.
115. Olson, R., Bartley, N., Urbanowicz, R., Moore, J.: Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In: Friedrich, T. (ed.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'16). pp. 485–492. ACM (2016).
116. Olson, R., La Cava, W., Mustahsan, Z., Varik, A., Moore, J.: Data-driven advice for applying machine learning to bioinformatics problems. In: Proceedings of the Pacific Symposium in Biocomputing 2018. pp. 192–203 (2018).
117. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NeurIPS Autodiff Workshop (2017).
118. Pedregosa, F.: Hyperparameter optimization with approximate gradient. In: Balcan and Weinberger [8], pp. 737–746.
119. Peng-Wei Chen, Jung-Ying Wang, Hahn-Ming Lee: Model selection of SVMs using GA approach. In: Proceedings of the 2004 IEEE International Joint Conference on Neural Networks (IJCNN). vol. 3, pp. 2035–2040. IEEE Computer Society Press (2004).

120. Petrak, J.: Fast subsampling performance estimates for classification algorithm selection. Technical Report TR-2000-07, Austrian Research Institute for Artificial Intelligence (2000).
121. Poloczek, M., Wang, J., Frazier, P.: Multi-Information Source Optimization. In: Guyon et al. [48], pp. 4288–4298.
122. Precup, D., Teh, Y. (eds.): Proceedings of the 34th International Conference on Machine Learning (ICML'17), vol. 70. Proceedings of Machine Learning Research (2017).
123. Provost, F., Jensen, D., Oates, T.: Efficient progressive sampling. In: Fayyad, U., Chaudhuri, S., Madigan, D. (eds.) The 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99). pp. 23–32. ACM Press (1999).
124. Rasmussen, C., Williams, C.: Gaussian Processes for Machine Learning. The MIT Press (2006).
125. Rendle, S.: Factorization machines. In: Webb, G., Liu, B., Zhang, C., Gunopulos, D., Wu, X. (eds.) Proceedings of the 10th IEEE International Conference on Data Mining (ICDM'06). pp. 995–1000. IEEE Computer Society Press (2010).
126. Ripley, B.D.: Statistical aspects of neural networks. Networks and chaos – statistical and probabilistic aspects 50, 40–123 (1993).
127. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A., Fei-Fei, L.: Imagenet large scale visual recognition challenge. International Journal of Computer Vision 115(3), 211–252 (2015).
128. Sabharwal, A., Samulowitz, H., Tesauro, G.: Selecting Near-Optimal Learners via Incremental Data Allocation. In: Schuurmans, D., Wellman, M. (eds.) Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16). AAAI Press (2016).
129. Samanta, B.: Gear fault detection using artificial neural networks and support vector machines with genetic algorithms. Mechanical Systems and Signal Processing 18(3), 625–644 (2004).
130. Sanders, S., Giraud-Carrier, C.: Informing the Use of Hyperparameter Optimization Through Metalearning. In: Gottumukkala, R., Ning, X., Dong, G., Raghavan, V., Aluru, S., Karypis, G., Miele, L., Wu, X. (eds.) 2017 IEEE International Conference on Big Data (Big Data). IEEE Computer Society Press (2017).
131. Schilling, N., Wistuba, M., Drumond, L., Schmidt-Thieme, L.: Hyperparameter optimization with factorized multilayer perceptrons. In: Appice, A., Rodrigues, P., Costa, V., Gama, J., Jorge, A., Soares, C. (eds.) Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'15). Lecture Notes in Computer Science, vol. 9285, pp. 87–103. Springer (2015).
132. Schilling, N., Wistuba, M., Drumond, L., Schmidt-Thieme, L.: Joint Model Choice and Hyperparameter Optimization with Factorized Multilayer Perceptrons. In: 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI). pp. 72–79. IEEE Computer Society Press (2015).
133. Sculley, D., Snoek, J., Wiltschko, A., Rahimi, A.: Winner's curse? on pace, progress, and empirical rigor. In: International Conference on Learning Representations Workshop track (2018), published online: https://iclr.cc.

134. Shah, A., Ghahramani, Z.: Pareto Frontier Learning with Expensive Correlated Objectives. In: Balcan and Weinberger [8], pp. 1919–1927.
135. Shahriari, B., Swersky, K., Wang, Z., Adams, R., de Freitas, N.: Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE 104(1), 148–175 (2016).
136. Shahriari, B., Bouchard-Cote, A., de Freitas, N.: Unbounded Bayesian optimization via regularization. In: Gretton and Robert [47], pp. 1168–1176.
137. SIGOPT: Improve ML models 100x faster (2018), https://sigopt.com/.
138. Simon, D.: Evolutionary optimization algorithms. John Wiley & Sons (2013).
139. Snoek, J.: Bayesian optimization and semiparametric models with applications to assistive technology. PhD Thesis, University of Toronto (2013).
140. Snoek, J., Larochelle, H., Adams, R.: Practical Bayesian optimization of machine learning algorithms. In: Bartlett et al. [9], pp. 2960–2968.
141. Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, Adams, R.: Scalable Bayesian optimization using deep neural networks. In: Bach and Blei [7], pp. 2171–2180.
142. Snoek, J., Swersky, K., Zemel, R., Adams, R.: Input warping for Bayesian optimization of non-stationary functions. In: Xing and Jebara [157], pp. 1674–1682.
143. Sparks, E., Talwalkar, A., Haas, D., Franklin, M., Jordan, M., Kraska, T.: Automating model search for large scale machine learning. In: Balazinska, M. (ed.) Proceedings of the Sixth ACM Symposium on Cloud Computing – SoCC '15. pp. 368–380. ACM Press (2015).
144. Springenberg, J., Klein, A., Falkner, S., Hutter, F.: Bayesian optimization with robust Bayesian neural networks. In: Lee et al. [87]
145. Sun, Q., Pfahringer, B., Mayo, M.: Towards a Framework for Designing Full Model Selection and Optimization Systems. In: Multiple Classifier Systems, vol. 7872, pp. 259–270. Springer (2013).
146. Swersky, K., Duvenaud, D., Snoek, J., Hutter, F., Osborne, M.: Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces. In: NeurIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'14) (2014).
147. Swersky, K., Snoek, J., Adams, R.: Multi-task Bayesian optimization. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. (eds.) Proceedings of the 27th International Conference on Advances in Neural Information Processing Systems (NeurIPS'13). pp. 2004– 2012 (2013).
148. Swersky, K., Snoek, J., Adams, R.: Freeze-thaw Bayesian optimization arXiv:1406.3896v1 [stats.ML] (2014).
149. Thornton, C., Hutter, F., Hoos, H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Dhillon, I., Koren, Y., Ghani, R., Senator, T., Bradley, P., Parekh, R., He, J., Grossman, R., Uthurusamy, R. (eds.) The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13). pp. 847–855. ACM Press (2013).
150. Wainer, J., Cawley, G.: Empirical Evaluation of Resampling Procedures for Optimising SVM Hyperparameters. Journal of Machine Learning Research 18, 1–35 (2017).

151. Wang, J., Xu, J., Wang, X.: Combination of hyperband and Bayesian optimization for hyperparameter optimization in deep learning. arXiv:1801.01596v1 [cs.CV] (2018).
152. Wang, L., Feng, M., Zhou, B., Xiang, B., Mahadevan, S.: Efficient Hyperparameter Optimization for NLP Applications. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. pp. 2112–2117. Association for Computational Linguistics (2015).
153. Wang, Z., Hutter, F., Zoghi, M., Matheson, D., de Feitas, N.: Bayesian optimization in a billion dimensions via random embeddings. Journal of Artificial Intelligence Research 55, 361–387 (2016).
154. Wang, Z., Gehring, C., Kohli, P., Jegelka, S.: Batched Large-scale Bayesian Optimization in High-dimensional Spaces. In: Storkey, A., Perez-Cruz, F. (eds.) Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS). vol. 84. Proceedings of Machine Learning Research (2018).
155. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Automatic Frankensteining: Creating Complex Ensembles Autonomously. In: Proceedings of the 2017 SIAM International Conference on Data Mining (2017).
156. Wolpert, D.: Stacked generalization. Neural Networks 5(2), 241–259 (1992).
157. Xing, E., Jebara, T. (eds.): Proceedings of the 31th International Conference on Machine Learning, (ICML'14). Omnipress (2014).
158. Zabinsky, Z.: Pure Random Search and Pure Adaptive Search. In: Stochastic Adaptive Search for Global Optimization, pp. 25–54. Springer (2003).
159. Zeng, X., Luo, G.: Progressive sampling-based Bayesian optimization for efficient and automatic machine learning model selection. Health Information Science and Systems 5(1) (2017).
160. Zhang, Y., Bahadori, M.T., Su, H., Sun, J.: FLASH: Fast Bayesian Optimization for Data Analytic Pipelines. In: Krishnapuram, B., Shah, M., Smola, A., Aggarwal, C., Shen, D., Rastogi, R. (eds.) Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). pp. 2065–2074. ACM Press (2016).

Глава 2

Метаобучение

Хоакин Ваншорен (✉), факультет математики и компьютерных наук Технического университета Эйндховена, Эйндховен, Северный Брабант, Нидерланды, e-mail: j.vanschoren@tue.nl

Метаобучение, или *обучение обучению*, – это наука о систематическом наблюдении за тем, как различные подходы машинного обучения справляются с широким спектром задач обучения, а затем на основе этого опыта, или *метаданных*, обучении новым задачам гораздо быстрее, чем это было бы возможно в ином случае. Метаобучение не только значительно ускоряет и улучшает разработку конвейеров машинного обучения или нейронных архитектур, но и позволяет нам заменить алгоритмы, разработанные вручную, новыми подходами, полученными на основе данных. В этой главе представлен обзор состояния дел в этой увлекательной и постоянно развивающейся области.

2.1. Введение

Когда мы осваиваем новые навыки, мы редко – если вообще когда-либо – начинаем с нуля. Мы начинаем с навыков, полученных ранее при решении смежных задач, повторно используем подходы, которые хорошо работали раньше, и сосредотачиваемся на том, что, вероятно, стоит попробовать, исходя из опыта [82]. С каждым приобретенным навыком обучение новым навыкам становится проще, требуя все меньше проб и ошибок. Короче говоря, мы *учимся учиться* на разных задачах. Точно так же, создавая модели машинного обучения для конкретной задачи, мы часто опираемся на опыт решения смежных задач или используем наше (часто неявное) понимание поведения методов машинного обучения, чтобы сделать правильный выбор.

Задача метаобучения заключается в том, чтобы учиться на основе предыдущего опыта систематическим способом, в полной мере используя доступные данные. Во-первых, нам нужно собрать *метаданные*, которые описывают

предыдущие задачи обучения и ранее изученные модели. Они включают точные *конфигурации алгоритмов*, использованных для обучения моделей, в том числе настройки гиперпараметров, составы конвейеров и/или архитектуры сетей, результирующие *оценки моделей*, такие как точность и время обучения, изученные параметры модели, например обученные веса нейронной сети, а также измеряемые свойства самой задачи, также известные как *метапризнаки*. Во-вторых, нам нужно учиться на имеющихся метаданных, чтобы извлекать и передавать знания, которые направляют поиск оптимальных моделей для новых задач. В этой главе представлен краткий обзор наиболее эффективных подходов к метаобучению.

Термин «метаобучение» охватывает любой тип обучения, основанный на предыдущем опыте решения других задач. Чем больше схожи между собой эти предыдущие задачи, тем больше типов метаданных мы можем использовать, и определение сходства задач становится ключевой всеобъемлющей проблемой. Возможно, нет нужды напоминать, что бесплатных обедов не бывает [57, 188]. Когда новая задача представляет собой совершенно несвязанные явления или случайный шум, использование предыдущего опыта будет неэффективным. К счастью, в реальных задачах существует множество возможностей извлечь уроки из предыдущего опыта.

В оставшейся части этой главы мы классифицируем методы метаобучения в зависимости от типа метаданных, которые они используют, от наиболее общих до наиболее специфичных для конкретной задачи. Сначала в разделе 2.2 мы обсудим, как учиться *исключительно на оценках моделей*. Эти методы можно использовать для рекомендации общеполезных конфигураций и пространств поиска конфигураций, а также для переноса знаний из *эмпирически схожих* задач. В разделе 2.3 мы рассмотрим, как можно *характеризовать* задачи, чтобы более явно выразить сходство задач и построить метамодели, которые изучают взаимосвязи между характеристиками данных и эффективностью обучения. Наконец, в разделе 2.4 вы узнаете, как можно *передавать параметры обученной модели* между задачами, которые по своей сути похожи, например имеют одинаковые входные признаки, что позволяет использовать трансферное обучение (обучение с переносом) [111] и обучение в несколько этапов [126].

Обратите внимание, что, хотя *многозадачное обучение* [25] (обучение нескольким связанным задачам одновременно) и *ансамблевое обучение* [35] (построение нескольких моделей для одной задачи) часто могут быть эффективно объединены с системами метаобучения, они сами по себе не предполагают обучения на основе предыдущего опыта решения других задач.

Эта глава основана на обзорной статье [176].

2.2. Обучение на основе оценок моделей

Предположим, что у нас есть доступ к предшествующим задачам $t_j \in T$, множеству всех известных задач, а также множеству алгоритмов обучения, полностью определенных их *конфигурациями* $\theta_i \in \Theta$; здесь Θ представля-

ет дискретное, непрерывное или смешанное пространство конфигураций, которое может охватывать настройки гиперпараметров, компоненты контейнера и/или компоненты сетевой архитектуры. $\mathbf{P}$ – это множество всех предыдущих скалярных оценок $P_{i,j} = P(\theta_i, t_j)$ конфигурации θ_i на задаче t_j в соответствии с заранее определенной мерой оценки, например точностью, и методом оценки модели, например перекрестной проверкой. $\mathbf{P}_{new}$ – это множество известных оценок $P_{i,new}$ на новой задаче t_{new}. Теперь мы хотим обучить *метаобучатель* (metalearner) L, который предсказывает рекомендуемые конфигурации Θ^*_{new} для новой задачи t_{new}. Метаобучатель обучается на метаданных $\mathbf{P} \cup \mathbf{P}_{new}$. $\mathbf{P}$ обычно собирают заранее или извлекают из репозиториев метаданных [174, 177]. $\mathbf{P}_{new}$ изучается самим методом метаобучения итеративным способом, иногда методом теплого старта с начальным $\mathbf{P}'_{new}$, сгенерированным другим методом.

2.2.1. Независимые от задачи рекомендации

Сначала представим, что у нас нет доступа к оценкам t_{new}, следовательно, $\mathbf{P}_{new} = \emptyset$. Тогда мы все еще можем выучить функцию $f : \Theta \times T \to \{\theta^*_k\}$, $k = 1..K$, дающую набор рекомендуемых конфигураций, *не зависящих* от t_{new}. Затем эти θ^*_k могут быть оценены на t_{new} для выбора лучшей конфигурации или для применения дальнейших методов оптимизации наподобие тех, что обсуждались в разделе 2.2.3.

Такие подходы часто дают ранжирование, т. е. *упорядоченное* множество θ^*_k. Обычно это делается путем разбиения Θ на множество конфигураций-кандидатов θ_i, также называемое *портфелем*, оцененных на большом количестве задач t_j. Затем мы можем получить рейтинг для каждой задачи, например, используя *показатели успешности*, *AUC* или *значительные победы* [21, 34, 85]. Однако часто бывает желательно, чтобы одинаково хорошие, но более быстрые алгоритмы занимали более высокие места, поэтому было предложено множество методов, позволяющих найти компромисс между точностью и временем обучения [21, 134]. Далее мы можем объединить эти рейтинги по отдельным задачам в *глобальный рейтинг*, например, вычислив средний рейтинг [1, 91] по всем задачам. Если данных для построения глобального рейтинга недостаточно, можно рекомендовать *подмножества конфигураций* на основе лучших известных конфигураций для каждой предыдущей задачи [70, 173] или возвращать *квазилинейные рейтинги* [30].

Чтобы найти наилучшую конфигурацию θ^* для новой, ранее не встречавшейся задачи t_{new}, можно воспользоваться простым универсальным подходом: выбрать K лучших конфигураций [21], пройти по списку и оценить каждую конфигурацию на задаче t_{new} по очереди. Этот процесс оценивания может быть остановлен после достижения заданного значения K, исчерпания бюджета времени или нахождения достаточно точной модели. В ряде исследований было показано, что в условиях ограниченного времени многоцелевые ранжирования (включая время обучения) сходятся к почти оптимальным моделям гораздо быстрее [1, 134] и обеспечивают надежную основу для сравнения алгоритмов [1, 85].

Подход, сильно отличающийся от приведенного выше, заключается в том, чтобы сначала обучить дифференцируемую функцию $f_j(\theta_i) = P_{i,j}$ на всех предыдущих оценках конкретной задачи t_j, а затем использовать градиентный спуск для поиска оптимизированной конфигурации θ_j^* для каждой предыдущей задачи [186]. Если предположить, что некоторые из задач t_j будут похожи на t_{new}, эти θ_j^* будут полезны для теплого старта методов байесовской оптимизации.

2.2.2. Построение пространства конфигураций

Предварительные оценки также могут быть использованы для обучения лучшего пространства конфигураций Θ^*. Хотя это опять же не зависит от t_{new}, удачно подобранное пространство конфигураций может радикально ускорить поиск оптимальных моделей, поскольку исследуются только наиболее подходящие области пространства. Это очень важно, когда вычислительные ресурсы ограничены, и на практике оказалось значимым фактором при практическом сравнении систем AutoML [33].

В функциональном подходе ANOVA [67] гиперпараметры считаются важными, если они объясняют большую часть дисперсии результатов применения алгоритма на данной задаче. В [136] было проведено исследование с помощью 250 000 экспериментов OpenML с тремя алгоритмами на 100 наборах данных.

Альтернативный подход заключается в том, чтобы сначала обучить оптимальное значение гиперпараметра по умолчанию, а затем определить важность гиперпараметра как *выигрыш в производительности*, который может быть достигнут при настройке гиперпараметра вместо того, чтобы оставить значение по умолчанию. Действительно, даже если гиперпараметр имеет большую дисперсию значений, может найтись одно конкретное значение, которое всегда приводит к хорошей производительности. В [120] это наблюдение было сделано с использованием около 500 000 экспериментов OpenML на шести алгоритмах и 38 наборах данных. Значения по умолчанию обучались *совместно* для всех гиперпараметров алгоритма путем первого обучения суррогатных моделей для этого алгоритма на большом количестве задач. Затем извлекалась выборка множества конфигураций, и конфигурация, которая минимизирует средний риск по всем задачам, становилась рекомендуемой конфигурацией по умолчанию. Наконец, важность (или *настраиваемость*) каждого гиперпараметра оценивалась путем наблюдения за тем, какого улучшения можно добиться путем его настройки.

В [183] значения по умолчанию обучаются *независимо* от других гиперпараметров и определяются как конфигурации, которые наиболее часто встречаются в топ-K конфигураций для каждой задачи. В том случае, если оптимальное значение по умолчанию зависит от метапризнаков (например, количества обучающих экземпляров или признаков), обучаются простые функции, включающие эти метапризнаки. Затем с помощью статистического теста определяется, допустимо ли оставить гиперпараметр по умолчанию. Для этого измеряют потерю производительности, наблюдаемую при отсут-

ствии настройки гиперпараметра (или набора гиперпараметров), в то время как все остальные параметры настроены. Этот подход был протестирован с помощью 118 000 экспериментов OpenML с двумя алгоритмами (опорные векторы и случайный лес) на 59 наборах данных.

2.2.3. Перенос конфигурации

Если мы хотим предоставить рекомендации для конкретной задачи t_{new}, нам нужна дополнительная информация о том, насколько t_{new} похожа на предыдущие задачи t_j. Один из способов сделать это – оценить ряд рекомендованных (или случайных) конфигураций t_{new}, получив новые данные $\mathbf{P}_{new}$. Если мы заметим, что оценки $P_{i,new}$ похожи на $P_{i,j}$, то t_j и t_{new} можно считать внутренне похожими, основываясь на эмпирических данных. Мы можем использовать эти знания для обучения метаобучателя, который предсказывает рекомендуемый набор конфигураций Θ^*_{new} для t_{new}. Более того, каждую выбранную конфигурацию θ^*_{new} можно оценить и добавить в $\mathbf{P}_{new}$, итеративно собирая больше эмпирических данных, чтобы узнать, какие задачи похожи друг на друга.

2.2.3.1. Относительные ориентиры

Первая мера сходства задач рассматривает относительные (парные) различия в производительности, также называемые *относительными ориентирами* (relative landmark) $RL_{a,b,j} = P_{a,j} - P_{b,j}$ между двумя конфигурациями θ_a и θ_b на конкретной задаче t_j [53]. Метод *активного тестирования* [85] использует их следующим образом: он начинает с глобально лучшей конфигурации (см. раздел 2.2.1), обозначив ее θ_{best}, и продолжает в форме соревнования. В каждом раунде выбирается «конкурент» θ_c, который наиболее убедительно превосходит θ_{best} на похожих задачах. Задачи считаются похожими, если относительные ориентиры всех оцениваемых конфигураций схожи; иными словами, если конфигурации работают одинаково как на t_j, так и на t_{new}, то задачи считаются схожими. Далее метод оценивает конкурента θ_c, получая $P_{c,new}$, обновляет сходство задач и повторяет процедуру. Ограничением этого метода является то, что он может рассматривать только конфигурации θ_i, которые были оценены на многих предыдущих задачах.

2.2.3.2. Суррогатные модели

Более гибким способом переноса информации является построение *суррогатных моделей* $s_j(\theta_i) = P_{i,j}$ для всех предшествующих задач t_j, обученных с использованием всех доступных $\mathbf{P}$. Затем можно определить сходство задач по критерию ошибки между $s_j(\theta_i)$ и $P_{i,new}$: если суррогатная модель для t_j может генерировать точные предсказания для t_{new}, то эти задачи внутренне похожи. Обычно это делается в сочетании с байесовской оптимизацией (глава 1) для определения следующей θ_i.

Вистуба и др. [187] обучают суррогатные модели на основе гауссовых процессов (GP) для каждой предыдущей задачи, плюс одну для t_{new} и объединяют

их во взвешенную нормализованную сумму, при этом новое предсказанное среднее μ определяется как взвешенная сумма отдельных μ_j (полученных из предыдущих задач t_j). Веса μ_j рассчитываются с помощью взвешенного по ядру среднего Надарая–Уотсона, где каждая задача представлена в виде вектора относительных ориентиров, а для измерения сходства между векторами относительных ориентиров t_j и t_{new} используется квадратичное ядро Эпанечникова [104]. Чем больше сходство t_j с t_{new}, тем больше вес s_j, что увеличивает влияние суррогатной модели для t_j.

Фойрер и др. [45] предлагают объединить предсказательные распределения отдельных гауссовых процессов, что делает комбинированную модель снова гауссовым процессом. Веса вычисляются в соответствии с агностическим байесовским ансамблем Лакоста и др. [81], который взвешивает предикторы в соответствии с оценкой их способности к обобщению.

Метаданные также могут быть переданы в функции сбора, а не в суррогатной модели [187]. Суррогатная модель обучается только на $P_{i,new}$, но следующая θ_i для оценки предоставляется функцией сбора, которая является средневзвешенным ожидаемого улучшения [69] на $P_{i,new}$ и предсказанных улучшений на всех предыдущих $P_{i,j}$. Вес предшествующих задач может быть снова определен через точность суррогатной модели или через относительные ориентиры. Вес компонента, от которого ожидается улучшение, постепенно увеличивается с каждой итерацией по мере сбора большего количества данных $P_{i,new}$.

2.2.3.3. Многозадачное обучение с теплым стартом

Еще один подход к сопоставлению предыдущих задач t_j заключается в обучении совместного представления задачи с использованием **P** предшествующих оценок. В [114] ориентированные на конкретную задачу байесовские линейные регрессионные суррогатные модели $s_j(\theta_i^z)$ [20] обучаются на новой конфигурации θ^z, изученной нейронной сетью прямого распространения $NN(\theta_i)$. Эта сеть обучается подходящему расширению базиса θ^z исходной конфигурации θ, в котором линейные суррогатные модели могут точно предсказать $P_{i,new}$. Суррогатные модели предварительно обучаются на метаданных OpenML, чтобы обеспечить теплый старт оптимизации $NN(\theta_i)$ в условиях многозадачного обучения. В более ранней работе по многозадачному обучению [166] предполагалось, что у нас уже есть набор «похожих» исходных задач t_j. Перенос информации между задачами t_j и t_{new} происходит путем построения совместной модели GP для байесовской оптимизации, которая изучает и использует точную связь между задачами. Обучение совместного GP масштабируется хуже, чем построение отдельного GP для каждой задачи. Спрингенберг и др. [161] также предполагают, что задачи связаны и похожи, но изучают отношения между задачами в процессе оптимизации с помощью байесовских нейронных сетей. Таким образом, их метод представляет собой некий гибрид двух предыдущих подходов. Головин и др. [58] предполагают последовательный порядок (например, по времени) между задачами. Они строят стек регрессоров GP по одному на задачу, обучая каждый GP на остатках относительно регрессора, распо-

ложенного ниже. Таким образом, каждая задача использует предыдущие задачи для определения своих приоритетов.

2.2.3.4. Другие методы

Еще один подход к поиску исходных задач t_j, наиболее похожих на t_{new} [125], основан на стратегии многоруких бандитов [139]. В этой аналогии каждый t_j – это одна «рука», а (стохастическая) награда за выбор определенной задачи (рывок «руки») определяется по критерию ошибки в предсказаниях байесовского оптимизатора на основе GP, который моделирует предварительные оценки t_j как зашумленные измерения и комбинирует их с существующими оценками t_{new}. Однако кубический рост вычислительной нагрузки GP делает этот подход менее масштабируемым.

Другой способ определения сходства задач – взять существующие оценки $P_{i,j}$, использовать выборку Томпсона [167] для получения оптимального распределения $\rho^j_{\max}$ и затем измерить расхождение Кульбака–Лейблера [80] между $\rho^j_{\max}$ и $\rho^{new}_{\max}$ [124]. Затем эти распределения объединяются в смешанное распределение на основе сходства и используются для построения функции сбора, которая предсказывает следующую наиболее перспективную конфигурацию для оценки. Этот подход был испытан на настройке двух гиперпараметров модели SVM с использованием пяти задач.

Наконец, дополнительным способом использования **P** является рекомендация того, какие конфигурации *не* следует использовать. После обучения суррогатных моделей для каждой задачи мы можем посмотреть, какие t_j наиболее похожи на t_{new}, а затем использовать $s_j(\theta_i)$ для обнаружения областей Θ, где прогнозируемая производительность будет низкой. Исключение этих областей может ускорить поиск более эффективных конфигураций. Вистуба и др. [185] делают это, используя меру сходства задач, основанную на коэффициенте *ранговой корреляции Кендалла* [73] между рангами, полученными при ранжировании конфигураций θ_i с помощью $P_{i,j}$ и $P_{i,new}$ соответственно.

2.2.4. Кривые обучения

Мы также можем извлечь метаданные о самом процессе обучения, например как быстро улучшается производительность модели при добавлении большего количества обучающих данных. Если разделить обучение на шаги s_t, обычно добавляя фиксированное количество обучающих выборок на каждом шаге, мы можем измерить производительность $P(\theta_i, t_j, s_t) = P_{i,j,t}$ конфигурации θ_i на задаче t_j после шага s_t, получая *кривую обучения* (learning curve) для всех точек времени s_t. Как было сказано в главе 1, кривые обучения также используются для ускорения оптимизации гиперпараметров определенной задачи. В метаобучении информация о кривой обучения переносится между задачами.

Оценивая конфигурацию на новой задаче t_{new}, мы можем остановить обучение после определенного числа итераций $r < t$ и по частично наблюдаемой кривой обучения предсказать, насколько хорошо конфигурация будет

работать на полном наборе данных, на основе предыдущего опыта на других задачах и решить, продолжать обучение или нет. Это может значительно ускорить поиск хороших конфигураций.

Один из подходов заключается в предположении, что схожие задачи дают схожие кривые обучения. Сначала определим расстояние между задачами на основе того, насколько похожи частичные кривые обучения: $dist(t_a, t_b) = f(P_{i,a,t}, P_{i,b,t})$ при $t = 1, \ldots, r$. Далее находим k наиболее похожих задач $t_{1\ldots k}$ и используем их полные кривые обучения для прогнозирования того, насколько хорошо конфигурация будет работать на новом полном наборе данных. Сходство задач может быть измерено путем сравнения форм частичных кривых для всех опробованных конфигураций, а прогноз делается путем адаптации «ближайшей» полной кривой (кривых) к новой частичной кривой [83, 84]. Этот подход был также успешен в сочетании с активным тестированием [86], и его можно еще больше ускорить, используя многоцелевые меры оценки, которые включают время обучения [134].

Интересно отметить, что, хотя несколько методов направлены на предсказание кривых обучения во время поиска нейронной архитектуры (глава 3), пока ни одна из этих работ не использует кривые обучения, наблюдавшиеся ранее в других задачах.

2.3. Обучение на основе свойств задачи

Еще одним богатым источником метаданных являются характеристики (метапризнаки) рассматриваемой задачи. Каждая задача $t_j \in T$ описывается вектором $m(t_j) = (m_{j,1}, \ldots, m_{j,K})$ из K метапризнаков $m_{j,k} \in M$ – множества всех известных метапризнаков. Он может быть использован для определения меры сходства задач, основанной, например, на евклидовом расстоянии между $m(t_i)$ и $m(t_j)$, чтобы мы могли перенести информацию из наиболее похожих задач в новую задачу t_{new}. Более того, вместе с предыдущими оценками $\mathbf{P}$ мы можем обучить метаобучатель L предсказывать производительность $P_{i,new}$ конфигураций θ_i на новой задаче t_{new}.

2.3.1. Метапризнаки

В табл. 2.1 приведен краткий обзор наиболее часто используемых метапризнаков, а также краткое пояснение того, почему они являются показателем качества модели. Там, где это возможно, мы также приводим формулы для их вычисления. Более полные обзоры можно найти в литературе [26, 98, 130, 138, 175].

Чтобы построить вектор метапризнаков $m(t_j)$, необходимо выбрать и обработать эти метапризнаки. Исследования метаданных OpenML показали, что оптимальный набор метапризнаков зависит от приложения [17]. Многие метапризнаки вычисляются по отдельным признакам или их комбинациям,

Таблица 2.1. Обзор часто используемых метапризнаков

Название	Формула	Характеристика модели	Варианты
Кол-во элементов	n	Скорость, масштабируемость [99]	p/n, $\log(n)$, $\log(n/p)$
Кол-во признаков	p	«Проклятие размерности» [99]	$\log(p)$, % категорий
Кол-во классов	c	Сложность, дисбаланс [99]	отн. классов min/maj
Кол-во отс. значений	m	Эффекты от подстановки [70]	% выпадений
Кол-во выбросов	o	Зашумленность данных [141]	o/n
Асиммметрия	$\frac{E(X-\mu_X)^3}{\sigma_X^3}$	Нормальность признаков [99]	min, max, μ, σ, q_1, q_3
Коэф. эксцесса	$\frac{E(X-\mu_X)^4}{\sigma_X^4}$	Нормальность признаков [99]	min, max, μ, σ, q_1, q_3
Корреляция	$\rho_{X_1X_2}$	Взаимозависимость призн. [99]	min, max, μ, σ, ρ_{XY} [158]
Ковариация	$conv_{X_1X_2}$	Взаимозависимость призн. [99]	min, max, μ, σ, cov_{XY}
Концентрация	$\tau_{X_1X_2}$	Взаимозависимость призн. [99]	min, max, μ, σ, τ_{XY}
Разреженность	sparsity(X)	Степень дискретности [143]	min, max, μ, σ
Притяжение	gravity(X)	Межклассовая дисперсия [5]	
p-значение ANOVA	$p_{val_{X_1X_2}}$	Избыточность признака [70]	$p_{val_{XY}}$ [158]
Коэф. отклонения	$\frac{\sigma_Y}{\mu_Y}$	Отклонение в цели [158]	
ρ_{λ_1} PCA	$\sqrt{\frac{\lambda_1}{1+\lambda_1}}$	Отклонение в первом PC [99]	$\frac{\lambda_1}{\sum_i \lambda_i}$ [99]
Асимметрия PCA		Асимметрия в первом PC [48]	Коэф. эксцесса PCA [48]
95% PCA	$\frac{dim_{95\%var}}{p}$	Внутренняя размерность [9]	
Вероятность класса	$P(\mathrm{C})$	Распределение класса [99]	min, max, μ, σ
Энтропия класса	$H(\mathrm{C})$	Дисбаланс класса [99]	
Норм. энтропия	$\frac{H(\mathrm{X})}{\log_2 n}$	Информативность признака [26]	min, max, μ, σ
Взаимн. информация	$MI(\mathrm{C}, \mathrm{X})$	Важность признака [99]	min, max, μ, σ
Коэф. неопределенности	$\frac{MI(\mathrm{C},\mathrm{X})}{H(\mathrm{C})}$	Важность признака [3]	min, max, μ, σ
Эквив. кол-во признаков	$\frac{H(C)}{MI(C,X)}$	Внутренняя размерность [99]	
Отношение сигнал/шум	$\frac{\overline{H(X)}-\overline{MI(C,X)}}{\overline{MI(C,X)}}$	Зашумленность данных [99]	
Дискриминатор Фишера	$\frac{(\mu_{c1}-\mu_{c2})^2}{\sigma_{c1}^2-\sigma_{c2}^2}$	Разделимость классов c_1, c_2 [64]	См. [64]
Уровень перекрытия		Перекрытие распред. классов [64]	См. [64]

Таблица 2.1 *(окончание)*

Название	Формула	Характеристика модели	Варианты
Отклонение условий		Сложность задачи [180]	См. [179, 180]
Целостность данных		Качество данных [76]	См. [76]
Кол-во узлов, листьев	$\|\eta\|$, $\|\psi\|$	Базовая сложность [113]	Глубина дерева
Длина ветви		Базовая сложность [113]	min, max, μ, σ
Кол-во узлов на признак	$\|\eta_X\|$	Важность признака [113]	min, max, μ, σ
Кол-во листов на класс	$\frac{\|\psi_c\|}{\|\psi\|}$	Сложность класса [49]	min, max, μ, σ
Совпадение листьев	$\frac{n_{\psi_i}}{n}$	Разделимость классов [16]	min, max, μ, σ
Усиление информации		Важность признака [6]	min, max, μ, σ, коэф. Джини
Ориентир (1NN)	$P(\theta_{1NN}, t_j)$	Разреженность данных [115]	Элита 1NN [115]
Ориентир (дерево)	$P(\theta_{Tree}, t_j)$	Разделимость данных [115]	Остаток случайного дерева
Ориентир (линейный)	$P(\theta_{Lin}, t_j)$	Линейная разделимость [115]	Линейный дискриминант
Ориентир (NB)	$P(\theta_{NB}, t_j)$	Независимость признаков [115]	Больше моделей [14, 88]
Относит. ориентир	$P_{a,j} - P_{b,j}$	Производит. зондирования [53]	
Выборочный ориентир	$P(\theta_i, t_j, s_t)$	Производит. зондирования [53]	

Группы сверху вниз: простые, статистические, информационно-теоретические, сложностные, основанные на моделях и ориентирные. Непрерывные признаки X и цель Y имеют среднее значение μ_X, стандартное отклонение σ_X, дисперсию σ_X^2. Категориальные признаки X и класс C имеют категориальные значения π_i, условные вероятности $\pi_{i|j}$, совместные вероятности $\pi_{i,j}$, маргинальные вероятности $\pi_{i+} = \sum_j \pi_{ij}$, энтропию $H(X) = \sum_j \pi_{i+} \log 2(\pi_{i+})$.

и их необходимо агрегировать с помощью сводной статистики (min, max, μ, σ, квартили $q_{1\ldots4}$) или гистограмм [72]. Извлечение и агрегирование необходимо выполнять систематически [117]. При вычислении сходства задач также важно нормализовать все метапризнаки [9], выполнить отбор признаков [172] или использовать методы снижения размерности (например, PCA) [17]. При обучении метамоделей можно также использовать реляционные метамодели [173] или контекстно-зависимые методы вывода [63, 71, 92].

Помимо этих метапризнаков общего назначения, было сформулировано множество более специфических вариантов. Для потоковых данных можно использовать потоковые ориентиры [135, 137], для данных временных рядов можно вычислять коэффициенты автокорреляции или наклоны регрессионных моделей [7, 121, 147], а для задач обучения без учителя можно кластеризовать данные различными способами и извлекать свойства этих класте-

ров [159]. Во многих приложениях можно также использовать информацию, специфичную для конкретной предметной области [109, 156].

2.3.2. Обучение метапризнаков

Вместо того чтобы вручную определять метапризнаки, мы можем *обучить* совместные представления для групп задач. Один из подходов заключается в построении метамоделей, которые генерируют представление ориентироподобного метапризнака M' исходя из других метапризнаков, полученных из задачи M, обученных на метаданных о производительности $\mathbf{P}$; т. е. $f : M \mapsto M'$. Сан и Фаринджер [165] делают это, оценивая предопределенный набор конфигураций θ_i на всех предыдущих задачах t_j и генерируя бинарный метапризнак $m_{j,a,b} \in M'$ для каждой парной комбинации конфигураций θ_a и θ_b, указывая, больше ли θ_a, чем θ_b; таким образом, $m'(t_j) = (m_{j,a,b}, m_{j,a,c}, m_{j,b,c}, \ldots)$. Чтобы вычислить $m_{new,a,b}$, для каждой парной комбинации (a, b) изучаются *метаправила*, каждое из которых предварительно определяет, будет ли θ_a превосходить θ_b в задаче t_j, учитывая другие метапризнаки $m(t_j)$.

Можно также выучить совместное представление, основанное полностью на доступных метаданных $\mathbf{P}$, т. е. $f : \mathbf{P} \times \Theta \mapsto M'$. Ранее в разделе 2.2.3 мы обсуждали, как это сделать с помощью нейронных сетей прямого распространения [114]. Если задачи имеют одно и то же входное пространство, например это изображения с одинаковым разрешением, можно также использовать глубокое метрическое обучение для изучения представления метафункций, например с помощью *сиамских сетей* (Siamese network) [75].

Они обучаются путем подачи данных двух различных задач в две сети-близнецы и использования разницы между предсказанной и наблюдаемой производительностью $P_{i,new}$ в качестве сигнала ошибки. Поскольку параметры модели обеих сетей связаны в сиамскую сеть, две очень похожие задачи отображаются на одни и те же области в скрытом пространстве метапризнаков. Они могут быть использованы для оптимизации гиперпараметров байесовской сети с теплым стартом [75] и поиска нейронной архитектуры [2].

2.3.3. Оптимизация с теплым стартом на основе схожих задач

Использование метапризнаков – это естественный и удобный способ оценить сходство задач и инициализировать процедуры оптимизации на основе перспективных конфигураций в схожих задачах. Это похоже на то, как опытные эксперты начинают ручной поиск хороших моделей, учитывая опыт решения схожих задач.

Запуск алгоритма *генетического поиска* в областях пространства поиска с перспективными решениями может значительно ускорить сходимость к хорошему решению. Гомес и др. [59] рекомендуют создавать начальную конфигурацию путем нахождения k наиболее похожих предшествующих за-

дач t_j на основе расстояния L1 между векторами $m(t_j)$ и $m(t_{new})$, где каждый $m(t_j)$ включает 17 простых и статистических метапризнаков. Для каждой из k наиболее похожих задач оценивается наилучшая конфигурация на t_{new}, которая затем используется для инициализации алгоритма генетического поиска (оптимизация методом роя частиц), а также *поиска с запретами* (tabu search). Рейф и др. [129] придерживаются очень похожего подхода, используя 15 простых, статистических и ориентирных метапризнаков. Они используют метод прямого отбора для поиска наиболее полезных метапризнаков и запускают стандартный генетический алгоритм (GAlib) с модифицированной операцией гауссовой мутации. Варианты активного тестирования (раздел 2.2.3), использующие метапризнаки, также были опробованы [85, 100], но не показали лучших результатов, чем подходы, основанные на относительных ориентирах.

Подходы к оптимизации на основе моделей тоже могут значительно выиграть от удачного начального набора перспективных конфигураций. Алгоритм SCoT [9] обучает одну суррогатную модель ранжирования $f : M \times \Theta \rightarrow R$, предсказывающую ранг θ_i в задаче t_j. M содержит четыре метапризнака (три простых и один на основе PCA). Суррогатная модель обучается на всех рангах, в том числе и на t_{new}. Ранжирование используется, потому что шкала значений оценки может сильно различаться между задачами. Регрессия GP преобразует ранги в вероятности для проведения байесовской оптимизации, и каждая новая $P_{i,new}$ используется для повторного обучения суррогатной модели после каждого шага.

Шиллинг и др. [148] используют модифицированный многослойный перцептрон в качестве суррогатной модели в виде $s_j(\theta_i, m(t_j), b(t_j)) = P_{i,j}$, где $m(t_j)$ – метапризнаки, а $b(t_j)$ – вектор из j двоичных признаков, которые равны 1, если метаобъект относится к t_j, и 0 в противном случае. Многослойный перцептрон использует модифицированную функцию активации на основе факторизационных машин [132] в первом слое, направленную на обучение скрытого представления каждой задачи для моделирования сходства задач. Поскольку эта модель не может представлять неопределенности, для получения прогнозируемых средних и моделирования дисперсий обучается ансамбль из 100 многослойных перцептронов.

Обучению единственной суррогатной модели на всех предыдущих метаданных зачастую не хватает масштабируемости. Йогатама и Манн [190] также строят единую байесовскую суррогатную модель, но включают только задачи, похожие на t_{new}, где сходство задач определяется как евклидово расстояние между векторами метапризнаков, состоящими из трех простых метапризнаков. Значения $P_{i,j}$ стандартизированы, чтобы преодолеть проблему различных шкал для каждого t_j. Суррогатная модель обучается гауссову процессу с определенной комбинацией ядер на всех экземплярах.

Фойрер и др. [48] предлагают более простой и масштабируемый метод, который запускает байесовскую оптимизацию путем сортировки всех предварительных задач t_j аналогично [59], но включая 46 простых, статистических и ориентирующих метапризнаков, а также $H(\mathrm{C})$. Для теплого запуска суррогатной модели используются t лучших конфигураций на d наиболее похожих задачах. При этом перебирается гораздо больше гиперпараметров,

чем в предыдущих работах, включая этапы предварительной обработки. Этот подход с теплым стартом был также использован в более поздней работе [46], которая подробно обсуждается в главе 6.

Наконец, для рекомендации перспективных конфигураций можно также использовать *совместную фильтрацию* (collaborative filtering) [162]. По аналогии задачи t_j (пользователи) предоставляют оценки ($P_{i,j}$) для конфигураций θ_i (элементов), а методы матричной факторизации используются для предсказания неизвестных значений $P_{i,j}$ и рекомендации лучших конфигураций для произвольной задачи. Важным вопросом здесь является проблема холодного старта, поскольку матричная факторизация требует по крайней мере некоторых оценок на t_{new}. Янг и др. [189] используют D-оптимальную схему эксперимента для выборки начального набора оценок $P_{i,new}$. Они прогнозируют как прогностическую производительность, так и время работы, чтобы рекомендовать набор конфигураций для теплого старта, которые являются одновременно точными и быстрыми. Мисир и Себаг [102, 103] используют метапризнаки для решения проблемы холодного старта. Фуси и др. [54] также используют метапризнаки, следуя той же процедуре, что и в [46], и применяют подход вероятностной матричной факторизации, который позволяет им выполнять дальнейшую байесовскую оптимизацию конфигураций конвейера θ_i. Этот подход позволяет получить полезные скрытые встраивания как задач, так и конфигураций, в которых байесовская оптимизация может быть выполнена более эффективно.

2.3.4. Метамодели

Мы также можем изучить сложную взаимосвязь между метапризнаками задачи и полезностью конкретных конфигураций, построив метамодель L, которая рекомендует наиболее полезные конфигурации Θ^*_{new} исходя из заданных метапризнаков M новой задачи t_{new}. Существует большое количество работ [22, 56, 87, 94] по построению метамоделей для выбора алгоритмов [15, 19, 70, 115] и рекомендации гиперпараметров [4, 79, 108, 158]. Эксперименты показали, что расширяемые деревья (boosted tree) и бэггинг-деревья (bagged tree) часто дают лучшие предсказания, хотя многое зависит от конкретных используемых метапризнаков [72, 76].

2.3.4.1. Ранжирование

Метамодели также могут генерировать *рейтинг* из топ-K наиболее перспективных конфигураций. Один из подходов заключается в построении метамодели k-ближайших соседей (kNN) для предсказания того, какие задачи похожи, а затем ранжирования лучших конфигураций для этих похожих задач [23, 147]. Это похоже на работу, рассмотренную в разделе 2.3.3, но без привязки к последующему подходу оптимизации. Метамодели, специально предназначенные для ранжирования, такие как *деревья предсказательной кластеризации* (predictive clustering trees) [171] и *деревья ранжирования меток* (label ranking trees) [29], также показали хорошие результаты. Алгоритмы

Approximate Ranking Tree Forests (ART Forests) [165] – ансамбли быстрых ранжирующих деревьев – оказываются особенно эффективными, поскольку они имеют «встроенный» выбор метапризнаков, хорошо работают, даже если доступно мало предыдущих задач, а ансамблирование делает метод более надежным. Алгоритм autoBagging [116] ранжирует рабочие процессы бэггинга, включая четыре различных гиперпараметра, используя ранжировщик на основе XGBoost, обученный на 140 наборах данных OpenML и 146 метапризнаках. Лорена и др. [93] рекомендуют конфигурации SVM для задач регрессии при помощи метамодели kNN и нового набора метапризнаков, основанных на сложности данных.

2.3.4.2. Прогнозирование производительности

Метамодели также могут напрямую предсказывать производительность, например точность или время обучения, в конфигурации для данной задачи исходя из ее метапризнаков. Это позволяет нам понять, будет ли конфигурация достаточно интересна для оценки в какой-либо процедуре оптимизации. В ранних работах для прогнозирования производительности дискретного набора конфигураций и их последующего ранжирования применялись линейная регрессия или регрессоры на основе правил [14, 77]. Гуэрра и др. [61] обучают метарегрессор SVM для каждого алгоритма классификации, чтобы предсказать его точность на новой задаче t_{new} исходя из метапризнаков. Рейф и др. [130] обучают аналогичный метарегрессор на большем количестве метаданных, чтобы предсказать его *оптимизированную* производительность. Дэвис и др. [32] вместо этого используют метарегрессор на основе многослойного перцептрона, предсказывая производительность конкретной конфигурации алгоритма.

Вместо прогнозирования производительности метарегрессор также может быть обучен для предсказания времени обучения/прогнозирования алгоритма, например, с помощью SVM-регрессора, обученного на метапризнаках [128], который сам настраивается с помощью генетических алгоритмов [119]. Янг и др. [189] предсказывают время работы конфигурации с помощью полиномиальной регрессии, основываясь только на количестве экземпляров и признаков. Хаттер и др. [68] описали общие соображения по прогнозированию времени работы алгоритмов в различных областях.

Большинство этих метамоделей генерируют перспективные конфигурации, но фактически не настраивают эти конфигурации на t_{new}. Вместо этого прогнозы могут быть использованы для теплого старта или задания рабочего направления для любого другого метода оптимизации, что позволяет использовать всевозможные комбинации метамоделей и техник оптимизации. Действительно некоторые работы, обсуждаемые в разделе 2.3.3, можно рассматривать как использование метамодели на основе расстояния для теплого старта байесовской оптимизации [48, 54] или эволюционных алгоритмов [59, 129]. В принципе, здесь могут быть использованы и другие метамодели.

Вместо того чтобы изучать связь между метапризнаками задачи и производительностью конфигурации, можно построить суррогатные модели,

предсказывающие производительность конфигураций на конкретных задачах [40]. Затем можно научиться комбинировать эти прогнозы для каждой задачи, чтобы запустить или направить методы оптимизации для новой задачи t_{new} [45, 114, 161, 187], как сказано в разделе 2.2.3. Хотя метапризнаки также можно использовать для объединения прогнозов по задачам на основе сходства задач, в конечном итоге эффективнее собирать новые наблюдения $P_{i,new}$, поскольку они позволяют нам уточнять оценки сходства задач с каждым новым наблюдением [47, 85, 187].

2.3.5. Синтез конвейера

При создании полных конвейеров машинного обучения [153] количество вариантов конфигурации резко возрастает, что делает еще более важным использование предыдущего опыта. Можно ограничить пространство поиска, накладывая на конвейер фиксированную структуру, полностью описываемую набором гиперпараметров. Затем можно использовать наиболее перспективные конвейеры для аналогичных задач, чтобы запустить байесовскую оптимизацию [46, 54].

Другие подходы дают рекомендации для определенных этапов конвейера [118, 163] и могут быть использованы в более масштабных подходах к построению конвейера, таких как планирование [55, 74, 105, 184] или эволюционные методы [110, 164]. Нгуен и др. [105] строят новые конвейеры с помощью *лучевого поиска* (beam search), сосредоточенного на компонентах, рекомендованных метаобучающим методом, который сам обучен на примерах успешных предыдущих конвейеров. Билалли и др. [18] предсказывают, какие методы предварительной обработки рекомендуются для данного алгоритма классификации. Они строят метамодель для каждого целевого алгоритма классификации, которая, исходя из t_{new} новых метапризнаков, предсказывает, какие методы предварительной обработки должны быть включены в конвейер. Аналогично Шенфельд и др. [152] строят метамодели, предсказывающие, когда алгоритм предварительной обработки улучшит точность или время работы конкретного классификатора.

Метод AlphaD3M [38] использует подход *самовоспроизводящегося* (self-play) обучения с подкреплением, в котором текущее состояние представлено текущим конвейером, а действия включают добавление, удаление или замену компонентов конвейера. *Поиск по дереву Монте-Карло* (Monte Carlo tree search, MCTS) генерирует конвейеры, оцениваемые для обучения рекуррентной нейронной сети с долгой краткосрочной памятью (long short-term memory, LSTM), которая может предсказывать производительность конвейера, в свою очередь производя вероятности действий для MCTS в следующем раунде. Описание состояния также включает метапризнаки текущей задачи, что позволяет нейронной сети обучаться на разных задачах. Метод Mosaic [123] также генерирует конвейеры с помощью MCTS, но вместо этого использует для выбора перспективных конвейеров подход на основе многоруких бандитов.

2.3.6. Настраивать или не настраивать?

Для уменьшения количества параметров конфигурации, подлежащих оптимизации, и экономии ценного времени оптимизации в условиях жестких временных рамок проекта были также предложены метамодели, позволяющие предсказать, стоит ли настраивать данный алгоритм, *исходя из метапризнаков решаемой задачи* [133], и насколько ценных улучшений можно ожидать от настройки конкретного алгоритма по сравнению с дополнительными затратами времени [144]. Более целенаправленные исследования конкретных алгоритмов обучения позволили создать метамодели, предсказывающие, когда необходимо настраивать SVM [96], каковы хорошие гиперпараметры по умолчанию для SVM с учетом задачи (включая интерпретируемые метамодели) [97] и как настраивать деревья решений [95].

2.4. Обучение на основе предыдущих моделей

Последний тип метаданных, которые можно изучать, – это предшествующие модели машинного обучения, т. е. их структура и изученные параметры модели. Если кратко, мы хотим обучить *метаобучатель* (meta-learner) L, который учится тому, как готовить обучатель l_{new} для новой задачи t_{new}, на основе похожих задач $t_j \in T$ и соответствующих оптимизированных моделей $l_j \in \mathcal{L}$, где $\mathcal{L}$ – пространство всех возможных моделей. Обучатель l_j обычно определяется параметрами его модели $W = \{w_k\}$, $k = 1...K$ и/или конфигурацией $\theta_i \in \Theta$.

2.4.1. Трансферное обучение

В *трансферном обучении* (transfer learning) [170] мы берем модели, обученные на одной или нескольких *исходных* задачах t_j, и используем их как отправные точки для создания модели на аналогичной *целевой* задаче t_{new}. Это можно сделать, заставив целевую модель быть структурно или иным образом похожей на исходную модель (модели). Это обобщенная идея, на основании которой подходы трансферного обучения были предложены для ядерных методов [41, 42], параметрических байесовских моделей [8, 122, 140], байесовских сетей [107], кластеризации [168] и обучения с подкреплением [36, 62]. Однако лучше всего подходят для трансферного обучения нейронные сети, поскольку структура и параметры модели-источника могут быть использованы в качестве хорошей инициализации для целевой модели, создавая *предварительно обученную* модель, которая затем может быть доработана с использованием доступных обучающих данных t_{new} [11, 13, 24, 169]. В некоторых случаях перед переносом исходной сети может потребоваться ее модификация [155]. В оставшейся части этого раздела мы сосредоточимся на нейронных сетях.

Было показано, что достаточно большие наборы данных изображений, такие как ImageNet [78], позволяют получать предварительно обученные модели, которые исключительно хорошо переносятся на другие задачи [37, 154]. Однако было также показано, что этот подход не очень хорошо работает, когда целевая задача недостаточно похожа [191]. Вместо того чтобы надеяться, что предварительно обученная модель «случайно» хорошо переносится на новую задачу, мы можем целенаправленно наделить метаобучатели индуктивным предубеждением (полученным в результате решения многих похожих задач), что позволит им гораздо быстрее обучаться новым задачам, о чем мы поговорим ниже.

2.4.2. Метаобучение в нейронных сетях

Одним из первых подходов к метаобучению является создание *рекуррентных нейронных сетей* (recurrent neural network, RNN), способных изменять свои собственные веса [149, 150]. В процессе обучения они используют собственные веса в качестве дополнительных входных данных и наблюдают за собственными ошибками, чтобы научиться изменять эти веса в ответ на новую задачу. Обновление весов определяется в параметрической форме, которая полностью дифференцируема и может совместно оптимизировать как сеть, так и алгоритм обучения с помощью градиентного спуска, но при этом очень сложна для обучения. В более поздних работах предложено использовать обучение с подкреплением для разных задач, чтобы адаптировать стратегию поиска [151] или скорость обучения для градиентного спуска [31] к конкретной задаче.

Опираясь на интуитивное ощущение, что обратное распространение ошибки является маловероятным механизмом обучения для нашего мозга, Бенджио и др. [12] заменили обратное распространение простыми параметрическими правилами обновления синаптических весов (или эволюционировавшими правилами [27]), основанными на биологических принципах. Параметры оптимизируются, например, с помощью градиентного спуска или эволюции по набору входных задач. Рунарссон и Йонссон [142] заменили эти параметрические правила однослойной нейронной сетью. Санторо и др. [146] вместо этого используют нейронную сеть с расширенной памятью, чтобы научиться хранить и извлекать «воспоминания» о предыдущих задачах классификации. Хохрайтер и др. [65] используют LSTM [66] в качестве метаобучателя для обучения многослойных персептронов.

Андрихович и др. [6] тоже заменяют оптимизатор, например стохастический градиентный спуск, на LSTM, обученный на нескольких предыдущих задачах. Потери метаобучателя (оптимизатора) определяются как сумма потерь базовых обучателей (оптимизаторов) и оптимизируются с помощью градиентного спуска. На каждом шаге метаобучатель выбирает обновление веса, которое, как ожидается, уменьшает потери оптимизатора больше всего, основываясь на выученных весах модели $\{w_k\}$ на предыдущем шаге, а также на текущем градиенте производительности. Более поздние работы обобщают этот подход, обучая оптимизатор на синтетических функциях, используя

градиентный спуск [28]. Это позволяет метаобучающим системам оптимизировать оптимизаторы, даже если они не имеют доступа к градиентам.

Параллельно Ли и Малик [89] предложили схему обучения алгоритмов оптимизации с точки зрения обучения с подкреплением. Она представляет любой конкретный алгоритм оптимизации как стратегию, а затем обучается этой стратегии с помощью направленного поиска стратегий. В последующей работе [90] показано, как использовать этот подход для обучения алгоритмам оптимизации для (неглубоких) нейронных сетей.

Группа алгоритмов *поиска нейронных архитектур* включает в себя множество методов, которые строят модель производительности нейронной сети для конкретной задачи, например, с помощью байесовской оптимизации или обучения с подкреплением. Более подробное обсуждение приведено в главе 3. Однако большинство из этих методов пока не обобщаются на все задачи и поэтому в данной книге не рассматриваются.

2.4.3. Обучение на ограниченных данных

Особенно сложной проблемой метаобучения является обучение точной глубокой модели всего на нескольких обучающих выборках с использованием предыдущего опыта решения очень похожих задач, для которых у нас есть большие обучающие наборы. Это называется *пристрелочным обучением* (few-shot learning) или *обучением на ограниченных данных*. Люди обладают врожденной способностью к обучению на ограниченных данных, и разработчики стремятся создать агенты машинного обучения, способные делать то же самое [82]. В качестве примера можно привести классификацию «*K*-shot *N*-way» (*K* попыток, *N* вариантов), когда нам дано много образцов (например, изображений) определенных классов (например, объектов) и мы хотим обучить классификатор l_{new}, способный классифицировать *N* новых классов, используя только *K* примеров каждого из них.

Используя предыдущий опыт, мы можем, например, выучить общее представление признаков для всех задач, запустить обучение l_{new} с лучшей инициализации параметров модели W_{init} и получить *индуктивное смещение* (inductive bias), которое поможет направлять оптимизацию параметров модели, так что обучение l_{new} будет проходить гораздо быстрее, чем это возможно без смещения.

Более ранние работы по обучению на ограниченных данных в значительной степени основаны на ручном извлечении признаков [10, 43, 44, 50]. Однако метаобучение потенциально дает возможность выучить общее представление признаков для всех задач без применения ручного труда экспертов.

Виньялс и др. [181] утверждают, что для обучения на очень малом количестве данных следует обратиться к непараметрическим моделям (таким как *k*-ближайшие соседи), которые используют компонент памяти вместо обучения большого количества параметров модели. Их метаобучающая система – это *сопоставляющая сеть* (matching network), которая использует идею компонента памяти в нейронной сети. Она изучает общее представление для

помеченных обучающих образцов и *сопоставляет* каждый новый тестовый экземпляр с запомненными примерами, используя критерий косинусного сходства. Сеть обучается на мини-пакетах, каждый из которых содержит всего несколько примеров конкретной задачи.

Снелл и др. [157] предлагают *прототипические сети* (prototypical network), которые отображают точки данных в p-мерное векторное пространство таким образом, что экземпляры заданного выходного класса находятся близко друг к другу. Затем вычисляется *прототип* (средний вектор) для каждого класса. Новые тестовые экземпляры отображаются в то же векторное пространство, а для создания функции softmax по всем возможным классам применяется метрика расстояния. Рен и др. [131] расширяют этот подход до частичного обучения с учителем.

Рави и Ларошель [126] используют метаобучение на основе LSTM для изучения правила обновления при обучении нейронной сети. С каждым новым примером обучатель возвращает текущий градиент и потери метаобучателю LSTM, который затем обновляет параметры модели $\{w_k\}$ обучателя. Метаобучатель обучается на всех предыдущих задачах.

В свою очередь, метод *метаобучения*, *инвариантного к модели* (model-agnostic meta-learning, MAML), [51] не пытается выучить правило обновления, а вместо этого изучает инициализацию параметров модели W_{init}, которая лучше обобщается на похожие задачи. Начав со случайного набора $\{w_k\}$, он итеративно выбирает пакет предыдущих задач и для каждой из них учит обучатель на K образцах для вычисления градиента и потерь (на тестовом наборе). Затем при помощи обратного распространения *метаградиента* он обновляет веса $\{w_k\}$ в том направлении, в котором их было бы легче обновить. Другими словами, после каждой итерации веса $\{w_k\}$ становятся лучшим W_{init} для начала точной настройки любой из задач. Финн и Левин [52] также утверждают, что MAML способен аппроксимировать любой алгоритм обучения при использовании достаточно глубокой полносвязной сети ReLU и определенных потерях. Они также пришли к выводу, что инициализация MAML более устойчива к переобучению на малых выборках и обобщает более широко, чем подходы метаобучения на основе LSTM.

Метод REPTILE [106] – это аппроксимация MAML, которая выполняет стохастический градиентный спуск для K итераций на определенной задаче, а затем постепенно перемещает веса инициализации в направлении весов, полученных после K итераций. Интуиция подсказывает, что каждая задача, вероятно, имеет более одного набора оптимальных весов $\{w_i^*\}$, а цель состоит в том, чтобы найти W_{init}, который близок хотя бы к одному из этих $\{w_i^*\}$ для каждой задачи.

Наконец, мы также можем вывести метаобучатель из нейронной сети черного ящика. Санторо и др. [145] предлагают *нейронные сети с дополненной памятью* (memory-augmented neural network, MANN), которые обучают *нейронную машину Тьюринга* (neural Turing machine, NTM) [60] – нейронную сеть с расширенными возможностями запоминания – в качестве метаобучателя. Затем этот метаобучатель может запоминать информацию о предыдущих задачах и использовать ее для обучения нового обучателя l_{new}. SNAIL [101] – это общая архитектура метаобучателя, состоящая из чередующихся слоев

временной свертки и каузального внимания. Сверточные сети изучают общий вектор признаков для обучающих экземпляров (изображений), чтобы агрегировать информацию из прошлого опыта. Слои каузального внимания изучают, какие части информации следует выбрать из накопленного опыта, чтобы обобщить их для решения новых задач.

В целом пересечение глубокого обучения и метаобучения служит особенно плодородной почвой для новых революционных идей, и мы ожидаем, что со временем эта область станет еще более важной.

2.4.4. За рамками обучения с учителем

Метаобучение, конечно, не ограничивается задачами обучения с учителем и успешно применяется для решения таких разнообразных задач, как обучение с подкреплением, активное обучение, оценка плотности и рекомендация объектов. Базовый обучатель может быть основан на обучении без учителя, а метаобучатель – с учителем, но возможны и другие комбинации.

Дуан и др. [39] предлагают сквозной подход к обучению с подкреплением (RL), состоящий из *быстрого* алгоритма RL для конкретной задачи, который управляется *медленным* алгоритмом мета-RL общего назначения. Задачи представляют собой взаимосвязанные *марковские процессы принятия решений* (Markov decision processes, MDP). Алгоритм мета-RL реализован в форме RNN, которая получает наблюдения, действия, вознаграждения и флаги завершения. Активации RNN хранят состояние быстрого RL-обучателя, а веса RNN обучаются путем наблюдения за эффективностью быстрых обучающихся в разных задачах.

Примерно в то же время Ванг и др. [182] предложили использовать алгоритм глубокого RL для обучения RNN, получая действия и вознаграждения предыдущего интервала, с целью обучения алгоритма RL базового уровня для конкретных задач. Вместо использования относительно неструктурированных задач, таких как случайные MDP, они сосредоточились на структурированных распределениях задач (например, таких как зависимые бандиты), в которых алгоритм мета-RL может использовать присущую задаче структуру.

Панг и др. [112] предложили применить подход метаобучения к *активному обучению* (active learning, AL). Базовым обучателем может служить любой двоичный классификатор, а метаобучатель – это глубокая RL-сеть, состоящая из глубокой нейронной сети, которая изучает представление проблемы AL для всех задач, и сети стратегии, которая изучает оптимальную стратегию, параметризованную в виде весов в сети. Метаобучатель получает текущее состояние (набор неразмеченных точек и состояние базового классификатора) и вознаграждение (производительность базового классификатора) и выдает вероятность запроса, т. е. какие точки в неразмеченном наборе запрашивать следующими.

Рид и др. [127] предлагают подход с ограниченными данными на основе *оценки плотности* (density estimation, DE). Целью является изучение распределения вероятностей по небольшому количеству изображений опре-

деленного концепта (например, рукописной буквы), которое может быть использовано для создания изображений этого концепта или вычисления вероятности того, что этот концепт присутствует на изображении. В данном подходе используются авторегрессионные модели изображений, которые раскладывают совместное распределение на пиксельные коэффициенты. Обычно они определены через большое количество образцов целевого концепта. Вместо них можно использовать MAML-обучатель на основе ограниченных данных, предварительно обученный на большом количестве образцов других (похожих) концептов.

Наконец, Вартак и др. [178] рассматривают проблему холодного старта в матричном разложении. Они предлагают архитектуру глубокой нейронной сети, обучающей базовую нейронную сеть, смещения которой корректируются на основе информации о задаче. В то время как структура и веса рекомендательных нейронных сетей остаются фиксированными, метаобучатель учится настраивать смещения на основе истории запросов каждого пользователя.

Все эти последние разработки указывают на тот факт, что часто бывает полезно взглянуть на проблемы через призму метаобучения и найти новые основанные на данных подходы для замены разработанных вручную базовых обучающих систем.

2.5. Заключение

Возможности метаобучения проявляются по-разному, и их можно использовать с помощью широкого спектра методов обучения. Каждый раз, когда мы пытаемся обучить модель для определенной задачи, независимо от того, удалось это или нет, мы получаем полезный опыт, который можно использовать для изучения новых задач. Не следует начинать процесс разработки и обучения модели с нуля. Вместо этого мы должны систематически собирать «историю обучения» и использовать ее, чтобы создавать системы AutoML, которые постоянно совершенствуются, помогая нам обучать новые модели все более эффективно. Чем больше новых задач мы встречаем и чем более схожи эти новые задачи, тем больше мы можем использовать предыдущий опыт, добиваясь того, что большая часть необходимого обучения окажется сделанной заранее. Способность компьютерных систем хранить практически безграничный опыт предыдущих попыток обучения (в виде метаданных) открывает широкие возможности для использования этого опыта совершенно новыми способами, а мы только начинаем изучать, как эффективно учиться на предыдущем опыте. Тем не менее это достойная цель: изучение того, как научить модели решать любые задачи, дает нам гораздо больше возможностей, чем знание того, как научиться решать конкретные задачи.

Благодарности. Автор благодарит Павла Браздила, Матиаса Фойрера, Франка Хаттера, Рагху Раджана, Эрин Грант, Хьюго Ларошеля, Яна ван Рейна и Джейн Ванг за многочисленные бесценные советы и отзывы о рукописи главы.

2.6. Литература

1. Abdulrahman, S., Brazdil, P., van Rijn, J., Vanschoren, J.: Speeding up Algorithm Selection using Average Ranking and Active Testing by Introducing Runtime. Machine Learning 107, 79–108 (2018).
2. Afif, I.N.: Warm-Starting Deep Learning Model Construction using Meta-Learning. Master's thesis, TU Eindhoven (2018).
3. Agresti, A.: Categorical Data Analysis. Wiley Interscience (2002).
4. Ali, S., Smith-Miles, K.A.: Metalearning approach to automatic kernel selection for support vector machines. Neurocomputing 70(1), 173–186 (2006).
5. Ali, S., Smith-Miles, K.A.: On learning algorithm selection for classification. Applied Soft Computing 6(2), 119–138 (2006).
6. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., De Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Advances in Neural Information Processing Systems. pp. 3981–3989 (2016).
7. Arinze, B.: Selecting appropriate forecasting models using rule induction. Omega 22(6), 647– 658 (1994).
8. Bakker, B., Heskes, T.: Task Clustering and Gating for Bayesian Multitask Learning. Journal of Machine Learning Research 4, 83–999 (2003).
9. Bardenet, R., Brendel, M., Kégl, B., Sebag, M.: Collaborative hyperparameter tuning. In: Proceedings of ICML 2013. pp. 199–207 (2013).
10. Bart, E., Ullman, S.: Cross-generalization: Learning novel classes from a single example by feature replacement. In: Proceedings of CVPR 2005. pp. 672–679 (2005).
11. Baxter, J.: Learning Internal Representations. In: Advances in Neural Information Processing Systems, NeurIPS (1996).
12. Bengio, S., Bengio, Y., Cloutier, J.: On the search for new learning rules for anns. Neural Processing Letters 2(4), 26–30 (1995).
13. Bengio, Y.: Deep learning of representations for unsupervised and transfer learning. In: ICML Workshop on Unsupervised and Transfer Learning. pp. 17–36 (2012).
14. Bensusan, H., Kalousis, A.: Estimating the predictive accuracy of a classifier. Lecture Notes in Computer Science 2167, 25–36 (2001).
15. Bensusan, H., Giraud-Carrier, C.: Discovering task neighbourhoods through landmark learning performances. In: Proceedings of PKDD 2000. pp. 325–330 (2000).
16. Bensusan, H., Giraud-Carrier, C., Kennedy, C.: A higher-order approach to meta-learning. In: Proceedings of ILP 2000. pp. 33–42 (2000).
17. Bilalli, B., Abelló, A., Aluja-Banet, T.: On the predictive power of meta-features in OpenML. International Journal of Applied Mathematics and Computer Science 27(4), 697–712 (2017).
18. Bilalli, B., Abelló, A., Aluja-Banet, T., Wrembel, R.: Intelligent assistance for data preprocessing. Computer Standards and Interfaces 57, 101–109 (2018).
19. Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., Vanschoren, J.: ASLib:

A benchmark library for algorithm selection. Artificial Intelligence 237, 41–58 (2016).
20. Bishop, C.M.: Pattern recognition and machine learning. Springer (2006).
21. Brazdil, P., Soares, C., da Costa, J.P.: Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. Machine Learning 50(3), 251–277 (2003).
22. Brazdil, P., Giraud-Carrier, C., Soares, C., Vilalta, R.: Metalearning: Applications to Data Mining. Springer-Verlag Berlin Heidelberg (2009).
23. Brazdil, P.B., Soares, C., Da Coasta, J.P.: Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. Machine Learning 50(3), 251–277 (2003).
24. Caruana, R.: Learning many related tasks at the same time with backpropagation. Neural Information Processing Systems pp. 657–664 (1995).
25. Caruana, R.: Multitask Learning. Machine Learning 28(1), 41–75 (1997).
26. Castiello, C., Castellano, G., Fanelli, A.M.: Meta-data: Characterization of input features for meta-learning. In: 2nd International Conference on Modeling Decisions for Artificial Intelligence (MDAI). pp. 457–468 (2005).
27. Chalmers, D.J.: The evolution of learning: An experiment in genetic connectionism. In: Connectionist Models, pp. 81–90. Elsevier (1991).
28. Chen, Y., Hoffman, M.W., Colmenarejo, S.G., Denil, M., Lillicrap, T.P., Botvinick, M., de Freitas, N.: Learning to learn without gradient descent by gradient descent. In: Proceedings of ICML 2017, PMLR 70, pp. 748–756 (2017).
29. Cheng, W., Hühn, J., Hüllermeier, E.: Decision tree and instance-based learning for label ranking. In: Proceedings of ICML 2009. pp. 161–168 (2009).
30. Cook, W.D., Kress, M., Seiford, L.W.: A general framework for distance-based consensus in ordinal ranking models. European Journal of Operational Research 96(2), 392–397 (1996).
31. Daniel, C., Taylor, J., Nowozin, S.: Learning step size controllers for robust neural network training. In: Proceedings of AAAI 2016. pp. 1519–1525 (2016).
32. Davis, C., Giraud-Carrier, C.: Annotative experts for hyperparameter selection. In: AutoML Workshop at ICML 2018 (2018).
33. De Sa, A., Pinto, W., Oliveira, L.O., Pappa, G.: RECIPE: A grammar-based framework for automatically evolving classification pipelines. In: European Conference on Genetic Programming. pp. 246–261 (2017).
34. Demšar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research 7, 1–30 (2006).
35. Dietterich, T.: Ensemble methods in machine learning. In: International workshop on multiple classifier systems. pp. 1–15 (2000).
36. Dietterich, T., Busquets, D., Lopez de Mantaras, R., Sierra, C.: Action Refinement in Reinforcement Learning by Probability Smoothing. In: 19th International Conference on Machine Learning. pp. 107–114 (2002).
37. Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.: DeCAF: A deep convolutional activation feature for generic visual recognition. In: Proceedings of ICML 2014. pp. 647–655 (2014).
38. Drori, I., Krishnamurthy, Y., Rampin, R., de Paula Lourenco, R., Ono, J.P., Cho, K., Silva, C., Freire, J.: AlphaD3M: Machine learning pipeline synthesis. In: AutoML Workshop at ICML (2018).

39. Duan, Y., Schulman, J., Chen, X., Bartlett, P.L., Sutskever, I., Abbeel, P.: RL2: Fast reinforcement learning via slow reinforcement learning. arXiv preprint arXiv:1611.02779 (2016).
40. Eggensperger, K., Lindauer, M., Hoos, H., Hutter, F., Leyton-Brown, K.: Efficient Benchmarking of Algorithm Configuration Procedures via Model-Based Surrogates. Machine Learning 107, 15–41 (2018).
41. Evgeniou, T., Micchelli, C., Pontil, M.: Learning Multiple Tasks with Kernel Methods. Journal of Machine Learning Research 6, 615–637 (2005).
42. Evgeniou, T., Pontil, M.: Regularized multi-task learning. In: Tenth Conference on Knowledge Discovery and Data Mining (2004).
43. Fei-Fei, L.: Knowledge transfer in learning to recognize visual objects classes. In: International Conference on Development and Learning. Art. 51 (2006).
44. Fei-Fei, L., Fergus, R., Perona, P.: One-shot learning of object categories. Pattern analysis and machine intelligence 28(4), 594–611 (2006).
45. Feurer, M., Letham, B., Bakshy, E.: Scalable meta-learning for Bayesian optimization. arXiv 1802.02219 (2018).
46. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Advances in Neural Information Processing Systems 28. pp. 2944–2952 (2015).
47. Feurer, M., Letham, B., Bakshy, E.: Scalable meta-learning for Bayesian optimization using ranking-weighted gaussian process ensembles. In: AutoML Workshop at ICML 2018 (2018).
48. Feurer, M., Springenberg, J.T., Hutter, F.: Using meta-learning to initialize Bayesian optimization of hyperparameters. In: International Conference on Metalearning and Algorithm Selection. pp. 3–10 (2014).
49. Filchenkov, A., Pendryak, A.: Dataset metafeature description for recommending feature selection. In: Proceedings of AINL-ISMW FRUCT 2015. pp. 11–18 (2015).
50. Fink, M.: Object classification from a single example utilizing class relevance metrics. In: Advances in Neural information processing systems, NeurIPS 2005. pp. 449–456 (2005).
51. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of ICML 2017. pp. 1126–1135 (2017).
52. Finn, C., Levine, S.: Meta-learning and universality: Deep representations and Gradient Descent can Approximate any Learning Algorithm. In: Proceedings of ICLR 2018 (2018).
53. Fürnkranz, J., Petrak, J.: An evaluation of landmarking variants. ECML/PKDD 2001 Work- shop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning pp. 57–68 (2001).
54. Fusi, N., Sheth, R., Elibol, H.M.: Probabilistic matrix factorization for automated machine learning. In: Advances in Neural information processing systems, NeurIPS 2018, pp. 3352– 3361 (2018).
55. Gil, Y., Yao, K.T., Ratnakar, V., Garijo, D., Ver Steeg, G., Szekely, P., Brekelmans, R., Kejriwal, M., Luo, F., Huang, I.H.: P4ML: A phased performance-based pipeline planner for automated machine learning. In: AutoML Workshop at ICML 2018 (2018).
56. Giraud-Carrier, C.: Metalearning tutorial. In: Tutorial at the International Conference on Machine Learning and Applications. pp. 1–45 (2008).

57. Giraud-Carrier, C., Provost, F.: Toward a justification of meta-learning: Is the no free lunch theorem a show-stopper. In: Proceedings of the ICML-2005 Workshop on Meta-learning. pp. 12–19 (2005).
58. Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., Sculley, D.: Google vizier: A service for black-box optimization. In: Proceedings of ICDM 2017. pp. 1487–1495 (2017).
59. Gomes, T.A., Prudêncio, R.B., Soares, C., Rossi, A.L., Carvalho, A.: Combining metalearning and search techniques to select parameters for support vector machines. Neurocomputing 75(1), 3–13 (2012).
60. Graves, A., Wayne, G., Danihelka, I.: Neural turing machines. arXiv preprint arXiv:1410.5401 (2014).
61. Guerra, S.B., Prudêncio, R.B., Ludermir, T.B.: Predicting the performance of learning algorithms using support vector machines as metaregressors. In: Proceedings of ICANN. pp. 523–532 (2008).
62. Hengst, B.: Discovering Hierarchy in Reinforcement Learning with HEXQ. In: International Conference on Machine Learning. pp. 243–250 (2002).
63. Hilario, M., Kalousis, A.: Fusion of meta-knowledge and meta-data for case-based model selection. Lecture Notes in Computer Science 2168, 180–191 (2001).
64. Ho, T.K., Basu, M.: Complexity measures of supervised classification problems. Pattern Analysis and Machine Intelligence. 24(3), 289–300 (2002).
65. Hochreiter, S., Younger, A., Conwell, P.: Learning to learn using gradient descent. In: Lecture Notes on Computer Science, 2130. pp. 87–94 (2001).
66. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735– 1780 (1997)
67. Hutter, F., Hoos, H., Leyton-Brown, K.: An Efficient Approach for Assessing Hyperparameter Importance. In: Proceedings of ICML (2014).
68. Hutter, F., Xu, L., Hoos, H., Leyton-Brown, K.: Algorithm runtime prediction: Methods & evaluation. Artificial Intelligence 206, 79–111 (2014).
69. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global Optimization 13(4), 455–492 (1998).
70. Kalousis, A.: Algorithm Selection via Meta-Learning. Ph.D. thesis, University of Geneva, Department of Computer Science (2002).
71. Kalousis, A., Hilario, M.: Representational issues in meta-learning. Proceedings of ICML 2003 pp. 313–320 (2003).
72. Kalousis, A., Hilario, M.: Model selection via meta-learning: a comparative study. International Journal on Artificial Intelligence Tools 10(4), 525–554 (2001).
73. Kendall, M.G.: A new measure of rank correlation. Biometrika 30(1/2), 81–93 (1938).
74. Kietz, J.U., Serban, F., Bernstein, A., Fischer, S.: Designing KDD-workflows via HTN-planning for intelligent discovery assistance. In: 5th Planning to Learn Workshop at ECAI 2012 (2012).
75. Kim, J., Kim, S., Choi, S.: Learning to warm-start Bayesian hyperparameter optimization. arXiv preprint arXiv:1710.06219 (2017).

76. Köpf, C., Iglezakis, I.: Combination of task description strategies and case base properties for meta-learning. ECML/PKDD Workshop on Integration and Collaboration Aspects of Data Mining pp. 65–76 (2002).
77. Köpf, C., Taylor, C., Keller, J.: Meta-analysis: From data characterization for meta-learning to meta-regression. In: PKDD Workshop on Data Mining, Decision Support, Meta-Learning and ILP. pp. 15–26 (2000).
78. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012).
79. Kuba, P., Brazdil, P., Soares, C., Woznica, A.: Exploiting sampling and meta-learning for parameter setting support vector machines. In: Proceedings of IBERAMIA 2002. pp. 217– 225 (2002).
80. Kullback, S., Leibler, R.A.: On information and sufficiency. The annals of mathematical statistics 22(1), 79–86 (1951).
81. Lacoste, A., Marchand, M., Laviolette, F., Larochelle, H.: Agnostic Bayesian learning of ensembles. In: Proceedings of ICML. pp. 611–619 (2014).
82. Lake, B.M., Ullman, T.D., Tenenbaum, J.B., Gershman, S.J.: Building machines that learn and think like people. Behavior and Brain Science 40 (2017).
83. Leite, R., Brazdil, P.: Predicting relative performance of classifiers from samples. Proceedings of ICML pp. 497–504 (2005).
84. Leite, R., Brazdil, P.: An iterative process for building learning curves and predicting relative performance of classifiers. Lecture Notes in Computer Science 4874, 87–98 (2007).
85. Leite, R., Brazdil, P., Vanschoren, J.: Selecting Classification Algorithms with Active Testing. Lecture Notes in Artificial Intelligence 10934, 117–131 (2012).
86. Leite, R., Brazdil, P.: Active testing strategy to predict the best classification algorithm via sampling and metalearning. In: Proceedings of ECAI 2010. pp. 309–314 (2010).
87. Lemke, C., Budka, M., Gabrys, B.: Metalearning: a survey of trends and technologies. Artificial intelligence review 44(1), 117–130 (2015).
88. Ler, D., Koprinska, I., Chawla, S.: Utilizing regression-based landmarkers within a meta-learning framework for algorithm selection. Technical Report 569. University of Sydney pp. 44–51 (2005).
89. Li, K., Malik, J.: Learning to optimize. In: Proceedings of ICLR 2017 (2017).
90. Li, K., Malik, J.: Learning to optimize neural nets. arXiv preprint arXiv: 1703.00441 (2017).
91. Lin, S.: Rank aggregation methods. WIREs Computational Statistics 2, 555–570 (2010).
92. Lindner, G., Studer, R.: AST: Support for algorithm selection with a CBR approach. In: ICML Workshop on Recent Advances in Meta-Learning and Future Work. pp. 38–47. J. Stefan Institute (1999).
93. Lorena, A.C., Maciel, A.I., de Miranda, P.B.C., Costa, I.G., Prudêncio, R.B.C.: Data complexity meta-features for regression problems. Machine Learning 107(1), 209–246 (2018).
94. Luo, G.: A review of automatic selection methods for machine learning algorithms and hyper- parameter values. Network Modeling Analysis in Health Informatics and Bioinformatics 5(1), 18 (2016).

95. Mantovani, R.G., Horváth, T., Cerri, R., Vanschoren, J., de Carvalho, A.C.: Hyper-parameter tuning of a decision tree induction algorithm. In: Brazilian Conference on Intelligent Systems. pp. 37–42 (2016).
96. Mantovani, R.G., Rossi, A.L., Vanschoren, J., Bischl, B., Carvalho, A.C.: To tune or not to tune: recommending when to adjust SVM hyper-parameters via meta-learning. In: Proceedings of IJCNN. pp. 1–8 (2015).
97. Mantovani, R.G., Rossi, A.L., Vanschoren, J., Carvalho, A.C.: Meta-learning recommendation of default hyper-parameter values for SVMs in classifications tasks. In: ECML PKDD Workshop on Meta-Learning and Algorithm Selection (2015).
98. Mantovani, R.: Use of meta-learning for hyperparameter tuning of classification problems. Ph.D. thesis, University of Sao Carlos, Brazil (2018).
99. Michie, D., Spiegelhalter, D.J., Taylor, C.C., Campbell, J.: Machine Learning, Neural and Statistical Classification. Ellis Horwood (1994).
100. Miranda, P., Prudêncio, R.: Active testing for SVM parameter selection. In: Proceedings of IJCNN. pp. 1–8 (2013).
101. Mishra, N., Rohaninejad, M., Chen, X., Abbeel, P.: A simple neural attentive meta-learner. In: Proceedings of ICLR (2018).
102. Misir, M., Sebag, M.: Algorithm Selection as a Collaborative Filtering Problem. Research report, INRIA (2013).
103. Mısır, M., Sebag, M.: Alors: An algorithm recommender system. Artificial Intelligence 244, 291–314 (2017).
104. Nadaraya, E.A.: On estimating regression. Theory of Probability & Its Applications 9(1), 141–142 (1964).
105. Nguyen, P., Hilario, M., Kalousis, A.: Using meta-mining to support data mining workflow planning and optimization. Journal of Artificial Intelligence Research 51, 605–644 (2014).
106. Nichol, A., Achiam, J., Schulman, J.: On first-order meta-learning algorithms. arXiv 1803.02999v2 (2018).
107. Niculescu-Mizil, A., Caruana, R.: Learning the Structure of Related Tasks. In: Proceedings of NIPS Workshop on Inductive Transfer (2005).
108. Nisioti, E., Chatzidimitriou, K., Symeonidis, A.: Predicting hyperparameters from meta-features in binary classification problems. In: AutoML Workshop at ICML (2018).
109. Olier, I., Sadawi, N., Bickerton, G., Vanschoren, J., Grosan, C., Soldatova, L., King, R.: Meta- QSAR: learning how to learn QSARs. Machine Learning 107, 285–311 (2018).
110. Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of GECCO. pp. 485–492 (2016).
111. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on knowledge and data engineering 22(10), 1345–1359 (2010).
112. Pang, K., Dong, M., Wu, Y., Hospedales, T.: Meta-learning transferable active learning policies by deep reinforcement learning. In: AutoML Workshop at ICML (2018).
113. Peng, Y., Flach, P., Soares, C., Brazdil, P.: Improved dataset characterisation for meta-learning. Lecture Notes in Computer Science 2534, 141–152 (2002).

114. Perrone, V., Jenatton, R., Seeger, M., Archambeau, C.: Multiple adaptive Bayesian linear regression for scalable Bayesian optimization with warm start. In: Advances in Neural information processing systems, NeurIPS 2018 (2018).
115. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.G.: Meta-learning by landmarking various learning algorithms. In: 17th International Conference on Machine Learning (ICML). pp. 743–750 (2000).
116. Pinto, F., Cerqueira, V., Soares, C., Mendes-Moreira, J.: autoBagging: Learning to rank bagging workflows with metalearning. arXiv 1706.09367 (2017).
117. Pinto, F., Soares, C., Mendes-Moreira, J.: Towards automatic generation of metafeatures. In: Proceedings of PAKDD. pp. 215–226 (2016).
118. Post, M.J., van der Putten, P., van Rijn, J.N.: Does Feature Selection Improve Classification? A Large Scale Experiment in OpenML. In: Advances in Intelligent Data Analysis XV. pp. 158–170 (2016).
119. Priya, R., De Souza, B.F., Rossi, A., Carvalho, A.: Using genetic algorithms to improve prediction of execution times of ML tasks. In: Lecture Notes in Computer Science. vol. 7208, pp. 196–207 (2012).
120. Probst, P., Bischl, B., Boulesteix, A.L.: Tunability: Importance of hyperparameters of machine learning algorithms. ArXiv 1802.09596 (2018).
121. Prudêncio, R., Ludermir, T.: Meta-learning approaches to selecting time series models. Neurocomputing 61, 121–137 (2004)
122. Raina, R., Ng, A.Y., Koller, D.: Transfer Learning by Constructing Informative Priors. In: Proceedings of ICML (2006).
123. Rakotoarison, H., Sebag, M.: AutoML with Monte Carlo Tree Search. In: ICML Workshop on AutoML 2018 (2018).
124. Ramachandran, A., Gupta, S., Rana, S., Venkatesh, S.: Information-theoretic transfer learning framework for Bayesian optimisation. In: Proceedings of ECMLPKDD (2018).
125. Ramachandran, A., Gupta, S., Rana, S., Venkatesh, S.: Selecting optimal source for transfer learning in Bayesian optimisation. In: Proceedings of PRICAI. pp. 42–56 (2018).
126. Ravi, S., Larochelle, H.: Optimization as a model for few-shot learning. In: Proceedings of ICLR (2017)
127. Reed, S., Chen, Y., Paine, T., Oord, A.v.d., Eslami, S., Rezende, D., Vinyals, O., de Freitas, N.: Few-shot autoregressive density estimation: Towards learning to learn distributions. In: Proceedings of ICLR 2018 (2018).
128. Reif, M., Shafait, F., Dengel, A.: Prediction of classifier training time including parameter optimization. In: Proceedings of GfKI 2011. pp. 260–271 (2011).
129. Reif, M., Shafait, F., Dengel, A.: Meta-learning for evolutionary parameter optimization of classifiers. Machine learning 87(3), 357–380 (2012).
130. Reif, M., Shafait, F., Goldstein, M., Breuel, T., Dengel, A.: Automatic classifier selection for non-experts. Pattern Analysis and Applications 17(1), 83–96 (2014).
131. Ren, M., Triantafillou, E., Ravi, S., Snell, J., Swersky, K., Tenenbaum, J.B., Larochelle, H., Zemel, R.S.: Meta-learning for semi-supervised few-shot classification. In: Proceedings of ICLR 2018 (2018).

132. Rendle, S.: Factorization machines. In: Proceedings of ICDM 2015. pp. 995–1000 (2010).
133. Ridd, P., Giraud-Carrier, C.: Using metalearning to predict when parameter optimization is likely to improve classification accuracy. In: ECAI Workshop on Meta-learning and Algorithm Selection. pp. 18–23 (2014).
134. van Rijn, J., Abdulrahman, S., Brazdil, P., Vanschoren, J.: Fast Algorithm Selection Using Learning Curves. In: Proceedings of IDA (2015).
135. van Rijn, J., Holmes, G., Pfahringer, B., Vanschoren, J.: The Online Performance Estimation Framework. Heterogeneous Ensemble Learning for Data Streams. Machine Learning 107, 149–176 (2018).
136. van Rijn, J.N., Hutter, F.: Hyperparameter importance across datasets. In: Proceedings of KDD. pp. 2367–2376 (2018).
137. van Rijn, J.N., Holmes, G., Pfahringer, B., Vanschoren, J.: Algorithm selection on data streams. In: Discovery Science. pp. 325–336 (2014).
138. Rivolli, A., Garcia, L., Soares, C., Vanschoren, J., de Carvalho, A.: Towards reproducible empirical research in meta-learning. arXiv preprint 1808.10406 (2018).
139. Robbins, H.: Some aspects of the sequential design of experiments. In: Herbert Robbins Selected Papers, pp. 169–177. Springer (1985).
140. Rosenstein, M.T., Marx, Z., Kaelbling, L.P.: To Transfer or Not To Transfer. In: NIPS Workshop on transfer learning (2005).
141. Rousseeuw, P.J., Hubert, M.: Robust statistics for outlier detection. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1(1), 73–79 (2011).
142. Runarsson, T.P., Jonsson, M.T.: Evolution and design of distributed learning rules. In: IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. pp. 59–63 (2000).
143. Salama, M.A., Hassanien, A.E., Revett, K.: Employment of neural network and rough set in meta-learning. Memetic Computing 5(3), 165–177 (2013).
144. Sanders, S., Giraud-Carrier, C.: Informing the use of hyperparameter optimization through metalearning. In: Proceedings of ICDM 2017. pp. 1051–1056 (2017).
145. Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., Lillicrap, T.: Meta-learning with memory-augmented neural networks. In: International conference on machine learning. pp. 1842–1850 (2016).
146. Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., Lillicrap, T.: One-shot learning with memory-augmented neural networks. arXiv preprint arXiv:1605.06065 (2016).
147. dos Santos, P., Ludermir, T., Prudêncio, R.: Selection of time series forecasting models based on performance information. 4th International Conference on Hybrid Intelligent Systems pp. 366–371 (2004).
148. Schilling, N., Wistuba, M., Drumond, L., Schmidt-Thieme, L.: Hyperparameter optimization with factorized multilayer perceptrons. In: Proceedings of ECML PKDD. pp. 87–103 (2015).
149. Schmidhuber, J.: Learning to control fast-weight memories: An alternative to dynamic recurrent networks. Neural Computing 4(1), 131–139 (1992).
150. Schmidhuber, J.: A neural network that embeds its own meta-levels. In: Proceedings of ICNN. pp. 407–412 (1993).
151. Schmidhuber, J., Zhao, J., Wiering, M.: Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. Machine Learning 28(1), 105–130 (1997).

152. Schoenfeld, B., Giraud-Carrier, C., Poggeman, M., Christensen, J., Seppi, K.: Feature selection for high-dimensional data: A fast correlation-based filter solution. In: AutoML Workshop at ICML (2018).
153. Serban, F., Vanschoren, J., Kietz, J., Bernstein, A.: A survey of intelligent assistants for data analysis. ACM Computing Surveys 45(3), Art.31 (2013).
154. Sharif Razavian, A., Azizpour, H., Sullivan, J., Carlsson, S.: Cnn features off-the-shelf: an astounding baseline for recognition. In: Proceedings of CVPR 2014. pp. 806–813 (2014).
155. Sharkey, N.E., Sharkey, A.J.C.: Adaptive Generalization. Artificial Intelligence Review 7, 313–328 (1993).
156. Smith-Miles, K.A.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Computing Surveys 41(1), 1–25 (2009).
157. Snell, J., Swersky, K., Zemel, R.: Prototypical networks for few-shot learning. In: Neural Information Processing Systems. pp. 4077–4087 (2017).
158. Soares, C., Brazdil, P., Kuba, P.: A meta-learning method to select the kernel width in support vector regression. Machine Learning 54, 195–209 (2004).
159. Soares, C., Ludermir, T., Carvalho, F.D.: An analysis of meta-learning techniques for ranking clustering algorithms applied to artificial data. Lecture Notes in Computer Science 5768, 131–140 (2009).
160. Soares, C., Petrak, J., Brazdil, P.: Sampling based relative landmarks: Systematically testdriving algorithms before choosing. Lecture Notes in Computer Science 3201, 250–261 (2001).
161. Springenberg, J., Klein, A., Falkner, S., Hutter, F.: Bayesian optimization with robust Bayesian neural networks. In: Advances in Neural Information Processing Systems (2016).
162. Stern, D.H., Samulowitz, H., Herbrich, R., Graepel, T., Pulina, L., Tacchella, A.: Collaborative expert portfolio management. In: Proceedings of AAAI. pp. 179–184 (2010).
163. Strang, B., van der Putten, P., van Rijn, J.N., Hutter, F.: Don't Rule Out Simple Models Prematurely. In: Advances in Intelligent Data Analysis (2018).
164. Sun, Q., Pfahringer, B., Mayo, M.: Towards a Framework for Designing Full Model Selection and Optimization Systems. In: International Workshop on Multiple Classifier Systems. pp. 259–270 (2013)
165. Sun, Q., Pfahringer, B.: Pairwise meta-rules for better meta-learning-based algorithm ranking. Machine Learning 93(1), 141–161 (2013).
166. Swersky, K., Snoek, J., Adams, R.P.: Multi-task Bayesian optimization. In: Advances in neural information processing systems. pp. 2004–2012 (2013).
167. Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika 25(3/4), 285–294 (1933).
168. Thrun, S.: Lifelong Learning Algorithms. In: Learning to Learn, chap. 8, pp. 181–209. Kluwer Academic Publishers, MA (1998).
169. Thrun, S., Mitchell, T.: Learning One More Thing. In: Proceedings of IJCAI. pp. 1217–1223 (1995).
170. Thrun, S., Pratt, L.: Learning to Learn: Introduction and Overview. In: Learning to Learn, pp. 3–17. Kluwer (1998).
171. Todorovski, L., Blockeel, H., Džeroski, S.: Ranking with predictive clustering trees. Lecture Notes in Artificial Intelligence 2430, 444–455 (2002).

172. Todorovski, L., Brazdil, P., Soares, C.: Report on the experiments with feature selection in meta-level learning. PKDD 2000 Workshop on Data mining, Decision support, Meta-learning and ILP pp. 27–39 (2000).
173. Todorovski, L., Dzeroski, S.: Experiments in meta-level learning with ILP. Lecture Notes in Computer Science 1704, 98–106 (1999).
174. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: networked science in machine learning. ACM SIGKDD Explorations Newsletter 15(2), 49–60 (2014).
175. Vanschoren, J.: Understanding Machine Learning Performance with Experiment Databases. Ph.D. thesis, Leuven Univeristy (2010).
176. Vanschoren, J.: Meta-learning: A survey. arXiv:1810.03548 (2018).
177. Vanschoren, J., Blockeel, H., Pfahringer, B., Holmes, G.: Experiment databases. Machine Learning 87(2), 127–158 (2012).
178. Vartak, M., Thiagarajan, A., Miranda, C., Bratman, J., Larochelle, H.: A meta-learning perspective on cold-start recommendations for items. In: Advances in Neural Information Processing Systems. pp. 6904–6914 (2017).
179. Vilalta, R.: Understanding accuracy performance through concept characterization and algorithm analysis. ICML Workshop on Recent Advances in Meta-Learning and Future Work (1999).
180. Vilalta, R., Drissi, Y.: A characterization of difficult problems in classification. Proceedings of ICMLA (2002).
181. Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al.: Matching networks for one shot learning. In: Advances in Neural Information Processing Systems. pp. 3630–3638 (2016).
182. Weerts, H., Meuller, M., Vanschoren, J.: Importance of tuning hyperparameters of machine learning algorithms. Technical report, TU Eindhoven (2018).
183. Weerts, H., Meuller, M., Vanschoren, J.: Importance of tuning hyperparameters of machine learning algorithms. Tech. rep., TU Eindhoven (2018).
184. Wever, M., Mohr, F., Hüllermeier, E.: Ml-plan for unlimited-length machine learning pipelines. In: AutoML Workshop at ICML 2018 (2018).
185. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Hyperparameter search space pruning, a new component for sequential model-based hyperparameter optimization. In: ECML PKDD 2015. pp. 104–119 (2015).
186. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Learning hyperparameter optimization initializations. In: 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA). pp. 1–10 (2015).
187. Wolpert, D., Macready, W.: No free lunch theorems for search. Technical Report SFI-TR-95- 02-010, The Santa Fe Institute (1996).
188. Yang, C., Akimoto, Y., Kim, D., Udell, M.: OBOE: Collaborative filtering for AutoML initialization. In: NeurIPS 2018 Workshop on Metalearning (2018).
189. Yang, C., Akimoto, Y., Kim, D., Udell, M.: Oboe: Collaborative filtering for AutoML initialization. arXiv preprint arXiv:1808.03233 (2018).
190. Yogatama, D., Mann, G.: Efficient transfer learning method for automatic hyperparameter tuning. In: AI and Statistics. pp. 1077–1085 (2014).
191. Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? In: Advances in neural information processing systems. pp. 3320–3328 (2014).

Глава 3

Поиск нейронной архитектуры

Томас Эльскен (✉), Центр искусственного интеллекта Bosch, Robert Bosch GmbH, Реннинген, Баден-Вюртемберг, Германия; факультет компьютерных наук, Университет Фрайбурга, Фрайбург, Баден-Вюртемберг, Германия, e-mail: elsken@informatik.uni-freiburg.de; thomas.elsken@de.bosch.com

Ян Хендрик Метцен, Центр искусственного интеллекта Bosch, Robert Bosch GmbH, Реннинген, Баден-Вюртемберг, Германия

Франк Хуттер, факультет компьютерных наук, Университет Фрайбурга, Фрайбург, Германия

За последние годы глубокое обучение позволило добиться значительного прогресса в решении различных задач, таких как распознавание изображений, распознавание речи и машинный перевод. Одним из важнейших аспектов этого прогресса являются новые нейронные архитектуры. В настоящее время прикладные архитектуры в основном разрабатываются вручную квалифицированными экспертами, что является трудоемким и подверженным ошибкам процессом. В связи с этим растет интерес к автоматизированным методам поиска нейронных архитектур. Мы представляем обзор существующих работ в этой области исследований и классифицируем их по трем параметрам: пространству поиска, стратегии поиска и стратегии оценки производительности.

3.1. Введение

Успех глубокого обучения в задачах восприятия во многом обусловлен автоматизацией процесса конструирования признаков: иерархические экстракторы признаков обучаются на основе данных, а не разрабатываются вручную. Однако этот успех сопровождается ростом спроса на *конструирование архитектуры* (architecture engineering), когда все более сложные ней-

ронные архитектуры приходится проектировать вручную. Таким образом, *поиск нейронной архитектуры* (neural architecture search, NAS), представляющий собой процесс автоматизации проектирования архитектуры, является логическим следующим шагом в автоматизации машинного обучения. NAS можно рассматривать как область AutoML, которая значительно пересекается с оптимизацией гиперпараметров и метаобучением (описаны в главах 1 и 2 этой книги соответственно). Мы классифицируем методы NAS по трем параметрам: пространству поиска, стратегии поиска и стратегии оценки производительности:

- *пространство поиска* – определяет, какие архитектуры могут быть представлены в принципе. Наличие предварительных знаний о свойствах, хорошо подходящих для выполнения задачи, может уменьшить размер пространства поиска и упростить поиск. Однако это также привносит человеческую предвзятость, которая может помешать найти новые архитектурные компоненты, выходящие за рамки текущих знаний человека;
- *стратегия поиска* – подробно описывает, как исследовать пространство поиска. Она представляет собой классический компромисс между исследованием и использованием, поскольку, с одной стороны, желательно быстро найти хорошо работающие архитектуры, а с другой стороны, следует избегать преждевременной конвергенции в область субоптимальных архитектур;
- *стратегия оценки производительности* – целью NAS обычно является поиск архитектур, которые обладают высокой предсказательной способностью на невидимых данных. Оценка производительности относится к процессу оценки этой производительности: самым простым вариантом является стандартное обучение и проверка архитектуры на данных, но это, к сожалению, требует больших вычислительных затрат и ограничивает количество архитектур, которые могут быть исследованы. Поэтому значительная часть последних исследований направлена на разработку методов, снижающих стоимость оценки производительности.

Упомянутые параметры схематически представлены на рис. 3.1. Глава также выстроена в соответствии с этими тремя критериями: мы начинаем с обсуждения пространств поиска в разделе 3.2, рассматриваем стратегии поиска в разделе 3.3 и описываем подходы к оценке производительности в разделе 3.4. В заключение в разделе 3.5 мы рассуждаем о перспективных направлениях исследований. Глава основана на обзорной статье [23].

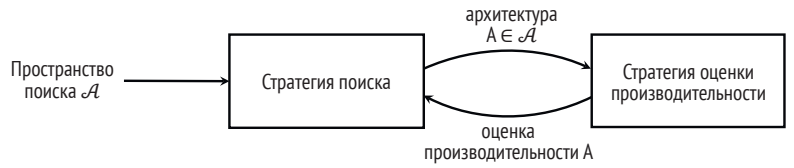


Рис. 3.1 ❖ Обобщенная иллюстрация методов поиска нейронных архитектур. Стратегия поиска выбирает архитектуру A из заранее определенного пространства поиска. Архитектура передается стратегии оценки производительности

3.2. Пространство поиска

От пространства поиска зависит, какие нейронные архитектуры в принципе может обнаружить подход NAS. Далее мы обсудим распространенные пространства поиска из последних работ в области AutoML.

Относительно простым пространством поиска является пространство *цепочечных нейронных сетей* (chain-structured neural network), показанных на рис. 3.2 (слева). Архитектура цепочечной нейронной сети A может быть записана как последовательность из n слоев, где i-й слой L_i получает вход от слоя $i - 1$, а его выход служит входом для слоя $i + 1$, т. е. $A = L_n \circ \ldots L_1 \circ L_0$. Пространство поиска определяется такими параметрами, как: 1) максимальное число слоев n (возможно, неограниченное); 2) тип операции, которую может выполнять каждый слой, например объединение, свертка или более продвинутые типы слоев, такие как свертки с разделением по глубине [13] или расширенные свертки [68]; и 3) гиперпараметры, связанные с операцией, например число фильтров, размер ядра и шаг для сверточного слоя [4, 10, 59] или просто число блоков для полносвязных сетей [41]. Заметим, что параметры из (3) зависят от параметра (2), поэтому параметризация пространства поиска не имеет фиксированной длины, а скорее описывает условное пространство. Последние работы в области NAS [9, 11, 21, 22, 49, 75] включают в себя элементы схем, позаимствованные из архитектур, созданных вручную, такие как обходные соединения, которые позволяют строить сложные *многократно разветвленные сети* (multi-branch networks), как показано на

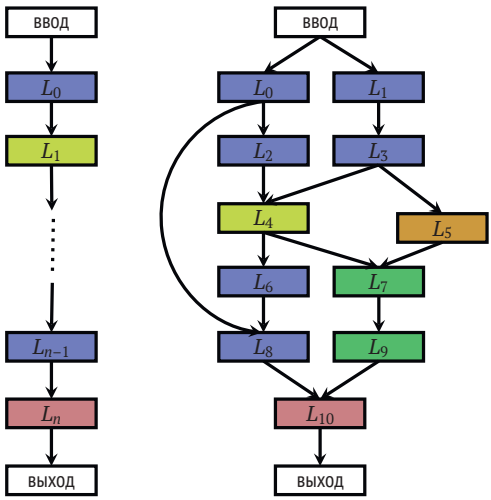


Рис. 3.2 ❖ Иллюстрация различных архитектурных пространств. Каждый узел графа соответствует слою в нейронной сети, например свертке или пулингу. Различные типы слоев обозначены разными цветами. Стрелка от слоя L_i к слою L_j означает, что L_j получает выход L_i в качестве входа. Слева: элемент пространства цепочечных структур. Справа: элемент более сложного пространства поиска с дополнительными типами слоев и множеством ветвей и обходных путей

рис. 3.2 (справа). В этом случае вход слоя i может быть формально описан как функция $g_i(L_{i-1}^{out}, \ldots, L_0^{\text{out}})$, объединяющая выходы предыдущих слоев. Использование такой функции дает значительно больше степеней свободы. Частными случаями таких разветвленных архитектур являются: 1) сети с цепной структурой (когда задано $g_i(L_{i-1}^{out}, \ldots, L_0^{\text{out}}) = L_{i-1}^{out}$), 2) остаточные сети [28], где выходы предыдущих слоев суммируются ($g_i(L_{i-1}^{out}, \ldots, L_0^{\text{out}}) = L_{i-1}^{out} + L_j^{\text{out}}, j < i$), и 3) плотные сети [29], где выходы предыдущих слоев конкатенируются ($g_i(L_{i-1}^{out}, \ldots, L_0^{\text{out}}) = concat(L_{i-1}^{out}, \ldots, L_0^{\text{out}})$).

Вдохновившись созданными вручную архитектурами, состоящими из повторяющихся *мотивов* (motif, по аналогии с повторяющимися участками ДНК) [28, 29, 62], Зоф и др. [75], а также Чжун и др. [71] предлагают искать такие мотивы, называемые в их работах ячейками или блоками соответственно, а не полные архитектуры. Зоф и др. [75] оптимизируют два различных типа ячеек: обычную ячейку, которая сохраняет размерность входных данных, и редукционную ячейку, которая уменьшает пространственную размерность. Окончательная архитектура строится путем укладки этих ячеек в заранее определенном порядке, как показано на рис. 3.3. Это пространство поиска имеет два основных преимущества по сравнению с рассмотренными выше.

1. Размер пространства поиска резко сокращается, поскольку ячейки могут быть сравнительно небольшими. Например, Зоф и др. [75] отмечают семикратное ускорение по сравнению с предыдущей работой [74] при достижении лучшей производительности.
2. Ячейки можно легче переносить на другие наборы данных, адаптируя количество ячеек, используемых в модели. Действительно, Зоф и др. [75] перенесли ячейки, оптимизированные на CIFAR-10, в ImageNet и достигли самой высокой производительности.

Неудивительно, что это ячеечное пространство поиска также успешно использовалось во многих последующих работах [11, 22, 37, 39, 46, 49, 72]. Однако при использовании такого пространства поиска возникает новый вопрос, а именно как выбрать *метаархитектуру*: сколько ячеек следует использовать и как они должны быть соединены для построения реальной модели? Например, Зоф и др. [75] строят последовательную модель из ячеек, в которой каждая ячейка получает на вход выходы двух предыдущих ячеек, а Цай и др. [11] используют высокоуровневую структуру уже известных вручную разработанных архитектур, таких как DenseNet [29], и используют свои ячейки в этих моделях. В принципе, ячейки можно комбинировать произвольно, например в рамках описанного выше пространства цепочечных сетей, просто заменяя слои ячейками. В идеале метаархитектура должна оптимизироваться автоматически как часть NAS; в противном случае можно легко увлечься проектированием метаархитектуры, и поиск ячейки станет чрезмерно простым, если большая часть сложности уже заложена в метаархитектуру.

Одним из шагов в направлении оптимизации метаархитектур является иерархическое пространство поиска, представленное Лю и др. [38], которое состоит из нескольких уровней мотивов. Первый уровень состоит из набора примитивных операций, второй – из различных мотивов, которые соединяют примитивные операции через прямой ациклический граф, тре-

тий – из мотивов, которые описывают, как соединяются мотивы второго уровня, и т. д. Пространство поиска на основе ячеек можно рассматривать как частный случай этого иерархического пространства поиска, где количество уровней равно трем, мотивы второго уровня соответствуют ячейкам, и третий уровень – это жестко закодированная метаархитектура.

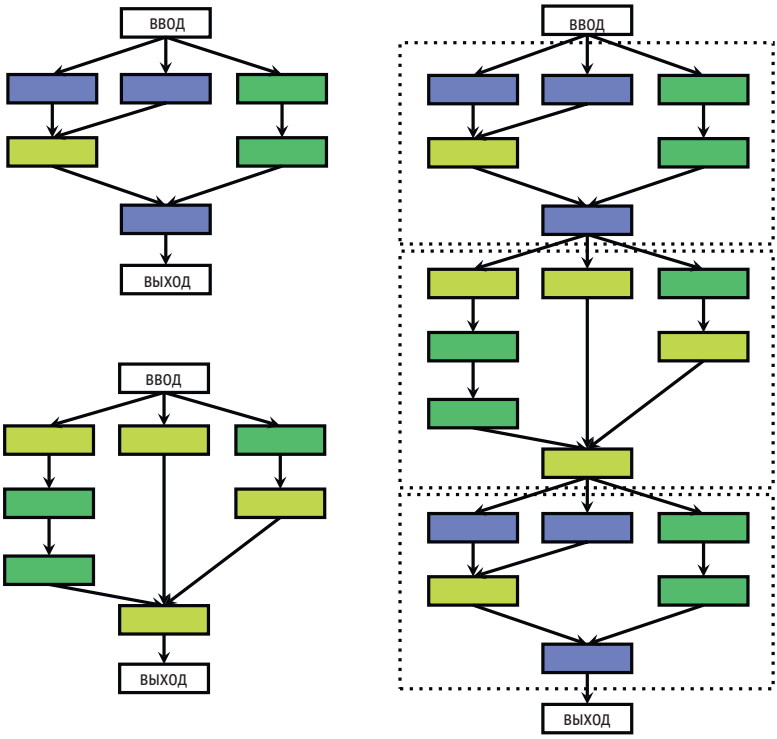


Рис. 3.3 ❖ Иллюстрация ячеечного пространства поиска. Слева: две различные ячейки, например обычная (вверху) и редукционная ячейка (внизу) [75]. Справа: архитектура, построенная путем последовательного укладывания ячеек. Обратите внимание, что ячейки можно комбинировать и более сложным образом, например в пространствах разветвленных сетей, просто заменяя слои ячейками

Выбор пространства поиска в значительной степени определяет сложность проблемы оптимизации: даже в случае пространства поиска на основе одной ячейки с фиксированной метаархитектурой, проблема оптимизации остается 1) не непрерывной и 2) относительно высокоразмерной (поскольку более сложные модели имеют тенденцию работать лучше, что приводит к большему количеству вариантов схемы). Отметим, что архитектуры во многих пространствах поиска могут быть записаны в виде векторов фиксированной длины; например, пространство поиска для каждой из двух ячеек в работе Зофа и др. [75] может быть записано как 40-мерное пространство поиска с категориальными измерениями, каждое из которых определяет выбор неболь-

шого количества различных строительных блоков и входов. Аналогично неограниченные пространства поиска могут быть ограничены максимальной глубиной, что приводит к появлению пространств поиска фиксированного размера с (потенциально многими) условными измерениями.

В следующем разделе мы обсудим стратегии поиска, которые хорошо подходят для таких пространств.

3.3. Стратегия поиска

Для исследования пространства нейронных архитектур можно использовать множество различных стратегий поиска, включая случайный поиск, байесовскую оптимизацию, эволюционные методы, обучение с подкреплением (RL) и градиентные методы. Исторически эволюционные алгоритмы уже использовались многими исследователями для эволюционного подбора нейронных архитектур (и часто также их весов) больше десяти лет назад [см., например, 2, 25, 55, 56]. В работе Яо [67] приводится обзор литературы по работам, выполненным ранее 2000 г.

Байесовская оптимизация отметилась несколькими успехами в области NAS в период с 2013 г., что привело к созданию современных архитектур технического зрения [7], лучшей производительности для CIFAR-10 без дополнения набора данных [19] и первым автоматически настроенным нейронным сетям, которые выиграли соревнования с экспертами [41]. Поиск нейросетевых архитектур стал основной темой исследований в сообществе машинного обучения после того, как Зоф и Ле [74] получили конкурентоспособную производительность на эталонах CIFAR-10 и Penn Treebank с помощью стратегии поиска, основанной на обучении с подкреплением. Хотя Зоф и Ле [74] использовали для достижения этого результата огромные вычислительные ресурсы (800 GPU в течение трех-четырех недель), вскоре после их работы было опубликовано множество методов снижения вычислительных затрат и дальнейшего повышения производительности.

Если рассматривать NAS как задачу обучения с подкреплением [4, 71, 74, 75], то генерацию нейронной архитектуры можно рассматривать как действие агента, при этом пространство действий идентично пространству поиска. Вознаграждение агента основано на оценке эффективности обученной архитектуры на незнакомых данных (см. раздел 3.4). Различные методы обучения с подкреплением отличаются тем, как они представляют стратегию агента и как они ее оптимизируют: Зоф и Ле [74] используют стратегию рекуррентной нейронной сети для последовательной выборки строки, которая в свою очередь кодирует нейронную архитектуру. Первоначально они обучали эту сеть с помощью градиентного алгоритма REINFORCE, но в последующей работе вместо него использовали *проксимальную оптимизацию стратегии* (proximal policy optimization, PPO) [75]. Бейкер и др. [4] используют Q-обучение для обучения стратегии, которая последовательно выбирает тип слоя и соответствующие гиперпараметры. В качестве альтернативы можно рассматривать последовательные процессы принятия решений, в которых

стратегия последовательно выбирает действия для создания архитектуры, «состояние» среды содержит сводку действий, выбранных на данный момент, а вознаграждение (как правило, не дисконтированное) получается только после последнего действия. Однако, поскольку во время этого последовательного процесса не происходит никакого взаимодействия с окружающей средой (не наблюдается внешнее состояние и нет промежуточных вознаграждений), мы считаем интуитивно более понятной интерпретацию процесса выборки архитектуры как последовательной генерации одного действия; это упрощает задачу обучения с подкреплением до задачи многорукого бандита без состояния.

Смежный подход предложили Цай и др. [10], которые представляют NAS как последовательный процесс принятия решений: в их варианте состояние – это текущая (частично обученная) архитектура, вознаграждение – это оценка производительности архитектуры, а действие соответствует применению сохраняющих функциональность мутаций, называемых морфизмами сети [12, 63] (см. также раздел 3.4), после чего следует этап обучения сети. Для работы с сетевыми архитектурами переменной длины используется двунаправленная сеть LSTM, кодирующая архитектуру в представление фиксированной длины. На основе этого кодированного представления сети акторов принимают решение о выборе действия. Комбинация этих двух компонентов составляет стратегию, которая полностью обучается с помощью градиентного алгоритма REINFORCE. Отметим, что при таком подходе одно и то же состояние (архитектура) не будет посещаться дважды, поэтому от стратегии требуется сильное обобщение по пространству архитектуры.

Альтернативой использованию обучения с подкреплением являются нейроэволюционные методы, которые используют эволюционные алгоритмы для оптимизации нейронной архитектуры. Первый известный нам нейроэволюционный подход к проектированию нейронных сетей появился почти три десятилетия назад: Миллер и др. [44] используют генетические алгоритмы для предложения архитектур и обратное распространение для оптимизации их весов. С тех пор многие нейроэволюционные подходы [2, 55, 56] используют генетические алгоритмы для оптимизации как нейронной архитектуры, так и ее весов; однако при масштабировании на современные нейронные архитектуры с миллионами весов для задач обучения с учителем методы оптимизации весов на основе SGD в настоящее время превосходят эволюционные[1]. Поэтому более современные нейроэволюционные подходы [22, 38, 43, 49, 50, 59, 66] снова используют градиентные методы для оптимизации весов и исключительно эволюционные алгоритмы для оптимизации самой нейронной архитектуры. Эволюционные алгоритмы развивают популяцию моделей, т. е. набор (возможно, обученных) сетей; на каждом шаге эволюции по крайней мере одна модель из популяции выбирается и служит

[1] В некоторых работах показано, что эволюционная адаптация даже миллионов весов конкурентоспособна по сравнению с оптимизацией на основе градиента, когда доступны только высоковариационные оценки градиента, например, для задач обучения с подкреплением [15, 51, 57]. Тем не менее для задач обучения с учителем оптимизация на основе градиента является наиболее распространенным подходом.

родителем для создания потомков путем применения к ней мутаций. В контексте NAS мутации – это локальные операции, такие как добавление или удаление слоя, изменение гиперпараметров слоя, добавление пропущенных связей, а также изменение гиперпараметров обучения. После обучения потомков оценивается их приспособленность (например, производительность на валидационном множестве), и они добавляются в популяцию.

Нейроэволюционные методы отличаются тем, как они отбирают родителей, обновляют популяции и генерируют потомков. Например, Реал и др. [49, 50] и Лю и др. [38] используют турнирный отбор [27] для выборки родителей, в то время как Элскен и др. [22] выбирают родителей из многоцелевого фронта Парето, используя обратную плотность. В работе Реал и др. [50] удаляют худшую особь из популяции, в то время как Реал и др. [49] считают полезным удаление самой старой особи (что уменьшает жадность алгоритма), а в Лю и др. [38] вообще не удаляют особей. Для создания потомства большинство подходов инициализирует дочерние сети случайным образом, в то время как Элскен и др. [22] используют ламаркистское наследование, т. е. знания (в виде выученных весов) передаются от родительской сети к дочерним с помощью сетевых морфизмов. Реал и др. [50] также позволяют потомку наследовать все параметры своего родителя, на которые не влияет применяемая мутация; хотя такое наследование не является строго сохраняющим функциональность, оно может ускорить обучение по сравнению со случайной инициализацией. Более того, они также позволяют мутировать скорость обучения, что можно рассматривать как способ оптимизации графика скорости обучения в ходе NAS.

Реал и др. [49] провели тематическое исследование, сравнивая обучение с подкреплением (RL), эволюцию и случайный поиск (RS), и пришли к выводу, что RL и эволюция работают одинаково хорошо с точки зрения точности конечного теста, причем эволюция имеет лучшую производительность в любое время и находит меньшие модели. В экспериментах оба подхода постоянно работают лучше, чем RS, но с довольно небольшим отрывом: RS достиг уровня тестовых ошибок около 4 % на CIFAR-10, в то время как RL и эволюция достигли примерно 3.5 % (после «увеличения модели», где глубина и количество фильтров были увеличены; разница на реальном, неувеличенном пространстве поиска составила около 2 %). Разница была еще меньше у Лю и др. [38], которые сообщили об ошибке тестирования 3.9 % на CIFAR-10 и ошибке валидации 21.0 % на ImageNet для RS, по сравнению с 3.75 % и 20.3 % для их метода, основанного на эволюции, соответственно.

Байесовская оптимизация (BO, см., например, [53]) – один из самых популярных методов оптимизации гиперпараметров (см. также главу 1 этой книги), но многие исследователи не применяли его к NAS, поскольку типичные инструментальные средства BO основаны на гауссовых процессах и ориентированы на решение непрерывных задач оптимизации с низкой размерностью. Сверски и др. [60] и Кандасами и др. [31] выводят ядерные функции для пространств поиска архитектуры, чтобы использовать классические методы байесовской оптимизации на основе GP, но пока не достигли современной производительности. В других работах используют древовидные модели (в частности, древовидные оценщики Парзена [8] или случай-

ные леса [30]) для эффективного поиска в очень многомерных условных пространствах и достижения наилучшей производительности для широкого круга задач, совместно оптимизируя как нейронные архитектуры, так и их гиперпараметры [7, 19, 41, 69]. Хотя полное сравнение не проводилось, есть предварительные свидетельства того, что эти подходы также могут превосходить эволюционные алгоритмы [33].

Пространства поиска архитектуры также пытались исследовать иерархическим путем, например в сочетании с эволюцией [38], или путем последовательной оптимизации на основе моделей [37]. Негриньо и Гордон [45] и Вистуба [65] используют древовидную структуру пространства поиска и применяют поиск по дереву Монте-Карло. Элскен и др. [21] предлагают простой, но хорошо работающий алгоритм *восхождения по выпуклой поверхности* (hill climbing), который обнаруживает высококачественные архитектуры путем жадного движения в направлении более эффективных архитектур, не требуя более сложных механизмов поиска.

В отличие от описанных выше методов безградиентной оптимизации Лю и др. [39] предлагают непрерывную релаксацию пространства поиска для оптимизации на основе градиента: вместо того чтобы фиксировать одну операцию o_i (например, свертку или объединение) для выполнения на определенном слое, авторы вычисляют выпуклую комбинацию из набора операций $\{o_1, \ldots, o_m\}$. Более конкретно при заданном входе слоя x, выход слоя y вычисляется как $y = \sum_{i=1}^{m} \lambda_i o_i(x)$, $\lambda_i \geqslant 0$, $\sum_{i=1}^{m} \lambda_i = 1$, где коэффициент свертки λ_i эффективно параметризует архитектуру сети. Затем Лю и др. [39] оптимизируют веса и архитектуру сети путем чередования шагов градиентного спуска на обучающих данных для весов и на проверочных данных для архитектурных параметров, таких как λ. В конечном итоге дискретная архитектура получается путем выбора операции i такой, что $i = \text{argmax}_i \lambda_i$ для каждого слоя. Шин и др. [54] и Ахмед и Торресани [1] также используют градиентную оптимизацию нейронных архитектур, однако они рассматривают только оптимизацию гиперпараметров слоев или схем связности соответственно.

3.4. Стратегия оценки производительности

Стратегии поиска, рассмотренные в предыдущем разделе, направлены на поиск нейронной архитектуры A, которая обеспечивает максимум некоторой меры производительности, например точности модели на незнакомых данных. Чтобы направлять процесс поиска, эти стратегии должны оценить производительность текущей архитектуры A. Самый простой способ сделать это – обучить A на обучающем наборе и оценить ее производительность на проверочных данных. Однако обучение каждой архитектуры для оценки с нуля часто приводит к вычислительным затратам порядка тысяч дней работы GPU [49, 50, 74, 75].

Чтобы уменьшить вычислительную нагрузку, производительность можно оценивать на основе менее точных показателей фактической производительности после полного обучения (также обозначаемых как *прокси-метрики*

или low-fidelity metric). Такие оценки получаются при более коротком времени обучения [69, 75], обучении на подмножестве данных [34], на изображениях с меньшим разрешением [14] или с меньшим количеством фильтров на слой [49, 75]. Хотя эти менее точные приближения снижают вычислительные затраты, они также вносят погрешность в оценку, поскольку производительность обычно недооценивается. Это может не вызывать проблем до тех пор, пока стратегия поиска основывается только на ранжировании различных архитектур и относительное ранжирование остается стабильным. Однако последние результаты показывают, что это относительное ранжирование может резко измениться, если разница между дешевыми приближениями и «полной» оценкой слишком велика [69], что говорит о необходимости постепенного увеличения точности по мере продвижения поиска [24, 35].

Другой возможный способ оценки производительности архитектуры основан на экстраполяции кривой обучения [5, 19, 32, 48, 61]. Домхан и др. [19] предлагают экстраполировать начальные кривые обучения и прекращать те, которые, по прогнозам, будут иметь низкую производительность, чтобы ускорить процесс поиска архитектуры. Бейкер и др. [5], Клейн и др. [32], Равал и Миикkулайнен [48], Сверски и др. [61] также рассматривают возможность использования гиперпараметров архитектуры для предсказания того, какие частичные кривые обучения являются наиболее перспективными. Обучение суррогатной модели для прогнозирования производительности новых архитектур было предложено Лю и др. [37]. Согласно их подходу экстраполяцию кривой обучения не используют, но выполняют прогнозирование производительности на основе свойств архитектуры/ячейки и экстраполируют прогноз на архитектуры/ячейки большего размера, чем наблюдалось во время обучения. Основная проблема прогнозирования производительности нейронных архитектур заключается в том, что для ускорения процесса поиска хорошие прогнозы в относительно большом пространстве поиска должны быть сделаны на основе относительно небольшого количества оценок.

Другой подход к ускорению оценки производительности заключается в инициализации весов новых архитектур на основе весов других архитектур, которые были обучены ранее. Один из способов достижения этой цели, получивший название морфизмов сети [64], позволяет модифицировать архитектуру, оставляя функцию, представленную сетью, неизменной [10, 11, 21, 22]. Это позволяет последовательно увеличивать мощность сетей и сохранять высокую производительность, не требуя обучения с нуля. Продолжение обучения в течение нескольких эпох также может использовать дополнительную емкость, введенную морфизмами сети. Преимуществом этих подходов является то, что они позволяют искать пространства без присущей верхней границы на размер архитектуры [21]; с другой стороны, строгие сетевые морфизмы могут только увеличить архитектуру, что может привести к чрезмерно сложным архитектурам. Это может быть ослаблено путем использования приближенных сетевых морфизмов, которые позволяют уменьшать архитектуру [22].

Однократный поиск архитектуры (one-shot architecture search) – еще один перспективный подход для ускорения оценки производительности, который рассматривает все архитектуры как различные подграфы суперграфа

(модель one-shot) и присваивает общие веса архитектурам, которые имеют общие ребра этого суперграфа [6, 9, 39, 46, 52]. Необходимо обучить (одним из имеющихся способов) веса только одной модели one-shot, а архитектуры (которые являются просто подграфами модели one-shot) могут затем оцениваться без отдельного обучения, наследуя обученные веса от модели. Это значительно ускоряет оценку производительности архитектур, поскольку обучение не требуется (только оценка производительности на проверочных данных). Этот подход обычно имеет большую погрешность, так как сильно недооценивает реальную производительность архитектур; тем не менее он позволяет надежно ранжировать архитектуры, так как оценочная производительность сильно коррелирует с реальной производительностью [6]. Различные методы однократного NAS отличаются способом обучения модели one-shot: алгоритм ENAS [46] обучает контроллер RNN, который, в свою очередь, выбирает архитектуры из пространства поиска и обучает модель на основе приблизительных градиентов, полученных с помощью REINFORCE. Алгоритм DARTS [39] оптимизирует все веса модели one-shot совместно с непрерывной релаксацией пространства поиска, полученного путем размещения смеси операций-кандидатов на каждом ребре модели. Бендер и др. [6] обучают модель one-shot только один раз и показывают, что этого достаточно, когда части этой модели деактивируются стохастически во время обучения с помощью отбрасывания путей. В то время как ENAS и DARTS оптимизируют распределение по архитектурам во время обучения, подход Бендера и др. [6] можно рассматривать как использование фиксированного распределения. Высокая производительность, достигнутая методом Бендера, указывает на то, что комбинация разделения веса и фиксированного (тщательно выбранного) распределения может быть (возможно, неожиданно) единственным необходимым компонентом для однократного NAS. С этими подходами связано метаобучение *гиперсетей* (hypernetwork), которое генерирует веса для новых архитектур и, таким образом, требует только обучения гиперсети, но не самих архитектур [9]. Основное отличие здесь заключается в том, что веса не являются строго общими, а генерируются общей гиперсетью (в зависимости от выбранной архитектуры).

Общий недостаток однократного NAS состоит в том, что априорно определенный суперграф ограничивает пространство поиска своими подграфами. Кроме того, методы, которые требуют, чтобы весь суперграф находился в памяти графического процессора во время поиска архитектуры, будут ограничены относительно небольшими суперграфами и областями поиска соответственно и поэтому обычно используются в сочетании с пространствами поиска на основе ячеек. И хотя методы, основанные на разделении веса, существенно сократили вычислительные ресурсы, требуемые для NAS (с тысяч до нескольких GPU-дней), в настоящее время не совсем понятно, какие смещения они вносят в поиск, если выборочное распределение архитектур оптимизировано наряду с однократной моделью. Например, первоначальная предвзятость, которая заключается в изучении одних частей пространства поиска больше, чем других, может привести к тому, что веса однократной модели будут лучше адаптированы для этих архитектур, что, в свою очередь, усилит предвзятость поиска к этим частям пространства. Это может привести

к преждевременной конвергенции NAS, что служит одним из аргументов в пользу фиксированного распределения выборки, используемого в работе Бендера и др. [6]. В целом более систематический анализ погрешностей, вносимых различными оценщиками эффективности, является одним из перспективных направлений будущих исследований.

3.5. Перспективные направления

В этом разделе мы обсудим несколько текущих и будущих направлений исследований в области NAS. Большинство существующих работ посвящено NAS для классификации изображений. С одной стороны, классификация изображений – это трудно достижимый идеал, поскольку в этой области очень много хороших архитектур, разработанных вручную, и не так-то легко превзойти их с помощью NAS. С другой стороны, опираясь на знание разработанных вручную архитектур, можно относительно легко определить подходящее пространство поиска. Это, в свою очередь, делает маловероятным, что NAS найдет архитектуры, существенно превосходящие существующие, поскольку найденные архитектуры не могут принципиально отличаться друг от друга. Поэтому мы считаем важным выйти за рамки проблем классификации изображений, применив NAS к менее изученным областям. Заметными шагами в этом направлении являются применение NAS для моделирования языка [74], моделирования музыки [48], восстановления изображений [58] и сжатия сетей [3]. Следующими перспективными направлениями для применения NAS могут быть обучение с подкреплением, генеративные состязательные сети, семантическая сегментация или объединение сенсорных данных.

Альтернативным направлением является разработка методов NAS для многозадачных сценариев [36, 42] и многообъектных сценариев [20, 22, 73], в которых меры эффективности использования ресурсов используются в качестве целей наряду с эффективностью прогнозирования на незнакомых данных. Аналогично было бы интересно расширить подходы обучения с подкреплением или на основе задачи о многоруких бандитах, такие как рассмотренные в разделе 3.3, для обучения стратегий, обусловленных состоянием, которые кодируют свойства задачи и/или требования к ресурсам (т. е. приводят к задаче контекстного бандита). Аналогичного направления придерживались Рамачандран и Ле [47], расширяя однократное NAS для генерации различных архитектур в зависимости от задачи или экземпляра «на лету». Применение NAS для поиска архитектур, более устойчивых к неблагоприятным примерам [17], является еще одним перспективным направлением.

С тематикой NAS также связаны исследования по определению более общих и гибких пространств поиска. Например, хотя ячеечное пространство поиска обеспечивает высокую переносимость между различными задачами классификации изображений, оно в значительной степени основано на человеческом опыте классификации изображений и не может быть легко обобщено на другие области, где жестко заданная иерархическая структура (повторение одних и тех же ячеек несколько раз в виде цепочки) неприменима.

К таки задачам относятся, например, семантическая сегментация текста или обнаружение объектов. Поисковое пространство, позволяющее представлять и идентифицировать более общую иерархическую структуру, сделало бы NAS более широко применимым, как показано в одной из первых работ в этом направлении (Лю и др. [38]). Более того, обычные пространства поиска также основаны на заранее определенных структурных компонентах, таких как различные виды сверток и объединение, но не позволяют идентифицировать новые компоненты на этом уровне; выход за рамки этого ограничения может существенно увеличить возможности NAS.

Сравнение различных методов NAS осложняется тем, что измерение производительности архитектуры зависит от многих факторов, помимо самой архитектуры. Хотя большинство авторов сообщает о результатах на наборе данных CIFAR-10, эксперименты часто отличаются в отношении пространства поиска, вычислительного бюджета, увеличения объема данных, процедур обучения, регуляризации и других факторов. Например, для CIFAR-10 производительность существенно повышается при использовании изменяемой скорости обучения по методу косинусного отжига [40], дополнения данных методом CutOut [18], методом MixUp [70] или комбинацией факторов [16], регуляризации методом Shake-Shake [26] или запланированного снижения скорости [75]. Поэтому можно предположить, что улучшения в этих компонентах оказывают большее влияние на заявленные показатели производительности, чем улучшение архитектуры за счет NAS. Поэтому мы считаем, что определение общих эталонов имеет решающее значение для справедливого сравнения различных методов NAS. Первым шагом в этом направлении является определение эталона для совместного поиска архитектуры и гиперпараметров для полностью подключенной нейронной сети с двумя скрытыми слоями [33]. В этом эталоне необходимо оптимизировать девять дискретных гиперпараметров, которые управляют как архитектурой, так и оптимизацией/регуляризацией. Все 62 208 возможных комбинаций гиперпараметров были предварительно оценены, что позволяет сравнивать различные методы с небольшими вычислительными ресурсами. Тем не менее пространство поиска остается очень простым по сравнению с пространствами, используемыми большинством методов NAS. Было бы также интересно оценить методы NAS не изолированно, а как часть полной системы AutoML с открытым исходным кодом, где также оптимизируются гиперпараметры [41, 50, 69] и конвейер дополнения данных [16] вместе с NAS.

Несмотря на то что в целом методика NAS достигла впечатляющей производительности, до сих пор она не дает представления о том, почему конкретные архитектуры работают хорошо и насколько похожи архитектуры, полученные в независимых прогонах. Было бы желательно выявить общие мотивы, понять, почему эти мотивы важны для высокой производительности, и выяснить, являются ли эти мотивы общими для различных проблем.

Благодарности. Мы благодарим Эстебана Реала, Арбера Зела, Габриэля Бендера, Кеннета Стэнли и Томаса Пфайла за отзывы о рукописи этой главы и о наших предыдущих исследованиях. Эта работа была частично поддержана Европейским исследовательским советом (ERC) в рамках программы исследований и инноваций Европейского союза «Горизонт 2020» по гранту № 716721.

3.6. Литература

1. Ahmed, K., Torresani, L.: Maskconnect: Connectivity learning by gradient descent. In: European Conference on Computer Vision (ECCV) (2018).
2. Angeline, P.J., Saunders, G.M., Pollack, J.B.: An evolutionary algorithm that constructs recurrent neural networks. IEEE transactions on neural networks 5 1, 54–65 (1994).
3. Ashok, A., Rhinehart, N., Beainy, F., Kitani, K.M.: N2n learning: Network to network compression via policy gradient reinforcement learning. In: International Conference on Learning Representations (2018).
4. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. In: International Conference on Learning Representations (2017a).
5. Baker, B., Gupta, O., Raskar, R., Naik, N.: Accelerating Neural Architecture Search using Performance Prediction. In: NIPS Workshop on Meta-Learning (2017b).
6. Bender, G., Kindermans, P.J., Zoph, B., Vasudevan, V., Le, Q.: Understanding and simplifying one-shot architecture search. In: International Conference on Machine Learning (2018).
7. Bergstra, J., Yamins, D., Cox, D.D.: Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: ICML (2013).
8. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R.S., Bartlett, P.L., Pereira, F., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 24. pp. 2546–2554 (2011).
9. Brock, A., Lim, T., Ritchie, J.M., Weston, N.: SMASH: one-shot model architecture search through hypernetworks. In: NIPS Workshop on Meta-Learning (2017).
10. Cai, H., Chen, T., Zhang, W., Yu, Y., Wang, J.: Efficient architecture search by network transformation. In: Association for the Advancement of Artificial Intelligence (2018a).
11. Cai, H., Yang, J., Zhang, W., Han, S., Yu, Y.: Path-Level Network Transformation for Efficient Architecture Search. In: International Conference on Machine Learning (Jun 2018b).
12. Chen, T., Goodfellow, I.J., Shlens, J.: Net2net: Accelerating learning via knowledge transfer. In: International Conference on Learning Representations (2016).
13. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. arXiv:1610.02357 (2016).
14. Chrabaszcz, P., Loshchilov, I., Hutter, F.: A downsampled variant of imagenet as an alternative to the CIFAR datasets. CoRR abs/1707.08819 (2017).
15. Chrabaszcz, P., Loshchilov, I., Hutter, F.: Back to basics: Benchmarking canonical evolution strategies for playing atari. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18. pp. 1419–1426. International Joint Conferences on Artificial Intelligence Organization (Jul 2018).

16. Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: AutoAugment: Learning Augmentation Policies from Data. In: arXiv:1805.09501 (May 2018).
17. Cubuk, E.D., Zoph, B., Schoenholz, S.S., Le, Q.V.: Intriguing Properties of Adversarial Examples. In: arXiv:1711.02846 (Nov 2017).
18. Devries, T., Taylor, G.W.: Improved regularization of convolutional neural networks with cutout. arXiv preprint abs/1708.04552 (2017).
19. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI) (2015).
20. Dong, J.D., Cheng, A.C., Juan, D.C., Wei, W., Sun, M.: Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In: European Conference on Computer Vision (2018).
21. Elsken, T., Metzen, J.H., Hutter, F.: Simple And Efficient Architecture Search for Convolutional Neural Networks. In: NIPS Workshop on Meta-Learning (2017).
22. Elsken, T., Metzen, J.H., Hutter, F.: Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution. In: International Conference on Learning Representations (2019).
23. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. arXiv:1808.05377 (2018).
24. Falkner, S., Klein, A., Hutter, F.: BOHB: Robust and efficient hyperparameter optimization at scale. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 1436–1445. PMLR, Stockholmsmässan, Stockholm Sweden (10–15 Jul 2018).
25. Floreano, D., Dürr, P., Mattiussi, C.: Neuroevolution: from architectures to learning. Evolutionary Intelligence 1(1), 47–62 (2008).
26. Gastaldi, X.: Shake-shake regularization. In: International Conference on Learning Represen- tations Workshop (2017).
27. Goldberg, D.E., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. In: Foundations of Genetic Algorithms. pp. 69–93. Morgan Kaufmann (1991).
28. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: Conference on Computer Vision and Pattern Recognition (2016).
29. Huang, G., Liu, Z., Weinberger, K.Q.: Densely Connected Convolutional Networks. In: Conference on Computer Vision and Pattern Recognition (2017).
30. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: LION. pp. 507–523 (2011).
31. Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B., Xing, E.: Neural Architecture Search with Bayesian Optimisation and Optimal Transport. arXiv:1802.07191 (Feb 2018).
32. Klein, A., Falkner, S., Springenberg, J.T., Hutter, F.: Learning curve prediction with Bayesian neural networks. In: International Conference on Learning Representations (2017a).
33. Klein, A., Christiansen, E., Murphy, K., Hutter, F.: Towards reproducible neural architecture and hyperparameter search. In: ICML 2018 Workshop on Reproducibility in ML (RML 2018) (2018).

34. Klein, A., Falkner, S., Bartels, S., Hennig, P., Hutter, F.: Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In: Singh, A., Zhu, J. (eds.) Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 54, pp. 528–536. PMLR, Fort Lauderdale, FL, USA (20–22 Apr 2017b).
35. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: bandit-based configuration evaluation for hyperparameter optimization. In: International Conference on Learning Representations (2017).
36. Liang, J., Meyerson, E., Miikkulainen, R.: Evolutionary Architecture Search For Deep Multitask Networks. In: arXiv:1803.03745 (Mar 2018).
37. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive Neural Architecture Search. In: European Conference on Computer Vision (2018a).
38. Liu, H., Simonyan, K., Vinyals, O., Fernando, C., Kavukcuoglu, K.: Hierarchical Representations for Efficient Architecture Search. In: International Conference on Learning Representations (2018b).
39. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. In: International Conference on Learning Representations (2019).
40. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. In: International Conference on Learning Representations (2017).
41. Mendoza, H., Klein, A., Feurer, M., Springenberg, J., Hutter, F.: Towards Automatically-Tuned Neural Networks. In: International Conference on Machine Learning, AutoML Workshop (Jun 2016).
42. Meyerson, E., Miikkulainen, R.: Pseudo-task Augmentation: From Deep Multitask Learning to Intratask Sharing and Back. In: arXiv:1803.03745 (Mar 2018).
43. Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., Hodjat, B.: Evolving Deep Neural Networks. In: arXiv:1703.00548 (Mar 2017).
44. Miller, G., Todd, P., Hedge, S.: Designing neural networks using genetic algorithms. In: 3rd International Conference on Genetic Algorithms (ICGA'89) (1989).
45. Negrinho, R., Gordon, G.: DeepArchitect: Automatically Designing and Training Deep Architectures. arXiv:1704.08792 (2017).
46. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. In: International Conference on Machine Learning (2018).
47. Ramachandran, P., Le, Q.V.: Dynamic Network Architectures. In: AutoML 2018 (ICML workshop) (2018).
48. Rawal, A., Miikkulainen, R.: From Nodes to Networks: Evolving Recurrent Neural Networks. In: arXiv:1803.04439 (Mar 2018).
49. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Aging Evolution for Image Classifier Architecture Search. In: AAAI Conference on Artificial Intelligence (2019).
50. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. International Conference on Machine Learning (2017).

51. Salimans, T., Ho, J., Chen, X., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint (2017).
52. Saxena, S., Verbeek, J.: Convolutional neural fabrics. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29, pp. 4053–4061. Curran Associates, Inc. (2016).
53. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., de Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE 104(1), 148–175 (Jan 2016).
54. Shin, R., Packer, C., Song, D.: Differentiable neural network architecture search. In: International Conference on Learning Representations Workshop (2018).
55. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large- scale neural networks. Artif. Life 15(2), 185–212 (Apr 2009), URL https://doi.org/10.1162/artl.2009.15.2.15202.
56. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 10, 99–127 (2002).
57. Such, F.P., Madhavan, V., Conti, E., Lehman, J., Stanley, K.O., Clune, J.: Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv preprint (2017).
58. Suganuma, M., Ozay, M., Okatani, T.: Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 4771–4780. PMLR, Stockholmsmässan, Stockholm Sweden (10–15 Jul 2018).
59. Suganuma, M., Shirakawa, S., Nagao, T.: A genetic programming approach to designing convolutional neural network architectures. In: Genetic and Evolutionary Computation Conference (2017).
60. Swersky, K., Duvenaud, D., Snoek, J., Hutter, F., Osborne, M.: Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. In: NIPS Workshop on Bayesian Optimization in Theory and Practice (2013).
61. Swersky, K., Snoek, J., Adams, R.P.: Freeze-thaw bayesian optimization (2014).
62. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the Inception Architecture for Computer Vision. In: Conference on Computer Vision and Pattern Recognition (2016).
63. Wei, T., Wang, C., Chen, C.W.: Modularized morphing of neural networks. arXiv:1701.03281 (2017).
64. Wei, T., Wang, C., Rui, Y., Chen, C.W.: Network morphism. In: International Conference on Machine Learning (2016).
65. Wistuba, M.: Finding Competitive Network Architectures Within a Day Using UCT. In: arXiv:1712.07420 (Dec 2017).
66. Xie, L., Yuille, A.: Genetic CNN. In: International Conference on Computer Vision (2017).
67. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE 87(9), 1423–1447 (Sept 1999).
68. Yu, F., Koltun, V.: Multi-scale context aggregation by dilated convolutions (2016).

69. Zela, A., Klein, A., Falkner, S., Hutter, F.: Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. In: ICML 2018 Workshop on AutoML (AutoML 2018) (2018).
70. Zhang, H., Cissé, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. arXiv preprint abs/1710.09412 (2017).
71. Zhong, Z., Yan, J., Wu, W., Shao, J., Liu, C.L.: Practical block-wise neural network architecture generation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2423–2432 (2018a).
72. Zhong, Z., Yang, Z., Deng, B., Yan, J., Wu, W., Shao, J., Liu, C.L.: Blockqnn: Efficient block- wise neural network architecture generation. arXiv preprint (2018b).
73. Zhou, Y., Ebrahimi, S., Arik, S., Yu, H., Liu, H., Diamos, G.: Resource-efficient neural architect. In: arXiv:1806.07912 (2018).
74. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: International Conference on Learning Representations (2017).
75. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Conference on Computer Vision and Pattern Recognition (2018).

Часть II

СИСТЕМЫ AutoML

Глава 4

Auto-WEKA: автоматический выбор модели и оптимизация гиперпараметров в WEKA

Ларс Коттхофф (✉), Университет Вайоминга, Ларами, штат Вайоминг, США, e-mail: larsko@uwyo.edu

Крис Торнтон, **Кевин Лейтон-Браун**, факультет компьютерных наук, Университет Британской Колумбии, Ванкувер, Британская Колумбия, Канада

Хольгер Хоос, Лейденский институт передовых компьютерных наук, Лейденский университет, Лейден, Нидерланды

Франк Хуттер, факультет компьютерных наук, Университет Фрайбурга, Фрайбург, Германия

К сегодняшнему дню разработано множество различных алгоритмов машинного обучения; с учетом гиперпараметров каждого алгоритма существует ошеломляюще большое количество возможных вариантов. В этой главе мы рассматриваем проблему одновременного выбора алгоритма обучения и задания его гиперпараметров. Мы показываем, что она может быть решена с помощью полностью автоматизированного подхода, использующего последние инновации в байесовской оптимизации. В частности, мы рассматриваем методы отбора признаков и все подходы машинного обучения, реализованные в стандартном дистрибутиве WEKA, включая два ансамблевых метода, 10 метаметодов, 28 базовых обучателей и настройки гиперпараметров для каждого обучателя. На каждом из 21 популярного набора данных из репозитория UCI, KDD Cup 09, вариантов набора данных MNIST и CIFAR-10 мы продемонстрируем производительность, часто значительно превосходящую стандартные методы отбора и оптимизации гиперпараметров. Мы

надеемся, что наш подход поможет пользователям, которые не являются специалистами в области машинного обучения, более эффективно выбирать алгоритмы и настройки гиперпараметров, подходящие для их приложений, и, следовательно, достигать более высокой производительности.

4.1. Введение

Все чаще пользователями инструментов машинного обучения становятся неспециалисты, которым требуются готовые решения. Сообщество машинного обучения старается помочь таким пользователям, предоставляя широкий выбор сложных алгоритмов обучения и методов выбора признаков в виде пакетов с открытым исходным кодом, таких как WEKA [15] и mlr [7]. Эти пакеты предлагают пользователю сделать два вида выбора: выбрать алгоритм обучения и настроить его, задав гиперпараметры (которые также управляют отбором признаков, если это применимо к алгоритму). Сделать правильный выбор при наличии таких степеней свободы может быть непросто, поэтому многие пользователи выбирают алгоритмы, основываясь на репутации или интуитивной привлекательности, и/или оставляют гиперпараметры по умолчанию. Конечно, такой подход может дать результаты гораздо хуже, чем при выборе лучшего метода и гиперпараметров.

Это наводит на мысль о естественной задаче машинного обучения: исходя из заданного набора данных автоматически выбрать алгоритм обучения и одновременно настроить его гиперпараметры для оптимизации эмпирической производительности. Мы называем эту задачу *комбинированным выбором алгоритма и оптимизацией гиперпараметров* (combined algorithm selection and hyperparameter optimization problem, задача CASH); формальное определение задачи мы дадим в разделе 4.3. В прошлом было много работ, посвященных выбору модели, например [1, 6, 8, 9, 11, 24, 25, 33], и оптимизации гиперпараметров, например [3-5, 14, 23, 28, 30]. Но, несмотря на практическую важность, мы с удивлением обнаружили в литературе лишь ограниченное число вариантов задачи CASH; кроме того, в них рассматривается фиксированное и относительно небольшое число конфигураций параметров для каждого алгоритма (см., например, [22]).

Вероятное объяснение заключается в том, что поиск в комбинированном пространстве алгоритмов обучения и их гиперпараметров очень сложен: функция отклика зашумлена, пространство имеет высокую размерность, включает как категориальный, так и непрерывный выбор и содержит иерархические зависимости (например, гиперпараметры алгоритма обучения имеют смысл только в том случае, если выбран этот алгоритм; выбор алгоритма в ансамблевом методе имеет смысл только в том случае, если выбран этот ансамблевый метод; и т. д.). Другое направление работы связано с процедурами метаобучения, которые используют характеристики набора данных, например производительность так называемых алгоритмов ориентирования, чтобы предсказать, какой алгоритм или конфигурация гиперпараметров будут работать хорошо [2, 22, 26, 32]. В то время как алгоритмы

CASH, которые мы исследуем в этой главе, начинают с нуля для каждого нового набора данных, упомянутые процедуры метаобучения используют информацию из предыдущих наборов данных, которая не всегда может быть доступна.

Далее мы покажем, что CASH можно рассматривать как единую иерархическую задачу оптимизации гиперпараметров, в которой даже выбор алгоритма сам по себе считается гиперпараметром. Мы также показываем, что на основе такой постановки задачи современные методы байесовской оптимизации позволяют получить высококачественные результаты за разумное время и с минимальными человеческими усилиями. После обсуждения некоторых предварительных условий (раздел 4.2) мы определяем задачу CASH и обсуждаем методы ее решения (раздел 4.3). Затем мы определяем конкретную задачу CASH, охватывающую широкий спектр обучающих алгоритмов и селекторов признаков в открытом пакете WEKA (раздел 4.4), и показываем, что поиск в комбинированном пространстве алгоритмов и гиперпараметров дает более эффективные модели, чем стандартные методы выбора алгоритмов и оптимизации гиперпараметров (раздел 4.5). В частности, мы показываем, что недавно разработанные процедуры байесовской оптимизации TPE [4] и SMAC [16] часто находят комбинации алгоритмов и гиперпараметров, которые превосходят существующие базовые методы, особенно на больших наборах данных.

Эта глава основана на двух статьях, опубликованных в материалах *KDD 2013* [31] и в журнале *Journal of Machine Learning Research* (*JMLR*) в 2017 г. [20].

4.2. Предварительные условия

Мы рассматриваем обучение функции $f: \mathcal{X} \mapsto \mathcal{Y}$, где $\mathcal{Y}$ является либо конечной (для классификации), либо непрерывной (для регрессии). *Алгоритм обучения* A отображает множество $\{d_1, \ldots, d_n\}$ обучающих точек данных $d_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ на такую функцию, которая часто выражается через вектор *параметров модели*. Большинство алгоритмов обучения A дополнительно содержит гиперпараметры $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$, которые изменяют способ работы самого алгоритма обучения $A_{\boldsymbol{\lambda}}$. Например, гиперпараметры используются для описания штрафа за длину описания, количества нейронов в скрытом слое, количества точек данных, которые должен содержать лист дерева решений, чтобы его можно было разделить, и т. д. Эти гиперпараметры обычно оптимизируются во «внешнем цикле», который оценивает производительность каждой конфигурации гиперпараметров с помощью перекрестной проверки.

4.2.1. Выбор модели

При заданном наборе алгоритмов обучения $\mathcal{A}$ и ограниченном количестве обучающих данных $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ целью выбора модели является определение алгоритма $A^* \in \mathcal{A}$, обладающего оптимальной способностью

к обобщению. Эффективность обобщения оценивается путем разбиения $\mathcal{D}$ на несовпадающие обучающее и проверочное множества $\mathcal{D}^{(i)}_{train}$ и $\mathcal{D}^{(i)}_{valid}$, обучения функций f_i путем применения A^* к $\mathcal{D}^{(i)}_{train}$ и оценки предсказательной эффективности этих функций на $\mathcal{D}^{(i)}_{valid}$. Это позволяет записать задачу выбора модели в виде:

$$A^* \in \operatorname*{argmin}_{A \in \mathcal{A}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A, \mathcal{D}^{(i)}_{train}, \mathcal{D}^{(i)}_{valid}),$$

где $\mathcal{L}(A, \mathcal{D}^{(i)}_{train}, \mathcal{D}^{(i)}_{valid})$ – это потеря, достигнутая A при обучении на $\mathcal{D}^{(i)}_{train}$ и оценке на $\mathcal{D}^{(i)}_{valid}$.

Мы используем k-кратную перекрестную проверку [19], которая разбивает обучающие данные на k равных по размеру частей $\mathcal{D}^{(1)}_{train}$, ..., $\mathcal{D}^{(k)}_{valid}$ и задает $\mathcal{D}^{(i)}_{train} = \mathcal{D} \backslash \mathcal{D}^{(i)}_{valid}$ для $i = 1, \ldots, k$[1].

4.2.2. Оптимизация гиперпараметров

Задача оптимизации гиперпараметров $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ заданного алгоритма обучения A концептуально схожа с задачей выбора модели. Некоторые ключевые отличия заключаются в том, что гиперпараметры часто непрерывны, что пространства гиперпараметров часто имеют высокую размерность и что мы можем воспользоваться наличием корреляции между различными настройками гиперпараметров $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \in \boldsymbol{\Lambda}$. Если даны n гиперпараметров $\lambda_1, \ldots, \lambda_n$ с областями $\boldsymbol{\Lambda}_1, \ldots, \boldsymbol{\Lambda}_n$, пространство гиперпараметров $\boldsymbol{\Lambda}$ является подмножеством перекрестного произведения этих областей: $\boldsymbol{\Lambda} \subset \Lambda_1 \times \cdots \times \Lambda_n$. Это подмножество часто является строгим, например когда определенные настройки одного гиперпараметра делают другие гиперпараметры неактивными. Например, параметры, определяющие особенности третьего слоя глубокой сети, не имеют значения, если глубина сети настроена на один или два слоя. Аналогично параметры полиномиального ядра модели опорных векторов не имеют значения, если используется другое ядро.

С более формальной точки зрения, следуя [17], мы говорим, что гиперпараметр λ_i *обусловлен* другим гиперпараметром λ_j, если λ_i активен только тогда, когда гиперпараметр λ_j принимает значения из заданного множества $V_i(j) \subsetneq \Lambda_j$; в этом случае мы называем λ_j *родителем* λ_i. Условные гиперпараметры могут быть родителями других условных гиперпараметров, что приводит к появлению древовидного структурированного пространства [4] или – в некоторых случаях – *направленного ациклического графа* (directed acyclic graph, DAG) [17]. С учетом такого структурированного пространства $\boldsymbol{\Lambda}$ задача оптимизации потенциально иерархических гиперпараметров может быть записана следующим образом:

---

[1] Существуют и другие способы оценки эффективности обобщения; например, мы также экспериментировали с повторной валидацией случайной подвыборки [19] и получили аналогичные результаты.

$$\boldsymbol{\lambda}^* \in \operatorname*{argmin}_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}_{train}^{(i)}, \mathcal{D}_{valid}^{(i)}).$$

4.3. Одновременный выбор алгоритмов и оптимизация гиперпараметров (CASH)

Учитывая набор алгоритмов $\mathcal{A} = \{A^{(1)}, \ldots, A^{(k)}\}$ с соответствующими гиперпараметрическими пространствами $\boldsymbol{\Lambda}^{(1)}, \ldots, \boldsymbol{\Lambda}^{(k)}$, мы определяем комбинированную задачу выбора алгоритма и оптимизации гиперпараметров (CASH) как вычисление:

$$A^*_{\boldsymbol{\lambda}^*} \in \operatorname*{argmin}_{A^{(j)} \in \mathcal{A},\ \boldsymbol{\lambda} \in \boldsymbol{\Lambda}^{(j)}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A_{\boldsymbol{\lambda}}^{(j)}, \mathcal{D}_{train}^{(i)}, \mathcal{D}_{valid}^{(i)}). \tag{4.1}$$

Заметим, что эту задачу можно переформулировать как единую комбинированную иерархическую задачу оптимизации гиперпараметров с пространством параметров $\boldsymbol{\Lambda} = \boldsymbol{\Lambda}^{(1)} \cup \cdots \cup \boldsymbol{\Lambda}^{(k)} \cup \{\lambda_r\}$, где λ_r – новый гиперпараметр корневого уровня, выбирающий между алгоритмами $A^{(1)}, \ldots, A^{(k)}$. Параметры корневого уровня каждого подпространства $\boldsymbol{\Lambda}^{(i)}$ обуславливаются тем, что λ_r воплощаются в A_i.

В принципе, задача (4.1) может быть решена различными способами. Перспективным подходом является байесовская оптимизация [10], и в частности *последовательная оптимизация на основе модели* (sequential model-based optimization, SMBO) [16] – универсальный механизм стохастической оптимизации, который может работать как с категориальными, так и с непрерывными гиперпараметрами и может использовать иерархическую структуру, вытекающую из условных параметров. SMBO (алгоритм 1) сначала строит модель $\mathcal{M}_{\mathcal{L}}$, которая отражает зависимость функции потерь $\mathcal{L}$ от настроек гиперпараметров $\boldsymbol{\lambda}$ (строка 1 в алгоритме 1). Затем он выполняет следующие итерации: использование $\mathcal{M}_{\mathcal{L}}$ для определения перспективной конфигурации-претендента гиперпараметров $\boldsymbol{\lambda}$ для следующей оценки (строка 3); оценка потери c для $\boldsymbol{\lambda}$ (строка 4); и обновление модели $\mathcal{M}_{\mathcal{L}}$ с полученной таким образом новой точкой данных $(\boldsymbol{\lambda}, c)$ (строки 5, 6).

Алгоритм 1. SMBO

1: инициализация модели $\mathcal{M}_{\mathcal{L}}$; $\mathcal{H} \leftarrow \emptyset$
2: **while** бюджет времени на оптимизацию не исчерпан **do**
3: $\quad\lambda \leftarrow$ конфигурация-претендент из $\mathcal{M}_{\mathcal{L}}$
4: $\quad$Вычисление $c = \mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}_{train}^{(i)}, \mathcal{D}_{valid}^{(i)})$
5: $\quad\mathcal{H} \leftarrow \mathcal{H} \cup \{(\boldsymbol{\lambda}, c)\}$
6: $\quad$Обновление $\mathcal{M}_{\mathcal{L}}$ исходя из $\mathcal{H}$
7: **end while**
8: **return** λ из $\mathcal{H}$ с минимальным значением c

Для выбора следующей гиперпараметрической конфигурации λ с помощью модели $\mathcal{M}_{\mathcal{L}}$ алгоритм SMBO использует так называемую *функцию сбора* (acquisition function) $a_{\mathcal{M}_{\mathcal{L}}} : \boldsymbol{\Lambda} \mapsto \mathbb{R}$, которая использует прогнозируемое распределение модели $\mathcal{M}_{\mathcal{L}}$ при произвольных гиперпараметрических конфигурациях $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ для количественной оценки (в аналитической форме) того, насколько полезным будет знание о $\boldsymbol{\lambda}$. Затем SMBO просто максимизирует эту функцию по $\boldsymbol{\Lambda}$, чтобы выбрать наиболее полезную конфигурацию $\boldsymbol{\lambda}$ для следующей оценки. Существует несколько хорошо изученных функций сбора [18, 27, 29]; все они нацелены на автоматический компромисс между использованием (локальная оптимизация гиперпараметров в областях, известных своей эффективностью) и исследованием (попытка выбора гиперпараметров в относительно неисследованной области пространства), чтобы избежать преждевременной сходимости. Мы максимизируем положительное *ожидаемое улучшение* (expected improvement, EI), достижимое по сравнению с существующей заданной потерей c_{min} [27]. Пусть $c(\boldsymbol{\lambda})$ обозначает потерю гиперпараметрической конфигурации $\boldsymbol{\lambda}$. Тогда положительная функция улучшения по c_{min} определяется как

$$I_{c_{\min}}(\boldsymbol{\lambda}) := \max\{c_{\min} - c(\boldsymbol{\lambda}), 0\}.$$

Конечно, мы не знаем $c(\boldsymbol{\lambda})$. Однако мы можем вычислить ее ожидание относительно текущей модели $\mathcal{M}_{\mathcal{L}}$:

$$\mathbb{E}_{\mathcal{M}_{\mathcal{L}}}[I_{c_{\min}}(\boldsymbol{\lambda})] = \int_{-\infty}^{c_{\min}} \max\{c_{\min} - c, 0\} \cdot p_{\mathcal{M}_{\mathcal{L}}}(c|\lambda) dc. \tag{4.2}$$

Далее кратко рассмотрим подход SMBO, используемый в этой главе.

4.3.1. Последовательный алгоритм конфигурации по модели (SMAC)

Последовательный алгоритм конфигурации по модели (sequential model-based algorithm configuration, SMAC)[1] [16] использует для отражения зависимости функции потерь c от гиперпараметров $\boldsymbol{\lambda}$ разнообразные модели $p(c|\boldsymbol{\lambda})$, включая приближенные гауссовы процессы и случайные леса. В этой главе мы используем модели случайного леса, поскольку они, как правило, хорошо работают с дискретными и высокоразмерными входными данными. SMAC обрабатывает условные параметры путем преобразования неактивных условных параметров в $\boldsymbol{\lambda}$ в значения по умолчанию для обучения модели и предсказания. Это позволяет отдельным деревьям решений содержать ветвления вида «активен ли гиперпараметр λ_i?», что позволяет им сосредоточиться на активных гиперпараметрах. Хотя случайные леса обычно не рассматриваются как вероятностные модели, SMAC получает предсказательное среднее $\mu_{\boldsymbol{\lambda}}$ и дисперсию $\sigma^2_{\boldsymbol{\lambda}}$ для $p(c|\boldsymbol{\lambda})$ в качестве частотных оценок предсказаний отдельных деревьев для $\boldsymbol{\lambda}$; затем он моделирует $p_{\mathcal{M}_{\mathcal{L}}}(c|\boldsymbol{\lambda})$ как гауссово значение $\mathcal{N}(\mu_{\boldsymbol{\lambda}}, \sigma^2_{\boldsymbol{\lambda}})$.

[1] В некоторых публикациях применяется аббревиатура SMBA. – *Прим. перев.*

SMAC использует критерий ожидаемого улучшения, определенный в уравнении 4.2, создавая экземпляр $c_{\min}$ – потерю наилучшей конфигурации гиперпараметров, измеренной на данный момент. Согласно прогнозному распределению SMAC $p_{\mathcal{M}_{\mathcal{L}}}(c|\boldsymbol{\lambda}) = \mathcal{N}(\mu_{\boldsymbol{\lambda}}, \sigma^2_{\boldsymbol{\lambda}})$ это математическое ожидание вычисляется в аналитическом виде следующим образом:

$$\mathbb{E}_{\mathcal{M}_{\mathcal{L}}}[I_{c_{\min}}(\boldsymbol{\lambda})] = \sigma_{\boldsymbol{\lambda}} \cdot [u \cdot \Phi(u) + \varphi(u)],$$

где $u = \frac{c_{\min} - \mu_{\boldsymbol{\lambda}}}{\sigma_{\boldsymbol{\lambda}}}$, а φ и Φ обозначают функцию плотности вероятности и кумулятивную функцию стандартного нормального распределения соответственно [18].

SMAC предназначен для робастной оптимизации при зашумленном вычислении функций, и поэтому в нем реализованы специальные механизмы для отслеживания наилучшей известной конфигурации и обеспечения высокой уверенности в оценке производительности этой конфигурации. Этой устойчивостью к зашумленным вычислениям функций можно воспользоваться при комбинированном выборе алгоритмов и оптимизации гиперпараметров, поскольку оптимизируемая функция в уравнении (4.1) представляет собой среднее по набору членов потерь (каждый из которых соответствует одной паре $\mathcal{D}^{(i)}_{train}$ и $\mathcal{D}^{(i)}_{valid}$, полученной из обучающего набора). Ключевая идея в SMAC заключается в последовательном улучшении оценок среднего путем оценки этих условий по одному за раз, таким образом, достигается компромисс между точностью и вычислительными затратами. Для того чтобы новая конфигурация стала текущей, она должна превзойти предыдущую в каждом сравнении: рассматривается только одна подвыборка, две подвыборки и т. д. до общего количества подвыборок, ранее использованных для оценки предыдущей конфигурации. Более того, каждый раз, когда конфигурация-кандидат выдерживает такое сравнение, ее оценивают на новой подвыборке, вплоть до общего числа доступных подвыборок. Таким образом, количество подвыборок, используемых для оценки кандидата, растет со временем. По этой причине плохо работающая конфигурация может быть отброшена после рассмотрения всего одной подвыборки.

Наконец, SMAC содержит механизм диверсификации для достижения устойчивой производительности, даже когда его модель введена в заблуждение, и исследует новые части пространства: каждая вторая конфигурация выбирается случайным образом. Благодаря только что описанной процедуре оценивания это требует меньше накладных расходов, чем можно предположить.

4.4. Auto-WEKA

Чтобы продемонстрировать осуществимость автоматического подхода к решению задачи CASH, мы разработали пакет автоматизированного машинного обучения Auto-WEKA, который решает эту задачу для обучающих алгоритмов и селекторов признаков, реализованных в пакете машинного обучения WEKA [15]. Отметим, что, хотя мы сосредоточились на алгоритмах классификации

в WEKA, нет никаких препятствий для распространения нашего подхода на другие параметры. К слову, другой успешной системой, использующей ту же технологию, является auto-sklearn [12].

В табл. 4.1 показаны все поддерживаемые алгоритмы обучения и селекторы признаков с указанием количества гиперпараметров алгоритмов. Метаметоды принимают на вход один базовый классификатор и его параметры, а ансамблевые методы могут принимать на вход любое количество базовых обучающих устройств. Мы разрешили метаметодам использовать любой базовый обучающий элемент с любыми настройками гиперпараметров, а ансамблевым методам – использовать до пяти обучающих элементов, опять же с любыми настройками гиперпараметров. Не все методы обучения применимы для всех наборов данных (например, из-за неспособности классификатора обрабатывать отсутствующие данные). В каждом наборе данных наша реализация Auto-WEKA автоматически рассматривает только подмножество применимых обучающих данных. Отбор признаков выполняется как этап предварительной обработки перед построением любой модели.

Таблица 4.1. Обучатели и методы, поддерживаемые Auto-WEKA, а также количество гиперпараметров |Λ|. Каждый обучатель поддерживает классификацию; обучатели, отмеченные звездочкой, также поддерживают регрессию

Базовые обучатели			
BayesNet	2	NaiveBayes	2
DecisionStump*	0	NaiveBayesMultinomial	0
DecisionTable*	4	OneR	1
Gaussian Processes*	10	PART	4
IBk*	5	RandomForest	7
J48	9	RandomTree*	11
JRip	4	REPTree*	6
KStar*	3	SGD*	5
Linear Regression*	3	SimpleLinearRegression*	0
LMT	9	SimpleLogistic	5
Logistic	1	SMO	11
M5P	4	SMOreg*	13
M5Rules	4	Voted Perceptron	3
Multilayer Perceptron*	8	ZeroR*	0
Ансамблевые методы			
Stacking	2	Vote	2
Метаметоды			
LWL	5	Bagging	4
AdaBoostM1	6	RandomCommittee	2
AdditiveRegression	4		
AttributeSelectedClassifier	2	RandomSubSpace	3
Методы на основе выбора признаков			
BestFirst	2	GreedyStepwise	4

Алгоритмы в табл. 4.1 имеют широкий спектр гиперпараметров, которые принимают значения из непрерывных интервалов, из диапазонов целых чисел и из других дискретных множеств. С каждым числовым параметром мы связали равномерное или логарифмически равномерное априорное распределение, в зависимости от его семантики. Например, для штрафа в алгоритме гребневой регрессии (ридж-регрессии) мы задали логарифмическое равномерное априорное распределение, а для максимальной глубины дерева в случайном лесу – равномерное распределение. Auto-WEKA работает с непрерывными значениями гиперпараметров вплоть до точности машины, на которой выполняется. Мы подчеркиваем, что это комбинированное гиперпараметрическое пространство *намного* больше, чем простое объединение гиперпараметрических пространств базовых обучателей, поскольку ансамблевые методы допускают до пяти *независимых* базовых обучателей. Мета- и ансамблевые методы, а также выбор признаков вносят дополнительный вклад в общий размер гиперпараметрического пространства Auto-WEKA.

Auto-WEKA использует оптимизатор SMAC, описанный выше, для решения задачи CASH и доступен пользователям через менеджер пакетов WEKA; исходный код можно найти по адресу https://github.com/AutoML/autoweka, а официальный сайт проекта находится по адресу http://www.cs.ubc.ca/labs/beta/Projects/autoweka. Для экспериментов, описанных в этой главе, мы использовали Auto-WEKA версии 0.5. Результаты, полученные более новыми версиями, схожи; мы не стали повторять полный набор экспериментов из-за больших вычислительных затрат.

4.5. Экспериментальная оценка

Мы оценивали Auto-WEKA на 21 известном эталонном наборе данных (табл. 4.2): 15 наборов из репозитория UCI [13]; задача convex, базовый MNIST и повернутый MNIST с фоновыми изображениями, использованные в [5]; задача appentency из конкурса KDD Cup 2009 и две версии задачи классификации изображений CIFAR-10 [21] (CIFAR-10-Small – подмножество CIFAR-10, где используются только первые 10 000 точек обучающих данных, а не все 50 000). Обратите внимание, что в экспериментальной оценке мы сосредоточились на классификации. Для наборов данных с заранее определенным разделением на обучающий и тестовый поднаборы мы использовали это разделение. В противном случае мы случайным образом делили набор данных на 70 % обучающих и 30 % тестовых данных. Мы скрывали тестовые данные от всех методов оптимизации; они использовались только один раз на этапе автономного анализа для оценки моделей, найденных различными методами оптимизации.

Для каждого набора данных мы запустили Auto-WEKA с каждым алгоритмом оптимизации гиперпараметров с общим бюджетом времени 30 ч. Для каждого метода мы выполнили 25 запусков этого процесса с различными инициациями генератора случайных чисел, а затем – для имитации распа-

раллеливания на типичной рабочей станции – использовали бутстреп-выборку для повторного выбора четырех случайных запусков и получали отчет о производительности того из них, который давал наилучший результат перекрестной проверки.

Таблица 4.2. Используемые наборы данных; «Кол-во д. э.» и «Кол-во н. э.» означают количество дискретных и непрерывных атрибутов элементов в наборе данных соответственно

Название	Кол-во д. э.	Кол-во н. э.	Кол-во классов	Кол-во обуч. точек данных	Кол-во тест. точек данных
Dexter	20 000	0	2	420	180
GermanCredit	13	7	2	700	300
Dorothea	100 000	0	2	805	345
Yeast	0	8	10	1038	446
Amazon	10 000	0	49	1050	450
Secom	0	591	2	1096	471
Semeion	256	0	10	1115	478
Car	6	0	4	1209	519
Made Ion	500	0	2	1820	780
KR-vs-KP	37	0	2	2237	959
Abalone	1	7	28	2923	1254
Wine Quality	0	11	11	3425	1469
Waveform	0	40	3	3500	1500
Gisette	5000	0	2	4900	2100
Convex	0	784	2	8000	50 000
CIFAR-10-Small	3072	0	10	10 000	10 000
MNIST Basic	0	784	10	12 000	50 000
Rot. MNIST + BI	0	784	10	12 000	50 000
Shuttle	9	0	7	43 500	14 500
KDD09-Appentency	190	40	2	35 000	15 000
CIFAR-10	3 072	0	10	50 000	10 000

В первых экспериментах мы наблюдали несколько случаев, когда метод SMBO в Auto-WEKA подбирал гиперпараметры, которые имели отличную эффективность при обучении, но плохую обобщаемость на тестовых данных. Чтобы Auto-WEKA мог обнаружить такое переобучение, мы разделили его обучающее множество на два подмножества: 70 % для использования внутри метода SMBO и 30 % проверочных данных, которые мы использовали только после завершения работы метода SMBO.

4.5.1. Эталонные методы

Назначение Auto-WEKA – помочь неспециалистам в использовании методов машинного обучения. Естественный подход, который может применять

такой пользователь, заключается в проведении 10-кратной перекрестной проверки на обучающем множестве для каждого метода с неизмененными гиперпараметрами и выборе классификатора с наименьшей средней ошибкой неправильной классификации по всем проверочным выборкам. Мы будем называть такой метод *Ex-Def*; это лучший выбор, который можно сделать для WEKA с гиперпараметрами по умолчанию.

Для каждого набора данных во втором и третьем столбцах табл. 4.3 представлены наилучшая и наихудшая «производительность оракула» обучателей по умолчанию, когда они были подготовлены на всех обучающих данных и оценены на тестовом наборе. Мы видим, что разрыв между лучшим и худшим обучателем огромен, например уровень ошибочной классификации 4.93 % против 99.24 % на наборе данных Dorothea. Это говорит о том, что для достижения хороших результатов необходим определенный отбор алгоритмов.

Более надежным эталоном, который мы будем использовать, является метод, который в дополнение к выбору обучателя, также оптимально задает его гиперпараметры из заранее определенного набора. Точнее, этот эталонный метод выполняет исчерпывающий поиск по сетке настроек гиперпараметров для каждого из эталонных обучателей, разбивая числовые параметры на три точки. Мы называем этот эталон *поиском по сетке* и отмечаем, что он представляет собой – как подход к оптимизации в совместном пространстве алгоритмов и настроек гиперпараметров – простой алгоритм CASH. Однако он довольно дорогой, требует более 10 000 процессорных часов на каждом из наборов Gisette, Convex, MNIST, Rot MNIST BI и обоих вариантов CIFAR, что делает его неприменимым в большинстве практических приложений. (В отличие от него мы предоставили Auto-WEKA только 120 процессорных часов.)

В табл. 4.2 (столбцы 4 и 5) показана лучшая и худшая «производительность оракула» на тестовом наборе для всех классификаторов, оцененных методом поиска по сетке. Сравнивая эти показатели с показателями по умолчанию, полученными с помощью Ex-Def, мы отмечаем, что в большинстве случаев даже лучший алгоритм WEKA по умолчанию может быть улучшен путем выбора лучших гиперпараметров, иногда довольно существенно: например, в небольшой задаче CIFAR-10 поиск по сетке обеспечил снижение ошибок на 13 % по сравнению с Ex-Def.

В предыдущей работе было продемонстрировано, что при постоянном общем бюджете времени поиск по сетке превосходит случайный поиск в пространстве гиперпараметров [5]. Наш последний эталонный вариант, случайный поиск, реализует такой метод, выбирая алгоритмы и гиперпараметры случайным образом, и рассчитывает их производительность на 10 выборках перекрестной проверки, пока не исчерпает свой бюджет времени. Для каждого набора данных мы сначала использовали 750 процессорных часов для вычисления производительности случайно выбранных комбинаций алгоритмов и гиперпараметров путем перекрестной проверки. Затем мы смоделировали случайный поиск, выбрав из этих результатов комбинации без замены, которые потребовали 120 процессорных часов, и определив комбинацию с наилучшей производительностью.

4.5.2. Результаты производительности, определенные перекрестной проверкой

В средней части табл. 4.3 представлены наши основные результаты. Во-первых, мы отмечаем, что поиск по сетке гиперпараметров всех базовых классификаторов дал лучшие результаты, чем Ex-Def, в 17 случаях из 21, что подчеркивает важность не только выбора правильного алгоритма, но и правильной настройки его гиперпараметров.

Тем не менее мы отмечаем, что дали поиску по сетке очень большой бюджет времени (часто более 10 000 процессорных часов для каждого набора данных, в общей сложности более 10 процессорных лет), что означает невозможность использования на практике в большинстве случаев.

В отличие от этого метода каждому из остальных методов мы давали только 4 × 30 процессорных часов на набор данных; тем не менее они все равно показали значительно лучшую производительность, чем поиск по сетке, превзойдя его в 14 случаях из 21. Случайный поиск превзошел поиск по сетке в 9 случаях из 21, тем самым красноречиво дав понять, что даже исчерпывающий поиск по сетке с большим бюджетом времени не всегда является правильным решением. Стоит заметить, что иногда улучшение производительности Auto-WEKA, по сравнению с базовыми версиями, было значительным, с относительным снижением потерь при перекрестной проверке (в данном случае коэффициента ошибочной классификации) более 10 % в шести случаях из 21.

4.5.3. Результаты тестирования производительности

Приведенные выше результаты показывают, что Auto-WEKA эффективно оптимизирует заданную целевую функцию, однако этого недостаточно, чтобы сделать вывод о том, что она подходит для моделей, которые хорошо обобщают. С ростом числа гиперпараметров алгоритма машинного обучения растет и его склонность к переобучению. Использование перекрестной проверки значительно повышает устойчивость Auto-WEKA к переобучению, но, поскольку пространство гиперпараметров значительно больше, чем у стандартных алгоритмов классификации, важно тщательно изучить, представляет ли переобучение проблему (и если да, то в какой степени).

Для оценки обобщающей способности мы определили комбинацию алгоритма и гиперпараметров A_λ, запустив Auto-WEKA, как и раньше (перекрестная проверка на обучающем наборе), обучили A_λ на всем обучающем наборе, а затем оценили полученную модель на тестовых данных. В правой части табл. 4.3 приведены результаты тестирования, полученные всеми методами.

В целом наблюдаются те же тенденции, что и в случае с перекрестной проверкой: Auto-WEKA превосходит эталонные методы, а поиск по сетке и случайный поиск превосходят Ex-Def. Однако различия в производительности были менее выражены: поиск по сетке показал лучшие результаты, чем Ex-Def, только в 15 случаях из 21, а случайный поиск, в свою очередь, пре-

Таблица 4.3. Производительность при 10-кратной перекрестной проверке и тестовых данных. Ex-Def и Grid Search являются детерминированными. Случайный поиск имеет временной бюджет в 120 процессорных часов. Для Auto-WEKA мы выполнили 25 запусков по 30 ч каждый. Мы приводим результаты как средние потери на 100 000 бутстрэп-образцов при моделировании 4 параллельных запусков. Мы определили потери при тестировании (уровень ошибочной классификации) путем обучения выбранной модели/гиперпараметров на всех 70 % обучающих данных и вычисления точности на ранее неиспользованных 30 % тестовых данных. Жирным шрифтом выделена самая низкая ошибка в блоке сравниваемых методов, которая была статистически значимой

Набор данных	Производительность оракула (%)				Производительность при 10-кратной перекрестной проверке (%)				Производительность на тестовых данных (%)			
	Ex-Def		Поиск по сетке		Ex-Def	Поиск по сетке	Случайный поиск	Auto-WEKA	Ex-Def	Поиск по сетке	Случайный поиск	Auto-WEKA
	Лучший	Худший	Лучший	Худший								
Dexter	7.78	52.78	**3.89**	63.33	10.20	**5.07**	10.60	5.66	8.89	**5.00**	9.18	7.49
GermanCredit	26.00	38.00	**25.00**	68.00	22.45	20.20	20.15	**17.87**	27.33	**26.67**	29.03	28.24
Dorothea	4.93	99.24	**4.64**	99.24	6.03	6.73	8.11	**5.62**	6.96	5.80	**5.22**	6.21
Yeast	40.00	68.99	**36.85**	69.89	39.43	39.71	38.74	35.51	40.45	42.47	43.15	40.67
Amazon	28.44	99.33	**17.56**	99.33	43.94	**36.88**	59.85	47.34	28.44	**20.00**	41.11	33.99
Secom	7.87	14.26	**7.66**	92.13	6.25	6.12	5.24	**5.24**	8.09	8.09	8.03	**8.01**
Semeion	8.18	92.45	**5.24**	92.45	6.52	4.86	6.06	**4.78**	8.18	6.29	6.10	**5.08**
Car	0.77	29.15	**0.00**	46.14	2.71	0.83	**0.53**	0.61	0.77	0.97	**0.01**	0.40
Madelon	**17.05**	50.26	**17.05**	62.69	25.98	26.46	27.95	**20.70**	21.38	21.15	24.29	**21.12**
KR-vs-KP	0.31	48.96	**0.21**	51.04	0.89	0.64	0.63	**0.30**	0.31	1.15	0.58	**0.31**
Abalone	73.18	84.04	**72.15**	92.90	73.33	72.15	72.03	**71.71**	73.18	73.42	74.88	73.51
Wine Quality	36.35	60.99	**32.88**	99.39	38.94	35.23	35.36	**34.65**	37.51	34.06	34.41	33.95
Waveform	14.27	68.80	**13.47**	68.80	12.73	12.45	12.43	**11.92**	14.40	14.66	14.27	14.42
Gisette	2.52	50.91	**1.81**	51.23	3.62	2.59	4.84	**2.43**	2.81	2.40	4.62	**2.24**
Convex	25.96	50.00	**19.94**	71.49	28.68	**22.36**	33.31	25.93	25.96	23.45	31.20	**23.17**
CIFAR-10-Small	65.91	90.00	**52.16**	90.36	66.59	**53.64**	67.33	58.84	65.91	56.94	66.12	**56.87**
MNIST Basic	5.19	88.75	**2.58**	88.75	5.12	**2.51**	5.05	3.75	5.19	**2.64**	5.05	3.64
Rot. MNIST + BI	63.14	88.88	**55.34**	93.01	66.15	**56.01**	68.62	57.86	63.14	57.59	66.40	**57.04**
Shuttle	0.0138	20.8414	**0.0069**	89.8207	0.0328	0.0361	0.0345	**0.0224**	0.0138	0.0414	0.0157	**0.0130**
KDD09-Appentency	1.7400	6.9733	**1.6332**	54.2400	1.8776	1.8735	1.7510	**1.7038**	1.7405	1.7400	1.7400	**1.7358**
C1FAR-10	64.27	90.00	**55.27**	90.00	65.54	**54.04**	69.46	62.36	64.27	63.13	69.72	**61.15**

взошел поиск по сетке в 7 случаях. Auto-WEKA превосходит базовые решения в 15 случаях. Примечательно, что на 12 из 13 самых больших наборов данных Auto-WEKA превосходит наши эталонные решения; мы объясняем это тем, что риск переобучения уменьшается с увеличением размера набора данных. Иногда улучшение производительности Auto-WEKA, по сравнению с другими методами, было значительным: в трех случаях из 21 относительное снижение уровня ошибочной классификации на тестовых данных превышало 16 %.

Как упоминалось ранее, пакет Auto-WEKA использовал только 70 % своего обучающего набора во время оптимизации производительности путем перекрестной проверки, оставляя оставшиеся 30 % для оценки риска переобучения. В любой момент времени метод SMBO в Auto-WEKA отслеживает лидера-претендента (конфигурацию гиперпараметров с самым низким показателем ошибочной классификации при перекрестной проверке). После завершения процедуры SMBO Auto-WEKA извлекает из нее претендентов и вычисляет их обобщающую способность на незнакомых 30 % данных. Затем вычисляется ранговый коэффициент Спирмена между последовательностью результатов обучения (оцененных методом SMBO с помощью перекрестной проверки) и результатами проверки обобщения.

4.6. Заключение

В этой главе мы показали, что сложная задача комбинированного выбора алгоритма и оптимизации гиперпараметров (CASH) может быть решена прикладным полностью автоматизированным инструментом. Это стало возможным благодаря современным методам байесовской оптимизации, которые итеративно строят модели ландшафта алгоритмов/гиперпараметров и используют эти модели для выявления новых точек в пространстве, заслуживающих исследования.

Мы создали инструмент Auto-WEKA, который использует весь спектр алгоритмов обучения в WEKA и позволяет неспециалистам легко создавать высококачественные классификаторы для заданных сценариев применения. Обширное эмпирическое сравнение на 21 известном наборе данных показало, что Auto-WEKA часто превосходит стандартные методы выбора алгоритмов и оптимизации гиперпараметров, особенно на больших наборах данных.

4.6.1. Популярность Auto-WEKA в сообществе

Auto-WEKA был первым методом, использующим байесовскую оптимизацию для автоматического создания высокопараметрической системы машинного обучения одним нажатием кнопки. С момента своего первого выпуска этот пакет скачали многие пользователи, работающие в области производства и науки; версия 2.0, которая интегрируется с менеджером пакетов WEKA, была загружена более 30 000 раз (в среднем более 550 загрузок в неделю). Auto-WEKA находится в стадии активной разработки, регулярно добавляются новые функции.

4.7. Литература

1. Adankon, M., Cheriet, M.: Model selection for the LS-SVM application to handwriting recognition. Pattern Recognition 42(12), 3264–3270 (2009).
2. Bardenet, R., Brendel, M., Kégl, B., Sebag, M.: Collaborative hyperparameter tuning. In: Proc. of ICML-13 (2013).
3. Bengio, Y.: Gradient-based optimization of hyperparameters. Neural Computation 12(8), 1889–1900 (2000).
4. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for Hyper-Parameter Optimization. In: Proc. of NIPS-11 (2011).
5. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. JMLR 13, 281–305 (2012)
6. Biem, A.: A model selection criterion for classification: Application to HMM topology optimization. In: Proc. of ICDAR-03. pp. 104–108. IEEE (2003).
7. Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., Jones, Z.M.: mlr: Machine Learning in R. Journal of Machine Learning Research 17(170), 1–5 (2016), http://jmlr.org/papers/v17/15-066.html.
8. Bozdogan, H.: Model selection and Akaike's information criterion (AIC): The general theory and its analytical extensions. Psychometrika 52(3), 345–370 (1987).
9. Brazdil, P., Soares, C., Da Costa, J.: Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. Machine Learning 50(3), 251–277 (2003).
10. Brochu, E., Cora, V.M., de Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Tech. Rep. UBC TR-2009-23 and arXiv:1012.2599v1, Department of Computer Science, University of British Columbia (2009).
11. Chapelle, O., Vapnik, V., Bengio, Y.: Model selection for small sample regression. Machine Learning (2001).
12. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 2962– 2970. Curran Associates, Inc. (2015), http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf.
13. Frank, A., Asuncion, A.: UCI machine learning repository (2010), http://archive.ics.uci.edu/ml, uRL: http://archive.ics.uci.edu/ml. University of California, Irvine, School of Information and Computer Sciences.
14. Guo, X., Yang, J., Wu, C., Wang, C., Liang, Y.: A novel LS-SVMs hyper-parameter selection based on particle swarm optimization. Neurocomputing 71(16), 3211–3215 (2008).
15. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: an update. ACM SIGKDD Explorations Newsletter 11(1), 10–18 (2009).
16. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proc. of LION-5. pp. 507–523 (2011).

17. Hutter, F., Hoos, H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. JAIR 36(1), 267–306 (2009).
18. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black box functions. Journal of Global Optimization 13, 455–492 (1998).
19. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proc. of IJCAI-95. pp. 1137–1145 (1995).
20. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. Journal of Machine Learning Research 18(25), 1–5 (2017), http://jmlr.org/papers/v18/16-261.html.
21. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto (2009).
22. Leite, R., Brazdil, P., Vanschoren, J.: Selecting classification algorithms with active testing. In: Proc. of MLDM-12. pp. 117–131 (2012).
23. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011), http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf.
24. Maron, O., Moore, A.: Hoeffding races: Accelerating model selection search for classification and function approximation. In: Proc. of NIPS-94. pp. 59–66 (1994).
25. McQuarrie, A., Tsai, C.: Regression and time series model selection. World Scientific (1998).
26. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Meta-learning by landmarking various learning algorithms. In: Proc. of ICML-00. pp. 743–750 (2000).
27. Schonlau, M., Welch, W.J., Jones, D.R.: Global versus local search in constrained optimization of computer models. In: Flournoy, N., Rosenberger, W., Wong, W. (eds.) New Developments and Applications in Experimental Design, vol. 34, pp. 11–25. Institute of Mathematical Statistics, Hayward, California (1998).
28. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Proc. of NIPS-12 (2012).
29. Srinivas, N., Krause, A., Kakade, S., Seeger, M.: Gaussian process optimization in the bandit setting: No regret and experimental design. In: Proc. of ICML-10. pp. 1015–1022 (2010).
30. Strijov, V., Weber, G.: Nonlinear regression model generation using hyperparameter optimization. Computers & Mathematics with Applications 60(4), 981–988 (2010).
31. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: KDD (2013).
32. Vilalta, R., Drissi, Y.: A perspective view and survey of meta-learning. Artif. Intell. Rev. 18(2), 77–95 (Oct 2002).
33. Zhao, P., Yu, B.: On model selection consistency of lasso. JMLR 7, 2541–2563 (Dec 2006).

Глава 5

Проект Hyperopt-sklearn

Брент Комер, Джеймс Бергстра, Крис Элиасмит
Центр теоретической нейрологии, Университет Ватерлоо, Ватерлоо, ОН, Канада, e-mail: bjkomer@uwaterloo.ca

Hyperopt-sklearn – это проект, направленный на разработку системы автоматической настройки алгоритмов библиотеки машинного обучения scikit-learn. Следуя примеру Auto-Weka, мы считаем, что выбор классификатора и даже выбор модуля предварительной обработки можно рассматривать как одну большую проблему оптимизации гиперпараметров. Мы используем Hyperopt для определения пространства поиска, которое охватывает множество стандартных компонентов (например, SVM, RF, KNN, PCA, TFIDF) и общих схем их комбинирования. Используя алгоритмы поиска в Hyperopt и стандартные эталонные наборы данных (MNIST, 20-Newsgroups, Convex Shapes), мы показываем, что поиск в этом пространстве практичен и эффективен. На момент написания этой главы нам удалось превзойти самые известные оценки для пространства моделей на наборах MNIST и Convex Shapes.

5.1. Введение

По сравнению с глубокими сетями такие алгоритмы, как метод опорных векторов (SVM) и случайные леса (RF), имеют достаточно малое количество гиперпараметров, чтобы ручная настройка и сеточный или случайный поиск давали удовлетворительные результаты. Однако с объективной точки зрения часто нет особых причин использовать именно SVM или RF, когда есть и другие модели, приемлемые в вычислительном отношении. Практикующий специалист, не ориентирующийся в моделях, может просто предпочесть ту из них, которая обеспечивает б*о*льшую точность. В этом свете выбор классификатора можно рассматривать как гиперпараметр наряду с параметром C в SVM и максимальной глубиной дерева в RF. Кроме того, выбор компонентов предварительной обработки и их конфигурации также можно рассматривать как часть задачи выбора модели/оптимизации гиперпараметров.

Проект Auto-Weka [19] первым показал, что даже в рамках настройки гиперпараметров может быть найдена целая библиотека подходов машинного обучения (Weka [8]). Однако Weka – это библиотека Java с лицензией GPL, и она не была написана с учетом масштабируемости, поэтому мы считаем, что существует необходимость в альтернативных проектах. Scikit-learn [16] – еще одна библиотека алгоритмов машинного обучения. Она написана на языке Python (со модулями на C для большего быстродействия) и имеет лицензию BSD. Scikit-learn широко используется в научном сообществе Python и поддерживает многие области применения машинного обучения.

В этой главе мы предлагаем вниманию читателей Hyperopt-sklearn – проект, который предоставляет пользователям Python и scikit-learn преимущества автоматической настройки алгоритмов. Hyperopt-sklearn использует Hyperopt [3] для описания пространства поиска возможных конфигураций компонентов scikit-learn, включая модули предварительной обработки, классификации и регрессии. Одной из основных особенностей этого проекта является наличие интерфейса, знакомого пользователям scikit-learn. Благодаря такому подходу для применения поиска гиперпараметров к существующей кодовой базе пользователей требуются лишь незначительные изменения. Эта глава начинается с описания библиотеки Hyperopt и пространства конфигураций, которое она использует в scikit-learn, затем приводятся примеры использования и результаты экспериментов с этим программным обеспечением.

Эта глава является расширенной версией нашей статьи 2014 г. о Hyperopt-sklearn, представленной на семинаре 2014 ICML по AutoML [10].

5.2. Оптимизация с помощью Hyperopt

Библиотека Hyperopt [3] предлагает алгоритмы оптимизации для пространств поиска, возникающих при настройке алгоритмов. Эти пространства характеризуются различными типами переменных (непрерывные, порядковые, категориальные), различными профилями чувствительности (например, равномерное или логарифмическое масштабирование) и наличием условий (когда есть выбор между двумя классификаторами, параметры одного классификатора не имеют значения при выборе другого классификатора). Чтобы использовать Hyperopt, пользователь должен определить три вещи:

- область поиска;
- целевую функцию;
- алгоритм оптимизации.

Область поиска задается с помощью случайных величин, распределения которых должны быть выбраны таким образом, чтобы наиболее перспективные комбинации имели высокую априорную вероятность. Область поиска может включать операторы и функции Python, которые объединяют случайные переменные в более удобные структуры данных для целевой функции. Любая условная структура определяется в пределах этой области. Целевая функция отображает совместную выборку этих случайных переменных в скалярную оценку, которую алгоритм оптимизации пытается минимизировать.

Пример задания области поиска с использованием Hyperopt показан на рис. 5.1.

```
from hyperopt import hp

space = hp.choice('my_conditional',
[
    ('case 1', 1 + hp.lognormal('c1', 0, 1)),
    ('case 2', hp.uniform('c2', -10, 10))
    ('case 3', hp.choice('c3', ['a', 'b', 'c']))
])
```

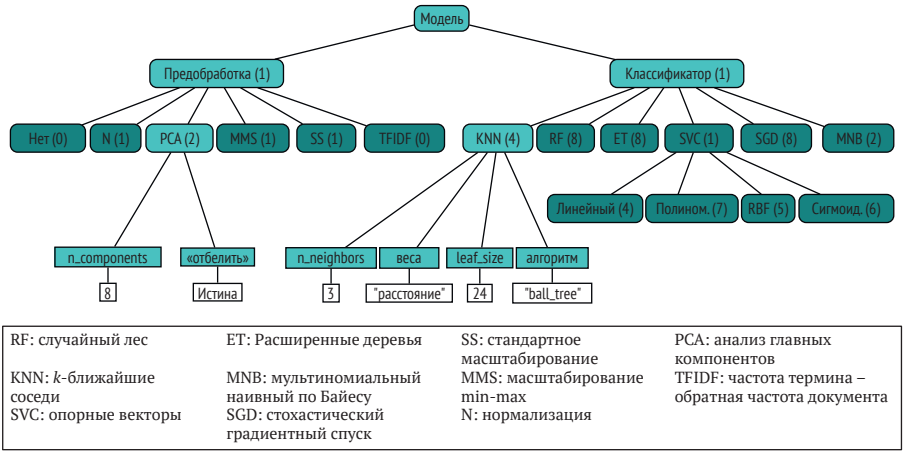


Рис. 5.1 ❖ Пример пространства поиска Hyperopt-sklearn, состоящего из этапа предварительной обработки, за которым следует классификатор. Существует шесть возможных модулей предварительной обработки и шесть возможных классификаторов. Выбор модели в этом пространстве конфигураций означает выбор путей в процессе выборки предков. Светло-голубые узлы представляют модель (PCA, *k*-ближайшие соседи). Белые листовые узлы внизу содержат примеры значений родительских гиперпараметров. Количество активных гиперпараметров в модели равно сумме чисел в выделенных родительских ячейках. Для комбинации PCA + KNN активировано восемь гиперпараметров

Здесь доступны четыре параметра – один для выбора случая, который активен, и по одному для каждого из трех случаев. К первому случаю относится положительный параметр чувствительности к логарифмическому масштабированию. Ко второму случаю относится вещественное число. Третий случай содержит категориальный параметр с тремя вариантами.

После выбора области поиска, целевой функции и алгоритма оптимизации функция `fmin` библиотеки Hyperopt выполняет оптимизацию и сохраняет результаты поиска в базе данных (например, в простом списке Python или в экземпляре MongoDB). Функция `fmin` выполняет простой анализ, находя конфигурацию с наилучшими характеристиками, и возвращает ее вызывающей стороне. Если используется бэкенд MongoDB, вызов `fmin` может задействовать несколько воркеров для реализации параллельного выбора модели на вычислительном кластере.

5.3. Выбор модели в scikit-learn как задача поиска

Выбор модели – это процесс оценки того, какая модель машинного обучения работает лучше всего из потенциально бесконечного множества вариантов. Если рассматривать выбор модели как задачу оптимизации, область поиска – это множество допустимых значений параметров конфигурации (гиперпараметров) модели машинного обучения. Объективная функция обычно представляет собой меру успеха (например, точность, F1-Score и т. д.) на отработанных примерах. Часто отрицательная мера успеха (потеря) используется для постановки задачи минимизации, а для получения более надежной итоговой оценки применяется перекрестная проверка. Практики обычно выполняют эту оптимизацию вручную, с помощью поиска по сетке или случайного поиска. В этой главе мы обсудим оптимизцию с помощью библиотеки Hyperopt. В общих чертах подход состоит в том, чтобы задать пространство поиска со случайными гиперпараметрами, использовать scikit-learn для реализации целевой функции, которая выполняет обучение и проверку модели, и Hyperopt для оптимизации гиперпараметров.

Scikit-learn содержит множество реализаций алгоритмов для обучения на основе данных (классификация или регрессия), а также алгоритмов для предварительной обработки данных в векторы, ожидаемые этими алгоритмами обучения. В перечень классификаторов входят, например, алгоритмы *k*-ближайших соседей, опорных векторов и случайного леса. Алгоритмы предварительной обработки включают такие преобразования, как Z-масштабирование по компонентам (нормализатор) и анализ главных компонент (principle components analysis, PCA). Полный алгоритм классификации обычно содержит ряд шагов предварительной обработки, за которыми следует классификатор. По этой причине scikit-learn предоставляет конвейер данных (объект `pipeline`) для представления и использования последовательности шагов предварительной обработки и классификации, как если бы они были одним компонентом (обычно с API, аналогичным классификатору). Хотя пакет Hyperopt-sklearn формально не использует объект `pipeline` от scikit-learn, он реализует связанную с ним функциональность. Hyperopt-sklearn предоставляет параметризацию пространства поиска над конвейерами, т. е. последовательностями шагов предварительной обработки и классификаторов или регрессоров.

На момент написания данной главы пространство конфигураций включало 24 классификатора, 12 регрессоров и 7 методов предобработки. Поскольку речь идет о проекте с открытым исходным кодом, это пространство, вероятно, будет расширяться в будущем по мере того, как все больше пользователей будут вносить свой вклад. В первом релизе проекта было доступно только подмножество пространства поиска, состоящее из шести классификаторов и пяти алгоритмов предварительной обработки. Это пространство применялось для первоначального анализа производительности, как показано на рис. 5.1. В целом эта параметризация содержит 65 гиперпараметров: 15 булевых переменных, 14 категориальных, 17 дискретных и 19 вещественных.

Хотя общее число гиперпараметров в полном пространстве конфигураций велико, число активных гиперпараметров, описывающих одну модель, гораздо меньше: например, модель, состоящая из алгоритмов PCA и RandomForest, будет иметь только 12 активных гиперпараметров (1 для выбора предобработки, 2 внутренних для PCA, 1 для выбора классификатора и 8 внутренних для RandomForest). Язык описания Hyperopt позволяет нам различать условные гиперпараметры (которым всегда должно быть присвоено корректное значение) и обычные неусловные гиперпараметры (которые могут оставаться без присвоенных значений, если они не используются). Мы используем этот факт, чтобы алгоритмы поиска Hyperopt не тратили время на изучение, например, гиперпараметров случайного леса, которые не влияют на производительность модели опорных векторов. Даже внутри классификаторов встречаются условные параметры: KNN имеет условные параметры, зависящие от метрики расстояния, а LinearSVC имеет три бинарных параметра (потери, штраф и удвоение), которые допускают только четыре допустимых совместных назначения. Hyperopt-sklearn также имеет «черный список» пар предобработчик–классификатор, которые не работают вместе. Например, PCA и MinMaxScaler несовместимы с MultinomialNB, TF-IDF может использоваться только для текстовых данных, а древовидные классификаторы не совместимы с разреженными признаками, полученными предобработчиком TF-IDF. Если допустить, что каждый гиперпараметр, представленный вещественным числом, дискретизирован всего на 10 допустимых значений, и учесть эти условные гиперпараметры, то поиск по сетке в нашем пространстве поиска все равно потребует недостижимого количества вычислений (порядка 10^{12}).

Наконец, пространство поиска становится задачей оптимизации и в том случае, если мы определяем цель поиска со скалярным значением. По умолчанию Hyperopt-sklearn использует метод оценки scikit-learn на проверочных данных для определения критерия поиска. Для классификаторов это так называемый критерий нуль-единичной потери (zero-one loss): количество правильных предсказаний меток среди данных, которые были исключены из набора данных, используемых для обучения (а также из данных, используемых для тестирования после процесса поиска при выборе модели).

5.4. Пример использования

Следуя традициям scikit-learn, пакет Hyperopt-sklearn предоставляет класс `Estimator` с методами `fit` и `predict`. Метод `fit` этого класса выполняет оптимизацию гиперпараметров, а после ее завершения метод `predict` применяет наилучшую модель к заданным тестовым данным. Каждый проход оценивания во время оптимизации выполняет обучение на большой части обучающего множества, оценивает точность на проверочном множестве и возвращает оценку оптимизатору. В конце поиска лучшая конфигурация повторно обучается на полном наборе данных для создания классификатора, который обрабатывает последующие вызовы метода `predict`.

Одной из важных целей создания Hyperopt-sklearn является простота изучения и использования. Поэтому синтаксис для описания процесса подгонки классификатора к данным и составления прогнозов очень похож на scikit-learn. Ниже показан простейший пример кода:

```
from hpsklearn import HyperoptEstimator

# Загрузка данных
train_data, train_label, test_data, test_label =
    load_my_data()

# Создание объекта оценивателя
estim = HyperoptEstimator()
# Поиск пространства классификаторов и шагов предобработки
# и их соответствующих гиперпараметров в scikit-learn
# для подгонки модели к данным
estim.fit(train_data, train_label)

# Выдача прогноза с помощью оптимизированной модели
prediction = estim.predict(test_data)

# Вычисление точности классификатора на текущем наборе данных
score = estim.score(test_data, test_label)

# Возвращаем объекты классификатора и шагов предобработки
model = estim.best_model()
```

Объект `HyperoptEstimator` содержит информацию о том, в каком пространстве искать, а также как искать. Он может быть настроен на использование различных алгоритмов поиска гиперпараметров, а также поддерживает использование комбинации алгоритмов. Здесь можно использовать любой алгоритм, который поддерживает тот же интерфейс, что и алгоритмы в hyperopt. Здесь же пользователь может указать максимальное количество вычислений оценок функции, а также тайм-аут в секундах между запусками.

```
from hpsklearn import HyperoptEstimator
from hyperopt import tpe
estim = HyperoptEstimator(algo=tpe.suggest,
                  max_evals=150,
                  trial_timeout=60)
```

Каждый алгоритм поиска может привносить в пространство поиска свое собственное смещение, и оттого не всегда ясно, что одна конкретная стратегия является лучшей во всех случаях. Иногда полезно использовать смесь алгоритмов поиска.

```
from hpsklearn import HyperoptEstimator
from hyperopt import anneal, rand, tpe, mix

# задаем алгоритм, применяющий случайный поиск в 5 % случаев,
# TPE в 75 % случаев и отжиг в остальных 20 %
mix_algo = partial(mix.suggest, p_suggest=[
    (0.05, rand.suggest),
```

```
    (0.75, tpe.suggest),
    (0.20, anneal.suggest)])
estim = HyperoptEstimator(algo=mix_algo,
                  max_evals=150,
                  trial_timeout=60)
```

Эффективный поиск по всему пространству классификаторов, доступных в scikit-learn, может потребовать много времени и вычислительных ресурсов. Иногда вам может быть известно определенное подпространство моделей, которое вас больше интересует. С помощью hyperopt-sklearn можно задать более узкое пространство поиска, что позволит исследовать его более глубоко.

```
from hpsklearn import HyperoptEstimator, svc

# ограничиваем поиск только моделями SVC models
estim = HyperoptEstimator(classifier=svc('my_svc'))
```

Также можно использовать комбинации различных пространств.

```
from hpsklearn import HyperoptEstimator, svc, knn
from hyperopt import hp

# ограничиваем пространство только случайным лесом,
# k-ближайшими соседями и моделями SVC.
clf = hp.choice('my_name',
    [random_forest('my_name.random_forest'),
     svc('my_name.svc'),
     knn('my_name.knn')])
estim = HyperoptEstimator(classifier=clf)
```

Модель опорных векторов, предоставляемая scikit-learn, позволяет использовать несколько различных ядер (линейное, RBF, полиномиальное, сигмоидное). Смена ядра может сильно повлиять на производительность модели, и каждое ядро имеет свои уникальные гиперпараметры. Чтобы учесть это, hyperopt-sklearn рассматривает каждый выбор ядра как уникальную модель в пространстве поиска. Если вы уже знаете, какое ядро лучше всего подходит для ваших данных, или вам просто интересно исследовать модели с определенным ядром, вы можете указать его напрямую:

```
from hpsklearn import HyperoptEstimator, svc_rbf
estim = HyperoptEstimator(classifier=svc_rbf('my_svc'))
```

Также можно указать, какие ядра вас интересуют, передав `svc` список:

```
from hpsklearn import HyperoptEstimator, svc
estim = HyperoptEstimator(
    classifier=svc('my_svc',
               kernels=['linear',
                        'sigmoid']))
```

Как и пространство классификаторов, пространство модулей предварительной обработки тоже допускает точную настройку. Несколько последова-

тельных этапов предварительной обработки может быть задано с помощью упорядоченного списка. Пустой список означает, что никакой предварительной обработки данных не будет:

```
from hpsklearn import HyperoptEstimator, pca
estim = HyperoptEstimator(preprocessing=[pca('my_pca')])
```

Здесь также можно использовать комбинации различных пространств:

```
from hpsklearn import HyperoptEstimator, tfidf, pca
from hyperopt import hp
preproc = hp.choice('my_name',
     [[pca('my_name.pca')],
     [pca('my_name.pca'), normalizer('my_name.norm')]
     [standard_scaler('my_name.std_scaler')],
     []])
estim = HyperoptEstimator(preprocessing=preproc)
```

Некоторые типы предварительной обработки будут работать только с определенными типами данных. Например, предобработчик `TfidfVectorizer`, который предоставляет scikit-learn, предназначен для работы с текстовыми данными и не подходит для других типов данных. Чтобы исключить несоответствие между предобработчиком и данными, hyperopt-sklearn поставляется с несколькими заранее определенными пространствами классификаторов и предобработчиков, адаптированных к определенным типам данных.

```
from hpsklearn import HyperoptEstimator, \
                any_sparse_classifier, \
                any_text_preprocessing
from hyperopt import tpe
   estim = HyperoptEstimator(
   algo=tpe.suggest,
   classifier=any_sparse_classifier('my_clf')
   preprocessing=any_text_preprocessing('my_pp')
   max_evals=200,
   trial_timeout=60)
```

В приведенных выше примерах кода происходит перебор всех гиперпараметров, доступных модели. Также можно указать значения конкретных гиперпараметров, и они будут оставаться постоянными во время поиска. Это может быть полезно, например, если вы знаете, что хотите использовать отбеленные данные PCA и SVM с полиномиальным ядром третьей степени:

```
from hpsklearn import HyperoptEstimator, pca, svc_poly
estim = HyperoptEstimator(
       preprocessing=[pca('my_pca', whiten=True)],
       classifier=svc_poly('my_poly', degree=3))
```

Также можно задать диапазоны отдельных параметров. Это делается с помощью стандартного синтаксиса hyperopt. Они отменяют значения по умолчанию, определенные в hyperopt-sklearn:

```
from hpsklearn import HyperoptEstimator, pca, sgd
from hyperopt import hp
import numpy as np

sgd_loss = hp.pchoice('loss',
                [(0.50, 'hinge'),
                 (0.25, 'log'),
                 (0.25, 'huber')])
sgd_penalty = hp.choice('penalty',
                 ['l2', 'elasticnet'])
sgd_alpha = hp.loguniform('alpha',
                   low=np.log(1e-5),
                   high=np.log(1) )
estim = HyperoptEstimator(
       classifier=sgd('my_sgd',
                  loss=sgd_loss,
                  penalty=sgd_penalty,
                  alpha=sgd_alpha) )
```

Все компоненты, доступные пользователю, можно найти в файле `components.py`. Полный рабочий пример использования hyperopt-sklearn при поиске модели для набора данных 20 новостных групп показан ниже:

```
from hpsklearn import HyperoptEstimator, tfidf,
    any_sparse_classifier
from sklearn.datasets import fetch_20newsgroups
from hyperopt import tpe
import numpy as np
# Загрузка данных и разделение на обучающий и тестовый наборы
train = fetch_20newsgroups(subset='train')
test = fetch_20newsgroups(subset='test')
X_train = train.data
y_train = train.target
X_test = test.data
y_test = test.target
estim = HyperoptEstimator(
       classifier=any_sparse_classifier('clf'),
       preprocessing=[tfidf('tfidf')],
       algo=tpe.suggest,
       trial_timeout=180)
estim.fit(X_train, y_train)
print(estim.score(X_test, y_test))
print(estim.best_model())
```

5.5. Эксперименты

Мы провели эксперименты на трех наборах данных, чтобы убедиться, что hyperopt-sklearn может находить точные модели на различных наборах данных за разумное время. Результаты были получены на трех наборах данных: MNIST, 20-Newsgroups и Convex Shapes. MNIST – это широко известный набор

данных, состоящий из 70 тыс. изображений 28×28 рукописных цифр в градациях серого [12]. 20-Newsgroups – это набор данных для классификации 20 тыс. сообщений новостных групп [13] (в наших экспериментах мы не удаляли заголовки). Convex Shapes – задача бинарной классификации, заключающаяся в различении выпуклых областей белого цвета на небольших (32×32) черно-белых изображениях [11].

На рис. 5.2 (слева) показано, что штраф за обширный поиск отсутствует. Мы выполнили оптимизацию на глубину до 300 оценок гиперпараметров, перебирая подмножество пространства поиска, изображенного на рис. 5.1, и сравнили качество решения со специализированным поиском конкретных типов классификаторов (включая наиболее известные классификаторы).

Рисунок 5.2 (справа) показывает, что поиск может находить различные хорошие модели. Этот рисунок был построен путем запуска hyperopt-sklearn с различными начальными условиями (количество оценок, выбор алгоритма оптимизации и затравка генератора случайных чисел) и отслеживания того, какая окончательная модель была выбрана после каждого запуска. Хотя модели опорных векторов всегда были одними из лучших, параметры наиболее удачных SVM выглядели очень по-разному в разных наборах данных. Например, на наборах данных изображений (MNIST и Convex) выбранные SVM никогда не имели сигмоидного или линейного ядра, в то время как на 20 новостных группах линейное и сигмоидное ядро часто оказывались лучшими.

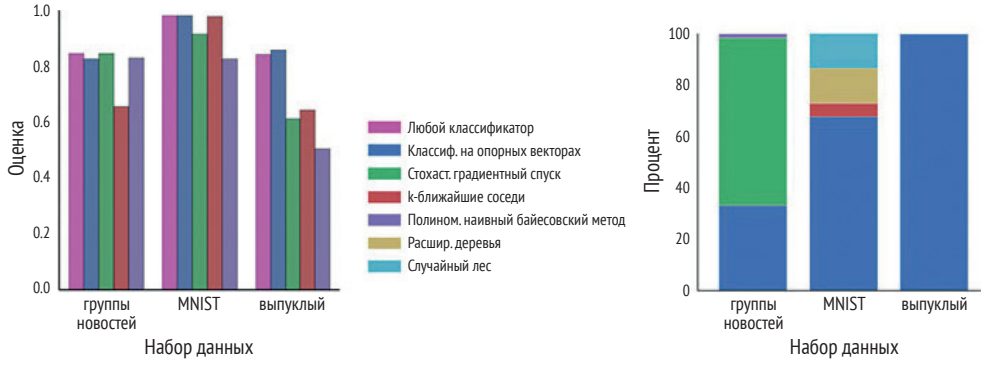


Рис. 5.2 ❖ *Слева*: наилучшая производительность модели. Для каждого набора данных поиск в полном пространстве конфигураций («Любой классификатор») дал результаты, примерно равные производительности поиска, ограниченного лучшим типом классификатора. Каждый столбик гистограммы представляет собой оценку, полученную при поиске, ограниченном этим конкретным классификатором. Для случая «Любой классификатор» нет ограничений на пространство поиска. Во всех случаях было выполнено 300 оценок гиперпараметров. Применялась оценка F1 для набора 20-Newsgroups и точность для MNIST и Convex Shapes. *Справа*: распределение выбора моделей. Рассматривая лучшие модели из всех оптимизационных прогонов, выполненных на полном пространстве поиска (любой классификатор, использование различных начальных условий и различных алгоритмов оптимизации), мы видим, что с различными наборами данных лучше всего справляются разные классификаторы. Модель SVC была единственным классификатором, выбранным в качестве лучшего варианта для Convex Shapes, и часто оказывалась лучшей для MNIST и 20-Newsgroups, однако оптимальные параметры SVC сильно отличались в разных наборах данных

Иногда исследователи, не знакомые с методами машинного обучения, могут просто использовать параметры по умолчанию доступных им классификаторов. Чтобы определить эффективность hyperopt-sklearn как замены для такого подхода, было проведено сравнение производительности параметров scikit-learn по умолчанию и небольшого поиска (25 оценок) в пространстве hyperopt-sklearn по умолчанию. Результаты на наборе данных 20-Newsgroups показаны на рис. 5.3. Улучшение производительности, по сравнению с базовым уровнем, наблюдается во всех случаях, что говорит о том, что эта методика поиска является ценной даже при небольшом вычислительном бюджете.

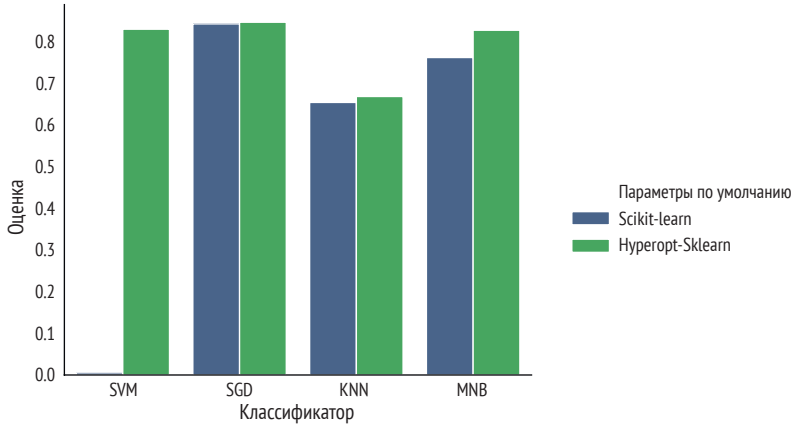


Рис. 5.3 ❖ Сравнение оценки F1 на наборе данных 20-Newsgroups с использованием параметров по умолчанию scikit-learn или пространства поиска по умолчанию hyperopt-sklearn. Результаты hyperopt-sklearn были получены в результате одного прогона с 25 оценками, ограниченного либо классификатором на основе модели опорных векторов, либо стохастическим градиентным спуском, либо *k*-ближайшими соседями, либо мультиномиальным наивным байесовским алгоритмом

5.6. Текущее состояние и перспективные направления исследований

В табл. 5.1 приведены оценки тестового набора лучших моделей, найденных с помощью перекрестной валидации, а также некоторые примеры из предыдущих работ. Оценки hyperopt-sklearn относительно высоки на каждом наборе данных, следовательно, при параметризации hyperopt-sklearn алгоритмы оптимизации Hyperopt вполне могут конкурировать с подготовленными экспертами-людьми.

Модель с наилучшими показателями на наборе данных MNIST использует глубокие нейронные сети. Небольшие рецептивные поля сверточных нейронов-победителей образуют большую сеть. Каждый нейронный столбец становится экспертом по входным данным, предварительно обработанным различными способами, и среднее предсказание 35 глубоких нейронных столбцов приходит к единому окончательному предсказанию [4]. Эта модель намного более продвинутая, чем те, которые доступны в scikit-learn. Ранее лучшей известной моделью в пространстве поиска scikit-learn была радиально-базисная SVM на центрированных данных, которая набрала 98.6 %, и hyperopt-sklearn соответствует этой производительности [15].

Таблица 5.1. Оценки hyperopt-sklearn по сравнению с данными из литературы на трех наборах данных, использованных в наших экспериментах

MNIST		**20-Newsgroups**		**Convex Shapes**	
Метод	**Точность, %**	**Метод**	**Оценка F1**	**Метод**	**Точность, %**
Групповые сверточные сети	99.8	CFC	0.928	**hyperopt-sklearn**	**88.7**
hyperopt-sklearn	**98.7**	**hyperopt-sklearn**	**0.856**	hp-dbnet	84.6
Поиск по сетке LibSVM	98.6	SVMTorch	0.848	dbn-3	81.4
Расширяемые деревья	98.5	LibSVM	0.843		

На наборе MNIST hyperopt-sklearn является одним из лучших методов, не использующих специфические для изображений знания предметной области (эти и другие результаты можно найти на сайте http://yann.lecun.com/exdb/mnist/). На наборе 20-Newsgroups hyperopt-sklearn конкурирует с аналогичными подходами, предложенными в литературе (оценки взяты из [7]). Для набора данных 20-Newsgroups оценка, полученная hyperopt-sklearn, является средневзвешенной оценкой F1, полученной sklearn. Другие подходы, представленные здесь, используют макросреднюю оценку F1. На наборе Convex Shapes hyperopt-sklearn превосходит ранее опубликованные методы автоматической настройки алгоритмов [6] и ручной настройки [11].

Модель CFC, которая показала хорошие результаты на наборе данных классификации документов 20 новостных групп, является классификатором типа Class-Feature-Centroid. Центроидные подходы обычно уступают SVM из-за того, что центроиды, найденные в процессе обучения, далеки от оптимального расположения. Представленный здесь метод CFC использует центроид, построенный на основе индексов межклассовой и внутриклассовой корреляций. Он использует новую комбинацию этих индексов вместе с денормализованной косинусной мерой для расчета оценки сходства между центроидом и вектором текста [7]. Этот тип модели в настоящее время не реализован в hyperopt-sklearn, и наши эксперименты показывают, что существующие компоненты hyperopt-sklearn не могут быть собраны так, чтобы соответство-

вать его уровню производительности. Возможно, когда такая модель будет реализована, Hyperopt сможет найти набор параметров, обеспечивающий еще большую точность классификации.

На наборе данных Convex Shapes наши эксперименты с hyperopt-sklearn выявили более точную модель, чем та, которая может существовать в произвольном пространстве поиска, как считалось ранее, не говоря уже о пространстве поиска с такими стандартными компонентами. Этот результат подчеркивает сложность и важность поиска гиперпараметров.

Hyperopt-sklearn предоставляет много возможностей для дальнейшей работы: в пространство поиска можно включить больше классификаторов и модулей предварительной обработки, и существует больше способов комбинировать даже существующие компоненты. Другие типы данных требуют иной предварительной обработки, и существуют другие задачи прогнозирования, помимо классификации. Прежде чем расширять пространство поиска, необходимо убедиться, что преимущества новых моделей перевешивают трудности поиска в более широком пространстве. Иногда scikit-learn оперирует параметрами, которые являются скорее деталями реализации, чем фактическими гиперпараметрами, влияющими на обучение (например, `algorithm` и `leaf_size` в модели *k*-ближайших соседей). Следует внимательно отнестись к определению этих параметров в каждой модели; возможно, в процессе исследования их придется обрабатывать по-разному.

Пользователь может добавить свой собственный классификатор в пространство поиска, если он соответствует интерфейсу scikit-learn. В настоящее время для этого требуется углубленное понимание исходного кода hyperopt-sklearn, и было бы неплохо усовершенствовать механизм добавления классификаторов, чтобы от пользователя требовался минимум усилий. Пользователь также может указать альтернативные методы оценки, помимо заданных по умолчанию, поскольку бывают случаи, когда они не подходят для решения конкретной задачи.

Мы показали, что алгоритмы случайного поиска, отжига и TPE делают hyperopt-sklearn достаточно эффективным инструментом, но медленная сходимость на рис. 5.4 говорит о том, что другие алгоритмы оптимизации могут быть более эффективными. Разработка алгоритмов байесовской оптимизации является областью активных исследований, и мы с нетерпением ждем возможности посмотреть, как другие алгоритмы поиска взаимодействуют с пространством поиска hyperopt-sklearn. Гиперпараметрическая оптимизация привела к появлению своеобразного искусства согласования пространств поиска с сильными сторонами поисковых алгоритмов.

Время, затрачиваемое на поиск, имеет большое практическое значение, и в настоящее время hyperopt-sklearn тратит значительное количество времени на оценку точек, которые являются бесперспективными. Техника раннего распознавания плохих результатов может значительно ускорить поиск [5, 18].

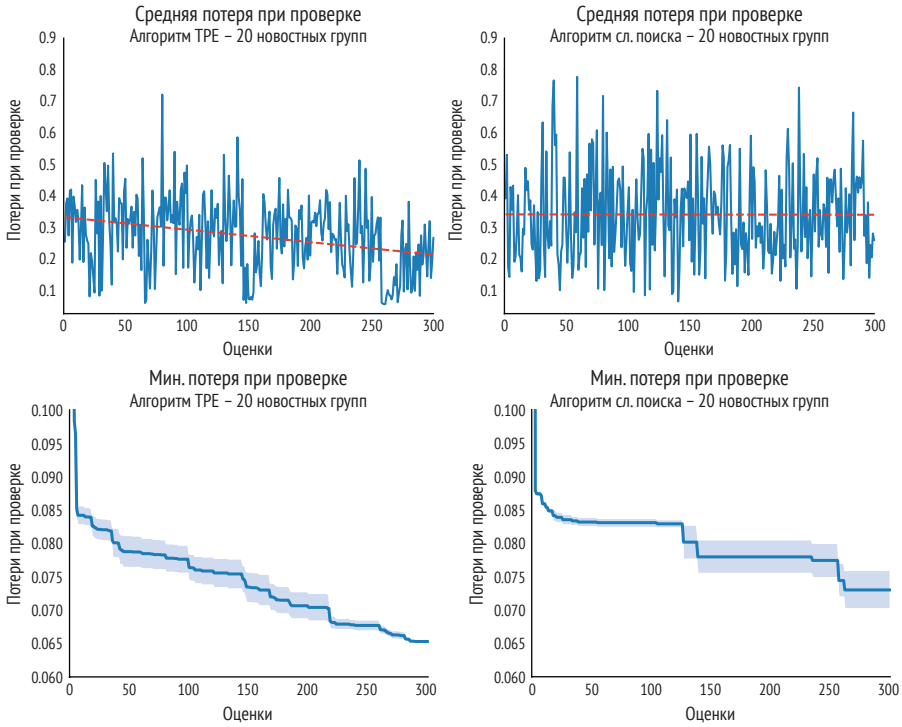


Рис. 5.4 ❖ Проверочные потери моделей, найденных для каждой последовательной оценки параметров с использованием набора данных 20-Newsgroups и области поиска Any Classifier. *Вверху слева*: средняя потеря на каждом шаге при различных затравках генератора случайных чисел для алгоритма TPE. Нисходящая тенденция указывает на то, что более перспективные области со временем исследуются чаще. *Вверху справа*: средняя потеря для случайного алгоритма. Горизонтальное расположение линии тренда свидетельствует об отсутствии обучения на предыдущих испытаниях. Большой разброс в производительности при разных оценках указывает на то, что задача очень чувствительна к настройке гиперпараметров. *Внизу слева*: минимальная потеря найденных на данный момент моделей для алгоритма TPE. На наборе 20-Newsgroups в течение 300 итераций достигается постепенный прогресс, но признаки сходимости не просматриваются. *Внизу справа*: минимальные потери для случайного алгоритма. Наблюдается довольно быстрый прогресс в течение приблизительно первых 40 оценок, а затем он надолго затихает. Улучшение все еще происходит, но со временем становится менее вероятным

5.7. Заключение

В этой главе был представлен hyperopt-sklearn – пакет Python для автоматической настройки стандартных алгоритмов машинного обучения, предоставляемых библиотекой scikit-learn. Hyperopt-sklearn предоставляет унифи-

цированный интерфейс к большому подмножеству алгоритмов машинного обучения, доступных в scikit-learn, и с помощью функций оптимизации способен как соперничать, так и превосходить экспертов-людей в настройке алгоритмов. Мы надеемся, что этот пакет даст практикующим специалистам полезный инструмент для разработки систем машинного обучения, а исследователям машинного обучения – ориентиры для будущих исследований по настройке алгоритмов.

Благодарности. Это исследование было поддержано программой NSERC Banting Fellowship, программой NSERC Engage и компанией D-Wave Systems. Мы также благодарим Христиана Богоевского за первые идеи по взаимодействию между Hyperopt и scikit-learn.

5.8. Литература

1. J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl. Algorithms for hyperparameter optimization, NIPS, 24:2546–2554, 2011.
2. J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, In Proc. ICML, 2013a.
3. J. Bergstra, D. Yamins, and D. D. Cox. Hyperopt: A Python library for optimizing the hyperparameters of machine learning algorithms, SciPy'13, 2013b.
4. D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column Deep Neural Networks for Image Classification, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3642– 3649. 2012.
5. T. Domhan, T. Springenberg, F. Hutter. Extrapolating Learning Curves of Deep Neural Networks, ICML AutoML Workshop, 2014.
6. K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters, NIPS workshop on Bayesian Optimization in Theory and Practice, 2013.
7. H. Guan, J. Zhou, and M. Guo. A class-feature-centroid classifier for text categorization, Proceedings of the 18th international conference on World wide web, 201–210. ACM, 2009.
8. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update, ACM SIGKDD explorations newsletter, 11(1):10–18, 2009.
9. F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration, LION-5, 2011. Extended version as UBC Tech report TR-2010-10.
10. B. Komer, J. Bergstra, and C. Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn, ICML AutoML Workshop, 2014.
11. H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation, ICML, 473–480, 2007.

12. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition, Proceedings of the IEEE, 86(11):2278–2324, November 1998.
13. T. Mitchell. 20 newsgroups data set, http://qwone.com/jason/20Newsgroups/, 1996.
14. J. Mockus, V. Tiesis, and A. Zilinskas. The application of Bayesian methods for seeking the extremum, L.C.W. Dixon and G.P. Szego, editors, Towards Global Optimization, volume 2, pages 117–129. North Holland, New York, 1978.
15. The MNIST Database of handwritten digits: http://yann.lecun.com/exdb/mnist/.
16. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12:2825–2830, 2011.
17. J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms, Neural Information Processing Systems, 2012.
18. K. Swersky, J. Snoek, R.P. Adams. Freeze-Thaw Bayesian Optimization, arXiv:1406.3896, 2014.
19. C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. AutoWEKA: Automated selection and hyper-parameter optimization of classification algorithms, KDD 847–855, 2013.

Глава 6

Auto-sklearn – эффективное и надежное автоматизированное машинное обучение

Маттиас Фойрер (✉), **Аарон Кляйн**, **Катарина Эггенспергер**, **Йост Тобиас Шпрингенберг**, **Мануэль Блюм**, **Франк Хуттер**, факультет компьютерных наук, Университет Фрайбурга, Фрайбург, Баден-Вюртемберг, Германия, e-mail: feurerm@informatik.uni-freiburg.de

Успех машинного обучения в широком спектре применений привел к постоянно растущему спросу на системы машинного обучения, которые могут быть использованы неспециалистами. Чтобы быть эффективными в повседневной практике, такие системы должны автоматически выбирать хороший алгоритм и шаги предварительной обработки признаков для нового набора данных, а также устанавливать соответствующие гиперпараметры. В современных исследовательских работах эту проблему решают путем автоматического машинного обучения (AutoML) с помощью эффективных методов байесовской оптимизации. Основываясь на этих достижениях, мы представляем новую надежную систему AutoML, основанную на пакете машинного обучения scikit-learn в Python (с использованием 15 классификаторов, 14 методов предварительной обработки признаков и четырех методов предварительной обработки данных, что в совокупности дает структурированное пространство гипотез со 110 гиперпараметрами). Эта система, которую мы назвали Auto-sklearn, улучшает существующие методы AutoML за счет того, что автоматически учитывает предыдущие результаты на аналогичных наборах данных и строит ансамбли из моделей, оцененных в ходе оптимизации. Наша система победила в шести из десяти этапов первого конкурса ChaLearn AutoML, и всесторонний анализ более 100 различных наборов данных

показал, что она значительно превосходит предыдущий уровень развития AutoML. Мы также демонстрируем прирост производительности благодаря каждому из наших новаторских решений и рассматриваем эффективность отдельных компонентов Auto-sklearn.

6.1. Введение

В последнее время машинное обучение достигло больших успехов во многих прикладных областях, что стимулирует растущий спрос на системы машинного обучения, которые могут быть эффективно использованы новичками. Соответственно, все больше коммерческих предприятий стремятся удовлетворить этот спрос (например, BigML.com, Wise.io, H2O.ai, feedzai.com, RapidMiner.com, Prediction.io, DataRobot.com, Microsoft Azure Machine Learning, Google Cloud Machine Learning Engine и Amazon Machine Learning). По своей сути каждый эффективный сервис машинного обучения должен решать фундаментальные проблемы, связанные с выбором алгоритма машинного обучения для определенного набора данных, необходимостью и способом предварительной обработки его признаков, а также настройкой всех гиперпараметров. Именно эту проблему мы рассмотрим в данной главе.

Более конкретно: мы исследуем *автоматизированное машинное обучение* (AutoML) – проблему автоматического (без участия человека) получения прогнозов относительно модели для нового набора данных в рамках фиксированного вычислительного бюджета. Формально эта проблема AutoML может быть сформулирована следующим образом.

Определение 1 (задача AutoML). Для $i = 1, \ldots, n + m$ пусть $\boldsymbol{x}_i$ обозначает вектор признаков, а y_i – соответствующее целевое значение. Если мы располагаем обучающим набором данных $D_{train} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ и векторами признаков $\boldsymbol{x}_{n+1}, \ldots, \boldsymbol{x}_{n+m}$ тестового набора данных $D_{test} = \{(\boldsymbol{x}_{n+1}, y_{n+1}), \ldots, (\boldsymbol{x}_{n+m}, y_{n+m})\}$, взятых из того же базового распределения данных, а также бюджетом b и метрикой потерь $\mathcal{L}(\cdot, \cdot)$, задача AutoML заключается в автоматическом создании механизма выдачи точных прогнозов на тестовом наборе $\hat{y}_{n+1}, \ldots, \hat{y}_{n+m}$. Потери решения $\hat{y}_{n+1}, \ldots, \hat{y}_{n+m}$ задачи AutoML определяются как $(1/m)\sum_{j=1}^{m}\mathcal{L}(\hat{y}_{n+j}, \ldots, y_{n+j})$.

На практике бюджет b представляет собой вычислительные ресурсы, такие как время работы процессора и/или использование памяти. Такая постановка задачи соответствует первому конкурсному заданию ChaLearn AutoML [23] (описание и анализ первой задачи AutoML см. также в главе 10). Система AutoML, которую мы опишем далее, победила в шести из десяти этапов этого конкурса.

Мы развиваем и расширяем методику AutoML, впервые представленную в Auto-WEKA [42]. По своей сути этот подход сочетает в себе высокопараметрическую структуру машинного обучения F с методом байесовской оптимизации [7, 40] для эффективной подгонки F под заданный набор данных.

Своим основным достижением мы считаем расширение этого подхода AutoML различными способами, которые значительно повышают его *эффективность* и *робастность*, основываясь на принципах, применимых к широкому спектру систем машинного обучения (например, используемых поставщиками услуг машинного обучения, упомянутыми выше). Во-первых, действуя в соответствии с успешной предыдущей работой о задачах оптимизации в низкой размерности [21, 22, 38], мы анализируем все наборы данных для выявления экземпляров фреймворков машинного обучения, которые хорошо работают на новом наборе данных, и запускаем с ними байесовскую оптимизацию (раздел 6.3.1). Во-вторых, мы автоматически строим ансамбли моделей, рассматриваемых байесовской оптимизацией (раздел 6.3.2). В-третьих, мы тщательно разрабатываем высокопараметризированную систему машинного обучения, состоящую из высокопроизводительных классификаторов и средств предварительной обработки, реализованных в популярной системе машинного обучения scikit-learn [36] (раздел 6.4). Наконец, мы проводим обширный эмпирический анализ с использованием разнообразных наборов данных, чтобы продемонстрировать, что полученная система Auto-sklearn превосходит предыдущие современные методы AutoML (раздел 6.5), показать, что каждое наше усовершенствование приводит к существенному улучшению производительности (раздел 6.6), а также получить представление о производительности отдельных классификаторов и предобработчиков, используемых в Auto-sklearn (раздел 6.7).

Эта глава является расширенной версией нашей статьи 2015 г., представляющей Auto-sklearn и опубликованной в материалах NeurIPS 2015 [20].

6.2. AutoML как задача CASH

Сначала мы рассмотрим формальное представление AutoML как задачи *комбинированного выбора алгоритмов и оптимизации гиперпараметров* (CASH), используемой в подходе Auto-WEKA. Две важные проблемы разработки AutoML заключаются в том, что: 1) ни один метод машинного обучения не работает лучше остальных на всех наборах данных и 2) некоторые методы машинного обучения (например, нелинейные SVM) в значительной степени зависят от оптимизации гиперпараметров. Последняя проблема была успешно решена с помощью байесовской оптимизации [7, 40], которая в настоящее время является основным компонентом многих систем AutoML. Первая проблема переплетается со второй, поскольку рейтинг алгоритмов зависит от того, правильно ли настроены их гиперпараметры. К счастью, эти две проблемы могут быть эффективно решены как единая, структурированная, совместная задача оптимизации.

Определение 2 (CASH). Пусть $\mathcal{A} = \{A^{(1)}, \ldots, A^{(R)}\}$ – множество алгоритмов, и пусть гиперпараметры каждого алгоритма $A^{(j)}$ имеют область $\mathbf{\Lambda}^{(j)}$. Далее, пусть $D_{train} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ – обучающее множество, которое разбива-

ется на K выборок перекрестной проверки $\{\mathcal{D}_{valid}^{(1)}, \ldots, \mathcal{D}_{valid}^{(K)}\}$ и $\{\mathcal{D}_{train}^{(1)}, \ldots, \mathcal{D}_{train}^{(K)}\}$ таких, что $\mathcal{D}_{train}^{(i)} = \mathcal{D}_{train} \backslash \mathcal{D}_{valid}^{(i)}$ для $i = 1, \ldots, K$. Наконец, пусть $\mathcal{L}(A_{\boldsymbol{\lambda}}^{(j)}, \mathcal{D}_{train}^{(i)}, \mathcal{D}_{valid}^{(i)})$ обозначает потерю, которую алгоритм $A^{(j)}$ достигает на $\mathcal{D}_{valid}^{(i)}$ при обучении на $\mathcal{D}_{train}^{(i)}$ с гиперпараметрами $\boldsymbol{\lambda}$. Тогда задача комбинированного выбора алгоритма и оптимизации гиперпараметров (CASH) состоит в том, чтобы найти сочетание алгоритма и гиперпараметров, которое минимизирует следующую потерю:

$$A^{\star}, \boldsymbol{\lambda}_{\star} \in \underset{A^{(j)} \in \mathcal{A},\, \boldsymbol{\lambda} \in \boldsymbol{\Lambda}^{(j)}}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A_{\boldsymbol{\lambda}}^{(j)}, \mathcal{D}_{train}^{(i)}, \mathcal{D}_{valid}^{(i)}). \tag{6.1}$$

Эта проблема CASH была впервые решена Торнтоном и др. [42] в системе Auto-WEKA с использованием механизма машинного обучения WEKA [25] и методов байесовской оптимизации на основе деревьев [5, 27]. Если коротко, байесовская оптимизация [7] настраивает вероятностную модель на отражение взаимосвязи между настройками гиперпараметров и их измеренной эффективностью; затем она использует эту модель для выбора наиболее перспективной настройки гиперпараметров (компромисс между исследованием новых частей пространства и использованием в известных «хороших» регионах), оценивает эту настройку гиперпараметров, обновляет модель в соответствии с результатом и выполняет (при необходимости) следующие итерации. В то время как байесовская оптимизация на основе моделей гауссовых процессов (например, [41]) лучше всего работает в задачах низкой размерности с числовыми гиперпараметрами, модели на основе деревьев оказались более успешными в структурированных и частично дискретных задачах высокой размерности [15] – таких как задача CASH – и также используются в системе AutoML hyperopt-sklearn [30]. Исследуя древовидные байесовские методы оптимизации, Торнтон и др. [42] обнаружили, что SMAC [27], основанный на случайных лесах, превосходит древовидный метод Парзена TPE [5], и поэтому мы используем SMAC для решения задачи CASH в данной работе. Наряду с использованием случайных лесов [6] главной отличительной особенностью SMAC является возможность быстрой перекрестной проверки за счет оценки одной выборки за раз и раннего отбрасывания плохо работающих настроек гиперпараметров.

6.3. Новые методы повышения эффективности и надежности AutoML

В этом разделе мы обсудим два улучшения стандартной методики AutoML. Во-первых, мы добавили шаг метаобучения для теплого старта процедуры байесовской оптимизации, что приводит к значительному повышению эффективности. Во-вторых, мы добавили шаг автоматического построения ансамбля, что позволяет нам использовать все классификаторы, найденные в результате байесовской оптимизации.

На рис. 6.1 представлена обобщенная схема рабочего процесса AutoML, содержащая оба наших улучшения. Отметим, что их эффективность должна быть выше для гибких систем ML, которые предлагают множество степеней свободы (например, множество алгоритмов, гиперпараметров и методов предварительной обработки).

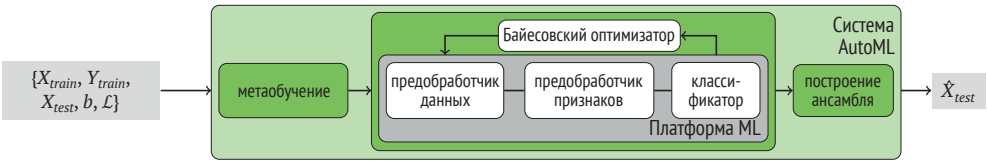


Рис. 6.1 ❖ Улучшенная методика AutoML. Мы добавляем два компонента к байесовской оптимизации гиперпараметров системы ML: метаобучение для инициализации байесовского оптимизатора и автоматическое построение ансамбля из конфигураций, оцененных в ходе оптимизации

6.3.1. Поиск перспективных вариантов при помощи метаобучения

Отраслевые эксперты извлекают знания из предыдущих задач: они стараются узнать о производительности алгоритмов машинного обучения на смежных задачах. Подход метаобучения (см. главу 2) имитирует эту стратегию, рассуждая о производительности алгоритмов обучения на разных наборах данных. Мы применяем метаобучение для выбора вариантов заданной системы машинного обучения, которые, как ожидается, будут хорошо работать на новом наборе данных. Точнее, для большого количества наборов данных мы собираем как данные о производительности, так и набор *метапризнаков*, т. е. характеристик набора данных, которые можно эффективно вычислить и которые помогают определить, какой алгоритм использовать на новом наборе данных.

Подход метаобучения дополняет байесовскую оптимизацию шаблонов машинного обучения. Метаобучение может быстро предложить несколько вариантов системы ML, которые, вероятно, будут работать достаточно хорошо, но оно не способно оперировать точной информацией о производительности. В отличие от метаобучения байесовская оптимизация медленно выполняется на таких больших пространствах гиперпараметров, как у целых ML-систем, но позволяет постепенно достичь точной оптимальной производительности. Мы используем эту взаимодополняемость, выбирая k конфигураций на основе метаобучения и используя их для запуска байесовской оптимизации. Такой подход к запуску оптимизации с помощью метаобучения уже успешно применялся ранее [21, 22, 38], но никогда к такой сложной проблеме оптимизации, как поиск в пространстве полноценной системы ML. Обучение по наборам данных также применялось в методах совместной байесовской оптимизации [4, 45]; хотя эти подходы перспективны, они

ограничены очень небольшим количеством метапризнаков и пока не могут справиться с частично дискретными пространствами конфигураций высокой размерности, с которыми сталкивается AutoML.

Наш подход к метаобучению работает следующим образом. На офлайновом этапе для каждого набора данных машинного обучения в хранилище данных (в нашем случае 140 наборов данных из хранилища OpenML [43]) мы оцениваем набор метапризнаков (описанных ниже) и используем байесовскую оптимизацию для определения и хранения выявленного ML-фреймворка с высокой эмпирической производительностью для этого набора данных. (В частности, мы запускали SMAC [27] в течение 24 ч с 10-кратной перекрестной проверкой на двух третях данных и сохраняли полученный экземпляр ML-фреймворка, который демонстрировал наилучшую производительность на оставшейся трети.) Затем, получив новый набор данных $\mathcal{D}$, мы вычисляем его метапризнаки, ранжируем все наборы данных по расстоянию L_1 до $\mathcal{D}$ в пространстве метапризнаков и выбираем для оценки сохраненные экземпляры ML-фреймворка для $k = 25$ ближайших наборов данных, после чего запускаем байесовскую оптимизацию для выбранных экземпляров.

Для того чтобы охарактеризовать наборы данных, мы использовали в общей сложности 38 метапризнаков из литературы, включая простые, информационно-теоретические и статистические метапризнаки [29, 33], такие как статистика о количестве точек данных, признаков и классов, а также перекос данных и энтропия целей. Все метахарактеристики перечислены в табл. 1 приложения к оригинальной публикации [20]. Примечательно, что нам пришлось исключить известную и эффективную категорию метапризнаков-ориентиров [37] (которые измеряют производительность простых базовых обучателей), поскольку они были слишком дорогими с вычислительной точки зрения, что неприемлемо на этапе онлайн-оценки. Необходимо отметить, что успех нашего подхода к метаобучению во многом основан на доступности различных наборов данных; благодаря таким проектам, как OpenML [43], мы ожидаем, что количество доступных наборов данных со временем будет расти, увеличивая важность метаобучения.

6.3.2. Автоматизированное построение ансамбля моделей, оцененных во время оптимизации

Хотя байесовская оптимизация гиперпараметров является эффективной с точки зрения данных для нахождения наилучшего значения гиперпараметров, это очень расточительная процедура, когда целью является просто способность делать хорошие предсказания: все модели, которые обучаются в ходе поиска, теряются, включая те из них, которые работают почти так же хорошо, как лучшие. Вместо того чтобы отбрасывать эти модели, мы предлагаем хранить их и использовать эффективный метод постобработки (который может быть запущен во втором процессе «на лету») для построения ансамбля из таких моделей. Подобное автоматическое построение ансамбля моделей позволяет избежать привязки к одному гиперпараметру и, таким

образом, является более робастным (и менее подверженным переобучению), чем использование точечной оценки, которую дает стандартная оптимизация гиперпараметров. Насколько нам известно, мы первыми сделали это простое наблюдение, которое можно применять для улучшения любого байесовского метода оптимизации гиперпараметров[1].

Хорошо известно, что ансамбли часто превосходят отдельные модели [24, 31] и что эффективные ансамбли могут быть созданы из библиотеки моделей [9, 10]. Ансамбли работают особенно хорошо, если модели, на которых они основаны, 1) сильны по отдельности и 2) делают некоррелированные ошибки [6]. Поскольку соблюдение этих условий гораздо более вероятно, когда отдельные модели различны по своей природе, построение ансамблей особенно хорошо подходит для объединения сильных экземпляров гибкой системы ML.

Однако простое построение равномерно взвешенного ансамбля моделей, найденных с помощью байесовской оптимизации, не дает хороших результатов. Напротив, мы обнаружили, что очень важно корректировать эти веса, используя прогнозы всех отдельных моделей на промежуточном множестве. Мы экспериментировали с различными подходами к оптимизации этих весов: *стекингом* [44], численной оптимизацией без градиента и методом *выбора ансамбля* [10]. Мы обнаружили, что численная оптимизация и стекинг приводят к переобучению и требуют больших вычислительных затрат, а метод выбора ансамбля оказался быстрым и надежным. В двух словах, выбор ансамбля (предложенный Каруаной и др. [10]), – это жадная процедура, которая начинается с пустого ансамбля, а затем итеративно добавляет модель, минимизирующую потери при проверке ансамбля (с равномерным весом, но с возможностью повторений). Мы использовали эту технику во всех наших экспериментах, создавая ансамбль размером 50, используя отбор с заменой [10]. Мы вычисляли потери ансамбля, используя тот же проверочный набор, который мы используем для байесовской оптимизации.

6.4. Практическая система автоматизированного машинного обучения

Для разработки надежной системы AutoML в качестве базовой платформы мы выбрали scikit-learn [36], одну из самых известных и широко используемых библиотек машинного обучения. Она предлагает широкий спектр хорошо зарекомендовавших себя и эффективно реализуемых алгоритмов ML и проста в использовании как для экспертов, так и для новичков. Поскольку наша система AutoML очень похожа на Auto-WEKA, но, как и Hyperopt-sklearn, основана на scikit-learn, мы назвали ее Auto-sklearn.

---

[1] Уже после публикации статьи [20] мы узнали, что Эскаланте и др. [16] и Бюргер и Паули [8] также применяли ансамбли в качестве этапа постобработки системы AutoML для улучшения обобщения. Однако в обеих работах обучаемые модели объединялись с помощью заранее определенной стратегии, не предусматривающей построение ансамбля на основе производительности отдельных моделей.

На рис. 6.2 показаны компоненты конвейера машинного обучения Auto-sklearn. Он содержит 15 алгоритмов классификации, 14 методов предварительной обработки признаков и 4 метода предварительной обработки данных. Мы параметризировали каждый из них, что привело к созданию пространства из 110 гиперпараметров. По большей части это условные гиперпараметры, которые активны только в том случае, если выбран соответствующий компонент. Отметим, что в SMAC [27] изначально заложена возможность обрабатывать эту условность.

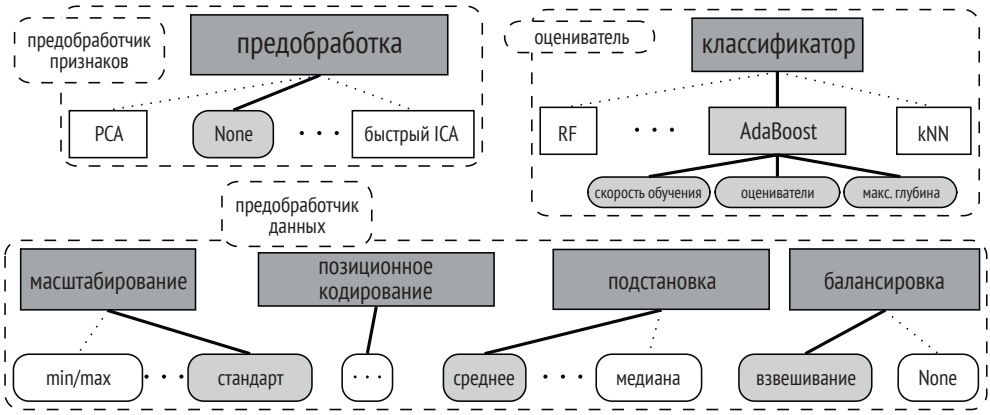


Рис. 6.2 ❖ Структурированное пространство конфигураций. Обычные прямоугольники обозначают родительские гиперпараметры, а прямоугольники с закругленными углами – конечные гиперпараметры. Серыми прямоугольниками обозначены активные гиперпараметры, образующие пример конфигурации и конвейера машинного обучения. Каждый конвейер содержит один предобработчик признаков, классификатор и до трех методов предобработки данных, а также соответствующие гиперпараметры

Все 15 алгоритмов классификации в Auto-sklearn перечислены в табл. 6.1. Они относятся к различным категориям, таким как общие линейные модели (2 алгоритма), метод опорных векторов (2), дискриминантный анализ (2), ближайшие соседи (1), наивные байесовские методы (3), деревья решений (1) и ансамбли (4). В отличие от Auto-WEKA [42] (описание Auto-WEKA приведено в главе 4) мы ограничили наше пространство конфигураций базовыми классификаторами и исключили метамодели и ансамбли, которые сами параметризованы одним или несколькими базовыми классификаторами. Хотя такие ансамбли увеличили число гиперпараметров Auto-WEKA почти в пять раз (до 786), Auto-sklearn имеет «всего» 110 гиперпараметров. Вместо этого мы строим сложные ансамбли, используя наш метод post-hoc, о котором говорится в разделе 6.3.2. По сравнению с Auto-WEKA это намного более экономично: в Auto-WEKA оценка эффективности ансамбля с пятью компонентами требует построения и оценки пяти моделей; в Auto-sklearn, напротив, ансамбли предоставляются в основном бесплатно, и можно комбинировать и сопоставлять модели, оцениваемые в произвольное время в ходе оптимизации.

Таблица 6.1. Количество гиперпараметров для каждого классификатора (вверху) и метода предварительной обработки признаков (внизу) для набора данных бинарной классификации в плотном представлении. Таблицы для разреженной бинарной классификации и наборов данных разреженной/плотной многоклассовой классификации можно найти в разделе E дополнительных материалов оригинальной публикации [20], табл. 2a, 3a, 4a, 2b, 3b и 4b. Мы различаем категориальные (cat) гиперпараметры с дискретными значениями и непрерывные (cont) числовые гиперпараметры. Числа в скобках – это условные гиперпараметры, которые имеют смысл только тогда, когда другой гиперпараметр имеет определенное значение

Тип классификатора	**#λ**	**cat (усл.)**	**cont (усл.)**
AdaBoost (AB)	4	1 (–)	3 (–)
Бернулли–Байес	2	1 (–)	1 (–)
Решающее дерево (DT)	4	1 (–)	3 (–)
Полностью рандомизированное дерево	5	2 (–)	3 (–)
Гауссов – наивный байесовский алг.	–	–	–
Градиентный бустинг (GB)	6	–	6 (–)
k-ближайшие соседи (kNN)	3	2 (–)	1 (–)
Линейный дискрим. анализ (LDA)	4	1 (–)	3 (1)
Линейный SVM	4	2 (–)	2 (–)
Ядерный SVM	7	2 (–)	5 (2)
Мультиномиальный наивный байесовский	2	1 (–)	1 (–)
Пассивно-агрессивный	3	1 (–)	2 (–)
Квадратичный дискрим. анализ (QDA)	2	–	2 (–)
Случайный лес (RF)	5	2 (–)	3 (–)
Линейный классификатор (SGD)	10	4 (–)	6 (3)

Метод предобработки	**#λ**	**cat (усл.)**	**cont (усл.)**
Полностью рандомизированное дерево	5	2 (–)	3 (–)
Быстрый ICA	4	3 (–)	1 (1)
Агломерация признаков	4	3 0	1 (–)
Ядерный PCA	5	1 (–)	4 (3)
Случайно-реалистический	2	–	2 (–)
Линейный SVM	3	1 (–)	2 (–)
Без предобработки	—	–	–
Выборки Нистрома	5	1 (–)	4 (3)
Анализ основных компонент (PCA)	2	1 (–)	1 (–)
Полиномиальный	3	2 (–)	1 (–)
Встраивания / случайные деревья	4	–	4 (–)
Выбор перцентиля	2	1 (–)	1 (–)
Выбор скорости	3	1 (–)	1 (–)
Позиционное кодирование	2	1 (–)	1 (1)
Подстановка	1	1 (–)	–
Балансировка	1	1 (–)	–
Масштабирование	1	1 (–)	–

Методы предобработки наборов данных в плотном представлении в Auto-sklearn также перечислены в табл. 6.1. В их число входят предварительные обработчики данных (которые изменяют значения признаков и всегда используются, когда они применимы) и предварительные обработчики признаков (которые изменяют фактический набор признаков, и либо используется только один из них, либо не используется ни один). К предварительной обработке данных относятся изменение масштаба входных данных, подстановка недостающих значений, позиционное кодирование и балансировка целевых классов. Четырнадцать возможных методов предварительной обработки признаков можно разделить на выбор признаков (2), ядерную аппроксимацию (2), разложение матрицы (3), встраивание (1), кластеризацию (1), полиномиальное расширение (1) и методы, использующие классификатор для выбора признаков (2). Например, L_1-регуляризованные линейные SVM, обученные на данных, могут быть использованы для отбора путем исключения признаков, соответствующих нулевым коэффициентам модели.

Подробное описание алгоритмов машинного обучения, используемых в Auto-sklearn, приведено в разделах A.1 и A.2 дополнительного материала оригинальной статьи [20], документации scikit-learn [36] и ссылках в них.

Чтобы максимально использовать доступные вычислительные мощности и не застрять на очень медленном прогоне определенной комбинации предварительной обработки и алгоритма машинного обучения, мы применили несколько мер для предотвращения таких длительных прогонов. Прежде всего мы ограничили время для каждой оценки экземпляра фреймворка ML. Мы также ограничили объем памяти, выделяемой для таких оценок, чтобы предотвратить свопинг или замораживание операционной системы. Когда процедура оценки превышала один из этих пределов, мы автоматически завершали ее и возвращали наихудший возможный результат для данной метрики оценки. Для некоторых моделей мы использовали итеративную процедуру обучения; мы настроили их так, чтобы они возвращали текущее значение производительности при достижении предела, прежде чем они будут завершены. Чтобы дополнительно уменьшить количество слишком длинных прогонов, мы запретили несколько комбинаций предварительных обработчиков и методов классификации: в частности, запрещено сочетать ядерную аппроксимацию с нелинейными и древовидными методами, а также с алгоритмом KNN (SMAC обрабатывает такие запрещенные комбинации естественным образом). По той же причине мы также исключили алгоритмы обучения признаков, такие как обучение по словарю.

Еще одной проблемой при оптимизации гиперпараметров является переобучение и повторная выборка данных, поскольку обучающие данные системы AutoML должны быть разделены на набор данных для обучения конвейера ML (обучающее множество) и набор данных, используемый для расчета функции потерь при байесовской оптимизации (проверочное множество). Здесь нам пришлось выбирать между более надежной перекрестной проверкой (которая в SMAC не требует дополнительных затрат) и оценкой моделей на всех выборках перекрестной проверки, чтобы можно было построить ансамбль с этими моделями. Так, для задач с жестким ограничением времени в 1 ч в разделе 6.6 мы использовали простое разделение набора на

обучающий и тестовый. В отличие от этого сценария мы смогли применить десятикратную перекрестную проверку в 24- и 30-часовых заданиях в разделах 6.5 и 6.7.

Наконец, не каждая задача обучения с учителем (например, классификация с несколькими целями) может быть решена всеми алгоритмами, доступными в Auto-sklearn. Поэтому, получив новый набор данных, Auto-sklearn предварительно выбирает методы, которые подходят для свойств набора данных. Поскольку методы scikit-learn ограничены числовыми входными значениями, мы всегда преобразовывали данные, применяя позиционное кодирование к категориальным признакам. Для того чтобы количество фиктивных признаков было минимальным, мы установили порог в процентах, а значение, встречающееся реже этого порога, преобразовывалось в специальное значение «другое» (other) [35].

6.5. Сравнение Auto-sklearn с Auto-WEKA и Hyperopt-sklearn

В качестве базового эксперимента мы сравнили производительность начального варианта Auto-sklearn (без наших улучшений метаобучения и построения ансамблей) с Auto-WEKA (глава 4) и Hyperopt-sklearn (глава 5), воспроизведя сценарий эксперимента с 21 набором данных из статьи, представляющей Auto-WEKA [42] (см. табл. 4.1 в главе 4 с описанием наборов данных). Следуя оригинальной схеме работы Auto-WEKA, мы использовали те же тренировочные и тестовые наборы данных [1], ограничение времени работы в 30 ч, 10-кратную перекрестную проверку (где оценка каждой выборки занимала 150 мин) и 10 независимых оптимизационных прогонов с SMAC на каждом наборе данных. Как и в Auto-WEKA, оценка ускоряется с помощью процедуры интенсификации SMAC, которая назначает прогоны на новых выборках перекрестной проверки только в том случае, если есть надежда, что конфигурация, оцениваемая в этом проходе, превзойдет конфигурацию с наилучшими показателями на данный момент [27]. Мы не модифицировали Hyperopt-sklearn, который всегда использует разделение 80/20 на обучающий и тестовый наборы соответственно. Все наши эксперименты проводились на восьмиядерных процессорах Intel Xeon E5-2650 v2 с частотой 2.60 ГГц и 4 Гб оперативной памяти. Мы позволили системе машинного обучения использовать 3 Гб, а остальное зарезервировали для SMAC. Во всех экспериментах использовались версии Auto-WEKA 0.5 и scikit-learn 0.16.1.

Результаты этого эксперимента мы представили в табл. 6.2. Так как наш сценарий в точности соответствовал тому, что описан в оригинальной статье по Auto-WEKA, в качестве проверки достоверности эксперимента мы сравнили результаты, полученные нами для Auto-WEKA (первая строка на рис. 6.2), с результатами, представленными авторами Auto-WEKA (глава 4),

и обнаружили, что в целом результаты достаточно близки. Более того, из таблицы видно, что Auto-sklearn показал себя значительно лучше, чем Auto-WEKA в шести случаях из 21, сравнялся с ним в 12 случаях и проиграл в трех. Для трех наборов данных, где Auto-WEKA показал лучшие результаты, мы обнаружили, что более чем в 50 % случаев лучший классификатор, который он выбрал, не реализован в scikit-learn (деревья с возможностью сокращения). Пока что Hyperopt-sklearn является скорее пробным вариантом, предлагающим пользователю адаптировать пространство конфигураций под свои нужды, чем полноценной системой AutoML. Текущая версия дает сбой, если ей предоставить разреженные данные с отсутствующими значениями. Она также дает сбой на наборе Cifar-10 из-за ограничения памяти, которое мы установили для всех оптимизаторов, чтобы обеспечить справедливое сравнение. На 16 наборах данных, на которых пакет Hyperopt-sklearn сработал успешно, он статистически сравнялся с лучшей конкурирующей системой AutoML в девяти случаях и проиграл ей в семи.

Таблица 6.2. Ошибка классификации тестового набора для Auto-WEKA (AW), базового варианта Auto-sklearn (AS) и Hyperopt-sklearn (HS) по методике оригинального оценивания Auto-WEKA [42] (см. также раздел 4.5)

	Abalone	Amazon	Car	Ci far-10	Ci far-10 (малый)	Convex	Dexter	Dorothea	German credit	Gisette	KDD09 appetency
AS	**73.50**	**16.00**	0.39	**51.70**	**54.81**	**17.53**	**5.56**	**5.51**	**27.00**	**1.62**	**1.74**
AW	**73.50**	30.00	**0.00**	56.95	56.20	21.80	8.33	6.38	28.33	2.29	**1.74**
HS	76.21	16.22	0.39	–	57.95	19.18	–	–	27.67	2.29	–
	KR-vs-KP	Made Ion	MNIST (базовый)	MRBI	Secom	Seme ion	Shuttle	Waveform	Wine quality	Yeast	
AS	0.42	**12.44**	2.84	**46.92**	**7.87**	**5.24**	**0.01**	14.93	33.76	40.67	
AW	**0.31**	18.21	2.84	60.34	8.09	5.24	**0.01**	14.13	33.36	37.75	
HS	0.42	14.74	**2.82**	55.79	–	5.87	0.05	**14.07**	34.72	38.45	

Здесь показан медианный процент ошибок теста на 100 000 бутстрэп-выборок (на основе 10 запусков), каждая выборка моделирует четыре параллельных прохода, и всегда остается лучший из них по результатам перекрестной проверки. Жирные цифры указывают на лучший результат. Подчеркнутые результаты статистически значительно отличаются от лучшего результата согласно бутстрэп-тесту с $p = 0.05$.

6.6. Оценка предложенных улучшений AutoML

Для того чтобы оценить надежность и общую применимость предложенной нами системы AutoML на широком спектре наборов данных, мы собрали 140 наборов данных бинарной и многоклассовой классификации из репозитория OpenML [43], выбирая только наборы, которые содержат не менее 1000 точек данных, чтобы обеспечить надежную оценку производительности. Эти наборы данных охватывают широкий спектр отраслей применения, таких как классификация текста, распознавание цифр и букв, классификация генных последовательностей и РНК, реклама, классификация объектов для данных телескопа и обнаружение рака в образцах тканей. Мы перечислили все наборы данных в табл. 7 и 8 в дополнительных материалах оригинальной публикации [20] и указали их уникальные идентификаторы OpenML для воспроизводимости. Мы случайным образом разделили каждый набор данных на две трети обучающего и одну треть тестового набора. Auto-sklearn имел доступ только к обучающему набору и, в свою очередь, разделил его на две трети для обучения и одну треть для вычисления валидационных потерь SMAC. В целом мы использовали четыре девятых данных для обучения моделей, две девятых – для расчета потерь при проверке, а последние три девятых – для отчета о результатах тестирования различных систем AutoML, которые мы сравнивали. Поскольку распределение классов во многих из этих наборов данных довольно несбалансированное, мы оценивали все методы AutoML с помощью меры, называемой *сбалансированным коэффициентом ошибок классификации* (balanced classification error rate, BER). Мы определяем коэффициент ошибок сбалансированной классификации как среднее значение доли неправильных классификаций в каждом классе. По сравнению со стандартной ошибкой классификации (средняя общая ошибка) эта мера (средняя ошибка по классам) присваивает равный вес всем классам. Отметим, что сбалансированные меры ошибок или точности часто используются в соревнованиях по машинному обучению, таких как AutoML challenge [23], который описан в главе 10.

Мы выполнили 10 запусков Auto-sklearn с метаобучением и без него, а также с построением ансамбля и без него на каждом из наборов данных. Для изучения их производительности в условиях жестких временных ограничений, а также из-за нехватки вычислительных ресурсов мы ограничили время работы процессора для каждого запуска до 1 ч; мы также ограничили время работы для оценки одной модели десятой частью этого времени (6 мин).

Чтобы не оценивать производительность на наборах данных, уже использованных для метаобучения, мы провели проверку с оставлением одного набора данных: при оценке на наборе данных $\mathcal{D}$ мы использовали метаинформацию только из 139 других наборов данных.

На рис. 6.3 показаны средние рейтинги четырех протестированных нами версий Auto-sklearn. Мы видим, что оба наших новых метода дали значительные улучшения по сравнению с базовым Auto-sklearn. Наиболее поразительным результатом является то, что метаобучение дало резкое улучшение, начиная с первой выбранной конфигурации и до конца эксперимента. Мы отмечаем, что улучшение было наиболее выраженным в начале и что со временем базовый Auto-sklearn также находил хорошие решения без метаобуче-

ния, что позволило ему догнать нас на некоторых наборах данных (тем самым улучшив свой общий рейтинг).

Более того, оба наших метода дополняют друг друга: предложенное нами автоматизированное построение ансамбля улучшило как базовый Auto-sklearn, так и Auto-sklearn с метаобучением. Интересно, что влияние ансамбля на производительность началось раньше для версии с метаобучением. Мы считаем, что это связано с тем, что метаобучение раньше создает лучшие модели машинного обучения, которые можно напрямую объединить в сильный ансамбль; но при более длительной работе базовый Auto-sklearn без метаобучения также выигрывает от автоматического построения ансамбля.

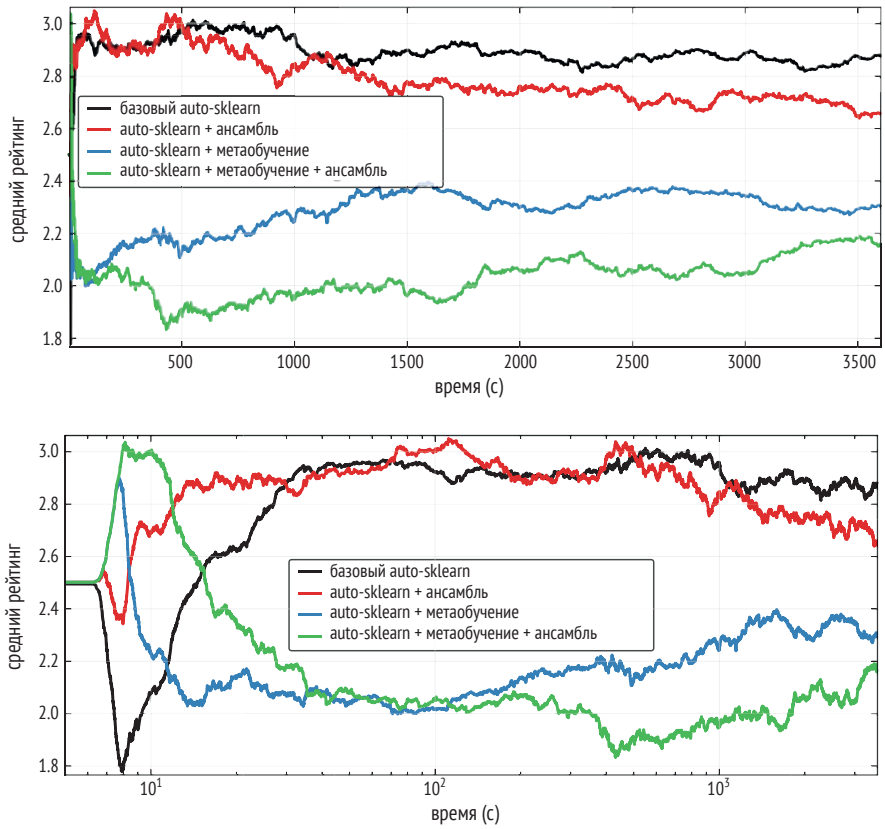


Рис. 6.3 ❖ Средний рейтинг всех четырех вариантов Auto-sklearn (ранжированных по сбалансированному коэффициенту ошибок теста BER) на 140 наборах данных. Обратите внимание, что рейтинг – это относительная мера производительности (здесь рейтинг всех методов должен быть равен 10), и поэтому улучшение BER одного метода может ухудшить рейтинг другого. *Вверху*: данные построены по линейной шкале *x*. *Внизу*: те же данные, что и на верхнем графике, но построенные в логарифмическом масштабе *x*. Из-за небольших дополнительных накладных расходов, связанных с метаобучением и выбором ансамбля, базовый Auto-sklearn способен достичь наилучшего рейтинга в течение первых 10 с, поскольку он производит прогнозы до того, как другие варианты Auto-sklearn закончат обучение своей первой модели. После этого метаобучение заметно ускоряется

6.7. Детальный анализ компонентов Auto-sklearn

В этом разделе мы рассмотрим отдельные классификаторы и предобработчики Auto-sklearn в сравнении с совместной оптимизацией всех методов, чтобы получить представление об их максимальной производительности и устойчивости. В идеале хотелось бы изучить все комбинации каждого классификатора и каждого предобработчика по отдельности, но при наличии 15 классификаторов и 14 предобработчиков это оказалось невозможным; поэтому при изучении производительности одного классификатора мы все равно оптимизировали все предобработчики, и наоборот. Чтобы выполнить более детальный анализ, мы сосредоточились на более узком подмножестве наборов данных, но расширили бюджет конфигурации для оптимизации всех методов с одного часа до одного дня и до двух дней для Auto-sklearn. В частности, мы кластеризовали наши 140 наборов данных с помощью метода g-средних [26] на основе метапризнаков набора данных и использовали по одному набору данных из каждого из 13 кластеров. Основное описание наборов данных приведено в табл. 6.3. В общей сложности эти обширные эксперименты потребовали 10.7 процессорных лет.

Таблица 6.3. Репрезентативные наборы данных для 13 кластеров, полученных с помощью кластеризации по алгоритму g-средних векторов метапризнаков для 140 наборов данных

ID	Название набора	#Cont	#Nom	Кол-во классов	Разреженность	Пропуски	|Обучение|	|Тесты|
38	Sick	7	22	2	—	X	2527	1245
46	Splice	0	60	3	—	—	2137	1053
179	Adult	2	12	2	—	X	32 724	16 118
184	KROPT	0	6	18	—	—	18 797	9259
554	MNIST	784	0	10	—	—	46 900	23 100
772	Quake	3	0	2	—	—	1459	719
917	fri_cl_1000_25 (бинаризован)	25	0	2	—	—	670	330
1049	pc4	37	0	2	—	—	976	482
1111	KDDCupO9 Appetency	192	38	2	—	X	33 500	16 500
1120	Magic Telescope	10	0	2	—	—	12 743	6277
1128	OVA Breast	10 935	0	2	—	—	1035	510
293	Covertype (бинаризован)	54	0	2	X	—	389 278	191 734
389	fbis_wc	2000	0	17	X	—	1651	812

В табл. 6.4 сравниваются результаты различных методов классификации с Auto-sklearn. В целом, как и ожидалось, случайные леса, чрезвычайно ран-

домизированные деревья, AdaBoost и градиентный бустинг показали наиболее устойчивую производительность, а алгоритмы SVM продемонстрировали сильную пиковую производительность для некоторых наборов данных. Помимо множества сильных классификаторов, есть также несколько моделей, которые показали неудовлетворительные результаты: дерево решений, пассивно-агрессивный, kNN, гауссов NB, LDA и QDA статистически значимо уступали лучшему классификатору на большинстве наборов данных. Наконец, таблица показывает, что ни один метод не является лучшим выбором для всех наборов данных. Как показано в таблице, а также визуально представлено для двух наборов данных на рис. 6.4, оптимизация совместного пространства конфигураций Auto-sklearn привела к наиболее устойчивой производительности. График рейтингов (рис. 2 и 3 в дополнительных материалах оригинальной публикации [20]) отражает количественную оценку на всех 13 наборах данных, показывая, что Auto-sklearn начинает с разумной, но не оптимальной производительности и эффективно ищет свое более общее пространство конфигураций, чтобы со временем прийти к наилучшей общей производительности.

В табл. 6.5 сравниваются результаты различных предобработчиков с Auto-sklearn. Как и при сравнении классификаторов выше, Auto-sklearn показал наиболее устойчивую производительность: фактически он показал наилучшие результаты на трех наборах данных и не был статистически значимо хуже лучшего предобработчика еще на 8 наборах из 13.

6.8. Обсуждение результатов и заключение

Представив результаты экспериментальной проверки, мы завершаем эту главу кратким обсуждением, простым примером использования Auto-sklearn, кратким обзором последних дополнений и заключительными замечаниями.

6.8.1. Обсуждение результатов

Мы продемонстрировали, что наша новая система Auto-sklearn для AutoML показывает хорошие результаты по сравнению с уровнем предыдущих методов и что наши усовершенствования в виде метаобучения и ансамбля моделей для AutoML обеспечивают прирост эффективности и надежности. Этот вывод подтверждается тем фактом, что Auto-sklearn выиграл три из пяти конкурсов, включая два последних, в первом соревновании ChaLearn по AutoML. В этой работе мы не оценивали использование Auto-sklearn для интерактивного машинного обучения с экспертом в цикле и неделями процессорного времени, но мы отмечаем, что использование этого режима позволило занять три первых места в человеческом (он же финальный) треке первой задачи ChaLearn AutoML (в дополнение к автоматическим трекам, в частности, табл. 10.5, фазы Final 0-4). Поэтому мы полагаем, что Auto-

Таблица 6.4. Медианный сбалансированный коэффициент ошибок (BER) на тестовых данных при оптимизации подпространств Auto-sklearn для каждого метода классификации (и всех предобработчиков), а также всего пространства конфигураций Auto-sklearn на 13 наборах данных. Все оптимизационные прогоны выполнялись в течение 24 ч, за исключением Auto-sklearn, который выполнялся в течение 48 ч. Жирным шрифтом обозначен лучший результат; подчеркнутые результаты статистически значимо не отличаются от лучшего согласно бутстрэп-тесту с использованием тех же условий, что и в табл. 6.2

OpenML dataset ID	Auto-sklearn	AdaBoost	Бернулли-NB	Дерево решений	Чрезвыч. рандом. деревья	Гаусс-NB	Град. бустинг	kNN	LDA	Линейн. SVM	Ядерный SVM	Multinomial naive Bayes	Passive aggresive	QDA	Random forest	Linear Class. (SGD)
38	2.15	2.68	50.22	2.15	18.06	11.22	**1.77**	50.00	8.55	16.29	17.89	46.99	50.00	8.78	2.34	15.82
46	3.76	4.65	–	5.62	4.74	7.88	**3.49**	7.57	8.67	8.31	5.36	7.55	9.23	7.57	4.20	7.31
179	**16.99**	17.03	19.27	18.31	17.09	21.77	17.00	22.23	18.93	17.30	17.57	18.97	22.29	19.06	17.24	17.01
184	**10.32**	10.52	–	17.46	11.10	64.74	10.42	31.10	35.44	15.76	12.52	27.13	20.01	47.18	10.98	12.76
554	1.55	2.42	–	12.00	2.91	10.52	3.86	2.68	3.34	2.23	**1.50**	10.37	100.00	2.75	3.08	2.50
772	46.85	49.68	47.90	47.75	**45.62**	48.83	48.15	48.00	46,74	48.38	48.66	47.21	48.75	47.67	47.71	47.93
917	10.22	9.11	25.83	11.00	10.22	33.94	10.11	11.11	34.22	18.67	**6.78**	25.50	20.67	30.44	10.83	18.33
1049	12.93	**12.53**	15.50	19.31	17.18	26.23	13.38	23.80	25.12	17.28	21.44	26.40	29.25	21.38	13.75	19.92
1111	23.70	23.16	28.40	24.40	24.47	29.59	**22.93**	50.30	24.11	23.99	23.56	27.67	43.79	25.86	28.06	23.36
1120	13.81	13.54	18.81	17.45	13.86	21.50	13.61	17.23	15.48	14.94	14.17	18.33	16.37	15.62	13.70	14.66
1128	4.21	4.89	4.71	9.30	3.89	4.77	4.58	4.59	4.58	4.83	4.59	4.46	5.65	5.59	**3.83**	4.33
293	2.86	4.07	24.30	5.03	3.59	32.44	24.48	4.86	24.40	14.16	100.00	24.20	21.34	28.68	**2.57**	15.54
389	19.65	22.98	–	33.14	19.38	29.18	19.20	30.87	19.68	**17.95**	22.04	20.04	20.14	39.57	20.66	17.99

Таблица 6.5. Аналогично табл. 6.4, но вместо этого оптимизируются подпространства для каждого метода предобработки (и всех классификаторов)

Open ML dataset ID	Auto-sklearn	Уплотн. данных	Чрезвыч. рандом. деревья	Быстр. ICA	Агломер. признаков	Ядерный PCA	Случайно-реалистический	Линейный SVM	Без пред. обраб.	Выборка Нистрома	PCA	Полином.	Встр. случ. деревья	Выбор перцент.	Выбор по рейтингу	Усечен. SVD
38	**2.15**	–	4.03	7.27	2.24	5.84	8.57	2.28	2.28	7.70	7.23	2.90	18.50	2.20	2.28	–
46	**3.76**	–	4.98	7.95	4.40	8.74	8.41	4.25	4.52	8.48	8.40	4.21	7.51	4.17	4.68	–
179	16.99	–	17.83	17.24	16.92	100.00	17.34	**16.84**	16.97	17.30	17.64	16.94	17.05	17.09	16.86	–
184	10.32	–	55.78	19.96	11.31	36.52	28.05	**9.92**	11.43	25.53	21.15	10.54	12.68	45.03	10.47	–
554	1.55	–	1.56	2.52	1.65	100.00	100.00	2.21	1.60	2.21	1.65	100.00	3.48	**1.46**	1.70	–
772	**46.85**	–	47.90	48.65	48.62	47.59	47.68	47.72	48.34	48.06	47.30	48.00	47.84	47.56	48.43	–
917	10.22	–	**8.33**	16.06	10.33	20.94	35.44	8.67	9.44	37.83	22.33	9.11	17.67	10.00	10.44	–
1049	12.93	–	20.36	19.92	13.14	19.57	20.06	13.28	15.84	18.96	17.22	12.95	18.52	**11.94**	14.38	–
1111	23.70	–	23.36	24.69	23.73	100.00	25.25	23.43	**22.27**	23.95	23.25	26.94	26.68	23.53	23.33	–
1120	13.81	–	16.29	14.22	13.73	14.57	14.82	14.02	13.85	14.66	14.23	**13.22**	15.03	13.65	13.67	–
1128	4.21	–	4.90	4.96	4.76	4.21	5.08	4.52	4.59	**4.08**	4.59	50.00	9.23	4.33	**4.08**	–
293	2.86	24.40	3.41	–	–	100.00	19.30	3.01	**2.66**	20.94	–	–	8.05	2.86	2.74	4.05
389	19.65	20.63	21.40	–	–	**17.50**	19.66	19.89	20.87	18.46	–	–	44.83	20.17	19.18	21.58

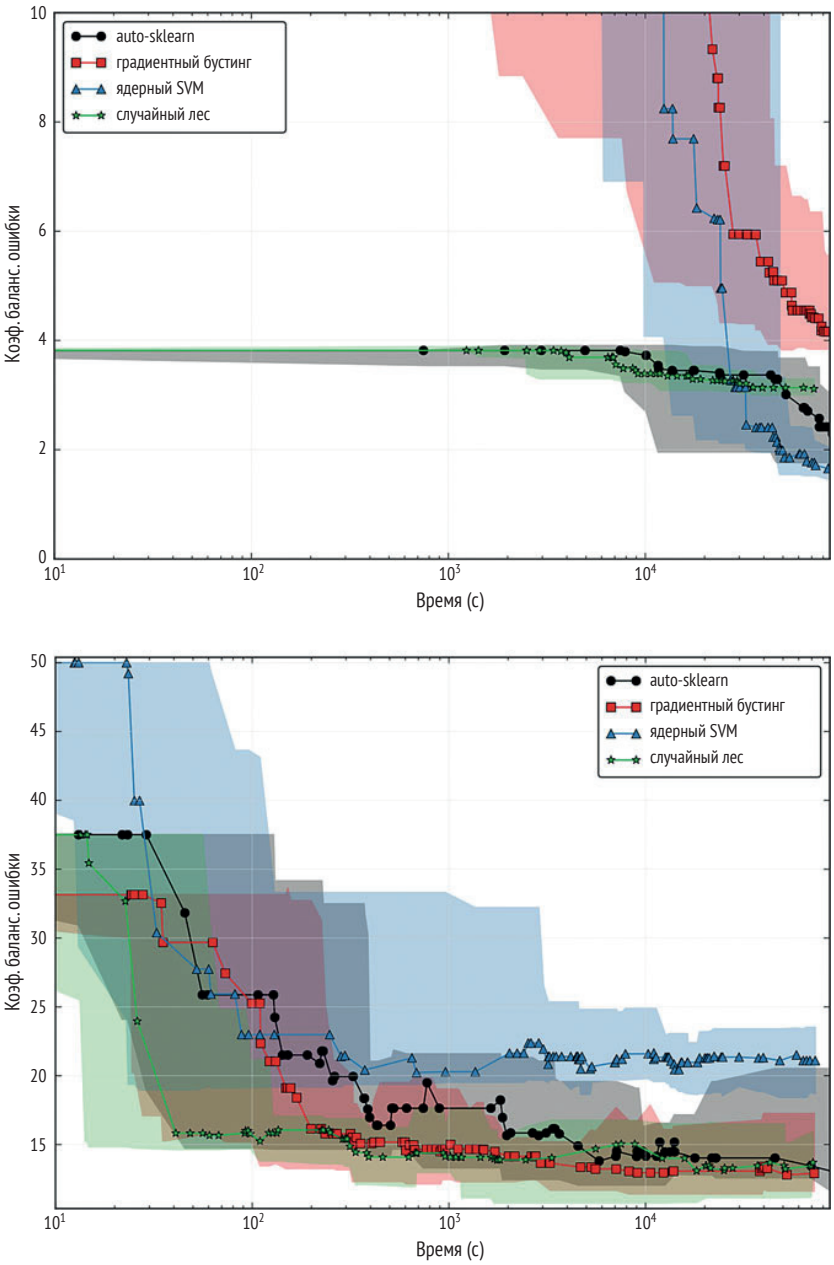


Рис. 6.4 ❖ Производительность подмножества классификаторов по сравнению с Auto-sklearn. *Вверху*: MNIST (набор данных OpenML ID 554). *Внизу*: Promise pc4 (набор данных OpenML ID 1049). Мы показываем медианный коэффициент ошибок при тестировании, а также пятый и 95-й процентили для оптимизации трех классификаторов по отдельности и оптимизации совместного пространства. График со всеми классификаторами можно найти на рис. 4 в дополнительных материалах оригинальной публикации [20]. Хотя Auto-sklearn уступает в начале, в конце его производительность близка к лучшему методу

sklearn является перспективной системой для использования как новичками, так и экспертами в области машинного обучения.

С момента публикации оригинальной статьи в NeurIPS [20] Auto-sklearn стал стандартной базой для новых подходов к автоматизированному машинному обучению, таких как FLASH [46], RECIPE [39], Hyperband [32], AutoPrognosis [3], ML-PLAN [34], Auto-Stacker [11] и AlphaD3M [13].

6.8.2. Практическое применение

Одним из важных результатов разработки Auto-sklearn является пакет auto-sklearn Python. Он представляет собой замену любого классификатора или регрессора scikit-learn аналогично классификатору, предоставляемому Hyperopt-sklearn [30], и может быть использован следующим образом:

```
import autosklearn.classification
cls = autosklearn.classification.AutoSklearnClassifier()
cls.fit(X_train, y_train)
predictions = cls.predict(X_test)
```

Auto-sklearn можно использовать с любой функцией потерь и стратегией повторной выборки для оценки потерь при проверке. Кроме того, можно расширить перечень классификаторов и предобработчиков, которые может выбрать Auto-sklearn. С момента первой публикации мы также добавили в Auto-sklearn поддержку регрессии. Мы размещаем новые версии на GitHub по адресу https://github.com/AutoML/auto-sklearn, а также в виде пакета Python на сайте https://pypi.org/. Документация представлена на сайте https://AutoML.github.io/auto-sklearn/master/#manual.

6.8.3. Расширения в PoSH Auto-sklearn

Хотя Auto-sklearn, описанный в этой главе, ограничен работой с наборами данных относительно скромного размера, в контексте конкурсной задачи AutoML 2 (глава 10) мы расширили его для эффективной работы с большими наборами данных. Auto-sklearn смог обрабатывать наборы данных в несколько сотен тысяч точек, используя кластер из 25 процессоров в течение двух дней, но не в рамках 20-минутного бюджета времени, требуемого задачей AutoML 2. Как подробно описано в докладе на семинаре [18], это подразумевало расширение рассматриваемых методов, включая экстремальный градиентный бустинг (в частности, XGBoost [12]), использование подхода последовательного уменьшения точности [28] (также описанного в главе 1) для решения задачи CASH и изменение нашего подхода к метаобучению. Теперь мы кратко опишем получившуюся систему под названием **PoSH Auto-sklearn** (сокращение от Portfolio Successive Halving в сочетании с Auto-sklearn), которая получила лучшие результаты в конкурсе 2018 г.

PoSH Auto-sklearn начинает с выполнения последовательного перебора с фиксированным портфелем из 16 конфигураций конвейеров машинного

обучения, а если остается время, то использует результаты этих проходов для теплого запуска комбинации байесовской оптимизации и последовательного перебора. Фиксированный портфель из 16 конвейеров был получен путем выполнения жадной максимизации субмодулярной функции с целью формирования сильного набора дополнительных конфигураций для оптимизации производительности, полученной на наборе из 421 набора данных; конфигурации-кандидаты, настроенные для этой оптимизации, представляли собой 421 конфигурацию, найденную путем выполнения SMAC [27] на каждом из этих наборов данных.

Комбинация байесовской оптимизации и последовательного деления пополам, которую мы использовали для получения надежных результатов в коротком временном окне, является адаптацией метода гиперпараметрической оптимизации BOHB (Bayesian Optimization and HyperBand) [17], рассмотренной в главе 1. В качестве бюджетов для этого подхода с переменной точностью мы использовали количество итераций для всех итеративных алгоритмов, за исключением SVM, где в качестве бюджета мы использовали размер набора данных.

Еще одним расширением для больших наборов данных, разработка которого в настоящее время продолжается, является наша работа по автоматическому глубокому обучению; этот проект обсуждается в следующей главе, посвященной Auto-Net.

6.8.4. Заключение и будущие исследования

Следуя подходу AutoML, использованному в Auto-WEKA, мы представили Auto-sklearn, который демонстрирует хорошие результаты по сравнению с предыдущим уровнем технологии в AutoML. Мы также показали, что наши механизмы метаобучения и ансамблирования еще больше повышают его эффективность и устойчивость.

Хотя Auto-sklearn выполняет настройку гиперпараметров за пользователя, этот инструмент имеет собственные гиперпараметры, которые влияют на его производительность для заданного бюджета времени, например временные ограничения, обсуждаемые в разделах 6.5, 6.6 и 6.7, или стратегия выборки, используемая для вычисления функции потерь. В предыдущей работе мы продемонстрировали, что выбор стратегии повторной выборки и выбор тайм-аутов можно рассматривать как проблему метаобучения [19], но мы хотели бы расширить ее на другие возможные варианты системы машинного обучения, с которыми сталкиваются пользователи Auto-sklearn.

Со времени написания оригинальной статьи область метаобучения значительно продвинулась вперед, открыв доступ к множеству новых методов включения метаинформации в байесовскую оптимизацию. Мы ожидаем, что использование одного из новых методов, рассмотренных в главе 2, может существенно улучшить процедуру оптимизации.

Наконец, наличие полностью автоматизированной процедуры, которая может тестировать сотни конфигураций гиперпараметров, подвергает нас повышенному риску переобучения на проверочном наборе. Чтобы избежать

этого, мы собираемся объединить Auto-sklearn с одним из методов, рассмотренных в главе 1, методами из области дифференциальной приватности [14] или другими методами, которые еще предстоит разработать.

Благодарности. Эта работа была поддержана Немецким исследовательским фондом (DFG) в рамках приоритетной программы «Автономное обучение» (SPP 1527, грант HU 1900/3-1), грантом Эмми Ноэтер HU 1900/2-1 и кластером BrainLinks-BrainTools (номер гранта EXC 1086).

6.9. Литература

1. Auto-WEKA website, http://www.cs.ubc.ca/labs/beta/Projects/autoweka.
2. Proc. of NeurIPS'15 (2015).
3. Ahmed, A., van der Schaar, M.: AutoPrognosis: Automated clinical prognostic modeling via Bayesian optimization with structured kernel learning. In: Proc. of ICML'18. pp. 139–148 (2018).
4. Bardenet, R., Brendel, M., Kégl, B., Sebag, M.: Collaborative hyperparameter tuning. In: Proc. of ICML'13. pp. 199–207 (2014).
5. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Proc. of NIPS'11. pp. 2546–2554 (2011).
6. Breiman, L.: Random forests. MLJ 45, 5–32 (2001).
7. Brochu, E., Cora, V., de Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv:1012.2599v1 [cs.LG] (2010).
8. Bürger, F., Pauli, J.: A holistic classification optimization framework with feature selection, preprocessing, manifold learning and classifiers. In: Proc. of ICPRAM'15. pp. 52–68 (2015).
9. Caruana, R., Munson, A., Niculescu-Mizil, A.: Getting the most out of ensemble selection. In: Proc. of ICDM'06. pp. 828–833 (2006).
10. Caruana, R., Niculescu-Mizil, A., Crew, G., Ksikes, A.: Ensemble selection from libraries of models. In: Proc. of ICML'04. p. 18 (2004).
11. Chen, B., Wu, H., Mo, W., Chattopadhyay, I., Lipson, H.: Autostacker: A compositional evolutionary learning system. In: Proc. of GECCO'18 (2018).
12. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proc. of KDD'16. pp. 785–794 (2016).
13. Drori, I., Krishnamurthy, Y., Rampin, R., Lourenco, R., One, J., Cho, K., Silva, C., Freire, J.: AlphaD3M: Machine learning pipeline synthesis. In: ICML AutoML workshop (2018).
14. Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., Roth, A.: Generalization in adaptive data analysis and holdout reuse. In: Proc. of NIPS'15 [2], pp. 2350–2358.
15. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., Leyton-Brown, K.: Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In: NIPS Workshop on Bayesian Optimization in Theory and Practice (2013).

16. Escalante, H., Montes, M., Sucar, E.: Ensemble particle swarm model selection. In: Proc. of IJCNN'10. pp. 1–8. IEEE (Jul 2010).
17. Falkner, S., Klein, A., Hutter, F.: BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In: Proc. of ICML'18. pp. 1437–1446 (2018).
18. Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., Hutter, F.: Practical automated machine learning for the AutoML challenge 2018. In: ICML AutoML workshop (2018).
19. Feurer, M., Hutter, F.: Towards further automation in AutoML. In: ICML AutoML workshop (2018).
20. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Proc. of NIPS'15 [2], pp. 2962–2970.
21. Feurer, M., Springenberg, J., Hutter, F.: Initializing Bayesian hyperparameter optimization via meta-learning. In: Proc. of AAAI'15. pp. 1128–1135 (2015).
22. Gomes, T., Prudêncio, R., Soares, C., Rossi, A., Carvalho, A.: Combining meta-learning and search techniques to select parameters for support vector machines. Neurocomputing 75(1), 3–13 (2012).
23. Guyon, I., Bennett, K., Cawley, G., Escalante, H., Escalera, S., Ho, T., N.Macià, Ray, B., Saeed, M., Statnikov, A., Viegas, E.: Design of the 2015 ChaLearn AutoML Challenge. In: Proc. of IJCNN'15 (2015).
24. Guyon, I., Saffari, A., Dror, G., Cawley, G.: Model selection: Beyond the Bayesian/Frequentist divide. JMLR 11, 61–87 (2010).
25. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: An update. ACM SIGKDD Explorations Newsletter 11(1), 10–18 (2009).
26. Hamerly, G., Elkan, C.: Learning the k in k-means. In: Proc. of NIPS'04. pp. 281–288 (2004).
27. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proc. of LION'11. pp. 507–523 (2011).
28. Jamieson, K., Talwalkar, A.: Non-stochastic best arm identification and hyperparameter optimization. In: Proc. of AISTATS'16. pp. 240–248 (2016).
29. Kalousis, A.: Algorithm Selection via Meta-Learning. Ph.D. thesis, University of Geneve (2002).
30. Komer, B., Bergstra, J., Eliasmith, C.: Hyperopt-sklearn: Automatic hyperparameter configu- ration for scikit-learn. In: ICML workshop on AutoML (2014).
31. Lacoste, A., Marchand, M., Laviolette, F., Larochelle, H.: Agnostic Bayesian learning of ensembles. In: Proc. of ICML'14. pp. 611–619 (2014).
32. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. JMLR 18(185), 1–52 (2018).
33. Michie, D., Spiegelhalter, D., Taylor, C., Campbell, J.: Machine Learning, Neural and Statistical Classification. Ellis Horwood (1994).
34. Mohr, F., Wever, M., Hüllermeier, E.: Ml-plan: Automated machine learning via hierarchical planning. Machine Learning (2018).
35. Niculescu-Mizil, A., Perlich, C., Swirszcz, G., Sindhwani, V., Liu, Y., Melville, P., Wang, D., Xiao, J., Hu, J., Singh, M., Shang, W., Zhu, Y.: Winning the KDD cup

orange challenge with ensemble selection. The 2009 Knowledge Discovery in Data Competition pp. 23–34 (2009).
36. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. JMLR 12, 2825–2830 (2011).
37. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Meta-learning by landmarking various learning algorithms. In: Proc. of ICML'00. pp. 743–750 (2000).
38. Reif, M., Shafait, F., Dengel, A.: Meta-learning for evolutionary parameter optimization of classifiers. Machine Learning 87, 357–380 (2012).
39. de Sá, A., Pinto, W., Oliveira, L., Pappa, G.: RECIPE: a grammar-based framework for automatically evolving classification pipelines. In: Proc. of ECGP'17. pp. 246–261 (2017).
40. Shahriari, B., Swersky, K., Wang, Z., Adams, R., de Freitas, N.: Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE 104(1), 148–175 (2016).
41. Snoek, J., Larochelle, H., Adams, R.: Practical Bayesian optimization of machine learning algorithms. In: Proc. of NIPS'12. pp. 2960–2968 (2012).
42. Thornton, C., Hutter, F., Hoos, H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Proc. of KDD'13. pp. 847–855 (2013)
43. Vanschoren, J., van Rijn, J., Bischl, B., Torgo, L.: OpenML: Networked science in machine learning. SIGKDD Explorations 15(2), 49–60 (2013).
44. Wolpert, D.: Stacked generalization. Neural Networks 5, 241–259 (1992).
45. Yogatama, D., Mann, G.: Efficient transfer learning method for automatic hyperparameter tuning. In: Proc. of AISTATS'14. pp. 1077–1085 (2014).
46. Zhang, Y., Bahadori, M., Su, H., Sun, J.: FLASH: Fast Bayesian Optimization for Data Analytic Pipelines. In: Proc. of KDD'16. pp. 2065–2074 (2016).

Глава 7

На пути к автоматически настраиваемым глубоким нейронным сетям

Гектор Мендоза, **Аарон Кляйн**, **Матиас Фойрер**, **Йост Тобиас Шпрингенберг**, **Матиас Урбан**, **Михаэль Буркарт**, **Максимилиан Диппель**, **Мариус Линдауэр**, **Франк Хуттер** (✉), факультет компьютерных наук, Университет Фрайбурга, Фрайбург, Баден-Вюртемберг, Германия, e-mail: fh@informatik.uni-freiburg.de

Последние достижения в области AutoML привели к созданию автоматизированных инструментов, способных конкурировать с экспертами в области машинного обучения при решении задач обучения с учителем. В этой работе мы представляем две версии Auto-Net – системы, которая создает автоматически настраиваемые глубокие нейронные сети без вмешательства человека. Первая версия, Auto-Net 1.0, основана на идеях системы Auto-sklearn, победившей в конкурсе с помощью метода байесовской оптимизации SMAC, и использует Lasagne в качестве базовой библиотеки глубокого обучения (DL). Более современная Auto-Net 2.0 основана на новой комбинации байесовской оптимизации и системы HyperBand, так называемой BOHB, и в качестве библиотеки DL использует PyTorch. Насколько нам известно, Auto-Net 1.0 была первой автоматически настраиваемой нейронной сетью, которая выиграла соревнование с человеческими экспертами (в рамках первого конкурса AutoML). Полученные впоследствии эмпирические результаты показывают, что объединение Auto-Net 1.0 с Auto-sklearn может работать лучше, чем любой из подходов по отдельности, и что Auto-Net 2.0 может работать еще лучше.

7.1. Введение

В последние годы нейронные сети добились значительных успехов в прохождении различных тестов и открыли много новых перспективных на-

правлений исследований [22, 27, 36, 39, 41]. Однако нейронные сети сложны в использовании для неспециалистов, поскольку их производительность в значительной степени зависит от правильных настроек большого набора гиперпараметров (например, скорости обучения и затухания веса) и выбора архитектуры (например, количества слоев и типа функций активации). В этой главе мы знакомим читателей с нашим исследованием по созданию эффективных нейронных сетей на основе подходов автоматизированного машинного обучения (AutoML).

Цель AutoML – предоставить эффективные готовые к использованию системы обучения, чтобы освободить экспертов и неспециалистов от утомительных и трудоемких задач выбора подходящего алгоритма для набора данных, а также подходящего метода предварительной обработки и различных гиперпараметров всех задействованных компонентов. Торнтон и др. [43] сформулировали эту проблему AutoML как комбинированную задачу выбора алгоритма и оптимизации гиперпараметров (CASH), целью которой является определение комбинации компонентов алгоритма, обеспечивающей наилучший результат проверки (как правило, перекрестной).

Один из мощных подходов к решению задачи CASH рассматривает упомянутую перекрестную проверку как вычислительно дорогостоящую функцию черного ящика и использует байесовскую оптимизацию [4, 35] для поиска ее оптимизатора. Хотя байесовская оптимизация обычно использует гауссовы процессы (GP) [32], они, как правило, плохо согласуются с особыми свойствами задачи CASH (высокая размерность; наличие одновременно как категориальных, так и непрерывных гиперпараметров; множество условных гиперпараметров, которые применимы только при определенных значениях других гиперпараметров). Адаптация GP для обработки этих характеристик является активной областью исследований [40, 44], но до сих пор в условиях CASH [9, 43] лучше всего работают методы байесовской оптимизации с использованием древовидных моделей [2, 17].

Auto-Net создана по образцу двух известных систем AutoML Auto-WEKA [43] и Auto-sklearn [11], рассмотренных в главах 4 и 6 этой книги соответственно. Обе они используют метод байесовской оптимизации SMAC [17] на основе случайного леса для решения задачи CASH – поиска наилучшей реализации классификаторов в WEKA [16] и scikit-learn [30] соответственно. Система Auto-sklearn использует два дополнительных метода для повышения производительности. Во-первых, она использует метаобучение [3], основанное на опыте предыдущих наборов данных, чтобы запустить поиск SMAC с изначально хороших конфигураций [12]. Во-вторых, поскольку конечная цель – сделать лучшие предсказания, было бы нерационально опробовать десятки моделей машинного обучения и затем использовать только одну лучшую модель; вместо этого Auto-sklearn сохраняет все модели, оцененные SMAC, и строит из них ансамбль с помощью техники выбора ансамбля [5]. Несмотря на то что и Auto-WEKA, и Auto-sklearn включают широкий спектр методов обучения с учителем, ни одна из них не использует современные нейронные сети.

Здесь мы представляем две версии системы, которую мы назвали Auto-Net, чтобы заполнить этот пробел. Auto-Net 1.0 основана на Theano и имеет относительно простое пространство поиска, в то время как более совре-

менная Auto-Net 2.0 реализована на PyTorch и использует более сложное пространство и последние достижения в области глубокого обучения. Еще одно различие заключается в процедуре поиска: Auto-Net 1.0 автоматически настраивает нейронные сети с помощью SMAC [17], следуя тому же подходу AutoML, что и Auto-WEKA и Auto-sklearn, а Auto-Net 2.0 базируется на BOHB [10], комбинации байесовской оптимизации (BO) и эффективных стратегий гонок на основе HyperBand (HB) [23].

Мы описываем пространство конфигураций и реализацию Auto-Net 1.0 в разделе 7.2 и Auto-Net 2.0 в разделе 7.3. Затем мы описываем исследование их производительности в разделе 7.4 и делаем выводы в разделе 7.5. Мы опускаем подробное обсуждение смежных работ и отсылаем читателей к главе 3 этой книги для знакомства с обзором чрезвычайно активно развивающейся области поиска нейронных архитектур. Тем не менее стоит отметить, что несколько современных инструментов машинного обучения преследует те же цели, что и Auto-Net в автоматизации глубокого обучения; например, Auto-Keras [20], Photon-AI, H2O.ai, DEvol или сервис Google Cloud AutoML.

Эта глава является расширенной версией нашей статьи о системе Auto-Net, представленной на семинаре *2016 ICML Workshop on AutoML* [26].

7.2. Auto-Net 1.0

В этом разделе мы представляем версию Auto-Net 1.0 и описываем ее реализацию. Мы решили реализовать эту первую версию Auto-Net как расширение Auto-sklearn [11], добавив новый компонент классификации (и регрессии); причина такого выбора в том, что он позволяет нам использовать существующие части конвейера машинного обучения: предварительную обработку признаков, предварительную обработку данных и построение ансамбля. Здесь мы ограничиваем Auto-Net полносвязными нейронными сетями с прямым распространением, поскольку они применимы к широкому спектру различных наборов данных; мы откладываем для будущих версий расширение на другие типы нейронных сетей, такие как сверточные или рекуррентные нейронные сети. Для доступа к методам нейронных сетей мы используем библиотеку глубокого обучения Lasagne [6] на языке Python, которая построена на базе Theano [42]. Однако отметим, что в целом наш подход не зависит от реализации нейронной сети.

Основываясь на работах [2] и [7], мы различаем независимые от слоя гиперпараметры сети, которые управляют архитектурой и процедурой обучения, и послойные гиперпараметры, которые задаются для каждого слоя. Всего мы оптимизируем 63 гиперпараметра (табл. 7.1), используя одно и то же пространство конфигураций для всех типов обучения с учителем (бинарная, многоклассовая и многометочная классификация, а также регрессия). Разреженные наборы данных используют то же самое конфигурационное пространство. (Поскольку нейронные сети очень плохо работают с разреженными наборами данных, мы преобразуем данные в плотное представление попакетно, перед подачей пакета в нейронную сеть.)

Таблица 7.1. Пространство конфигураций Auto-Net. Описание пространства для методов предобработки можно найти в [11]

	Название	Диапазон	По умолчанию	log масшт.	Тип	Условный
Гиперпараметры сети	Размер пакета	[32, 4096]	32	✓	float	—
	Кол-во обновлений	[50, 2500]	200	✓	int	—
	Кол-во слоев	[1, 6]	1	—	int	—
	Скорость обучения	$[10^{-6}, 1.0]$	10^{-2}	✓	float	—
	Регуляризация L_2	$[10^{-7}, 10^{-2}]$	10^{-4}	✓	float	—
	Прореживание	[0.0, 0.99]	0.5	✓	float	—
	Тип решателя	{SGD, импульс, Adam, Adadelta, Adagrad, smorm, по Нестерову}	smorm3s	—	cat	—
	Стратегия скорости обучения	{фиксированная, обратная, степенная, шаговая]	фиксир.	—	cat	—
Обусловлено типом решателя	β_1	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	β_2	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	ρ	[0.05, 0.99]	0.95	✓	float	ü
	Импульс	[0.3, 0.999]	0.9	✓	float	✓
Обусловлено стратегией скорости обуч.	γ	$[10^{-3}, 10^{-1}]$	10^{-2}	✓	float	✓
	k	[0.0, 1.0]	0.5	—	float	✓
	s	[2, 20]	2	—	int	✓
Гиперпараметры для каждого слоя	Тип активации	{сигмоидная, TanH, ScaledTanH, ELU, ReLU, с утечкой, линейная}	ReLU	—	cat	✓
	Кол-во нейронов	[64. 4096]	128	✓	int	✓
	Прореживание	[0.0. 0.99]	0.5	—	float	✓
	Инициализация весов	{константа, нормальная, равномерная, равномерная по Глоро, нормальная по Глоро, нормальная по Ге, равномерная по Ге, ортогональная, разреженная}	норм. по Ге	—	cat	✓
	Стандартная норм. инициализация	$[10^{-7}, 0.1]$	0.0005	—	float	✓
	Просачивание	[0.01, 0.99]	1/3	—	float	✓
	Танг. масшт. вниз	[0.5, 1.0]	2/3	—	float	✓
	Танг. масшт. вверх	[1.1, 3.0]	1.7159	✓	float	✓

Гиперпараметры каждого слоя k условно зависят от количества слоев не менее k. Из практических соображений мы ограничиваем количество слоев от одного до шести: во-первых, мы стремимся к тому, чтобы время обучения одной конфигурации было небольшим[1], а во-вторых, каждый слой добавляет восемь гиперпараметров слоя в пространство конфигурации, поэтому добавление новых слоев еще больше усложняет процесс конфигурирования.

Наиболее распространенным способом оптимизации внутренних весов нейронных сетей является стохастический градиентный спуск (stochastic gradient descent, SGD) с использованием частных производных, вычисляемых с помощью механизма обратного распространения. Стандартный SGD сильно зависит от правильной настройки гиперпараметра скорости обучения. Чтобы уменьшить эту зависимость, были предложены различные алгоритмы *решателей* (solver) для стохастического градиентного спуска. Мы включили в пространство конфигураций Auto-Net следующие хорошо известные из литературы методы: базовый стохастический градиентный спуск (SGD), стохастический градиентный спуск с импульсом (Momentum), Adam [21], Adadelta [48], импульс Нестерова [28] и Adagrad [8]. Кроме того, мы использовали вариант оптимизатора vSGD [33], получивший название «smorm», в котором оценка гессиана заменяется оценкой квадратичного градиента (вычисляется как в процедуре RMSprop). Каждый из этих методов имеет скорость обучения α и собственный набор гиперпараметров, например в Adam это векторы импульса β_1 и β_2. Гиперпараметр(ы) каждого решателя активен(ы) только в том случае, если выбран соответствующий решатель.

Мы также уменьшаем скорость обучения α со временем, используя следующие стратегии (которые умножают начальную скорость обучения на коэффициент α_{decay} после каждой эпохи $t = 0 \ldots T$):

- фиксированную: $\alpha_{decay} = 1$;
- обратную: $\alpha_{decay} = (1 + \gamma t)^{(-k)}$;
- степенную: $\alpha_{decay} = \gamma^t$;
- шаговую: $\alpha_{decay} = \gamma^{\lfloor t/s \rfloor}$.

Здесь гиперпараметры k, s и γ условно зависят от выбора стратегии.

Для поиска сильной реализации в этом условном пространстве поиска Auto-Net 1.0 по аналогии с Auto-WEKA и Auto-sklearn мы использовали метод байесовской оптимизации SMAC [17], основанный на методе случайного леса. SMAC – это подход к оптимизации, который отслеживает наилучшую конфигурацию, найденную на данный момент, и выдает ее при завершении.

7.3. Auto-Net 2.0

Версия AutoNet 2.0 отличается от AutoNet 1.0 в основном следующими тремя аспектами:

- использует PyTorch [29] вместо Lasagne в качестве библиотеки глубокого обучения;

[1] Мы стремились к тому, чтобы иметь возможность оценить несколько десятков конфигураций в течение двух дней на одном процессоре.

- использует большее пространство конфигураций, включая современные методы глубокого обучения, современные архитектуры (такие как ResNets), и включает более компактные представления пространства поиска;
- применяет BOHB [10] вместо SMAC для более эффективного построения хорошо работающей нейронной сети.

Далее мы обсудим эти моменты более подробно.

Поскольку разработка и поддержка Lasagne прекращена, нам пришлось выбрать другую библиотеку Python для Auto-Net 2.0. Наиболее популярными библиотеками глубокого обучения в настоящее время являются PyTorch [29] и Tensorflow [1]. Они обладают довольно схожими возможностями и в основном отличаются уровнем детализации, который они предоставляют. Например, PyTorch предлагает пользователю возможность отслеживать все вычисления во время обучения. Хотя у каждой из этих библиотек есть свои преимущества и недостатки, мы решили использовать PyTorch из-за ее способности динамически строить вычислительные графы. По этой причине мы также стали называть Auto-Net 2.0 альтернативным именем Auto-PyTorch.

Пространство поиска AutoNet 2.0 включает как гиперпараметры для выбора модуля (например, тип планировщика, архитектура сети), так и гиперпараметры для каждого из конкретных модулей. Система поддерживает различные модули глубокого обучения, такие как тип сети, планировщик скорости обучения, оптимизатор и метод регуляризации, как будет описано далее. Auto-Net 2.0 также легко расширяется; пользователи могут добавлять свои собственные модули.

В настоящее время Auto-Net 2.0 предлагает четыре различных типа сетей.

Многослойные перцептроны (MLP). Это стандартная реализация обычных MLP, расширенная *слоями прореживания* (dropout layer) [38]. Как и в AutoNet 1.0, каждый слой MLP параметризуется (например, количество нейронов и коэффициент прореживания).

Остаточные нейронные сети (ResNet). Это глубокие нейронные сети, которые обучаются *функциям невязки* или *остаточным функциям* (residual function) [47] с той разницей, что мы используем полностью связанные слои вместо сверточных. Как и положено для ResNet, архитектура состоит из M групп, в каждой из которых последовательно уложено N остаточных блоков. В то время как архитектура каждого блока неизменна, количество групп M, количество блоков в группе N, а также ширина каждой группы определяется гиперпараметрами, как показано в табл. 7.2.

Фигурные многослойные перцептроны (ShapedML). Чтобы избежать ситуации, когда каждый слой имеет свои собственные гиперпараметры (что является неэффективным представлением для поиска), в фигурных MLP общая форма слоев предопределена, например, в виде воронки, длинной воронки, ромба, шестиугольника, прямоугольника или треугольника. Мы использовали формы с сайта https://mikkokotila.github.io/slate/#shapes; параметризацию по таким формам предложил нам Илья Лощилов [25].

Фигурные остаточные сети (ShapedResNet), в которых общая форма слоев предопределена (например, воронка, длинная воронка, ромб, шестиугольник, прямоугольник, треугольник).

Типы сетей ResNets и ShapedResNets могут также использовать любой из методов регуляризации Shake-Shake [13] и ShakeDrop [46]. Метод MixUp [49] также может быть использован для всех сетей.

В настоящее время в Auto-Net 2.0 поддерживаются такие оптимизаторы, как Adam [21] и SGD с импульсом. Более того, Auto-Net 2.0 предлагает пять различных планировщиков, которые изменяют скорость обучения оптимизатора во времени (как функцию от количества эпох).

Экспоненциальный. Умножает скорость обучения на постоянный коэффициент в каждой эпохе.

Шаговый. Уменьшает скорость обучения на мультипликативный коэффициент после постоянного числа шагов.

Циклический. Изменяет скорость обучения в определенном диапазоне, чередуя увеличение и уменьшение [37].

Косинусный отжиг с теплым перезапуском [24]. Этот график скорости обучения реализует несколько фаз сходимости. Он охлаждает скорость обучения до нуля по косинусоидальному закону [24], а после каждой фазы сходимости нагревает ее, чтобы начать следующую фазу сходимости, часто к лучшему оптимуму. Веса сети не изменяются при нагревании (увеличении) скорости обучения, так что следующая фаза сходимости запускается в теплом режиме.

OnPlateau. Этот планировщик[1] изменяет скорость обучения всякий раз, когда метрика перестает улучшаться; в частности, он умножает текущую скорость обучения на коэффициент γ, если после p эпох не было улучшения.

Подобно Auto-Net 1.0, Auto-Net 2.0 может осуществлять поиск среди методов предварительной обработки. В настоящее время Auto-Net 2.0 поддерживает анализ главных компонент по Нюстрёму [45], ядерный анализ главных компонент [34], быстрый независимый анализ компонент [18], Random Kitchen Sink [31] и усеченное сингулярное разложение [15]. Пользователи могут указать список методов предварительной обработки, которые следует рассматривать, а также выбрать между различными стратегиями балансировки и нормализации (для стратегий балансировки доступно только взвешивание потерь, а для стратегий нормализации поддерживаются нормализация min-max и стандартизация). В отличие от Auto-Net 1.0, Auto-Net 2.0 не строит ансамбль в конце (хотя эта функциональность, вероятно, будет добавлена в ближайшее время). Все гиперпараметры Auto-Net 2.0 с соответствующими диапазонами и значениями по умолчанию приведены в табл. 7.2.

В качестве оптимизатора для этого высоко обусловленного пространства мы использовали BOHB [10], который сочетает обычную байесовскую оптимизацию со стратегией работы с однорукими бандитами Hyperband [23] для существенного повышения ее эффективности. Как и Hyperband, BOHB использует повторные прогоны алгоритма Successive Halving [19], чтобы потратить большую часть времени на работу с перспективными нейронными сетями и прекратить обучение нейронных сетей с низкой производительностью на ранней стадии. Аналогично байесовской оптимизации BOHB учится тому, какие виды нейронных сетей дают хорошие результаты. В частности,

[1] Реализован с помощью PyTorch.

как и в методе байесовской оптимизации TPE [2], BOHB использует ядерную оценку плотности (kernel density estimator, KDE) для описания областей высокой производительности в пространстве нейронных сетей (архитектур и настроек гиперпараметров) и на основании KDE делает выбор между исследованием и использованием. Одним из преимуществ алгоритма BOHB является то, что он легко распараллеливается, достигая почти линейного ускорения с увеличением числа воркеров [10].

Таблица 7.2. Пространство конфигураций Auto-Net 2.0. Всего существует 112 гиперпараметров

	Название гиперпараметра	**Диапазон значений**	**По умолчанию**	**Log масштаб**	**Тип**	**Условный**
Общие гиперпараметры	Размер пакета	[32,500]	32	✓	int	–
	Использ. подмешивания	{True, False}	Tine	–	bool	–
	Коэф. подмешивания	[0.0,1.0]	1.0	–	float	✓
	Сеть	{MLP. ResNet, ShapedMLP. ShapedResNet}	MLP	–	cat	–
	Оптимизатор	{Adam, SGD}	Adam	–	cat	–
	Обработчик	{nystroem, kernel pea. fast ica, kitchen sinks, truncated svd}	Nystroem	–	cat	–
	Подстановка	{most frequent, median, mean}	Most frequent	–	cat	–
	Исп. стратегию весов потерь	{True, False}	Tine	–	cat	–
	Планировщик скор. обучения	{Step, Exponential, OnPlateau, Cyclic, Cosine Annealing}	Step	–	cat	–
Предобработка						
Нистром	Коэффициент	[-1.0.1.0]	0.0	–	float	✓
	Степень	[2,5]	3	–	int	✓
	Гамма	[0.00003.8.0]	0.1	✓	float	✓
	Ядро	{poly, rbf, sigmoid, cosine}	rbf	–	cat	✓
	Кол-во компонент	[50.10000]	100	✓	int	✓
Случайно-реалистический	Гамма	[0.00003.8.0]	1.0	✓	float	✓
	Кол-во компонент	[50.10000]	100	✓	int	✓
Усеченный SVD	Целевое разрешение	[10,256]	128	–	int	✓
Ядерный PCA	Коэффициент	[-1.0.1.0]	0.0	–	float	✓
	Степень	[2,5]	3	–	int	✓
	Гамма	[0.00003.8.0]	0.1	✓	float	✓
	Ядро	{poly, rbf, sigmoid, cosine}	rbf	–	cat	✓
	Кол-во компонент	[50.10000]	100	✓	int	✓
Быстрый ICA	Алгоритм	{parallel, deflation}	Parallel	–	cat	✓
	Отжиг	{logcosh, exp, cube}	Logcosh	–	cat	✓
	Отбеливание	{True, False}	Tine	–	cat	✓
	Кол-во компонент	[10,2000]	1005	–	int	✓

Таблица 7.2 *(продолжение)*

	Название гиперпараметра	Диапазон значений	По умолчанию	Log масштаб	Тип	Условный
Сети						
MLP	Функция активации	{Sigmoid. Tanh. ReLu}	Sigmoid	–	cat	✓
	Кол-во слоев	[1,15]	9	–	int	✓
	Кол-во эл. (для слоя z)	[10.1024]	100	✓	int	✓
	Усечение (для слоя *i*)	[0.0.0.5]	0.25	–	int	✓
ResNet	Функция активации	{Sigmoid. Tanh. ReLu}	Sigmoid	–	cat	✓
	Группы остаточных блоков	[1-9]	4	–	int	✓
	Блоков на группу	[13]	2	–	int	✓
	Кол-во эл. (для группы z)	[128,1024]	200	✓	int	✓
	Использовать усечение	{True, False}	Tine	–	bool	✓
	Усечение (для группы z)	[0.0.0.9]	0.5	–	int	✓
	Исп. усечение с перемеш.	{True, False}	Tine	–	bool	✓
	Исп. двойное перемеш.	{True, False}	True	–	bool	✓
	Коэффициент β_{max}	[0.0.1.0]	0.5	–	float	✓
Ограниченный MLP	Функция активации	{Sigmoid. Tanh. ReLu}	Sigmoid	–	cat	✓
	Кол-во слоев	[3.15]	9	–	int	✓
	Кол-во элементов на слой	[10.1024]	200	✓	int	✓
	Форма сети	{Funnel. LongFunnel, Diamond. Hexagon. Brick. Triangle, Stairs}	Funnel	–	cat	✓
	Макс. усечение на слой	[0.0.0.6	0.2	–	float	✓
	Форма усечения	{Funnel. LongFunnel. Diamond. Hexagon. Brick. Triangle, Stairs}	Funnel	–	cat	✓
Ограниченный ResNet	Функция активации	{Sigmoid. Tanh. ReLu}	Sigmoid	–	cat	✓
	Кол-во слоев	[3-9]	4	–	int	✓
	Блоков на слой	[1-4]	2	–	int	✓
	Исп. усечение	{True, False}	Tine	–	bool	✓
	Макс. элем. на слой	[10.1024]	200	✓	int	✓
	Форма сети	{Funnel. LongFunnel. Diamond. Hexagon. Brick. Triangle, Stairs}	Funnel	–	cat	✓
	Макс. усечение на слой	[0.0.0.6	0.2	–	float	✓

Таблица 7.2 *(окончание)*

	Название гиперпараметра	Диапазон значений	По умолчанию	Log масштаб	Тип	Условный
Ограниченный ResNet	Форма усечения	{Funnel. LongFunnel. Diamond. Hexagon. Brick. Triangle, Stairs}	Funnel	–	cat	✓
	Исп. усечение с перемеш.	{True, False}	True	–	bool	✓
	Исп. двойное перемеш.	{True, False}	Tine	–	bool	✓
	Коэффициент β_{max}	[0.0.1.0]	0.5	–	float	✓
Оптимизаторы						
Adam	Скорость обучения	[0.0001,0.1]	0.003	✓	float	✓
	Затухание веса	[0.0001,0.1]	0.05	–	float	✓
SGD	Скорость обучения	[0.0001,0.1]	0.003	✓	float	✓
	Затухание веса	[0.0001,0.1]	0.05	–	float	✓
	Импульс	[0.1,0.9]	0.3	✓	float	✓
Планировщики						
Шаг	γ	[0.001,0.9]	0.4505	–	float	✓
	Размер шага	[1,10]	6	–	int	✓
Экспоненциальный	γ	[0.8,0.9999]	0.89995	–	float	✓
OnPlateau	γ	[0.05.0.5]	0.275	–	float	✓
	Выдержка	[3,10]	6	–	int	✓
Циклический	Длина цикла	[3,10]	6	–	int	✓
	Макс. коэффициент	[1.0,2.0]	1.5	–	float	✓
	Мин. коэффициент	[0.001,1.0]	0.5	–	float	✓
Косинусный отжиг	T_o	[1,20]	10	–	int	✓
	T_{mult}	[1.0,2.0]	1.5	–	float	✓

В качестве бюджета для BOHB мы можем использовать либо эпохи, либо время в минутах; по умолчанию мы используем время выполнения, но пользователи могут свободно настраивать различные параметры бюджета. Пример использования Auto-Net 2.0 показан в алгоритме 1. Подобно Auto-sklearn, Auto-Net построен как плагин-оцениватель для scikit-learn. Пользователи должны предоставить ему обучающий набор и метрику производительности (например, точность). Опционально они могут указать проверочный и тестовый наборы. Проверочный набор используется во время обучения для получения оценки производительности сети и для обучения KDE-моделей BOHB.

Алгоритм 1. Пример использования Auto-Net 2.0.

```
from autonet import AutoNetClassification

cls = AutoNetClassification(min_budget=5, max_budget=20, max_runtime=120)
cls.fit(X_train, Y_train)
predictions = cls.predict(X_test)
```

7.4. Эксперименты

Проведем эмпирическую оценку наших методов. Реализации Auto-Net работают как на CPU, так и на GPU, но, поскольку нейронные сети интенсивно используют матричные операции, они работают намного быстрее на GPU. Наши эксперименты на базе CPU проводились на вычислительном кластере, каждый узел которого имеет два восьмиядерных процессора Intel Xeon E5-2650 v2, работающих на частоте 2.6 ГГц, и общую память 64 Гб. Наши эксперименты на базе GPU проводились на вычислительном кластере, каждый узел которого оснащен четырьмя графическими процессорами GeForce GTX TITAN X.

7.4.1. Первичная оценка Auto-Net 1.0 и Auto-sklearn

В нашем первом эксперименте мы сравниваем различные экземпляры Auto-Net 1.0 на пяти наборах данных фазы 0 конкурса AutoML. Во-первых, мы используем версии на базе CPU и GPU, чтобы изучить разницу в работе нейросетей на разном оборудовании. Во-вторых, мы допускаем комбинирование нейронных сетей с моделями из Auto-sklearn. В-третьих, мы также запустили Auto-sklearn без нейронных сетей в качестве исходного уровня. На каждом наборе данных мы выполнили по 10 однодневных прогонов каждого метода, выделив до 100 мин для оценки одной конфигурации путем пятикратной перекрестной проверки на обучающем наборе. Для каждого этапа каждого прогона, согласно [11], мы построили ансамбль из моделей, оцененных до этого момента, и построили график ошибки при тестировании этого ансамбля. На практике мы либо использовали бы отдельный процесс для параллельного расчета ансамблей, либо рассчитывали бы их после процесса оптимизации.

На рис. 7.1 показаны результаты на двух из пяти наборов данных. Прежде всего мы отмечаем, что версия Auto-Net на базе GPU постоянно была примерно на порядок быстрее, чем версия на базе CPU. В рамках заданного фиксированного бюджета вычислений версия на базе CPU постоянно показывала худшие результаты, в то время как версия на базе GPU показала лучшие результаты на наборе данных `newsgroups` (рис. 7.1a), сравнялась с Auto-sklearn на трех других наборах данных и показала худшие результаты на одном наборе. Несмотря на то что версия Auto-Net, выполняемая на CPU, была очень медленной, в 3/5 случаев комбинация Auto-sklearn и Auto-Net на CPU все равно превосходила Auto-sklearn; это можно наблюдать, например, для набора данных `dorothea` на рис. 7.1b.

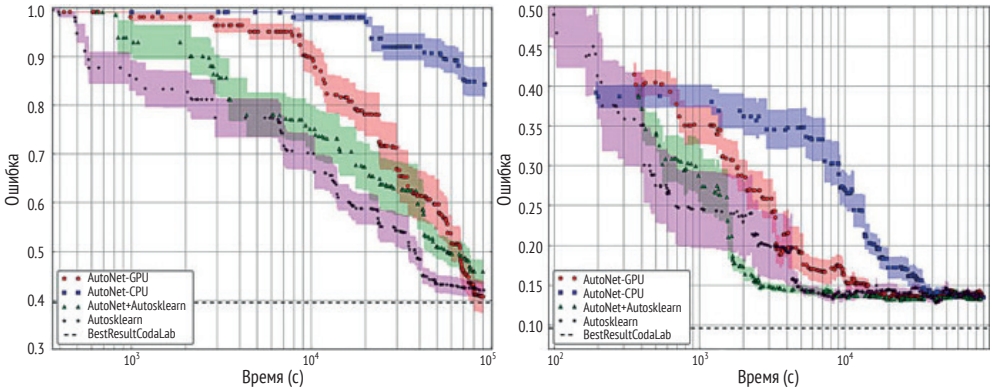


Рис. 7.1 ❖ Результаты четырех методов на двух наборах данных из этапа *Tweakathon0* конкурса AutoML. Мы показываем ошибки на проверочном наборе (не на тестовом, поскольку его истинные метки недоступны), а наши методы имеют доступ только к обучающему набору. Чтобы избежать путаницы, мы приводим среднюю ошибку ±1/4 стандартного отклонения за 10 запусков каждого метода. Здесь использованы наборы данных: (a) `newsgroups` (b) `dorothea`

7.4.2. Результаты для наборов данных конкурса AutoML

Разработав Auto-Net 1.0 во время первого конкурса AutoML, мы использовали комбинацию Auto-sklearn и Auto-Net на базе GPU на последних двух этапах, чтобы победить в соответствующих треках экспертов. Алгоритм Auto-sklearn разрабатывали гораздо дольше, и он является гораздо более надежным, чем Auto-Net, поэтому для 4/5 наборов данных на третьем этапе и 3/5 наборов данных на четвертом этапе Auto-sklearn показал наилучшие результаты сам по себе, и мы представили только его результаты. Здесь мы обсудим три набора данных, для которых мы использовали Auto-Net. На рис. 7.2 показаны официальные результаты конкурса AutoML среди экспертов (людей) для трех наборов данных, для которых мы использовали Auto-Net. Набор данных `alexis` был частью 3-го этапа («продвинутой фазы») конкурса. Для этого мы запустили Auto-Net на пяти GPU параллельно (задействуя SMAC в режиме совместного использования моделей) в течение 18 ч. Наша работа представляла собой автоматически построенный ансамбль из 39 моделей и явно превзошла всех экспертов-людей, достигнув оценки AUC 90 %, в то время как лучший конкурент-человек (Ideal Intel Analytics) достиг только 80 %. Насколько нам известно, это первый случай, когда автоматически построенная нейронная сеть победила на конкурсном наборе данных. Наборы данных `yolanda` и `tania` были частью 4-го этапа («экспертной фазы») конкурса. Для `yolanda` мы запускали Auto-Net в течение 48 ч на восьми графических процессорах и автоматически построили ансамбль из пяти нейронных сетей, заняв третье место. Для `tania` мы запускали Auto-Net в течение 48 ч на восьми GPU

вместе с Auto-sklearn на 25 CPU, и в итоге наш скрипт автоматической сборки построил ансамбль из восьми однослойных нейронных сетей, двух двухслойных нейронных сетей и одной модели логистической регрессии, обученной с помощью SGD. Этот ансамбль занял первое место на наборе данных `tania`.

Для набора данных `tania` мы также повторили эксперименты из раздела 7.4.1. На рис. 7.3 показано, что для этого набора данных Auto-Net работает явно лучше, чем Auto-sklearn, даже при работе только на CPU. Вариант Auto-Net на базе GPU показал лучшие результаты.

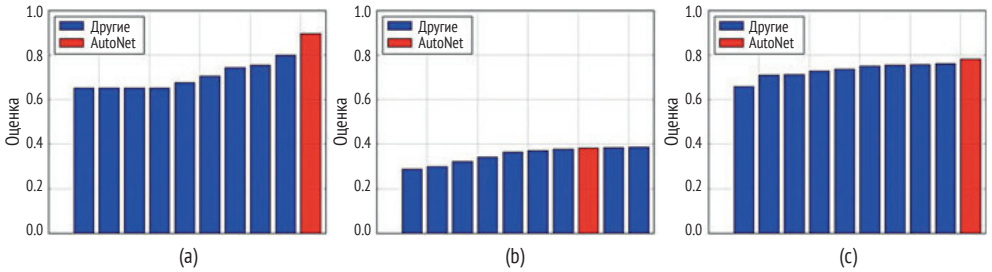


Рис. 7.2 ❖ Официальные результаты конкурса AutoML, показанные экспертами-людьми на трех наборах данных, для которых мы использовали Auto-Net. Мы показываем только 10 лучших работ на следующих наборах: (a) `alexis` (b) `yolanda` (c) `tania`

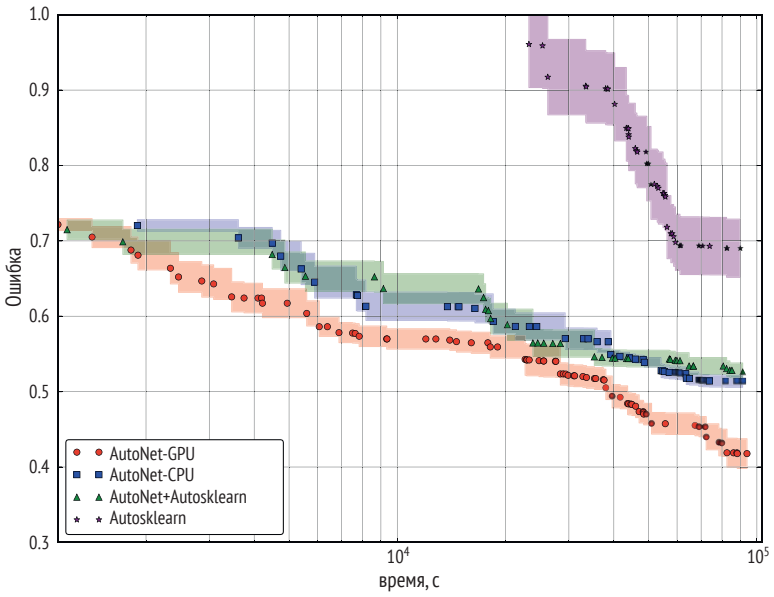


Рис. 7.3 ❖ Производительность на наборе данных `tania`. Показаны результаты перекрестной проверки на обучающем множестве, поскольку эталонные метки для проверочного или тестового множества конкурса недоступны. Чтобы избежать путаницы, мы показываем среднюю ошибку ±1/4 стандартного отклонения за 10 запусков каждого метода

7.4.3. Сравнение AutoNet 1.0 и 2.0

Наконец, проведем наглядное сравнение между Auto-Net 1.0 и 2.0. Необходимо отметить, что Auto-Net 2.0 имеет гораздо более обширное пространство поиска, чем Auto-Net 1.0, и поэтому мы ожидаем, что при достаточном количестве времени он будет лучше работать с большими наборами данных. Мы также ожидаем, что поиск в большом пространстве сложнее, чем поиск в меньшем пространстве Auto-Net 1.0; однако, поскольку Auto-Net 2.0 использует эффективный оптимизатор BOHB для раннего завершения плохо работающих нейронных сетей, он может получить высокую производительность в любое время. С другой стороны, Auto-Net 2.0 пока не реализует ансамблирование, и из-за отсутствия этого компонента регуляризации и большего пространства гипотез он может быть более склонен к переобучению, чем Auto-Net 1.0.

Чтобы проверить эти ожидания относительно производительности на различных по размеру наборах данных, мы использовали средний набор данных (`newsgroups` с 13 тыс. точек обучающих данных) и небольшой (`dorothea` с 800 точками обучающих данных). Результаты представлены в табл. 7.3.

На среднем наборе данных `newsgroups` Auto-Net 2.0 показал гораздо лучшие результаты, чем Auto-Net 1.0, а использование четырех воркеров также привело к значительному ускорению, что сделало Auto-Net 2.0 конкурентоспособным по сравнению с ансамблем Auto-sklearn и Auto-Net 1.0. Мы обнаружили, что, несмотря на большее пространство поиска Auto-Net 2.0, его производительность (при использовании метода переменной точности BOHB) всегда была лучше, чем у Auto-Net 1.0 (при использовании метода оптимизации черного ящика SMAC). На небольшом наборе данных `dorothea`, версия Auto-Net 2.0 также показала лучшие результаты, чем Auto-Net 1.0 на ранних этапах, но через некоторое время Auto-Net 1.0 показала несколько лучшие результаты. Мы объясняем это явление отсутствием ансамбля в Auto-Net 2.0 в сочетании с большим пространством поиска.

Таблица 7.3. Метрика ошибок различных версий Auto-Net, выполняемых в течение разного времени (все на CPU). Мы сравниваем Auto-Net 1.0, ансамбли Auto-Net 1.0 и Auto-sklearn, Auto-Net 2.0 с одним воркером и Auto-Net 2.0 с четырьмя воркерами. Все результаты являются средними по 10 запускам каждой системы. Мы показываем ошибки на проверочном конкурсном множестве (не на тестовом множестве, поскольку его эталонные метки недоступны)

	`newsgroups`			`dorothea`		
	10^3 с	**10^4 с**	**1 день**	**10^3 с**	**10^4 с**	**1 день**
Auto-Net 1.0	0.99	0.98	0.85	0.38	0.30	0.13
Auto-sklearn + Auto-Net 1.0	0.94	0.76	0.47	0.29	0.13	0.13
Auto-Net 2.0: 1 воркер	1.0	0.67	0.55	0.88	0.17	0.16
Auto-Net 2.0: 4 воркера	0.89	0.57	0.44	0.22	0.17	0.14

7.5. Заключение

Мы рассмотрели систему Auto-Net, которая создает автоматически настраиваемые глубокие нейронные сети без вмешательства человека. Несмотря на то что нейронные сети демонстрируют превосходную производительность на многих наборах данных, для традиционных наборов данных с заданными вручную характеристиками они не всегда демонстрируют наилучшие результаты. Однако мы показали, что даже в тех случаях, когда другие методы работают лучше, объединение Auto-Net и Auto-sklearn в ансамбль часто приводит к равной или лучшей производительности, чем любой из подходов в отдельности. Наконец, мы представили результаты по трем наборам данных конкурса AutoML, в котором Auto-Net заняла одно третье место и два первых места. Мы показали, что ансамбли Auto-sklearn и Auto-Net могут дать пользователям лучшее из обоих миров и довольно часто превосходят отдельные инструменты. Первые эксперименты с новой версией Auto-Net 2.0 показали, что использование более обширного пространства поиска в сочетании с BOHB в качестве оптимизатора дает многообещающие результаты.

В дальнейшем мы намерены расширить Auto-Net на более общие архитектуры нейронных сетей, включая сверточные и рекуррентные нейронные сети.

Благодарности. Эта работа была частично поддержана Европейским исследовательским советом (ERC) в рамках программы исследований и инноваций Европейского союза «Горизонт 2020» по гранту № 716721.

7.6. Литература

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). pp. 265–283 (2016), https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.
2. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K. (eds.) Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NIPS'11). pp. 2546–2554 (2011).
3. Brazdil, P., Giraud-Carrier, C., Soares, C., Vilalta, R.: Metalearning: Applications to Data Mining. Springer Publishing Company, Incorporated, 1 edn. (2008).
4. Brochu, E., Cora, V., de Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hi-

erarchical reinforcement learning. Computing Research Repository (CoRR) abs/1012.2599 (2010).
5. Caruana, R., Niculescu-Mizil, A., Crew, G., Ksikes, A.: Ensemble selection from libraries of models. In: In Proceedings of the 21st International Conference on Machine Learning. pp. 137–144. ACM Press (2004).
6. Dieleman, S., Schlüter, J., Raffel, C., Olson, E., Sønderby, S., Nouri, D., Maturana, D., Thoma, M., Battenberg, E., Kelly, J., Fauw, J.D., Heilman, M., diogo149, McFee, B., Weideman, H., takacsg84, peterderivaz, Jon, instagibbs, Rasul, K., CongLiu, Britefury, Degrave, J.: Lasagne: First release. (Aug 2015), https://doi.org/10.5281/zenodo.27878.
7. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Yang, Q., Wooldridge, M. (eds.) Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'15). pp. 3460–3468 (2015).
8. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. 12, 2121–2159 (Jul 2011).
9. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., Leyton-Brown, K.: Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In: NIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'13) (2013).
10. Falkner, S., Klein, A., Hutter, F.: Combining hyperband and bayesian optimization. In: NIPS 2017 Bayesian Optimization Workshop (Dec 2017).
11. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J.T., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NIPS'15) (2015).
12. Feurer, M., Springenberg, T., Hutter, F.: Initializing Bayesian hyperparameter optimization via meta-learning. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-nineth National Conference on Artificial Intelligence (AAAI'15). pp. 1128–1135. AAAI Press (2015).
13. Gastaldi, X.: Shake-shake regularization. CoRR abs/1705.07485 (2017).
14. Guyon, I., Bennett, K., Cawley, G., Escalante, H.J., Escalera, S., Ho, T.K., Macià, N., Ray, B., Saeed, M., Statnikov, A., Viegas, E.: Design of the 2015 chalearn AutoML challenge. In: 2015 International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (July 2015).
15. Halko, N., Martinsson, P., Tropp, J.: Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions (2009).
16. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: An update. SIGKDD Explorations 11(1), 10–18 (2009).
17. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C. (ed.) Proceedings of the Fifth International Conference on Learning and Intelligent Optimiza-

tion (LION'11). Lecture Notes in Computer Science, vol. 6683, pp. 507–523. Springer-Verlag (2011).
18. Hyvärinen, A., Oja, E.: Independent component analysis: algorithms and applications. Neural networks 13(4–5), 411–430 (2000).
19. Jamieson, K., Talwalkar, A.: Non-stochastic best arm identification and hyperparameter optimization. In: Gretton, A., Robert, C. (eds.) Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS. JMLR Workshop and Conference Proceedings, vol. 51, pp. 240–248. JMLR.org (2016).
20. Jin, H., Song, Q., Hu, X.: Efficient neural architecture search with network morphism. CoRR abs/1806.10282 (2018).
21. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: Proceedings of the International Conference on Learning Representations (2015).
22. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. In: Bartlett, P., Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NIPS'12). pp. 1097–1105 (2012).
23. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research 18, 185:1–185:52 (2017).
24. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. In: International Conference on Learning Representations (ICLR) 2017 Conference Track (2017).
25. Loshchilov, I.: Personal communication (2017).
26. Mendoza, H., Klein, A., Feurer, M., Springenberg, J., Hutter, F.: Towards automatically-tuned neural networks. In: ICML 2016 AutoML Workshop (2016).
27. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015).
28. Nesterov, Y.: A method of solving a convex programming problem with convergence rate O(1/sqr(k)). Soviet Mathematics Doklady 27, 372–376 (1983).
29. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: Autodiff Workshop at NIPS (2017).
30. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830 (2011).
31. Rahimi, A., Recht, B.: Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In: Advances in neural information processing systems. pp. 1313–1320 (2009).
32. Rasmussen, C., Williams, C.: Gaussian Processes for Machine Learning. The MIT Press (2006).

33. Schaul, T., Zhang, S., LeCun, Y.: No More Pesky Learning Rates. In: Dasgupta, S., McAllester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning (ICML'13). Omnipress (2014).
34. Schölkopf, B., Smola, A., Müller, K.: Kernel principal component analysis. In: International Conference on Artificial Neural Networks. pp. 583–588. Springer (1997).
35. Shahriari, B., Swersky, K., Wang, Z., Adams, R., de Freitas, N.: Taking the human out of the loop: A Review of Bayesian Optimization. Proc. of the IEEE 104(1) (12/2015 2016).
36. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. Nature 529, 484–503 (2016).
37. Smith, L.N.: Cyclical learning rates for training neural networks. In: Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on. pp. 464–472. IEEE (2017).
38. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15(1), 1929–1958 (2014).
39. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. CoRR abs/1409.3215 (2014), http://arxiv.org/abs/1409.3215.
40. Swersky, K., Duvenaud, D., Snoek, J., Hutter, F., Osborne, M.: Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces. In: NIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'13) (2013).
41. Taigman, Y., Yang, M., Ranzato, M., Wolf, L.: Deepface: Closing the gap to human-level performance in face verification. In: Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'14). pp. 1701–1708. IEEE Computer Society Press (2014).
42. Theano Development Team: Theano: A Python framework for fast computation of mathematical expressions. Computing Research Repository (CoRR) abs/1605.02688 (may 2016).
43. Thornton, C., Hutter, F., Hoos, H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: I. Dhillon, Koren, Y., Ghani, R., Senator, T., Bradley, P., Parekh, R., He, J., Grossman, R., Uthurusamy, R. (eds.) The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13). pp. 847–855. ACM Press (2013).
44. Wang, Z., Hutter, F., Zoghi, M., Matheson, D., de Feitas, N.: Bayesian optimization in a billion dimensions via random embeddings. Journal of Artificial Intelligence Research 55, 361–387 (2016).
45. Williams, C., Seeger, M.: Using the nyström method to speed up kernel machines. In: Advances in neural information processing systems. pp. 682–688 (2001).
46. Yamada, Y., Iwamura, M., Kise, K.: Shakedrop regularization. CoRR abs/1802.02375 (2018).

47. Zagoruyko, S., Komodakis, N.: Wide residual networks. CoRR abs/1605.07146 (2016).
48. Zeiler, M.: ADADELTA: an adaptive learning rate method. CoRR abs/1212.5701 (2012), http://arxiv.org/abs/1212.5701.
49. Zhang, H., Cissé, M., Dauphin, Y., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. CoRR abs/1710.09412 (2017).

Глава 8

TPOT: инструмент оптимизации конвейеров на основе деревьев для автоматизации машинного обучения

Рэндал Олсон (✉), Life Epigenetics, Миннеаполис, штат Массачусетс, США, e-mail: rso@randalolson.com

Джейсон Мур, Институт биомедицинской информатики, Университет Пенсильвании, Филадельфия, штат Пенсильвания, США

По мере того как наука о данных становится все более популярной, будет расти спрос на более доступные, гибкие и масштабируемые инструменты для работы с данными. В ответ на этот спрос исследователи в области автоматизированного машинного обучения (AutoML) начали создавать системы, автоматизирующие процесс проектирования и оптимизации конвейеров машинного обучения. В этой главе мы представляем TPOT v0.3 – систему AutoML с открытым исходным кодом на основе генетического программирования, которая оптимизирует ряд предобработчиков признаков и моделей машинного обучения с целью максимизации точности в задаче классификации с учителем. Мы протестировали TPOT на серии из 150 задач классификации с учителем и обнаружили, что она значительно превосходит базовые методы машинного обучения в 21 из них, а в четырех из них точность минимально снижается – и все это без каких-либо знаний о предметной области или участия человека. Следовательно, системы на основе генетического программирования обладают значительными перспективами в области AutoML.

8.1. Введение

Машинное обучение обычно определяют как «область исследований, которая дает компьютерам способность обучаться без прямого программирования» [19]. Несмотря на это распространенное утверждение, опытные специалисты в области машинного обучения знают, что разработка эффективных конвейеров машинного обучения часто является утомительным занятием и обычно требует значительного опыта работы с алгоритмами машинного обучения, экспертных знаний проблемной области и трудоемкого перебора [13]. Так что, вопреки мнению энтузиастов-исследователей, машинное обучение все еще нуждается в значительном объеме прямого программирования.

В ответ на этот вызов было разработано несколько автоматизированных методов машинного обучения [10]. В течение последних нескольких лет мы разрабатывали *инструмент оптимизации конвейеров на основе деревьев* (tree-based pipeline optimization tool, TPOT), который автоматически проектирует и оптимизирует конвейеры машинного обучения для заданной проблемной области [16], не требуя вмешательства человека. Вкратце TPOT оптимизирует конвейеры машинного обучения, используя разновидность метода генетического программирования (GP), хорошо известного метода эволюционных вычислений для автоматического построения компьютерных программ [1]. Ранее мы продемонстрировали, что сочетание GP с оптимизацией по Парето позволяет TPOT автоматически конструировать высокоточные и компактные конвейеры, которые постоянно превосходят базовые системы машинного обучения [13]. В этой главе мы расширили перечень достижений, включив в него 150 задач классификации с учителем, и оценили TPOT в широком спектре прикладных областей – от генетического анализа до классификации изображений и др.

Эта глава является расширенной версией нашей статьи о TPOT, которая была представлена на семинаре по AutoML 2016 ICML [15].

8.2. Базовые принципы TPOT

В следующих разделах мы представим обзор TPOT версии 0.3, включая операторы машинного обучения, используемые в качестве примитивов генетического программирования, древовидные конвейеры, используемые для объединения примитивов в рабочие конвейеры машинного обучения, и GP-алгоритм, используемый для эволюции этих древовидных конвейеров. Далее в этой главе мы приводим описание наборов данных, использованных для оценки последней версии TPOT. Открытый исходный код TPOT является проектом с открытым исходным кодом на GitHub, а основной код на Python можно найти по адресу https://github.com/rhiever/tpot.

8.2.1. Конвейерные операторы машинного обучения

По своей сути TPOT является оберткой для пакета машинного обучения Python scikit-learn [17]. Таким образом, каждый оператор конвейера машинного обучения (т. е. примитив GP) в TPOT соответствует алгоритму машинного обучения, такому как модель классификации с учителем или стандартный нормализатор признаков. Все перечисленные ниже реализации алгоритмов машинного обучения взяты из scikit-learn (кроме XGBoost), и мы отсылаем читателей к документации scikit-learn [17] и [9] для подробного объяснения алгоритмов машинного обучения, используемых в TPOT.

Операторы классификации с учителем: DecisionTree, RandomForest, eXtreme Gradient Boosting Classifier (из XGBoost, [3]), LogisticRegression и KNearestNeighborClassifier. Операторы классификации хранят прогнозы классификатора в виде нового признака, а также классификацию для конвейера.

Операторы предварительной обработки признаков: StandardScaler, RobustScaler, MinMaxScaler, MaxAbsScaler, RandomizedPCA [12], Binarizer и PolynomialFeatures. Операторы предварительной обработки модифицируют набор данных определенным образом и возвращают модифицированный набор данных.

Операторы отбора признаков: VarianceThreshold, SelectKBest, SelectPercentile, SelectFwe и Recursive Feature Elimination (RFE). Операторы отбора уменьшают количество признаков в наборе данных, используя некоторые критерии, и возвращают модифицированный набор данных.

Сюда также входит оператор, объединяющий разрозненные наборы данных, как показано на рис. 8.1, который позволяет объединить несколько модифицированных вариантов набора данных в один набор. Имейте в виду, что TPOT v0.3 не содержит операторы подстановки отсутствующих значе-

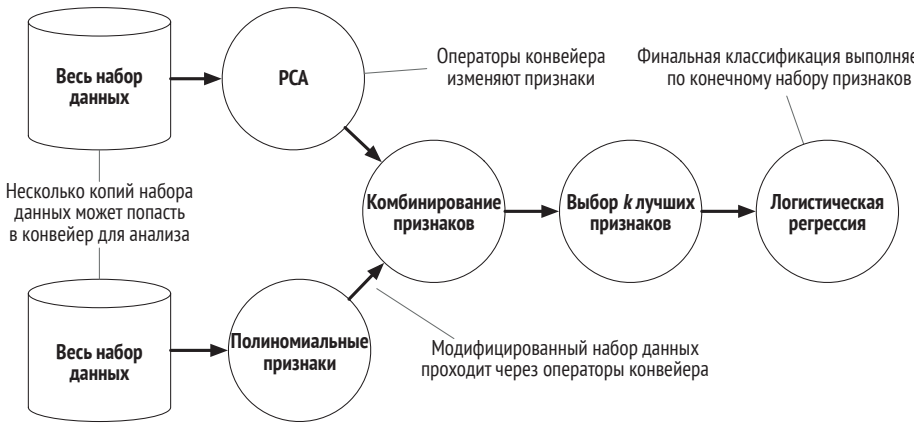


Рис. 8.1 ❖ Пример древовидного конвейера TPOT. Каждый кружок соответствует оператору машинного обучения, а стрелки указывают направление потока данных

ний и, следовательно, не поддерживает наборы данных с отсутствующими данными. Наконец, пользователю доступны целочисленные типы и типы с плавающей запятой для параметризации различных операторов, таких как количество соседей *k* в классификаторе *k*-ближайших соседей.

8.2.2. Построение конвейеров на основе деревьев

Чтобы объединить эти операторы в конвейер машинного обучения, мы рассматриваем их как GP-примитивы и строим из них GP-деревья. На рис. 8.1 показан пример древовидного конвейера, где два набора данных поступают на вход конвейера, последовательно модифицируются каждым оператором, объединяются в один набор данных и, наконец, используются для классификации. За исключением ограничения, что каждый конвейер должен иметь классификатор в качестве конечного оператора, можно построить конвейеры машинного обучения произвольной формы, способные обрабатывать несколько наборов данных. Таким образом, GP-деревья обеспечивают гибкое представление конвейеров машинного обучения.

Мы храним три дополнительные служебные переменные для каждой записи в наборе данных. Переменная `class` указывает истинную метку для каждой записи и используется при оценке точности каждого конвейера. Переменная `guess` указывает последнее предположение конвейера для каждой записи, где в качестве `guess` хранятся предсказания последнего оператора классификации в конвейере. Наконец, переменная `group` указывает, будет ли запись использоваться в качестве части внутреннего обучающего или тестового набора, поскольку древовидные конвейеры обучаются только на обучающих данных и оцениваются на тестовых данных. Отметим, что обучающий набор данных TPOT далее разделяется на внутреннее стратифицированное множество с соотношением между обучающей и проверочной частью 75 и 25 % соответственно.

8.2.3. Оптимизация конвейеров на основе деревьев

Для автоматического создания и оптимизации древовидных конвейеров мы используем алгоритм генетического программирования (GP) [1], реализованный в пакете Python DEAP [7]. Алгоритм TPOT GP следует стандартному процессу GP. Для начала алгоритм GP генерирует 100 случайных древовидных конвейеров и оценивает их сбалансированную точность путем перекрестной проверки на наборе данных. Для каждого поколения алгоритм GP выбирает 20 лучших конвейеров в популяции в соответствии со схемой выбора NSGA-II [4], где конвейеры выбираются так, чтобы одновременно максимизировать точность классификации на наборе данных и минимизировать количество операторов в конвейере. Каждый из 20 лучших отобранных конвейеров про-

изводит пять копий (т. е. потомков) в популяцию следующего поколения, 5 % этих потомков скрещиваются с другими потомками с помощью одноточечного кроссинговера, затем 90 % оставшихся незатронутых потомков случайным образом изменяются с помощью точечной, добавляющей или сокращающей мутации (вероятность каждой 1/3). Каждое поколение алгоритма обновляет фронт Парето решений без доминирования [4], найденных в любой момент времени в ходе работы GP. Алгоритм повторяет этот процесс оценки-выбора-кроссинговера-мутации в течение 100 поколений – добавляя и настраивая операторы конвейера, которые улучшают точность классификации, и обрезая операторы, которые ухудшают точность классификации, – после чего алгоритм выбирает конвейер с наивысшей точностью из фронта Парето в качестве репрезентативного «лучшего» конвейера из прохода.

8.2.4. Эталонные данные

Мы собрали 150 эталонов[1] (бенчмарков) для задачи классификации с учителем из самых разных источников, включая репозиторий машинного обучения UCI [11], большой предшествующий репозиторий бенчмарков из [18] и имитационные наборы данных генетического анализа из [20].

Эти эталонные наборы данных содержат от 60 до 60 000 записей, от единиц до сотен признаков и включают как бинарные, так и многоклассовые задачи классификации с учителем. Мы выбрали наборы данных из широкого спектра областей применения, включая генетический анализ, классификацию изображений, анализ временных рядов и многие другие. Благодаря столь обширному охвату совокупный бенчмарк – получивший название Penn Machine Learning Benchmark (PMLB) [14] – представляет собой полный набор тестов, с помощью которых можно оценить автоматизированные системы машинного обучения.

8.3. Результаты

Для оценки TPOT мы запустили 30 копий процесса на каждом из 150 эталонных наборов, где каждая копия имела 8 ч для завершения 100 поколений оптимизации (т. е. 100 × 100 = 10 000 оценок конвейера). В каждой репликации мы выполняли стратифицированное разбиение набора данных на обучение/тестирование в отношении 75 и 25 % соответственно и использовали отдельные затравки генератора случайных чисел для каждого разбиения и последующего запуска TPOT.

Для того чтобы найти разумно обоснованный исходный уровень для сравнения, мы аналогичным образом оценили 30 копий процесса оптимизации конвейеров методом случайного леса с 500 деревьями на 150 эталонных

[1] Эталонные данные на https://github.com/EpistasisLab/penn-ml-benchmarks.

наборах. Это базовый уровень использования машинного обучения, которым должен владеть начинающий специалист. Мы также запустили 30 копий версии TPOT, которая случайным образом генерирует и оценивает такое же количество конвейеров (10 000), что представляет собой случайный поиск в пространстве конвейеров TPOT. Во всех случаях мы измеряли точность полученных конвейеров или моделей как сбалансированную точность [21], которая корректирует дисбаланс частот классов в наборах данных путем вычисления точности на основе каждого класса, а затем усреднения точности по классам. В оставшейся части этой главы мы будем называть сбалансированную точность просто точностью.

Как показано на рис. 8.2, средняя производительность TPOT и метода случайного леса с 500 деревьями на большинстве наборов данных одинакова. В целом алгоритм TPOT обнаружил конвейеры, которые статистически значимо лучше случайного леса на 21 эталоне, значительно хуже на четырех эталонах и не имеют статистически значимой разницы на 125 эталонах. (Мы определяли статистическую значимость с помощью теста ранговой суммы Вилкоксона, где использовали консервативный порог p-значения с поправкой Бонферрони $< 0.000\overline{333}\left(\frac{0.05}{150}\right)$ для значимости.) На рис. 8.3 показаны распределения точности по 25 эталонам, по которым наблюдались значительные различия, где эталоны отсортированы по разнице в медианной точности между двумя экспериментами.

Примечательно, что большинство улучшений TPOT по этим эталонам довольно значительны, причем по нескольким из них медианная точность улучшилась от 10 до 60 % по сравнению с методом случайного леса. Напротив, в четырех эталонах, где TPOT ухудшил медианную точность, снижение точности составило всего 2–5 %. В некоторых случаях улучшения TPOT были достигнуты за счет обнаружения полезных предобработчиков признаков, которые позволяют моделям лучше классифицировать данные[1], например TPOT обнаружила, что применение предобработчика признаков RandomizedPCA перед моделированием эталона Hill_valley позволяет случайному лесу классифицировать набор данных с почти идеальной точностью. В других случаях улучшения TPOT были достигнуты путем применения другой модели к эталону, например TPOT обнаружил, что классификатор k-ближайших соседей с $k = 10$ соседями способен классифицировать эталон parity5, в то время как алгоритм случайного леса постоянно достигал на том же эталоне 0 % точности.

Когда мы сравнили базовый TPOT с версией TPOT, использующей случайный поиск (TPOT Random на рис. 8.3), мы обнаружили, что случайный поиск обычно обнаруживает конвейеры, которые достигают сравнимой точности с конвейерами, обнаруженными TPOT, за исключением эталонного набора «dis», где TPOT постоянно обнаруживал конвейеры с лучшими показателями. Для 17 из представленных эталонов ни один из случайных поисков не завер-

[1] Полный список: https://gist.github.com/rhiever/578cc9c686ffd873f46bca29406dde1d.

шился в течение 24 ч, что мы обозначили, оставив блочную диаграмму пустой на рис. 8.3. Мы обнаружили, что случайный поиск часто создает излишне сложные конвейеры для эталонных задач, даже когда для классификации достаточно простого конвейера с настроенной моделью. Таким образом, даже если случайный поиск иногда работает так же хорошо, как TPOT, с точки зрения точности, направленный поиск конвейеров, которые достигают высокой точности с минимально возможным количеством операций на конвейере, все равно дает значительные преимущества с точки зрения времени выполнения поиска, сложности и интерпретируемости модели.

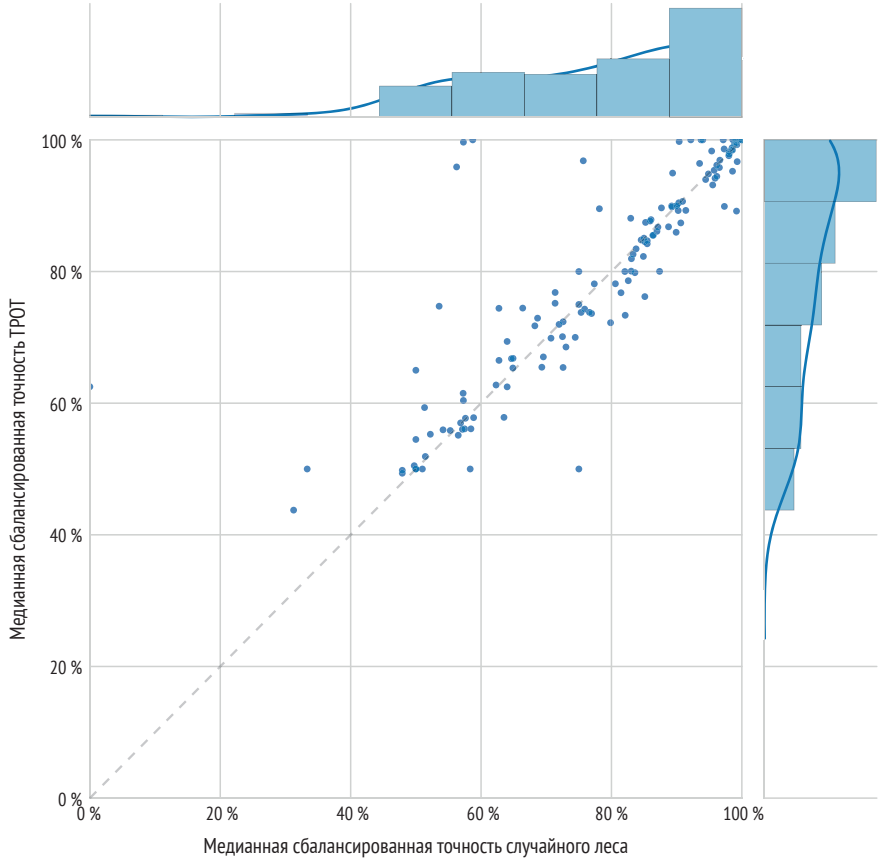


Рис. 8.2 ❖ Диаграмма рассеяния, показывающая медианные сбалансированные точности TPOT и случайного леса с 500 деревьями на 150 эталонных наборах данных. Каждая точка представляет собой точность на одном эталонном наборе данных, а диагональная линия –линию равенства (т. е. когда оба алгоритма достигают одинаковой точности). Точки над линией представляют наборы данных, где TPOT показал лучшие результаты, чем случайный лес, а точки под линией представляют наборы данных, где случайный лес показал лучшие результаты

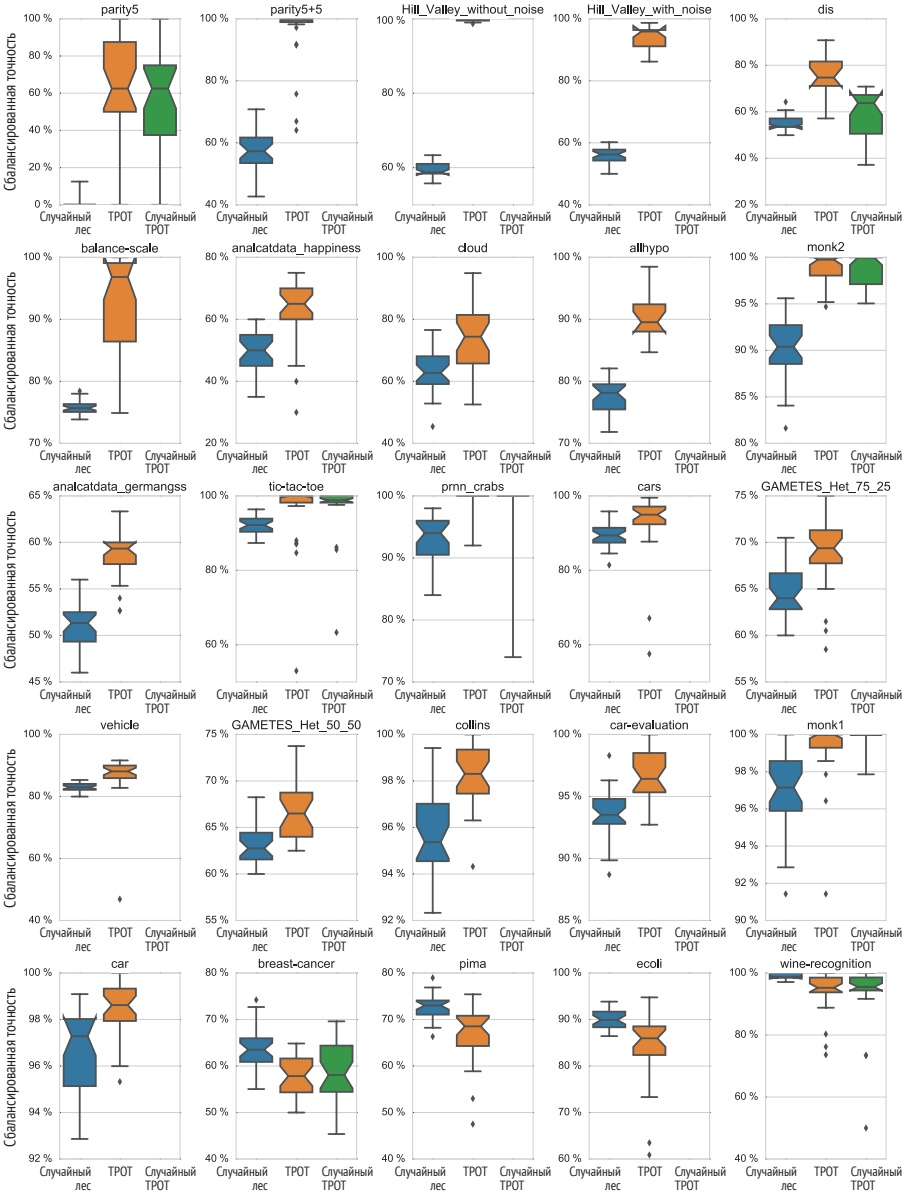


Рис. 8.3 ❖ Диаграммы, показывающие распределение сбалансированной точности для 25 эталонов со значительной разницей в медианной точности между TPOT и случайным лесом с 500 деревьями. Каждая диаграмма представляет 30 копий, внутренняя линия показывает медиану, вырезы представляют 95%-ный доверительный интервал медианы с бутстрэпом, концы блока представляют первый и третий квартили, а точки – выбросы

8.4. Выводы и перспективные направления исследований

Мы протестировали TPOT v0.3 на 150 наборах данных для классификации с учителем и обнаружили, что он обнаруживает конвейеры машинного обучения, которые могут превзойти базовые модели машинного обучения по нескольким показателям. В частности, мы отмечаем, что TPOT обнаружил эти конвейеры без каких-либо знаний о предметной области и без участия человека. Таким образом, TPOT демонстрирует значительные перспективы в области автоматизированного машинного обучения (AutoML), и мы будем продолжать совершенствовать TPOT до тех пор, пока он не будет постоянно находить конвейеры машинного обучения, успешно конкурирующие с моделями, которые разработал человек. Некоторые из этих предстоящих усовершенствований мы рассмотрим ниже.

Во-первых, мы будем изучать методы обеспечения направленной (разумной) инициализации [8] для систем AutoML на основе генетического программирования (GP), таких как TPOT. Например, мы можем использовать методы метаобучения для разумного подбора конфигураций конвейера, которые могут хорошо работать для конкретной решаемой задачи [6]. Вкратце метаобучение использует информацию из предыдущих запусков машинного обучения для предсказания того, насколько хорошо каждая конфигурация конвейера будет работать на конкретном наборе данных. Чтобы обеспечить сопоставимость наборов данных, алгоритмы метаобучения вычисляют метапризнаки наборов данных, такие как размер набора данных, количество признаков и различные аспекты признаков, которые затем используются для сопоставления метапризнаков наборов данных с соответствующими конфигурациями конвейеров, способных хорошо работать на наборах данных с такими метапризнаками. Такой интеллектуальный алгоритм метаобучения, вероятно, улучшит процесс направленной инициализации TPOT.

Кроме того, мы попытаемся охарактеризовать идеальную «форму» конвейера машинного обучения. В auto-sklearn [5] навязывается короткая и фиксированная структура конвейера, состоящая из предобработчика данных, предобработчика признаков и модели. В другой системе AutoML, основанной на GP [22], алгоритму GP позволили проектировать конвейеры произвольной формы и обнаружили, что сложные конвейеры с несколькими предобработчиками и моделями полезны для задач обработки сигналов. Таким образом, для достижения конкурентоспособности на уровне человека может быть жизненно важно позволить системам AutoML проектировать конвейеры произвольной формы.

Наконец, методы оптимизации с использованием генетического программирования обычно критикуют за оптимизацию слишком большой популяции решений, что иногда может быть медленным и расточительным для определенных задач оптимизации. Вместо этого можно обратить предполагаемую слабость GP в силу, создав ансамбль из популяций GP. Бхован и др. [2] исследовали один такой метод ансамблирования популяций со стандартным алгоритмом GP и показали, что он значительно улучшает производитель-

ность и создание ансамблей из популяций конвейеров машинного обучения TPOT является естественным направлением развития.

Проведенные ранее эксперименты показывают, что можно значительно улучшить качество конвейеров, если использовать подход к машинному обучению, не зависящий от модели, и позволить машине автоматически определить, какая последовательность предобработчиков и моделей лучше всего работает для данной предметной области. Следовательно, AutoML может произвести революцию в науке о данных, автоматизировав некоторые из самых утомительных и в то же время самых важных аспектов машинного обучения.

8.5. Литература

1. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming: An Introduction. Morgan Kaufmann, San Meateo, CA, USA (1998).
2. Bhowan, U., Johnston, M., Zhang, M., Yao, X.: Evolving diverse ensembles using genetic programming for classification with unbalanced data. Trans. Evol. Comp 17(3), 368–386 (Jun 2013).
3. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 785– 794. KDD '16, ACM, New York, NY, USA (2016).
4. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6, 182–197 (2002).
5. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 2944–2952. Curran Associates, Inc. (2015).
6. Feurer, M., Springenberg, J.T., Hutter, F.: Initializing bayesian hyperparameter optimization via meta-learning. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence, January 25–30, 2015, Austin, Texas, USA. pp. 1128–1135 (2015).
7. Fortin, F.A., De Rainville, F.M., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: Evolutionary Algorithms Made Easy. Journal of Machine Learning Research 13, 2171–2175 (2012).
8. Greene, C.S., White, B.C., Moore, J.H.: An expert knowledge-guided mutation operator for genome-wide genetic analysis using genetic programming. In: Pattern Recognition in Bioinformatics, pp. 30–40. Springer Berlin Heidelberg (2007).
9. Hastie, T.J., Tibshirani, R.J., Friedman, J.H.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, New York, NY, USA (2009).
10. Hutter, F., Lücke, J., Schmidt-Thieme, L.: Beyond Manual Tuning of Hyperparameters. KI - Künstliche Intelligenz 29, 329–337 (2015).

11. Lichman, M.: UCI machine learning repository (2013), http://archive.ics.uci.edu/ml.
12. Martinsson, P.G., Rokhlin, V., Tygert, M.: A randomized algorithm for the decomposition of matrices. Applied and Computational Harmonic Analysis 30, 47–68 (2011).
13. Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 485–492. GECCO'16, ACM, New York, NY, USA (2016).
14. Olson, R.S., La Cava, W., Orzechowski, P., Urbanowicz, R.J., Moore, J.H.: PMLB: A Large Benchmark Suite for Machine Learning Evaluation and Comparison. arXiv e-print. https://arxiv.org/abs/1703.00512 (2017).
15. Olson, R.S., Moore, J.H.: Tpot: A tree-based pipeline optimization tool for automating machine learning. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) Proceedings of the Workshop on Automatic Machine Learning. Proceedings of Machine Learning Research, vol. 64, pp. 66–74. PMLR, New York, New York, USA (24 Jun 2016), http://proceedings.mlr.press/v64/olson_ tpot_2016.html.
16. Olson, R.S., Urbanowicz, R.J., Andrews, P.C., Lavender, N.A., Kidd, L.C., Moore, J.H.: Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 — April 1, 2016, Proceedings, Part I, chap. Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pp. 123–137. Springer International Publishing (2016).
17. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830 (2011).
18. Reif, M.: A comprehensive dataset for evaluating approaches of various meta-learning tasks. In: First International Conference on Pattern Recognition and Methods (ICPRAM) (2012).
19. Simon, P.: Too big to ignore: the business case for big data. Wiley & SAS Business Series, Wiley, New Delhi (2013).
20. Urbanowicz, R.J., Kiralis, J., Sinnott-Armstrong, N.A., Heberling, T., Fisher, J.M., Moore, J.H.: GAMETES: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures. BioData Mining 5 (2012).
21. Velez, D.R., White, B.C., Motsinger, A.A., Bush, W.S., Ritchie, M.D., Williams, S.M., Moore, J.H.: A balanced accuracy function for epistasis modeling in imbalanced datasets using multifactor dimensionality reduction. Genetic Epidemiology 31(4), 306–315 (2007).
22. Zutty, J., Long, D., Adams, H., Bennett, G., Baxter, C.: Multiple objective vector-based genetic programming using human-derived primitives. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1127–1134. GECCO '15, ACM, New York, NY, USA (2015).

Глава 9

Проект Automatic Statistician

Кристиан Штайнрюкен (✉), **Эмма Смит**, **Дэвид Янц**, **Джеймс Ллойд**, **Зубин Гахрамани**, инженерный факультет Кембриджского университета, Кембридж, Великобритания, e-mail: tcs27@cam.ac.uk

Проект Automatic Statistician направлен на автоматизацию науки о данных, создание прогнозов и человекочитаемых отчетов из необработанных наборов данных при минимальном вмешательстве человека. Наряду с привычными графиками и статистическими данными генерируемые отчеты содержат подборку высокоуровневых сведений о наборе данных, полученных в результате 1) автоматизированного построения моделей для набора данных, 2) сравнения этих моделей и 3) программного компонента, который превращает эти результаты в описания на естественном языке. В этой главе описывается общая архитектура таких систем автоматической статистики, а также обсуждаются некоторые проектные решения и технические проблемы.

9.1. Введение

Машинное обучение (ML) и наука о данных – это тесно связанные области исследований, которые сосредоточены на разработке алгоритмов для автоматического обучения на основе данных. Эти алгоритмы также лежат в основе многих последних достижений в области искусственного интеллекта (ИИ), которые оказали огромное влияние на промышленность, открыв новый золотой век ИИ. Однако многие современные подходы к машинному обучению, науке о данных и ИИ страдают от ряда важных, но взаимосвязанных ограничений.

Во-первых, многие из используемых подходов представляют собой сложные черные ящики, которые трудно интерпретировать, понимать, отлаживать и доверять им. Отсутствие возможности интерпретации затрудняет

развертывание систем ML. Возьмем, к примеру, очевидные юридические, технические и этические последствия использования неинтерпретируемой системы черного ящика, на предсказаниях которой основаны решения, связанные с медицинским диагнозом, приговором по уголовному делу или самодвижущимся автомобилем. Осознание того, что применение методов черного ящика в таких условиях сильно ограничено, привело к серьезным усилиям по разработке *объяснимого ИИ* (explainable AI) и систем, которые обеспечивают интерпретируемость, доверие и прозрачность.

Во-вторых, разработка систем ML фактически представляет собой кустарную индустрию, где эксперты решают проблемы, разрабатывая штучные решения, которые часто отражают последовательность специальных действий, принятых вручную, или предпочтения и предубеждения эксперта. Иронично, что машинное обучение – область, посвященная созданию систем, которые автоматически обучаются на основе данных, – настолько зависит от экспертов-людей и ручной настройки моделей и алгоритмов обучения. Ручной перебор возможных моделей и методов может привести к неоптимальным решениям по любому количеству метрик. Более того, огромный дисбаланс между предложением экспертов и спросом на решения в области науки о данных и ML, вероятно, приводит к тому, что многие возможности для приложений, которые могли бы принести большую пользу обществу, упущены.

Мы создали проект Automatic Statistician с целью автоматизации многих аспектов анализа данных, обнаружения моделей и объяснения их вывода. В некотором смысле наша цель состоит в разработке ИИ для науки о данных – системы, которая может рассуждать о закономерностях в данных и объяснять их пользователю. В идеале при наличии исходных данных такая система должна уметь:

- автоматизировать процесс выбора и преобразования признаков;
- справляться с несовершенством реальных данных, включая недостающие значения, выбросы, различные типы и кодировки переменных;
- осуществлять поиск в большом пространстве моделей, чтобы автоматически обнаруживать хорошую модель, которая отражает все достоверные закономерности в данных;
- находить такую модель, избегая при этом как переобучения, так и недообучения;
- объяснять найденные закономерности пользователю, в идеале – беседуя с ним о данных на естественном человеческом языке;
- делать все вышеперечисленное эффективно и надежно, учитывая ограничения на время вычислений, память, объем данных и другие необходимые ресурсы.

Хотя эта повестка дня, очевидно, очень амбициозна, работа над проектом Automatic Statistician на сегодняшний день позволила достичь прогресса во многих из вышеперечисленных устремлений. В частности, способность обнаруживать правдоподобные модели на основе данных и объяснять эти открытия на разговорном английском языке является одной из отличительных особенностей Automatic Statistician [18]. Такая особенность может быть полезна практически в любой области или сфере деятельности, которая зависит от извлечения знаний из данных.

В отличие от большей части литературы по машинному обучению, которая была сосредоточена на все большем увеличении производительности при решении задач распознавания образов (с использованием таких инструментов, как ядерные методы, случайные леса или глубокое обучение), Automatic Statistician должен строить модели, состоящие из *интерпретируемых* компонентов, и иметь принципиальный способ представления неопределенности в отношении структуры модели при заданных данных. Он также должен иметь способность давать обоснованные ответы не только для больших, но и для малых наборов данных.

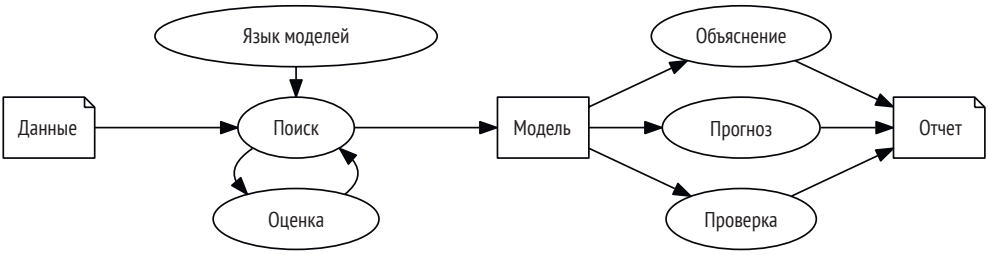


Рис. 9.1 ❖ Упрощенная блок-схема, описывающая работу автоматического статистического анализатора, составляющего отчеты. Модели для данных автоматически строятся (при помощи *расширяемого языка*[1] описания моделей) и оцениваются на этих данных. Оценка производится таким образом, чтобы модели можно было сравнивать друг с другом. Лучшие модели затем проверяются для составления отчета. Каждая модель может быть использована для экстраполяции или предсказания на основе данных, а схема модели может быть превращена в человекочитаемое описание. Для некоторых моделей также возможно генерировать критические замечания и сообщать о том, в каких случаях предположения моделирования не соответствуют данным

9.2. Базовые принципы Automatic Statistician

В основе Automatic Statistician лежит идея о том, что хорошее решение вышеперечисленных проблем может быть получено при работе в рамках *машинного обучения на основе моделей* (model-based machine learning) [2, 9]. В машинном обучении на основе моделей основная идея заключается в том, что вероятностные модели достоверно объясняют закономерности в данных и что вероятностный подход (или *байесовская бритва Оккама*) применим для обнаружения моделей, которые избегают как чрезмерного, так и недостаточного обучения [21]. Байесовские подходы обеспечивают элегантный способ компенсации сложности модели и сложности данных, а вероятностные модели являются композиционными (составляемыми из ограниченного

[1] Расширяемый, или открытый, язык (open-ended language) – язык программирования или описания структур, не имеющий завершенной (закрытой) формализации и дополняемый в контексте решаемой задачи. – *Прим. перев.*

набора компонентов) и интерпретируемыми, как было сказано ранее. Более того, методология, основанная на модели, утверждает, что такие задачи, как предварительная обработка и преобразование данных, являются частью модели и в идеале должны выполняться одновременно [35].

Automatic Statistician содержит следующие ключевые компоненты.

1. **Расширяемый язык моделей** – достаточно выразительный, чтобы отражать явления реального мира и позволять применять методы, используемые экспертами в области статистики и специалистами по обработке данных.
2. **Процедура поиска** для эффективного изучения языка моделей.
3. **Принципиальный метод оценки моделей** с учетом сложности, соответствия данным и использования ресурсов.
4. **Процедура автоматического объяснения моделей**, объясняющая прогнозы модели таким образом, чтобы они были одновременно точными и понятными для неспециалистов.

На рис. 9.1 схематически показано, как эти компоненты могут быть использованы для создания базовой версии приложения Automatic Statistician, пишущего отчеты.

Как будет показано далее в этой главе, можно построить системы на основе Automatic Statistician, которые заменяют процедуру автоматического объяснения на процедуры, производящие другие желаемые результаты, например необработанные прогнозы или решения. В таких случаях компоненты языка, поиска и оценки могут быть соответствующим образом модифицированы для определения приоритета выбранной цели.

9.2.1. Похожие исследования

К значимым достижениям других исследователей и разработчиков можно отнести статистические экспертные системы [11, 37] и обучение уравнениям [26, 27]. Робот-ученый [16] объединяет машинное обучение и научные открытия в области микробиологии в замкнутый цикл с экспериментальной платформой для автоматизации разработки и проведения новых экспериментов. Auto-WEKA [17, 33] и Auto-sklearn [6] – проекты, автоматизирующие обучение классификаторов, активно использующие методы байесовской оптимизации. Усилия по автоматизации применения методов машинного обучения к данным в последнее время набирают обороты и в конечном итоге могут привести к созданию практических систем ИИ для науки о данных.

9.3. Automatic Statistician и данные временных рядов

Системы автоматической статистики могут быть определены для множества различных целей и основываться на различных семействах моделей. Мы нач-

нем с описания одной такой системы, а более широкую таксономию обсудим позже, с примечаниями об общих элементах дизайна и общей архитектуре.

Первый автоматический статистический анализатор для задач одномерной регрессии был описан Ллойдом и др. [18]. Их система, названная Automatic Bayesian Covariance Discovery (ABCD), использует открытый язык моделей гауссовых процессов через композиционную грамматику над ядрами. Гауссов процесс (GP) определяет распределение по функциям, а параметры GP – его среднее и ядро – определяют свойства функций [25]. Существует широкий выбор доступных ядер, которые вызывают распределения функций с определенными свойствами; например, распределения по функциям, которые являются линейными, полиномиальными, периодическими или некоррелированными шумами. Блок-схема этой системы показана на рис. 9.2.

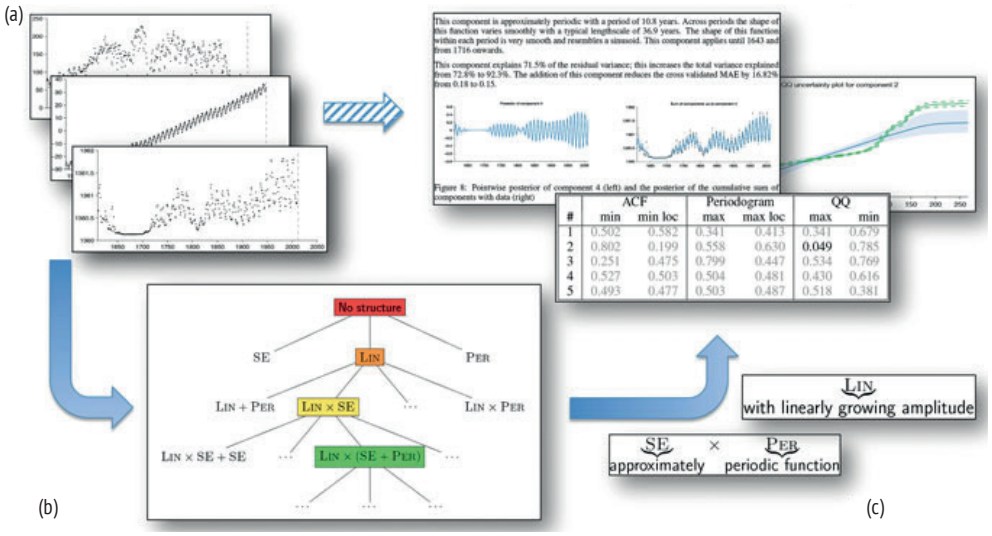


Рис. 9.2 ❖ Блок-схема, описывающая автоматическую статистику для составления отчетов по данным временных рядов. (a) Входом для системы являются данные, в данном случае представленные в виде временных рядов. (b) Система перебирает грамматику моделей, чтобы найти хорошую интерпретацию данных, используя байесовский вывод для оценки моделей. (c) Компоненты найденной модели переводятся в английские фразы. (d) Конечным результатом является отчет с текстом, рисунками и таблицами, в котором подробно описывается, какие выводы были сделаны о данных, включая раздел о проверке и критике модели [8, 20]

9.3.1. Грамматика операций над ядрами

Как упоминалось выше, грамматика над ядрами GP позволяет представить многие интересные свойства функций и дает систематический способ построения распределений для таких функций. Эта грамматика является композиционной: она включает в себя набор фиксированных *базовых ядер* и *опе-*

раторы ядер, которые позволяют составлять новые ядра из существующих. Эта грамматика была тщательно выбрана из соображений интерпретируемости: каждое выражение в грамматике определяет ядро, которое может быть описано простым, но описательным набором слов на человеческом языке.

Базовыми ядрами в грамматике являются: C (constant, постоянное), LIN (linear, линейное), SE (squared exponential, квадратичное экспоненциальное), PER (periodic, периодическое) и WN (white noise, белый шум). Операторами ядра являются: + (сложение), × (умножение) и CP (change point operator, оператор точки изменения), определяемые следующим образом:

$$(k_1 + k_2)(x, x') = k_1(x, x') + k_2(x, x'),$$

$$(k_1 \times k_2)(x, x') = k_1(x, x') \times k_2(x, x'),$$

$$\mathrm{CP}\ (k_1, k_2)\ (x, x') = k_1(x, x')\sigma(x)\sigma(x') + k_2(x, x')\ (1 - \sigma(x))\ (1 - \sigma(x')),$$

где $\sigma\sigma(x) = \frac{1}{2}\left(1 + \tanh\frac{l-x}{s}\right)$ – сигмоидальная функция, а l и s – параметры точки изменения. Базовые ядра могут быть произвольно скомбинированы с помощью вышеуказанных операторов для получения новых ядер.

Бесконечное пространство ядер, определяемое этой грамматикой, позволяет автоматизировать поиск, оценку и описание большого класса интересных распределений. Этот тип грамматики был впервые описан в [10] для задач факторизации матриц, а затем усовершенствован в [5] и [18] для моделей GP.

9.3.2. Процедура поиска и оценки

ABCD выполняет жадный поиск в пространстве моделей (как определено грамматикой). Параметры ядра каждой предложенной модели оптимизируются методом сопряженного градиента. Затем модель с оптимизированными параметрами оценивается с помощью байесовского информационного критерия [29]:

$$\mathrm{BIC}(M) = -2\log p(D|M) + |M|\log N, \tag{9.1}$$

где M – оптимизированная модель, $p(D|M)$ – маргинальное правдоподобие модели, интегрирующей латентную функцию GP, $|M|$ – число параметров ядра в M, а N – размер набора данных. Байесовский информационный критерий обеспечивает компромисс между сложностью модели и соответствием данным и аппроксимирует полное маргинальное правдоподобие (которое интегрирует латентные функции и гиперпараметры).

Модель, получившая наилучшую оценку в каждом раунде, используется для построения новых предложенных моделей по одному из двух путей: 1) расширения ядра с помощью правил производства из грамматики, таких как введение суммы, произведения или точки изменения; или 2) мутации ядра путем замены базового ядра на другое. Новый набор предложенных ядер затем оценивается в следующем раунде. В соответствии с приведенными выше

правилами возможно, что одно выражение ядра будет предложено несколько раз, но хорошо реализованная система будет вести учет и оценивать каждое выражение только один раз. Процедура поиска и оценки останавливается, либо когда результат всех вновь предложенных моделей хуже, чем у лучшей модели предыдущего раунда, либо когда превышена заданная глубина поиска.

Эта процедура жадного поиска не гарантирует нахождения лучшей модели в языке для любого набора данных: лучшая модель может скрываться в одном из поддеревьев, которые не были раскрыты. Нахождение глобально лучшей модели обычно не является обязательным, главное, чтобы за разумное время была найдена хорошая интерпретируемая модель. Существуют и другие способы проведения поиска и оценки моделей. Например, Малкомс и др. [22] описывают процедуру поиска ядра, основанную на байесовской оптимизации. Янц и др. [14] реализовали метод поиска ядра с использованием сигма-точечного фильтра и гамильтонова метода Монте-Карло.

9.3.3. Генерация описаний на естественном языке

После завершения процедуры поиска создается список выражений ядра и их оценки на наборе данных. Выражение с наилучшей оценкой затем используется для создания описания на естественном языке. Чтобы преобразовать ядро в описание на естественном языке, ядро сначала преобразуется в каноническую форму с помощью следующего процесса.

1. Вложенные суммы и произведения упрощаются до суммы произведений.
2. Некоторые произведения ядер могут быть упрощены до базовых ядер с измененными параметрами, например: $\mathrm{SE} \times \mathrm{SE} \rightarrow \mathrm{SE}^{*}$, $\mathrm{C} \times k \rightarrow k^{*}$ для любого k и $\mathrm{WN} \times k \rightarrow \mathrm{WN}^{*}$ для любого $k \in \{\mathrm{C}, \mathrm{SE}, \mathrm{WN}, \mathrm{PER}\}$.

После применения этих правил выражение ядра представляет собой сумму членов произведения, где каждый член произведения имеет следующий канонический вид:

$$k \times \prod_{m} \mathrm{LIN}^{(m)} \times \prod_{n} \boldsymbol{\sigma}^{(n)}, \tag{9.4}$$

где $\sigma(x, x') = \sigma(x)\sigma(x')$ является произведением двух сигмоидных функций и имеет одну из следующих форм: 1, WN, C, SE, $\prod_j \mathrm{PER}^{(j)}$ или $\mathrm{SE} \times \prod_j \mathrm{PER}^{(j)}$. Запись $\prod_j k^{(j)}$ означает произведение ядер, каждое со своими параметрами.

В этой канонической форме ядро представляет собой сумму произведений, и сначала описывается количество членов в сумме: «Алгоритм поиска структуры выявил N аддитивных компонентов в данных». За этим предложением следует описание каждого аддитивного компонента (т. е. каждого произведения в сумме) в соответствии со следующим алгоритмом.

1. Выбираем *одно* из ядер в произведении в качестве дескриптора существительного. Эвристика, рекомендованная Ллойдом и др. [18], заключается в том, чтобы выбирать в соответствии со следующим предпочтением: $\mathrm{PER} > \{\mathrm{C}, \mathrm{SE}, \mathrm{WN}\} > \prod_j \mathrm{LIN}^{(j)} > \prod_j \boldsymbol{\sigma}^{(j)}$, где PER – наиболее предпочтительный вариант.

2. Преобразовываем выбранный тип ядра в строку, используя следующую таблицу:

WN	«некоррелированный шум»	SE	«гладкая функция»
Per	«периодическая функция»	LIN	«линейная функция»
C	«константа»	$\prod_j \text{Lin}^{(j)}$	«полиномиальная функция»

3. Остальные ядра в произведении конвертируем в выражения *постмодификаторов* (post-modifier), которые добавляются к дескриптору существительного. Постмодификаторы конвертируются с помощью следующей таблицы:

SE	«форма которого изменяется плавно»
Per	«модулированный периодической функцией»
Lin	«с линейно изменяющейся амплитудой»
$\prod_j \text{Lin}^{(j)}$	«с полиномиально изменяющейся амплитудой»
$\prod_j \boldsymbol{\sigma}^{(j)}$	«который применяется с/до [точка изменения]»

4. Возможны дальнейшие уточнения описательной части, включая сведения о параметрах ядра или дополнительную информацию, вычисленную на основе данных. Некоторые из этих уточнений описаны в [18].

Более подробно о переводе выражений ядра на естественный язык можно прочитать в [18] и [19]. Пример выдержки из сгенерированного отчета показан на рис. 9.3.

Этот компонент является приблизительно периодическим с периодом 10.8 г. Между периодами форма этой функции изменяется плавно с типичным масштабом длины в 36.9 г. Форма этой функции в пределах каждого периода очень плавная и напоминает синусоиду. Этот компонент применяется до 1643 г. и с 1716 г.

Этот компонент объясняет 71.5 % остаточной дисперсии; он увеличивает общую объясненную дисперсию с 72.8 до 92.3 %. Добавление этого компонента уменьшает MAE перекрестной проверки на 16.82 %, с 0.18 до 0.15.

(Здесь для большей наглядности дан дословный перевод оригинального описания без редактирования стиля. – *Прим. перев.*)

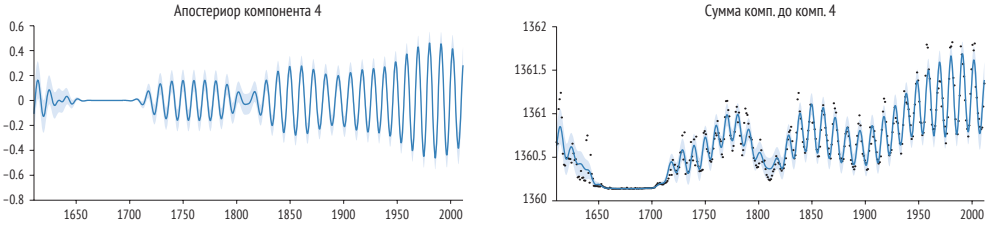


Рис. 9.3 ❖ Выдержка из автоматически созданного отчета, описывающего компоненты модели, обнаруженные ABCD. В этой части отчета выделен и описан приблизительно 11-летний цикл солнечных пятен, также отмечено его исчезновение в XVI веке, в период времени, известный как минимум Маундера. (Этот рисунок воспроизведен из [18])

9.3.4. Сравнение с людьми

Интересным вопросом является то, насколько прогнозы, сделанные автоматической статистической системой (например, алгоритмом ABCD), похожи на человеческие и как они соотносятся с прогнозами, сделанными другими методами, которые также основаны на гауссовых процессах. Чтобы ответить на этот вопрос, Шульц и др. [28] поставили перед участниками задачу экстраполяции из заданного набора данных и выбора предпочтительной экстраполяции из заданного набора. Результаты оказались обнадеживающими для метода поиска составного ядра по двум параметрам. Во-первых, участники эксперимента предпочли экстраполяции, сделанные ABCD, экстраполяциям, сделанным с помощью спектральных ядер [36] и с помощью простого ядра RBF (радиальная базисная функция). Во-вторых, когда участников попросили экстраполировать данные самостоятельно, их прогнозы были наиболее схожи с прогнозами, полученными с помощью процедуры композитного поиска ABCD.

Одной из целей разработки системы Automatic Statistician для составления отчетов является способность объяснять свои выводы в терминах, понятных человеку. Описанная ранее система ограничивает себя пространством моделей, которые могут быть объяснены на человеческом языке с помощью простых терминов, даже если это достигнуто ценой точности прогнозирования. В целом измерить интерпретируемость систем машинного обучения не так просто; одна из возможных схем предложена Доши-Велесом и Кимом [4]. Заметим попутно, что не все системы машинного обучения требуют такой функциональности. Например, когда результаты работы системы оказывают незначительное влияние на общество, особенно в плане социальных норм и взаимодействий, допустимо оптимизировать не интерпретируемость, а производительность или точность (например, распознавание почтовых индексов для автоматической сортировки почты).

9.4. Другие системы автоматической статистики

Способность генерировать человекочитаемые отчеты, возможно, является одной из отличительных особенностей систем автоматической статистики. Но, как упоминалось ранее, программное обеспечение такого рода может служить и другим целям. Например, пользователей могут интересовать необработанные прогнозы на основе данных (с пояснениями или без них), или они могут захотеть, чтобы система принимала решения на основе данных непосредственно от их имени.

Кроме того, можно построить системы автоматической статистики для семейств моделей, отличных от гауссовых процессов или грамматик. Например, мы создали системы Automatic Statistician для регрессии [5, 18],

классификации [12, 23], одномерных и многомерных данных; системы, основанные на различных классах моделей, а также системы с интеллектуальным управлением ресурсами и без него. В этом разделе обсуждаются некоторые элементы структуры, которые являются общими для многих систем Automatic Statistician.

9.4.1. Основные компоненты

Одной из ключевых задач, которые должен выполнять Automatic Statistician, является выбор, оценка и сравнение моделей. Эти типы задач могут выполняться параллельно, но они имеют взаимозависимость. Например, оценка одного набора моделей может повлиять на выбор следующего набора моделей.

Как правило, компонент **стратегии выбора** в нашей системе отвечает за выбор моделей для оценки: он может выбирать из фиксированного или открытого семейства моделей или генерировать и уточнять модели на основе оценки и сравнения ранее выбранных моделей. Иногда типы переменных в наборе данных (выведенные из данных или аннотированные пользователем) влияют на то, какие модели могут быть выбраны стратегией выбора. Например, может возникнуть желание различать непрерывные и дискретные данные, а также использовать различные методы для категориальных и порядковых данных.

Задача **оценки модели** обучает заданную модель на части предоставленного пользователем набора данных, а затем выдает оценку, тестируя модель на отложенных данных. Некоторые модели не требуют отдельного этапа обучения и могут напрямую выдавать логарифмическое правдоподобие для всего набора данных. Оценка модели, вероятно, является одной из самых важных задач для распараллеливания: в любой момент времени несколько выбранных моделей может оцениваться одновременно, на нескольких процессорах или даже на нескольких компьютерах.

Компонент **куратора отчета** – это часть программного обеспечения, которая решает, какие результаты включить в окончательный отчет. Например, он может включать разделы, описывающие наиболее подходящие модели, а также экстраполяцию, графики или таблицы данных. В зависимости от результатов оценки куратор отчета может решить включить дополнительные материалы, например разделы о фальсификации данных / критике моделей, рекомендации или резюме. В некоторых системах результат может быть не отчетом, а чем-то другим, например необработанными прогнозами, настройками параметров или исходным кодом модели.

В интерактивных системах **этап загрузки данных** предоставляет мгновенное резюме о загруженном наборе данных и позволяет пользователю скорректировать любые предположения о формате данных. Пользователь может сделать аннотации типов, удалить столбцы из набора данных, выбрать выходную переменную (например, для классификации) и указать виды анализов, которые должны быть выполнены.

9.4.2. Проблемы и задачи

9.4.2.1. Взаимодействие с пользователем

Хотя целью проекта Automatic Statistician является автоматизация *всех* аспектов обработки данных (от низкоуровневых задач, таких как форматирование и очистка данных, до высокоуровневых задач, таких как построение, оценка и критика модели), полезно также предоставить пользователям возможность взаимодействовать с системой и влиять на выбор, который она делает. Например, пользователи могут захотеть отдельно указать, какие части или какие аспекты данных их интересуют, а какие части можно игнорировать. Некоторые пользователи могут захотеть выбрать семейство моделей, которые система будет рассматривать на этапе построения или оценки модели. Наконец, система может вступить в диалог с пользователем, чтобы уточнить исследование или объяснить то, что она нашла в данных. Такая интерактивность должна поддерживаться базовой системой.

9.4.2.2. Отсутствующие и беспорядочные данные

Распространенной проблемой реальных наборов данных является то, что в них могут быть неполные или поврежденные записи, несоответствия единиц измерения или форматирования или другие виды дефектов. Подобные дефекты могут потребовать предварительной обработки данных, и, хотя многие решения могут быть приняты автоматически, иногда полезно взаимодействовать с пользователем. Хорошие модели могут напрямую работать с отсутствующими данными, и если на этапе загрузки данных отсутствующие данные идентифицированы правильно, то это не должно вызывать проблем. Но некоторые модели не могут обрабатывать отсутствующие данные в неизменном виде. В таких случаях может быть полезно выполнить предварительную подстановку данных, чтобы передать таким моделям версию набора данных с заполненными недостающими значениями. Задачу подстановки обычно решает отдельная модель, которая обучена на данных. Примерами таких методов являются, например, MissForest [31], MissPaLasso [30], mice [3], KNNimpute [34] и байесовские подходы [1, 7].

9.4.2.3. Распределение ресурсов

Еще одним важным аспектом работы автоматического статистического анализатора является использование ресурсов. Например, у пользователя может быть только ограниченное количество ядер процессора, или он может быть заинтересован в получении наилучшего отчета за определенное время, например до установленного срока. Чтобы сделать правильный выбор и оценку модели, интеллектуальная система может учитывать такие ограничения ресурсов. Возможность сделать это повлияет на общее удобство использования системы.

Даже если нет прямых ограничений на время вычислений, количество ядер процессора или использование памяти, интеллектуальная система мо-

жет выиграть от выделения ресурсов моделям, оценка которых наиболее перспективна для выбранного результата. Такая функциональность может быть реализована для моделей, которые поддерживают определенную форму постепенной оценки, например, путем постепенного обучения на все более крупных подмножествах набора данных. Одна из наших систем использовала для этой цели вариант байесовской оптимизации Freeze-thaw [32].

9.5. Заключение

Наше общество вступило в эпоху изобилия данных. Анализ и изучение данных необходимы для использования преимуществ этого растущего ресурса. К сожалению, в настоящее время рост объема данных опережает наши возможности по их анализу, особенно потому, что эта задача по-прежнему в значительной степени возлагается на экспертов-людей. Однако многие аспекты машинного обучения и анализа данных могут быть автоматизированы, и один из ключевых принципов в достижении этой цели – «применить машинное обучение к самому себе».

Проект Automatic Statistician призван автоматизировать науку о данных, взяв на себя все аспекты анализа данных, от предварительной обработки данных, моделирования и оценки до получения полезных и прозрачных результатов. Все эти задачи должны выполняться таким образом, чтобы не требовать от пользователя особых знаний, минимизировать его участие, а также разумно и контролируемо использовать вычислительные ресурсы.

Хотя эта цель весьма амбициозна и многое еще предстоит сделать, на пути к созданию таких автоматизированных систем достигнут обнадеживающий прогресс. Было создано несколько систем автоматической статистики, каждая из которых немного отличается по назначению и лежащей в основе технологии, но все они имеют общий замысел и во многом сходную философию проектирования. Мы надеемся, что создание таких инструментов даст возможность более широкому кругу людей получить представление о данных и поможет обществу более эффективно использовать наши информационные ресурсы.

Благодарности. Авторы выражают благодарность Тамиму Аделю Хешаму, Ларсу Коттхоффу и Франку Хуттеру за полезные отзывы.

9.6. Литература

1. Allingham, J.U.: Unsupervised automatic dataset repair. Master's thesis in advanced computer science, Computer Laboratory, University of Cambridge (2018).
2. Bishop, C.M.: Pattern recognition and machine learning. Information science and statistics, Springer (2006).

3. van Buuren, S., Groothuis-Oudshoorn, K.: mice: Multivariate imputation by chained equations in R. Journal of Statistical Software 45(3) (2011).
4. Doshi-Velez, F., Kim, B.: Towards a rigorous science of interpretable machine learning (Mar 2017), http://arxiv.org/abs/1702.08608.
5. Duvenaud, D., Lloyd, J.R., Grosse, R., Tenenbaum, J.B., Ghahramani, Z.: Structure discovery in nonparametric regression through compositional kernel search. In: Proceedings of the 30th International Conference on Machine Learning (Jun 2013).
6. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 2962–2970. Curran Associates, Inc. (2015).
7. Garriga Alonso, A.: Probability density imputation of missing data with Gaussian Mixture Models. MSc thesis, University of Oxford (2017).
8. Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A., Rubin, D.B.: Bayesian Data Analysis, Third Edition. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis (2013).
9. Ghahramani, Z.: Probabilistic machine learning and artificial intelligence. Nature 521, 452– 459 (2015).
10. Grosse, R.B., Salakhutdinov, R., Tenenbaum, J.B.: Exploiting compositionality to explore a large space of model structures. In: Uncertainty in Artificial Intelligence (2012).
11. Hand, D.J.: Patterns in statistical strategy. In: Gale, W.A. (ed.) Artificial intelligence and statistics (1986).
12. He, Q.: The Automatic Statistician for Classification. Master's thesis, Department of Engineering, University of Cambridge (May 2016).
13. Hwang, Y., Tong, A., Choi, J.: Automatic construction of nonparametric relational regression models for multiple time series. In: Balcan, M.F., Weinberger, K.Q. (eds.) ICML 2016: Proceedings of the 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 3030–3039. PLMR (2016).
14. Janz, D., Paige, B., Rainforth, T., van de Meent, J.W., Wood, F.: Probabilistic structure discovery in time series data (2016), https://arxiv.org/abs/1611.06863.
15. Kim, H., Teh, Y.W.: Scaling up the Automatic Statistician: Scalable structure discovery using Gaussian processes. In: Storkey, A., Perez-Cruz, F. (eds.) Proceedings of the 21st International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 84, pp. 575–584. PLMR (2018).
16. King, R.D., Whelan, K.E., Jones, F.M., Reiser, P.G.K., Bryant, C.H., Muggleton, S.H., Kell, D.B., Oliver, S.G.: Functional genomic hypothesis generation and experimentation by a robot scientist. Nature 427(6971), 247–252 (2004).
17. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. Journal of Machine Learning Research 18(25), 1–5 (2017).
18. Lloyd, J.R., Duvenaud, D., Grosse, R., Tenenbaum, J.B., Ghahramani, Z.: Automatic construction and natural-language description of nonparametric re-

gression models. In: Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-14) (2014).
19. Lloyd, J.R.: Representation, learning, description and criticism of probabilistic models with applications to networks, functions and relational data. Ph.D. thesis, Department of Engineering, University of Cambridge (Dec 2014).
20. Lloyd, J.R., Ghahramani, Z.: Statistical model criticism using kernel two sample tests. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28. pp. 829–837. Curran Associates, Inc. (2015).
21. MacKay, D.J.C.: Bayesian interpolation. Neural Computation 4(3), 415–447 (1992), see [24] for additional discussion and illustration.
22. Malkomes, G., Schaff, C., Garnett, R.: Bayesian optimization for automated model selection. In: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29, pp. 2900–2908. Curran Associates, Inc. (2016).
23. Mrkšić, N.: Kernel Structure Discovery for Gaussian Process Classification. Master's thesis, Computer Laboratory, University of Cambridge (Jun 2014).
24. Murray, I., Ghahramani, Z.: A note on the evidence and Bayesian Occam's razor. Tech. Rep. GCNU-TR 2005-003, Gatsby Computational Neuroscience Unit, University College London (2005).
25. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press (2006), http://www.gaussianprocess.org/gpml/.
26. Schmidt, M., Lipson, H.: Distilling free-form natural laws from experimental data. Science 324(5923), 81–85 (2009).
27. Schmidt, M., Lipson, H.: Symbolic regression of implicit equations. In: Riolo, R., O'Reilly, U.M., McConaghy, T. (eds.) Genetic Programming Theory and Practice VII, pp. 73–85. Springer, Boston, MA (2010).
28. Schulz, E., Tenenbaum, J., Duvenaud, D.K., Speekenbrink, M., Gershman, S.J.: Probing the compositionality of intuitive functions. In: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29, pp. 3729–3737. Curran Associates, Inc. (2016).
29. Schwarz, G.: Estimating the dimension of a model. The Annals of Statistics 6(2), 461–464 (1978).
30. Städler, N., Stekhoven, D.J., Bühlmann, P.: Pattern alternating maximization algorithm for missing data in high-dimensional problems. Journal of Machine Learning Research 15, 1903– 1928 (Jun 2014).
31. Stekhoven, D.J., Bühlmann, P.: MissForest – non-parametric missing value imputation for mixed-type data. Bioinformatics 28(1), 112–118 (Nov 2011).
32. Swersky, K., Snoek, J., Adams, R.P.: Freeze-thaw Bayesian optimization (Jun 2014), http://arxiv.org/abs/1406.3896.
33. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 847–855. KDD '13, ACM, New York, NY, USA (2013).

34. Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., Altman, R.B.: Missing value estimation methods for DNA microarrays. Bioinformatics pp. 520–525 (Jun 2001).
35. Valera, I., Ghahramani, Z.: Automatic discovery of the statistical types of variables in a dataset. In: Precup, D., Teh, Y.W. (eds.) ICML 2017: Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 3521–3529. PLMR (2017).
36. Wilson, A.G., Adams, R.P.: Gaussian process kernels for pattern discovery and extrapolation. In: Dasgupta, S., McAllester, D. (eds.) ICML 2013: Proceedings of the 30th International Conference on Machine Learning. JLMR Proceedings, vol. 28, pp. 1067–1075. JLMR.org (Jun 2013).
37. Wolstenholme, D.E., O'Brien, C.M., Nelder, J.A.: GLIMPSE: a knowledge-based front end for statistical analysis. Knowledge-Based Systems 1(3), 173–178 (1988).

Часть III

ПРОБЛЕМЫ AutoML

Глава 10

О чем говорят результаты конкурсов AutoML Challenge?

Изабель Гийон (✉), Университет Париж-Сюд, Орсэ, Франция, INRIA, Университет Париж-Сакле, Париж, Франция, ChaLearn и ClopiNet, Беркли, Калифорния, США, e-mail: guyon@chalearn.org

Лишенг Сун-Хосойя, **Чжэнъин Лю**, Лаборатория исследований в области информатики, Университет Париж-Сюд, Орсэ, Франция, Университет Париж-Сакле, Париж, Франция

Марк Буле, Группа машинного обучения, Orange Labs, Ланнион, Франция

Уго Жаир Эскаланте, Департамент вычислительных наук, INAOE и ChaLearn, Тонанцинтла, Мексика

Серхио Эскалера, Центр компьютерного зрения, Университет Барселоны, Барселона, Испания

Дамир Яджетич, IN2, Загреб, Хорватия

Бисакха Рэй, Медицинский центр Лангоне, Нью-Йоркский университет, Нью-Йорк, штат Нью-Йорк, США

Мехрин Саид, факультет компьютерных наук, Национальный университет компьютерных и новейших наук, Исламабад, Пакистан

Мишель Себаг, Лаборатория исследований в области информатики, CNRS, Париж, Франция Университет Париж-Сакле, Париж, Франция

Александр Статников, SoFi San Francisco, Калифорния, США

Вей-Вей Ту, 4Paradigm, Пекин, Китай

Эвелин Виегас, Microsoft Research, Редмонд, штат Вашингтон, США

(Авторы расположены в алфавитном порядке фамилий, написанных на английском языке, за исключением первого автора, который выполнил большую часть работы, и второго автора, который провел большую часть численного анализа и построил графики.)

Конкурс ChaLearn AutoML Challenge (NIPS 2015 – ICML 2016) состоял из шести раундов соревнования по машинному обучению с нарастающей сложностью при ограниченных вычислительных ресурсах. За ним последовал одноразундовый конкурс AutoML (PAKDD 2018). Задачи конкурсов по AutoML отличаются от традиционных задач по подбору моделей / подбору гиперпараметров на обычных конкурсах машинного обучения. Участники ставят перед собой цель разработать полностью автоматизированные и вычислительно эффективные системы, способные выполнять обучение и тестирование без вмешательства человека, причем на основе полностью открытого кода. В этой главе мы анализируем результаты проведенных конкурсов и приводим подробности о наборах данных, которые не были раскрыты участникам на момент соревнования. Мы систематически протестировали решения победителей на всех наборах данных всех раундов и сравнили с каноническими алгоритмами машинного обучения, доступными в scikit-learn. Все материалы, обсуждаемые в этой главе (данные и код), были выложены в открытый доступ на сайте http://AutoML.chalearn.org/.

10.1. Введение

Еще относительно недавно машинное обучение (ML) было дисциплиной, малоизвестной широкой публике. Для ученых, занимающихся ML, это был «рынок продавца»: они разрабатывали множество алгоритмов и постоянно искали новые интересные наборы данных. Крупные интернет-корпорации, накапливающие огромные объемы данных, такие как Google, Facebook, Microsoft и Amazon, популяризировали использование ML, а соревнования в области науки о данных привлекли к нему новое поколение молодых ученых. В настоящее время правительства разных стран и корпорации продолжают находить новые области применения ML, а с ростом доступности открытых данных мы перешли к «рынку покупателя»: складывается впечатление, что всем нужна самообучающаяся машина. Однако, к сожалению, самообучающиеся машины еще не полностью автоматизированы: все еще трудно понять, какое программное обеспечение применимо к той или иной задаче, как лучше всего отформатировать данные для программы и как правильно выбрать (гипер-)параметры. Цель серии конкурсов ChaLearn AutoML – направить энергию сообщества ML на то, чтобы постепенно уменьшить необходимость вмешательства человека при применении ML для решения широкого спектра практических задач.

Полная автоматизация – это безграничная задача, поскольку всегда могут возникнуть новые условия, с которыми раньше никто не сталкивался. Наши первые конкурсные задачи AutoML1 были ограничены следующими условиями:

- **обучением с учителем** (классификация и регрессия);
- **векторными представлениями признаков**;
- **однородными наборами данных** (одинаковое распределение в обучающем, проверочном и тестовом наборе);

- **наборами данных среднего размера** менее 200 Мб;
- **ограниченными компьютерными ресурсами**: время выполнения менее 20 мин для каждого набора данных на 8-ядерной машине x86_64 с 56 Гб оперативной памяти.

Мы исключили обучение без учителя, активное обучение, трансферное обучение и проблемы обнаружения причинно-следственных связей, которые очень важны для нас и рассматривались в прошлых задачах ChaLearn [31], но которые требуют каждый раз различных условий оценки, что делает сравнение результатов очень трудным. Мы не исключали обработку видео, изображений, текста и (в более общем случае) временных рядов, и выбранные наборы данных действительно содержат несколько экземпляров таких модальностей. Однако сначала они были предварительно обработаны в модели извлечения признаков, что ослабило акцент на обучении признакам. Тем не менее обучение на данных, предварительно обработанных в плане представлений, основанных на признаках, уже охватывает много областей, и полностью автоматизированный метод, решающий эту ограниченную проблему, уже был бы большим достижением в этой области.

В рамках этой ограниченной задачи мы ввели целый ряд затруднений, с которыми столкнулись участники конкурсов. Это:

- **различные распределения данных**: внутренняя/геометрическая сложность набора данных;
- **различные задачи**: регрессия, бинарная классификация, многоклассовая классификация, многозначная классификация;
- **различные метрики оценки**: AUC, BAC, MSE, F1 и т. д. (см. раздел 10.4.2);
- **баланс классов**: сбалансированные и несбалансированные соотношения классов;
- **разреженность**: полные и разреженные матрицы;
- **пропущенные значения**: наличие или отсутствие пропущенных значений;
- **категориальные переменные**: наличие или отсутствие категориальных переменных;
- **нерелевантные переменные**: наличие или отсутствие дополнительных нерелевантных переменных (дистракторов);
- **количество P_{tr} обучающих примеров**: малое или большое количество обучающих примеров;
- **количество N переменных/признаков**: малое или большое количество переменных;
- **отношение P_{tr}/N матрицы обучающих данных**: $P_{tr} \gg N$, $P_{tr} = N$ или $P_{tr} \ll N$.

В этой ситуации участникам пришлось столкнуться с множеством вариантов выбора модели и/или гиперпараметров. Некоторые другие, не менее важные аспекты автоматизации машинного обучения не были рассмотрены в данной задаче и оставлены для будущих исследований. К ним относятся: получение и форматирование данных, предварительная обработка и обучение признакам/представлениям, обнаружение и обработка перекошенных/смещенных данных, неоднородных, дрейфующих, мультимодальных или

мультивидовых данных (в зависимости от трансферного обучения), подбор алгоритмов к задачам (которые могут включать обучение с учителем и без учителя, обучение с подкреплением или другие подходы), получение новых данных (активное обучение, обучение на вопросах-ответах, обучение с подкреплением, причинные эксперименты), управление большими объемами данных, включая создание соответствующих по размеру и стратифицированных обучающих, проверочных и тестовых наборов, выбор алгоритмов, удовлетворяющих произвольным ограничениям ресурсов во время обучения и выполнения, возможность создания и повторного использования рабочих процессов, а также создание содержательных отчетов.

Эта серия конкурсов отсчитывает свою историю с «игры по выбору модели»[1] на NIPS 2006 [37], где участникам был предоставлен инструментарий машинного обучения, основанный на наборе инструментов CLOP [1], построенном на базе пакета Spider [69]. Инструментарий предоставлял участникам гибкий способ построения моделей путем комбинирования модулей предварительной обработки, отбора признаков, классификации и последующей обработки, а также позволял создавать ансамбли классификаторов. Целью игры было построение лучшей гипермодели: основное внимание уделялось выбору модели, а не разработке новых алгоритмов. Все задачи были задачами бинарной классификации на основе признаков. Организаторы предоставили пять наборов данных, а участники должны были предложить схему своей модели. Игра по выбору модели подтвердила эффективность перекрестной проверки (победитель изобрел новый вариант, названный перекрестной индексацией) и подчеркнула необходимость уделять больше внимания эффективности поиска с применением новых методов поиска, таких как оптимизация роя частиц.

В новой серии конкурсов AutoML 2015/2016 мы ввели понятие «задача»: каждый набор данных снабжался определенной метрикой оценки, которую нужно было оптимизировать, и бюджетом времени. Изначально мы планировали варьировать бюджет времени от набора к набору произвольным образом. В итоге по практическим соображениям мы установили его на уровне 20 мин (за исключением раунда 0, где бюджет времени варьировался от 100 до 300 с). Однако, поскольку наборы данных различались по размеру, это заставляло участников продуманно распределять отведенное им время. Другие элементы новизны включали свободу представления любого исполняемого файла Linux. Это стало возможным благодаря использованию автоматического выполнения на платформе Codalab с открытым исходным кодом[2]. Чтобы помочь участникам, мы предоставили стартовый набор на языке Python, основанный на библиотеке scikit-learn [55][3]. Это побудило многих из них написать обертку вокруг scikit-learn. Такова была, например, стратегия победителя конкурса Auto-sklearn [25–28][4]. После конкурса AutoML мы организовали дополнительную игру «Победи Auto-sklearn» на одном наборе данных (madeline), в которой участники могли задавать гиперпараметры «вручную»,

[1] http://clopinet.com/isabelle/Projects/NIPS2006/.
[2] http://competitions.codalab.org.
[3] http://scikit-learn.org/.
[4] https://AutoML.github.io/auto-sklearn/stable/.

чтобы попытаться победить Auto-sklearn. Но никто не смог победить этот алгоритм, даже его разработчики! Участники могли отправить json-файл, описывающий модель sklearn и настройки гиперпараметров, через интерфейс GUI. Этот интерфейс позволяет исследователям, которые хотят сравнить свои методы поиска с auto-sklearn, использовать точно такой же набор гипермоделей.

В 2015/2016 гг. вокруг конкурса AutoML было организовано большое количество сопутствующих мероприятий, включая буткампы, летние школы и семинары[1]. Задача AutoML была частью официального отбора конкурсной программы IJCNN 2015 и 2016 гг., а результаты обсуждались на семинарах AutoML и CiML на ICML и NIPS в 2015 и 2016 гг. По результатам мероприятий было подготовлено несколько публикаций: в [33] мы детально описываем структуру конкурса AutoML[2].

В [32] и [34] мы рассматриваем основные и окончательные результаты, представленные на семинарах по AutoML ICML 2015 и 2016. Конкурс AutoML 2015/2016 состоял из шести раундов, в каждом из которых было представлено по пять наборов данных. Мы также организовали последующее мероприятие для конференции PAKDD 2018[3] всего в два этапа, с пятью наборами данных на этапе разработки и пятью наборами данных в финальном раунде в форме слепого тестирования.

В этой главе мы выходим за рамки ранее опубликованных обзоров, представляем систематизированное исследование победивших решений на всех наборах данных конкурса и проводим сравнения с широко используемыми машинными обучателями, реализованными в scikit-learn. Здесь также приводятся не публиковавшиеся ранее подробности о наборах данных и рефлексивном анализе.

Эта глава частично основана на опубликованном ранее материале [32–34, 36]. Глава дополнена 46-страничным онлайновым приложением, доступ к которому можно получить по адресу http://AutoML.org/book.

10.2. Формализация задачи и обзор условий

10.2.1. Предметная область задачи

Эта серия задач посвящена обучению с учителем и, в частности, решению задач классификации и регрессии без дополнительного вмешательства человека в рамках заданных ограничений. Для этого мы выпустили большое количество наборов данных, предварительно отформатированных в заданных представлениях признаков (т. е. каждый пример состоит из фиксированного числа числовых коэффициентов; подробнее об этом в разделе 10.3).

---

1 http://AutoML.chalearn.org.

2 http://codalab.org/AutoML.

3 https://www.4paradigm.com/competition/pakdd2018.

В приложениях ML не всегда проводится различие между входными и выходными переменными. Например, в рекомендательных системах задача часто формулируется как прогнозирование отсутствующих значений для каждой переменной, а не прогнозирование значений конкретной переменной [58]. В обучении без учителя [30] целью является простое и компактное объяснение данных в конечном счете с привлечением предполагаемых скрытых переменных (например, принадлежности к классу, полученной алгоритмом кластеризации).

Мы рассматриваем только строгую схему обучения с учителем, в которой данные представлены в виде идентично и независимо распределенных пар вход–выход. Используемые модели ограничены векторными представлениями фиксированной длины, исключая задачи предсказания временных рядов. Задачи обработки текста, речи и видео, включенные в задачу, были предварительно переведены в соответствующие векторные представления фиксированной длины.

Сложность предложенных задач пропорциональна сложности данных (дисбаланс классов, разреженность, пропущенные значения, категориальные переменные). «Испытательный стенд» для моделей состоит из данных из самых разных областей. Хотя существуют наборы инструментов ML, которые могут решать все вышеперечисленные задачи, все же требуются значительные человеческие усилия, чтобы найти для заданных условий оптимальные методы и настройки гиперпараметров, которые максимизируют производительность при соблюдении вычислительных ограничений. Задача участников состоит в том, чтобы создать идеальный черный ящик, который устранит необходимость в трудоемком участии человека, компенсировав нехватку специалистов по обработке данных в ближайшее десятилетие.

10.2.2. Выбор полной модели

Мы называем решения участников *гипермоделями*, чтобы показать, что они построены из более простых компонентов. Например, для задач классификации участники могут предложить гипермодель, объединяющую несколько методов классификации, таких как ближайшие соседи, линейные модели, ядерные методы, нейронные сети и случайные леса. Более сложные гипермодели могут также включать модули предварительной обработки данных, извлечения и выбора признаков.

Как правило, прогностическая модель вида $y = f(\mathbf{x}; \boldsymbol{\alpha})$ имеет:

- множество параметров $\boldsymbol{\alpha} = [\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_n]$;
- алгоритм обучения (обычно называемый *учителем*), который служит для оптимизации параметров на обучающих данных;
- обученную модель (называемую предиктором) в виде $y = f(\mathbf{x})$, созданную обучающим алгоритмом;
- целевую функцию $J(f)$, которая может быть использована для оценки работы модели на тестовых данных.

Рассмотрим теперь пространство гипотез модели, определяемое вектором гиперпараметров $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_n]$. Вектор гиперпараметров может включать

не только параметры, отвечающие за переключение между альтернативными моделями, но и такие параметры моделирования, как параметры предварительной обработки, тип ядра в ядерном методе, количество блоков и слоев в нейронной сети или параметры регуляризации алгоритма обучения [59]. Некоторые авторы называют эту задачу *полным выбором модели* (full model selection) [24, 62], другие – задачей CASH (combined algorithm selection and hyperparameter optimization) [65]. Мы будем обозначать гипермодели как

$$y = f(\mathbf{x}; \boldsymbol{\theta}) = f(\mathbf{x}; \boldsymbol{\alpha}(\boldsymbol{\theta}), \boldsymbol{\theta}), \tag{10.1}$$

где вектор параметров модели $\boldsymbol{\alpha}$ является неявной функцией вектора гиперпараметров $\boldsymbol{\theta}$, полученной с помощью учителя для фиксированного значения $\boldsymbol{\theta}$, и обучающих данных, состоящих из пар вход–выход $\{\mathbf{x}_i, y_i\}$. Участники должны разработать алгоритмы, способные обучить гиперпараметры $\boldsymbol{\theta}$. Для этого может потребоваться осмысленная выборка пространства гиперпараметров и разделение имеющихся обучающих данных на подмножества для обучения и оценки предсказательной способности решений – один или несколько раз.

С точки зрения оптимизации выбор модели представляет собой двухуровневую задачу оптимизации [7, 18, 19]; существует нижняя цель J_1 для обучения параметров $\boldsymbol{\alpha}$ модели и верхняя цель J_2 для обучения гиперпараметров $\boldsymbol{\theta}$, обе они оптимизируются одновременно (рис. 10.1). С точки зрения статистики выбор модели представляет собой задачу множественного тестирования, в которой планки ошибок при прогнозировании производительности ϵ ухудшаются с ростом числа опробованных моделей/гиперпараметров или (в более общем случае) сложности гипермодели $C_2(\boldsymbol{\theta})$. Ключевой аспект AutoML заключается в том, чтобы избежать переобучения цели верхнего уровня J_2 путем ее регуляризации; аналогично регуляризируются цели нижнего уровня J_1.

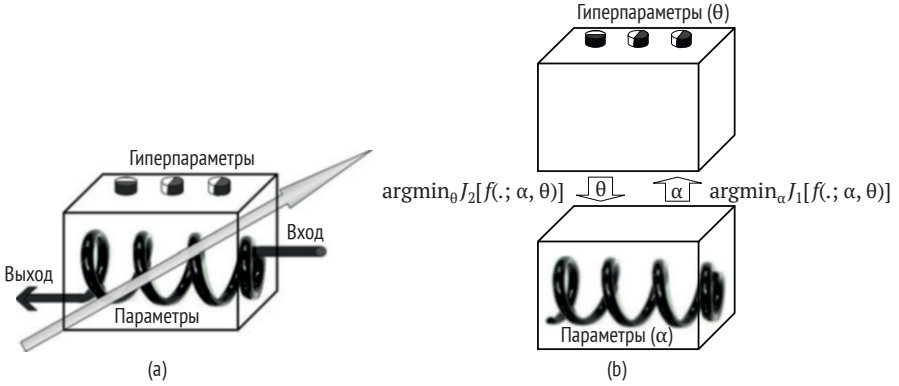


Рис. 10.1 ❖ Двухуровневая оптимизация. (a) Разделение обучаемой машины на два уровня, подлежащих настройке. (b) Разделение настройки параметров и гиперпараметров на два уровня. Цель верхнего уровня J_2 заключается в оптимизации гиперпараметров $\boldsymbol{\theta}$; цель нижнего уровня J_1 – в оптимизации параметров $\boldsymbol{\alpha}$

Указанная постановка задачи также допускает использование ансамблевых методов, которые позволяют нескольким «простым» моделям голосовать для принятия окончательного решения [15, 16, 29]. В этом случае гиперпараметры $\boldsymbol{\theta}$ можно интерпретировать как веса для голосования. Для простоты мы объединяем все параметры в один вектор, но для определения пространства гиперпараметров можно использовать более сложные структуры, такие как деревья или графы [66].

10.2.3. Оптимизация гиперпараметров

Каждый, кто работал с данными, сталкивался с необходимостью выбирать параметры модели, например такие, как масштабирование, нормализация, вменение отсутствующих значений, форма кодирования переменных (для категориальных переменных), дискретизация переменных, степень нелинейности и архитектура. С помощью ML удалось сократить число гиперпараметров и создать черные ящики для выполнения таких задач, как классификация и регрессия [21, 40]. Тем не менее любая реальная задача требует хотя бы некоторой подготовки данных, прежде чем их можно будет подать в «автоматический» метод, что требует выбора модели данных. В настоящее время наблюдается значительный прогресс в области сквозного автоматизированного ML для более сложных задач, таких как обработка текста, изображений, видео и речи с помощью методов глубокого обучения [6]. Однако даже эти методы имеют множество вариантов моделирования и гиперпараметров.

В то время как создание моделей для различных приложений было в центре внимания сообщества ML, оптимизации гиперпараметров уделяли слишком мало усилий. Обычная практика, включающая метод проб и ошибок и поиск по сетке, может привести к переобучению моделей на небольших наборах данных или к недообучению на больших наборах. Под переобучением мы подразумеваем создание моделей, которые хорошо работают на обучающих данных, но плохо работают на незнакомых данных, т. е. модели, которые не обобщают. Под недообучением мы подразумеваем выбор слишком простой модели, которая не отражает сложность данных и, следовательно, плохо работает как на обучающих, так и на тестовых данных. Несмотря на хорошо оптимизированные готовые алгоритмы оптимизации параметров, конечные пользователи все еще несут ответственность за организацию численных экспериментов для выявления лучшей из ряда рассматриваемых моделей. Из-за нехватки времени и ресурсов они часто выполняют выбор модели/гиперпараметров с помощью специальных методов. Иоаннидис и Лэнгфорд [42, 47] рассматривают фундаментальные распространенные ошибки, такие как плохое разбиение на обучающие и тестовые наборы, неподходящая сложность модели, выбор гиперпараметров с использованием тестовых наборов, неправильное использование вычислительных ресурсов и вводящие в заблуждение метрики тестов, которые могут свести на нет все исследование. Участники должны избегать этих недостатков и разрабатывать системы, которые можно тестировать слепыми методами.

Дополнительная изюминка нашей постановки задачи заключается в том, что код тестируется с ограниченными вычислительными ресурсами. То есть для каждой задачи устанавливается произвольное ограничение на время выполнения и предоставляется максимальный объем памяти. Это накладывает на участника ограничение на получение решения за заданное время и, следовательно, на оптимизацию поиска модели с вычислительной точки зрения. В итоге участники должны одновременно решать задачу избыточного/недостаточного обучения и задачу эффективного поиска оптимального решения, как указано в [43]. На практике вычислительные ограничения оказались гораздо более сложной задачей для участников, чем проблема переобучения. Поэтому основной вклад был сделан в разработку новых эффективных методов поиска с использованием передовых методов оптимизации.

10.2.4. Стратегии поиска моделей

Большинство практиков используют эвристику, такую как поиск по сетке или равномерная выборка для выбора пространства $\boldsymbol{\theta}$, и используют k-кратную перекрестную проверку в качестве цели верхнего уровня J_2 [20]. В этой схеме оптимизация $\boldsymbol{\theta}$ не выполняется последовательно [8]. Все параметры дискретизируются по регулярной схеме, обычно в линейном или логарифмическом масштабе. Это приводит к тому, что число вариантов экспоненциально увеличивается с ростом размерности $\boldsymbol{\theta}$. k-кратная перекрестная проверка заключается в разбиении набора данных на k выборок; $(k - 1)$ выборок используют для обучения, а оставшуюся выборку – для тестирования; в конечном итоге сообщается среднее значение тестовых оценок, полученных на k выборках. Обратите внимание, что некоторые наборы инструментов ML в настоящее время поддерживают перекрестную проверку. Отсутствуют принципиальные рекомендации по определению количества точек сетки и значения k (за исключением [20]), а также рекомендации по регуляризации J_2, однако этот простой метод является хорошим базовым подходом.

Были предприняты попытки оптимизировать непрерывные гиперпараметры с помощью двухуровневых методов оптимизации с использованием либо k-кратной перекрестной оценки [7, 50], либо оценки с исключением по одному в качестве цели верхнего уровня J_2. Оценку с исключением по одному можно эффективно вычислить в аналитической форме как побочный продукт обучения только одного предиктора на всех учебных примерах (например, виртуальное исключение по одному [38]). Метод был улучшен путем добавления регуляризации J_2 [17]. Градиентный спуск использовался для ускорения поиска путем локальной квадратичной аппроксимации J_2 [44]. В некоторых случаях полный $J_2(\boldsymbol{\theta})$ может быть вычислен на основе нескольких ключевых примеров [39, 54]. Другие подходы минимизируют приближение или верхнюю границу ошибки при исключении по одному, а не ее точную форму [53, 68]. Тем не менее эти методы все еще ограничены конкретными моделями и непрерывными гиперпараметрами.

Одной из первых попыток полного выбора модели был метод поиска моделей, который использует k-кратную перекрестную проверку для J_2. Он исследует пространство гиперпараметров шагами одинаковой величины, и когда никакое изменение любого параметра не уменьшает J_2, размер шага уменьшается вдвое, и процесс повторяется до тех пор, пока шаги не будут сочтены достаточно малыми [49]. Эскаланте и др. [24] рассмотрели задачу полного выбора модели с помощью оптимизации методом роя частиц, который работает, имея популяцию решений-кандидатов (частиц) и перемещая эти частицы по пространству гиперпараметров путем изменения положения и скорости частицы. Для J_2 также используется k-кратная перекрестная проверка. Этот подход позволил получить выигрышную модель в 76 % случаев. Переобучение ограничивали эвристически с помощью ранней остановки, а соотношение обучающих и проверочных данных не оптимизировалось. Несмотря на прогресс в разработке экспериментов для снижения риска переобучения [42, 47], в частности, путем теоретически обоснованного разделения данных [61], насколько нам известно, никто не занимался целенаправленно проблемой оптимального разделения данных.

Хотя регуляризация второго уровня вывода является относительно недавним нововведением в сообществе частотного ML, она была неотъемлемой частью байесовского моделирования через понятие гиперприоритета. Некоторые методы многоуровневой оптимизации объединяют выборку важности и марковские цепи Монте-Карло [2]. Область байесовской оптимизации гиперпараметров быстро развивалась и дала многообещающие результаты, в частности, благодаря использованию гауссовых процессов (GP) для моделирования эффективности обобщения [60, 63]. Однако было установлено, что подходы на основе древовидного оценивателя Парзена (tree-structured Parzen estimator, TPE), моделирующие $P(\mathbf{x}|y)$ и $P(y)$, а не $P(y|\mathbf{x})$ напрямую [9, 10], превосходят байесовскую оптимизацию на основе GP для задач структурированной оптимизации со многими гиперпараметрами, включая дискретные [23]. Основная идея этих методов заключается в подгонке $J_2(\boldsymbol{\theta})$ к гладкой функции в попытке уменьшить дисперсию и оценить ее в областях пространства гиперпараметров, которые недостаточно дискретизированы, чтобы направить поиск к областям с высокой дисперсией. Эти методы показали перспективные результаты, и некоторые из идей могут быть использованы в частотной системе. Например, алгоритм SMAC на основе случайного леса [41], который помог на порядки ускорить алгоритмы локального поиска и древовидного поиска на определенных распределениях экземпляров, также оказался очень эффективным для оптимизации гиперпараметров алгоритмов машинного обучения и лучше других алгоритмов масштабируется для высоких размерностей и дискретных входных данных [23]. Мы также заметили, что методы байесовской оптимизации часто комбинируют с другими методами, такими как метаобучение и ансамблевые методы [25], чтобы получить преимущество в некоторых задачах с ограничением по времени [32]. Некоторые из этих методов совместно рассматривают двухуровневую оптимизацию и принимают временные затраты как критическое руководство для поиска гиперпараметров [45, 64].

Помимо байесовской оптимизации, в литературе представлено еще несколько семейств подходов, которые привлекли большое внимание в связи

с развитием глубокого обучения. Идеи, заимствованные из обучения с подкреплением, были использованы для построения оптимальных архитектур нейронных сетей [4, 70]. Эти подходы формулируют задачу оптимизации гиперпараметров в стиле обучения с подкреплением, где, например, состояниями являются фактические настройки гиперпараметров (например, архитектура сети), действиями – добавление или удаление модуля (например, слоя CNN или слоя объединения), а вознаграждением – точность модели на проверочных данных. Затем они могут применить готовые алгоритмы обучения с подкреплением (например, RENFORCE, Q-learning, поиск по дереву Монте-Карло) для решения задачи. Другие методы поиска архитектуры используют эволюционные алгоритмы [3, 57]. Эти подходы рассматривают набор (популяцию) настроек гиперпараметров (особей), изменяют их (мутируют и размножают) и устраняют неперспективные настройки в соответствии с их оценкой перекрестной проверки (пригодностью). После нескольких поколений глобальное качество популяции возрастает. Одна из важных общих черт обучения с подкреплением и эволюционных алгоритмов заключается в том, что они предусматривают компромисс между исследованием и использованием. Несмотря на впечатляющие результаты, эти подходы требуют огромного количества вычислительных ресурсов, а некоторые (особенно эволюционные алгоритмы) трудно масштабировать. Фам и др. [56] недавно предложили распределение веса между дочерними моделями для значительного ускорения процесса [70] при достижении сопоставимых результатов.

Отметим, что разбиение задачи подбора параметров на два уровня может быть расширено на большее количество уровней за счет дополнительной сложности – т. е. добавления иерархии разбиения данных для выполнения множественной или вложенной перекрестной проверки [22], недостаточного количества данных для обучения и проверки на разных уровнях и увеличения вычислительной нагрузки.

В табл. 10.1 показан типичный пример многоуровневой оптимизации параметров при частотном подходе. Мы предполагаем, что используем инструментарий ML с двумя обучающими машинами: Kridge (kernel ridge regression) и Neural (нейронная сеть, она же модель глубокого обучения). На верхнем уровне мы используем тестовую процедуру для оценки производительности конечной модели (это не уровень вывода). Алгоритм вывода верхнего уровня Validation({GridCV(Kridge, MSE), GridCV(Neural, MSE)}, MSE) рекурсивно разлагается на элементы. Алгоритм Validation использует разбиение данных $D = [D_{Tr}, D_{Va}]$ для сравнения обучаемых машин Kridge и Neural (обученных с помощью D_{Tr} на валидационном множестве D_{Va}, с помощью функции оценки среднеквадратичной ошибки MSE). Алгоритм GridCV – поиск по сетке с 10-кратной перекрестной проверкой (CV) с функцией оценки MSE, затем оптимизирует гиперпараметры $\boldsymbol{\theta}$. Как Kridge, так и Neural используют виртуальную перекрестную проверку с исключением по одному (LOO) для настройки γ и классический L_2-регуляризованный функционал риска переобучения для настройки $\boldsymbol{\alpha}$.

Таблица 10.1. Типичный пример многоуровневого алгоритма вывода

Уровень	Алгоритм	Параметры		Выполняемая оптимизация	Разделение данных
		Фиксир.	Перемен.		
–	f	Все	Все	Определение производительности (не вывод)	D_{Te}
4	Проверка	Нет	Все	Окончательный выбор алгоритма с использованием проверочных данных	$D = [D_{Tr}, D_{Va}]$
3	GridCV	Индекс модели i	$\boldsymbol{\theta}$, γ, $\boldsymbol{\alpha}$	10-кратная перекрестная проверка на регулярных выборках из $\boldsymbol{\theta}$	$D_{Tr} = [D_{tr}, D_{va}]_{CV}$
2	Kridge($\boldsymbol{\theta}$) Neural ($\boldsymbol{\theta}$)	i, $\boldsymbol{\theta}$	γ, $\boldsymbol{\alpha}$	Виртуальная перекрестная проверка с исключением по одному для выбора параметра регуляризации γ	$D_{tr} = [D_{tr}^{\setminus\{d\}}, d]_{CV}$
1	Kridge($\boldsymbol{\theta}$, γ) Neural($\boldsymbol{\theta}$, γ)	i, $\boldsymbol{\theta}$, γ	$\boldsymbol{\alpha}$	Матричная инверсия градиентного спуска для вычисления $\boldsymbol{\alpha}$	D_{tr}
0	Kridge($\boldsymbol{\theta}$, γ, $\boldsymbol{\alpha}$) Neural($\boldsymbol{\theta}$, γ, $\boldsymbol{\alpha}$)	Все	Нет	–	–

Алгоритм верхнего уровня Validation({GridCV(Kridge, MSE), GridCV(Neural, MSE)}, MSE) рекурсивно разлагается на элементы. Вызов метода train с использованием данных D_{TrVa} приводит к функции *f,* которая затем тестируется с помощью test(*f*, *MSE*, D_{Te}). Обозначение $[.]_{CV}$ означает, что результаты являются средними по нескольким разбиениям данных (перекрестная проверка). Прочерк «—» означает «неприменимо». Семейство моделей с параметрами $\boldsymbol{\alpha}$ и гиперпараметрами $\boldsymbol{\theta}$ представляется как $f(\boldsymbol{\theta}, \boldsymbol{\alpha})$. Здесь мы отступаем от обычая ставить гиперпараметры на последнее место, гиперпараметры перечислены в порядке убывания уровня вывода. $\mathcal{F}$, рассматриваемый как алгоритм нижнего уровня, не выполняет никакого обучения: train($f(\boldsymbol{\theta}, \boldsymbol{\alpha})$) просто возвращает функцию $f(\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\alpha})$.

Заимствуя традиционную классификацию методов выбора признаков [11, 38, 46], стратегии поиска модели можно разделить на фильтры, обертки и встроенные методы (рис. 10.2). *Фильтры* – это методы сужения пространства моделей без обучения специальной модели верхнего уровня. К таким методам относятся предварительная обработка, конструирование признаков, разработка ядра, проектирование архитектуры, выбор приоритета или регуляризаторов, выбор модели шума, а также методы фильтрации для отбора признаков. Хотя некоторые фильтры используют обучающие данные, многие из них используют предварительные знания человека о задаче или знания, собранные из предыдущих задач. Недавно в [5] было предложено применить методы коллаборативной фильтрации для поиска моделей. *Обертоочные методы* рассматривают обучаемые модели как черный ящик, способный обучаться на примерах и делать предсказания после обучения. Они работают с алгоритмом поиска в пространстве гиперпараметров (поиск по сетке или стохастический поиск) и функцией, оценивающей производительность обученной модели (ошибка перекрестной проверки или байесовское доказательство). *Встроенные методы* похожи на обертки, но они используют знание алгоритма машинного обучения, чтобы сделать поиск более эффективным. Например, некоторые встроенные методы находят решение вы-

числительным путем, ничего не опуская, т. е. выполняя однократное обучение модели на всех обучающих данных (например, [38]). Другие встроенные методы совместно оптимизируют параметры и гиперпараметры [44, 50, 51].

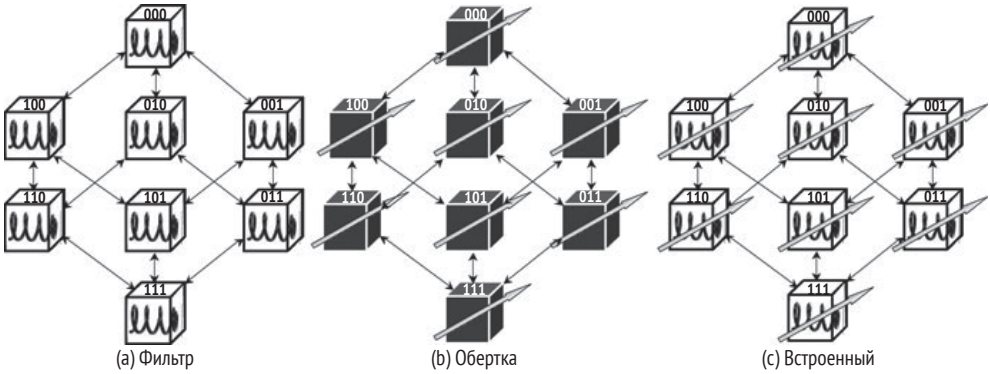


Рис. 10.2 ❖ Подходы к двухуровневому выводу. (a) *Методы фильтрации* выбирают гиперпараметры без настройки параметров обучаемой модели. (b) *Обертночные методы* выбирают гиперпараметры с помощью обученных моделей, рассматривая их как черные ящики. (c) *Встроенные методы* используют знания о структуре и/или параметрах обучаемой модели, чтобы направлять поиск гиперпараметров

В целом многие авторы фокусируются только на эффективности поиска, игнорируя проблему переобучения задачи второго уровня J_2, которая часто рассматривается как k-кратная перекрестная проверка с произвольным значением k. Байесовские методы предлагают способ избежать переобучения через понятие гипераприорных распределений, но за счет принятия предположений о том, как были сгенерированы данные, и без гарантий производительности. Во всех известных нам априорных подходах к полному выбору никто не пытался рассматривать задачу как оптимизацию регуляризованного функционала J_2 относительно 1) выбора модели и 2) разбиения данных. Для совместного решения статистических и вычислительных вопросов еще предстоит проделать большую работу. Серия конкурсов AutoML предлагает бенчмарки для сравнения и сопоставления методов решения этих проблем, свободные от предубеждения изобретателя/оценивателя.

10.3. Данные

Летом 2014 г. с помощью многочисленных коллег мы собрали первый пул из 70 наборов данных и в итоге выбрали из них 30 наборов данных для задачи 2015/2016 (см. табл. 10.2 и онлайн-приложение), отобранных для иллюстрации широкого спектра областей применения, таких как: биология и медицина, экология, энергетика и управление устойчивостью, обработка изображений, текста, аудио, речи, видео и других сенсорных данных, управление

Таблица 10.2. Наборы данных для конкурса AutoML 2015/2016. C – количество классов, Cbal – баланс классов, Sparse – разреженность, Miss. – доля пропущенных значений, Cat – категориальные переменные, Irr – доля нерелевантных переменных, Pte, Pva, Ptr – количество экземпляров тестового, проверочного и обучающего наборов соответственно, N – количество признаков, Ptr/N – соотношение параметров набора данных

Rnd	Набор данных	Задача	Метрика	Время	C	Cbal	Sparse	Miss.	Cat	Irr	Pte	Pva	Ptr	N	Ptr/N
0	1 ADULT	многомет.	Fl	300	3	1	0.16	0.011	1	0.5	9768	4884	34 190	24	1424.58
0	2 CADATA	регрессия	R2	200	0	NaN	0	0	0	0.5	10 640	5000	5000	16	312.5
0	3 DIGITS	многокл.	BAG	300	10	1	0.42	0	0	0.5	35 000	20 000	15 000	1568	9.57
0	4 DOROTHEA	бинарн.	AUC	100	2	0.46	0.99	0	0	0.5	800	350	800	100 000	0.01
0	5 NEWSGROUPS	многокл.	PAG	300	20	1	1	0	0	0	3755	1877	13 142	61 188	0.21
1	1 CHRISTINE	бинарн.	BAG	1200	2	1	0.071	0	0	0.5	2084	834	5418	1636	3.31
1	2 JASMINE	бинарн.	BAG	1200	2	1	0.78	0	0	0.5	1756	526	2984	144	20.72
1	3 MADELINE	бинарн.	BAG	1200	2	1	1.2e-06	0	0	0.92	3240	1080	3140	259	12.12
1	4 PHILIPPINE	бинарн.	BAG	1200	2	1	0.0012	0	0	0.5	4664	1166	5832	308	18.94
1	5 SYLVINE	бинарн.	BAG	1200	2	1	0.01	0	0	0.5	10 244	5124	5124	20	256.2
2	1 ALBERT	бинарн.	Fl	1200	2	1	0.049	0.14	1	0.5	51 048	25 526	425 240	78	5451.79
2	2 DILBERT	многокл.	PAG	1200	5	1	0	0	0	0.16	9720	4860	10 000	2000	5
2	3 FA BERT	многокл.	PAG	1200	7	0.96	0.99	0	0	0.5	2354	1177	8237	800	10.3
2	4 ROBERT	многокл.	BAG	1200	10	1	0.01	0	0	0	5000	2000	10 000	7200	1.39
2	5 VOLKERT	многокл.	PAG	1200	10	0.89	0.34	0	0	0	7000	3500	58 310	180	323.94
3	1 ALEXIS	многомет.	AUC	1200	18	0.92	0.98	0	0	0	15 569	7784	54 491	5000	10.9
3	2 DIONIS	многокл.	BAG	1200	355	1	0.11	0	0	0	12 000	6000	416 188	60	6936.47
3	3 GRIGORIS	многомет.	AUC	1200	91	0.87	1	0	0	0	9920	6486	45 400	301 561	0.15
3	4 JANNIS	многокл.	BAG	1200	4	0.8	7.3e-05	0	0	0.5	9851	4926	83 733	54	1550.61
3	5 WALLIS	многокл.	AUC	1200	11	0.91	1	0	0	0	8196	4098	10 000	193 731	0.05
4	1 EVITA	бинарн.	AUC	1200	2	0.21	0.91	0	0	0.46	14 000	8000	20 000	3000	6.67
4	2 FLORA	регрессия	ABS	1200	0	NaN	0.99	0	0	0.25	2000	2000	15 000	200 000	0.08
4	3 HELENA	многокл.	BAG	1200	100	0.9	6e-05	0	0	0	18 628	9314	65 196	27	2414.67
4	4 TANIA	многомет.	PAC	1200	95	0.79	1	0	0	0	44 635	22 514	157 599	47 236	3.34
4	5 YOLANDA	регрессия	R2	1200	0	NaN	1e-07	0	0	0.1	30 000	30 000	400 000	100	4000
5	1 ARTURO	многокл.	Fl	1200	20	1	0.82	0	0	0.5	2733	1366	9565	400	23.91
5	2 CARLO	бинарн.	PAC	1200	2	0.097	0.0027	0	0	0.5	10 000	10 000	50 000	1070	46.73
5	3 MARCO	многомет.	AUC	1200	24	0.76	0.99	0	0	0	20 482	20 482	163 860	15 299	10.71
5	4 PABLO	регрессия	ABS	1200	0	NaN	0.11	0	0	0.5	23 565	23 565	188 524	120	1571.03
5	5 WALDO	многокл.	BAC	1200	4	1	0.029	0	1	0.5	2430	2430	19 439	270	72

социальными сетями и реклама в интернете, анализ рынка и финансовое прогнозирование. Мы предварительно обработали данные, чтобы получить представления признаков (т. е. каждый экземпляр состоит из фиксированного количества числовых коэффициентов). Задачи обработки текста, речи и видео были включены в конкурсное задание, но не в своих родных представлениях переменной длины.

Для конкурса 2018 г. были отобраны три набора данных из первого пула (но не использованные в первой задаче) и добавлены семь новых наборов данных, собранных новыми организаторами и спонсорами (см. табл. 10.3 и онлайн-приложение).

Некоторые наборы данных были получены из открытых источников, но они были переформатированы в новые представления, чтобы скрыть их происхождение, за исключением финального раунда конкурсов 2015/2016 гг. и финального этапа конкурса 2018 г., которые включали совершенно новые данные.

Таблица 10.3. Наборы данных для конкурса AutoML 2018. Все задачи относятся к бинарной классификации. Метрикой является AUC для всех задач. Бюджет времени также одинаков для всех наборов данных (1200 с). Фаза 1 была фазой разработки, а фаза 2 – финальной фазой слепого тестирования. Обозначения столбцов те же, что в табл. 10.2

Этап	Набор данных	Cbal	Sparse	Miss	Cat	Irr	Pte	Pva	Ptr	N	Ptr/N
1	1 ADA	1	0.67	0	0	0	41 471	415	4147	48	86.39
1	2 ARCENE	0.22	0.54	0	0	0	700	100	100	10 000	0.01
1	3 GINA	1	0.03	0.31	0	0	31 532	315	3153	970	3.25
1	4 GUILLERMO	0.33	0.53	0	0	0	5000	5000	20 000	4296	4.65
1	5 RL	0.10	0	0.11	1	0	24 803	0	31 406	22	1427.5
2	1 PM	0.01	0	0.11	1	0	20 000	0	29 964	89	224.71
2	2 RH	0.04	0.41	0	1	0	28 544	0	31 498	76	414.44
2	3 RI	0.02	0.09	0.26	1	0	26 744	0	30 562	113	270.46
2	4 RICCARDO	0.67	0.51	0	0	0	5000	5000	20 000	4296	4.65
2	5 RM	0.001	0	0.11	1	0	26 961	0	28 278	89	317.73

В конкурсе 2015/2016 гг. сложность данных постепенно возрастала от раунда к раунду. В раунде 0 было представлено пять (публичных) наборов данных из предыдущих конкурсов, иллюстрирующих различные трудности, возникающие в последующих раундах.

Начальный. Только задачи бинарной классификации. Отсутствие недостающих данных; отсутствие категориальных признаков; умеренное количество признаков (<2000); сбалансированные классы. Сложность заключается в необходимости работать с разреженными и полными матрицами, наличием нерелевантных переменных и различными отношениями *Ptr/N*.

Промежуточный. Задачи бинарной и многоклассовой классификации. Сложность заключается в работе с несбалансированными классами, количеством классов, отсутствующими значениями, категориальными переменными и количеством признаков, достигающим 7000.

Продвинутый. Бинарные, многоклассовые и многозначные задачи классификации. Сложность заключается в работе с 300 000 признаков.

Эксперт. Задачи классификации и регрессии. Сложность заключается в работе со всем диапазоном сложности данных.

Мастер. Задачи классификации и регрессии любой сложности. Сложность состоит в обучении на совершенно новых наборах данных.

Все наборы данных в конкурсе 2018 г. представляли собой задачи бинарной классификации. Проверочные разделы не использовались, даже если они были доступны для некоторых задач. Три повторно использованных набора данных имели ту же сложность, что и наборы данных 1-го и 2-го раундов конкурса 2015/2016. Однако семь новых наборов данных создали участникам трудности, которых не было в предыдущем конкурсе. В частности, это экстремальный дисбаланс классов, наличие категориальных признаков и временная зависимость между экземплярами, которые могут быть использованы участниками для разработки своих методов[1]. Наборы данных по обоим конкурсам можно загрузить с сайта http://AutoML.chalearn.org/data.

10.4. Протокол конкурса

В этом разделе мы описываем схему проведения конкурса, которую мы разработали для обеспечения тщательной и справедливой оценки. Как было указано ранее, мы фокусируемся на задачах обучения с учителем (классификация и регрессия), без вмешательства человека, в рамках заданных ограничений по времени и компьютерным ресурсам (раздел 10.4.1) и с определенной метрикой (раздел 10.4.2), которая варьируется в зависимости от набора данных. В ходе конкурсов происхождение и описание наборов данных скрываются (за исключением самого первого раунда или этапа, где раздаются образцы данных), чтобы исключить использование знаний о предметной области и подтолкнуть участников к разработке полностью автоматизированных решений ML. В конкурсах AutoML 2015/2016 наборы данных были представлены в серии раундов (раздел 10.4.3), чередующих периоды разработки кода (этапы Tweakathon) и слепого тестирования кода без вмешательства человека (этапы AutoML). На этапах разработки можно было представить как результаты, так и код, но для участия в рейтинге слепого тестирования AutoML необходимо было представить код. В 2018 г. протокол конкурса AutoML был упрощен. У нас был только один раунд из двух этапов: этап разработки, в котором пять наборов данных было выпущено для практических целей, и заключительный этап слепого теста с пятью новыми наборами данных, которые никогда не использовались ранее.

---

[1] В наборах данных RL, PM, RH, RI и RM экземпляры были отсортированы в хронологическом порядке, эта информация была доступна участникам и могла быть использована для разработки их методов.

10.4.1. Бюджет времени и вычислительные ресурсы

Платформа Codalab предоставляет вычислительные ресурсы, общие для всех участников. Мы использовали до 10 вычислительных машин, параллельно обрабатывая очередь заявок, поданных участниками. Каждый экземпляр машины был оснащен 8 ядрами x86_64. Объем памяти был увеличен с 24 до 56 Гб после 3-го раунда конкурса AutoML 2015/2016. Для конкурса AutoML 2018 г. вычислительные ресурсы были сокращены, поскольку мы хотели стимулировать разработку более эффективных и в то же время результативных решений AutoML. Мы использовали шесть экземпляров машин, параллельно обрабатывающих очередь заявок. Каждый экземпляр был оснащен двумя ядрами x86_64 и 8 Гб памяти.

Для обеспечения справедливой оценки кода вычислитель выделялся только для обработки этого кода, и время его работы было ограничено заданным бюджетом времени (который мог варьироваться в зависимости от набора данных). Бюджет времени предоставлялся участникам вместе с каждым набором данных в информационном файле. Из практических соображений, за исключением первой фазы первого раунда, обычно мы выделяли 1200 с (20 мин) для каждого набора данных. Однако участники не знали этого заранее, и поэтому их код должен был иметь возможность управлять заданным временным бюджетом. Участники, представившие результаты вместо кода, не были ограничены временным бюджетом, поскольку их код выполнялся на их собственной платформе. Это было потенциально выгодно для работ, засчитываемых на финальных этапах (сразу после Tweakathon). Участники, желающие принять участие в этапах AutoML (слепое тестирование), для которых требовалось представить код, могли представить и результаты, и код одновременно. Когда результаты были представлены, они использовались как заявки на текущем этапе. Они не обязательно должны были быть произведены представленным кодом; т. е. если участник не хотел делиться личным кодом, он мог представить образец открытого конкурсного кода вместе с результатами выполнения закрытого кода. Код автоматически передавался на этапы AutoML для слепого тестирования. На этапах AutoML отправка результатов была невозможна.

Участникам предлагалось сохранить и отправить промежуточные результаты, чтобы мы могли построить кривые обучения. Эта возможность не была использована во время проведения конкурсов. Но мы рассмотрим кривые обучения в этой главе, чтобы оценить способность различных алгоритмов быстро достигать хороших результатов.

10.4.2. Метрики подсчета баллов

Баллы вычисляются путем сравнения представленных прогнозов с эталонными целевыми значениями. Для каждого экземпляра данных $i = 1 : P$ (где P – размер проверочного или тестового множества), целевое значение пред-

ставляет собой непрерывный числовой коэффициент y_i для задач регрессии, двоичный индикатор {0, 1} для задач различения двух классов или вектор двоичных индикаторов $[y_{il}]$ в {0, 1} для задач многоклассовой или многозначной классификации (по одному на класс l). Участники должны были представить прогнозные значения, максимально точно соответствующие целевым значениям, в виде непрерывного числового коэффициента q_i для задач регрессии и вектора числовых коэффициентов $[q_{il}]$ в диапазоне [0, 1] для задач многоклассовой или многозначной классификации (по одному на класс l).

Предоставляемый стартовый набор содержит реализацию на языке Python всех скоринговых метрик, используемых для оценки записей. Каждый набор данных имеет свой собственный критерий оценки, указанный в его информационном файле. Все оценки нормированы таким образом, что ожидаемое значение оценки для случайного предсказания, основанного на априорных вероятностях классов, равно 0, а оптимальная оценка равна 1. Многозначные задачи рассматриваются как несколько задач двоичной классификации и оцениваются с помощью среднего значения оценок каждой подзадачи двоичной классификации.

Сначала мы определим обозначение $\langle\cdot\rangle$ для среднего значения по всем экземплярам P, индексируемым по i. То есть

$$\langle y_i\rangle = (1/P)\sum_{i=1}^{P}(y_i). \tag{10.2}$$

Метрики оценки определены следующим образом.

R^2. Коэффициент смешанной корреляции (детерминации), используется только для задач регрессии. Метрика основана на среднеквадратичной ошибке (MSE) и дисперсии (VAR) и вычисляется как

$$R^2 = 1 - MSE/VAR, \tag{10.3}$$

где $MSE = \langle(y_i - q_i)^2\rangle$ и $VAR = \langle(y_i - m)^2\rangle$, причем $m = \langle y_i\rangle$.

ABS. Этот коэффициент аналогичен R^2, но основан на средней абсолютной ошибке (MAE) и среднем абсолютном отклонении (MAD) и рассчитывается как

$$ABS = 1 - MAE/MAD, \tag{10.4}$$

где $MAE = \langle\text{abs}(y_i - q_i)\rangle$ и $MAD = \langle\text{abs}(y_i - m)\rangle$.

BAC. Сбалансированная точность – это средняя точность по классам для задач классификации, а также средняя *чувствительность* (доля истинно положительных результатов) и *избирательность* (доля истинно отрицательных результатоы) для бинарной классификации:

$$BAC = \begin{cases} \frac{1}{2}\left[\frac{TP}{P} + \frac{TN}{N}\right] & \text{для бинарной классификации} \\ \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{N_i} & \text{для многоклассовой классификации} \end{cases}, \tag{10.5}$$

где $P(N)$ – количество положительных (отрицательных) экземпляров, TP (TN) – количество правильно классифицированных положительных (отрицательных) экземпляров, C – количество классов, TP_i – количество правильно классифицированных экземпляров класса i, N_i – количество экземпляров класса i.

Для задач бинарной классификации точность по классам – это доля правильных предсказаний классов при пороговом значении q_i, равном 0.5 для каждого класса. Для задач с несколькими метками точность по классу усредняется по всем классам. Для многозначных задач предсказания бинаризуются путем выбора класса с максимальным значением предсказания $\text{argmax}_l q_{il}$ перед вычислением точности по классам.

Мы нормализуем метрику следующим образом:

$$|BAC| = (BAC - R)/(1 - R), \tag{10.6}$$

где R – ожидаемое значение BAC для случайных прогнозов (т. е. $R = 0.5$ для бинарной классификации и $R = (1/C)$ для задач C-кратной классификации).

AUC. Площадь под кривой ROC используется для проблем ранжирования и бинарной классификации. ROC-кривая – это кривая зависимости *чувствительности* от l-избирательности при различных порогах предсказания. Значения AUC и BAC одинаковы для бинарных прогнозов. AUC рассчитывается для каждого класса отдельно, а затем усредняется по всем классам. Мы нормализуем метрику следующим образом:

$$|AUC| = 2AUC - 1. \tag{10.7}$$

Оценка F1. Среднее гармоническое значение точности и отклика вычисляется как

$$F1 = 2 * (\textit{точность} * \textit{отклик})/(\textit{точность} + \textit{отклик}), \tag{10.8}$$

$$\textit{точность} = \textit{истинно положит.} / (\textit{истинно положит.} + \textit{ложноположит.}), \tag{10.9}$$

$$\textit{отклик} = \textit{истинно положит.} / (\textit{истинно положит.} + \textit{ложноотрицат.}). \tag{10.10}$$

Пороговая оценка прогнозов и усреднение по классам обрабатываются аналогично BAC. Мы нормализуем метрику следующим образом:

$$|F1| = (F1 - R)/(1 - R), \tag{10.11}$$

где R – ожидаемое значение $F1$ для случайных предсказаний (см. BAC).

PAC. Вероятностная точность основана на перекрестной энтропии (или логарифмической потере) и вычисляется как

$$PAC = \exp(-CE), \tag{10.12}$$

$$CE = \begin{cases} average \sum_l \log(q_{il}) & \text{для многоклассовой} \\ -\langle y_i \log(q_i) & \\ +(1 - y_i)\log(1 - q_i)\rangle & \text{для бинарной и многозначной} \end{cases}. \tag{10.13}$$

В случае с несколькими метками мы нормализуем метрику следующим образом:

$$|PAC| = (PAC - R)/(1 - R), \tag{10.14}$$

где R – оценка, полученная с помощью $q_i = \langle y_i \rangle$ или $q_{il} = \langle y_{il} \rangle$ (т. е. с использованием в качестве предсказаний доли положительных экземпляров класса как оценки априорной вероятности).

Обратите внимание, что при нормализации R^2, ABS и PAC используется среднее целевое значение $q_i = \langle y_i \rangle$ или $q_{il} = \langle y_{il} \rangle$. В отличие от этого при нормализации BAC, AUC и F1 используется случайное предсказание одного из классов с равномерной вероятностью.

Для регрессии имеют значение только R^2 и ABS; остальные метрики мы вычисляем для полноты, заменяя целевые значения бинарными после сравнения с пороговым значением в среднем диапазоне.

10.4.3. Раунды и этапы в конкурсе 2015/2016

Конкурс 2015/2016 проходил в несколько этапов, сгруппированных в шесть раундов. Раунд 0 (подготовительный) был тренировочным раундом с использованием общедоступных наборов данных. За ним последовало пять раундов с постепенным повышением сложности (начальный, средний, продвинутый, эксперт и мастер). За исключением раундов 0 и 5, все раунды включали три этапа, в которых чередовались конкурсы AutoML и Tweakathons. Эти этапы описаны в табл. 10.4.

Таблица 10.4 Этапы раунда n в конкурсе 2015/2016. Для каждого набора данных предоставляется один размеченный обучающий набор и два неразмеченных набора (проверочный и тестовый) для тестирования

Этап раунда [n]	Цель	Длительность	Представление результатов	Данные	Критерии лидерства	Приз
* AutoML[n]	Слепой тест или код	Краткий	Нет (миграция кода)	Новые наборы, нескачиваемые	Результаты на тестовом наборе	Да
Tweakathon[n]	Ручная настройка	Месяцы	Код и/или результаты	Скачиваемые наборы	Результаты на проверочном наборе	Нет
* Final[n]	Результаты предыдущего раунда	Краткий	Нет (миграция результатов)	—	Результаты на тестовом наборе	Да

Представления результатов делались только на этапах Tweakathon. Результаты последней заявки отображались в таблице лидеров, и такая заявка автоматически переходила на следующий этап. Таким образом, код участников, выбывших до окончания конкурса, получал шанс быть протестированным в последующих раундах и этапах. Новые участники могли подключиться

к конкурсу в любое время. Призы присуждались на этапах, отмеченных знаком *, во время которых не было ни одной заявки. Чтобы принять участие в этапе AutoML[n], код должен быть представлен в Tweakathon[n – 1].

Чтобы поощрить участников использовать GPU и глубокое обучение, в 4-й раунд был включен GPU-трек, спонсором которого выступила NVIDIA.

Для участия в этапе Final[n] код или результаты должны были быть представлены в Tweakathon[n]. Если были представлены и код, и (хорошо оформленные) результаты, то результаты использовались для подсчета баллов, а не для повторного прохождения кода в Tweakathon[n] и Final[n]. Код выполнялся, если результаты были недоступны или плохо отформатированы. Таким образом, не было никаких минусов в том, чтобы присылать и результаты, и код. Если участник подавал и результаты, и код, то для участия в этапах Tweakathon/Final и AutoML можно было использовать разные методы. Заявки подавались только во время этапа Tweakathon, максимум пять заявок в день. На табло лидеров предоставлялась немедленная обратная связь по данным проверки. Участники ранжировались на основе результатов тестирования на этапах Final и AutoML.

Мы предоставили базовое программное обеспечение, используя библиотеку ML scikit-learn [55]. В ней используются ансамблевые методы, которые улучшаются со временем путем добавления большего количества базовых обучателей. Кроме количества базовых обучателей, использовались настройки гиперпараметров по умолчанию. Участники не были обязаны использовать язык Python или основной скрипт Python, который мы привели в качестве примера. Однако большинство участников сочли удобным использовать основной скрипт Python, который обрабатывал разреженный формат данных и управлял сквозными настройками обучения и метриками оценки. Многие ограничились поиском лучшей модели в библиотеке scikit-learn. Это говорит о важности предоставления хорошего стартового набора, но также и об опасности смещения результатов в сторону конкретных решений.

10.4.4. Этапы конкурса 2018 года

Соревнование 2015/2016 AutoML было очень длинным, и немногие команды приняли участие во всех раундах. Кроме того, даже несмотря на отсутствие обязательства участвовать в предыдущих раундах для участия в новых раундах, новые потенциальные участники опасались, что окажутся в невыгодном положении. Поэтому мы считаем, что предпочтительнее организовывать повторяющиеся ежегодные мероприятия, каждое из которых имеет свой семинар и возможность публикации. Это обеспечивает хороший баланс между конкуренцией и сотрудничеством.

В 2018 г. мы организовали один раунд конкурса AutoML в два этапа. В этом упрощенном протоколе участники могли практиковаться на пяти наборах данных во время первого этапа (разработки), представляя либо код, либо результаты. Их результаты отображались в таблице лидеров сразу же, как только они становились доступными.

Последняя заявка на этапе разработки автоматически передавалась на второй этап – слепое тестирование AutoML. На этом втором этапе (единственном предусматривающем призовые места) код участников автоматически оценивался на пяти новых наборах данных на платформе Codalab. Эти наборы данных не были раскрыты участникам. Таким образом, работы, не содержащие код, способный к автоматическому обучению и тестированию, не попадали в финальный этап и не могли претендовать на призы.

Мы предоставили тот же стартовый набор, что и в конкурсе AutoML 2015/2016, но участники также имели доступ к коду победителей предыдущего конкурса.

10.5. Результаты

В этом разделе приводится краткое описание результатов, полученных в ходе обоих конкурсов, объясняются методы, использованные участниками, и элементы их новизны, а также приводится анализ экспериментов, проведенных после конкурса, чтобы ответить на конкретные вопросы об эффективности методов поиска моделей.

10.5.1. Оценки, полученные в конкурсе 2015/2016

Конкурс AutoML 2015/2016 длился 18 месяцев (с 8 декабря 2014 г. по 1 мая 2016 г.). К его окончанию были получены практические решения, которые были выложены в открытый доступ, например решение победителей [25].

В табл. 10.5 представлены результаты на тестовом наборе на этапах AutoML (слепое тестирование) и Final (одноразовое тестирование на тестовом наборе, представленном в конце этапов Tweakathon). В случае ничьей приоритет получал участник, подавший заявку первым. В таблице представлены только результаты участников, занявших первые места. На рис. 10.3a мы также приводим сравнение результатов всех участников в таблице лидеров. На рис. 10.3a показаны результаты Tweakathon на финальном тестовом наборе в сравнении с результатами на проверочном наборе, что свидетельствует об отсутствии значительного переобучения на проверочном наборе, за исключением нескольких выбросов. На рис. 10.3b представлены результаты AutoML (слепое тестирование) и результаты финального теста Tweakathon (возможна ручная корректировка). Мы видим, что много заявок было сделано на этапе 1 (бинарная классификация), а затем количество участников снижалось по мере усложнения задач. Некоторые участники приложили много усилий на этапе Tweakathon и значительно превзошли свои показатели AutoML (например, Djajetic и AAD Freiburg).

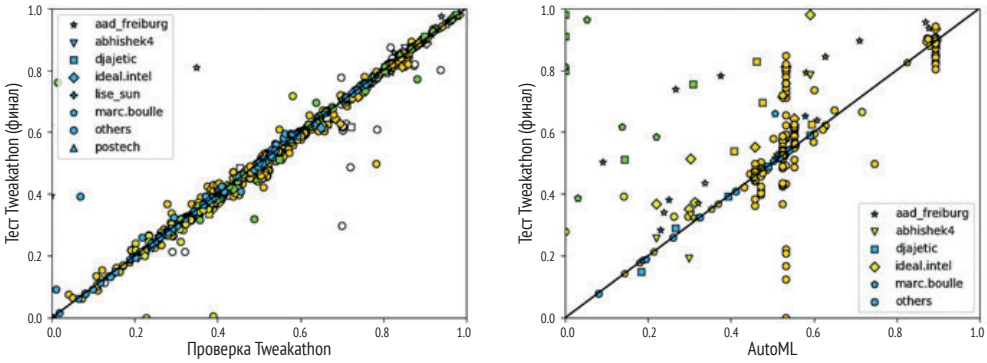
Остаются возможности для улучшения результатов за счет ручной настройки и/или дополнительных вычислительных ресурсов, о чем свидетельствуют значительные различия между результатами Tweakathon и AutoML (слепое тестирование) (табл. 10.5 и рис. 10.3b). В третьем раунде все участники, кроме одного, не смогли представить рабочие решения во время слепого тестиро-

вания, поскольку получили разреженные наборы данных, к которым не были готовы. К счастью, конкурсанты быстро сориентировались, и к концу испытания несколько участников смогло предоставить решения на всех наборах данных. Но схемы обучения все еще можно оптимизировать, поскольку, даже если отбросить 3-й раунд, существует разрыв в производительности между этапами AutoML (слепое тестирование с вычислительными ограничениями) и Tweakathon (вмешательство человека и дополнительные вычислительные мощности) на 15–35 %. Трек GPU предложил (только в 4-м раунде) платформу для опробования методов Deep Learning. Это позволило участникам продемонстрировать, что при наличии дополнительной вычислительной мощности методы глубокого обучения были конкурентоспособны с лучшими решениями из трека CPU. Однако ни один метод глубокого обучения не был конкурентоспособным при ограниченных вычислительных мощностях и бюджете времени, предложенных в треке CPU.

Таблица 10.5. Результаты победителей конкурса AutoML 2015/2016

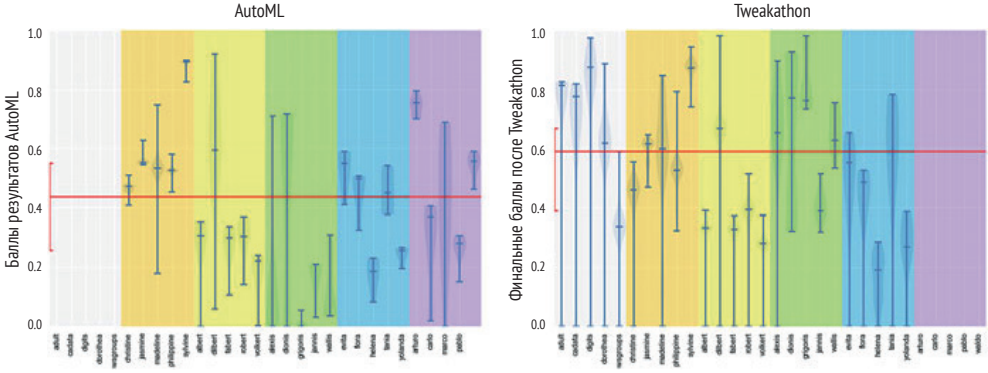
	AutoML				Final				UP, %
Раунд	**Оконч.**	**Победители**	**< *R* >**	**< *S* >**	**Оконч.**	**Победители**	**< *R* >**	**< *S* >**	
0	NA	NA	NA	NA	02/14/15	1. ideal	1.40	0.8159	NA
						2. abhi	3.60	0.7764	
						3. aad	4.00	0.7714	
1	02/15/15	1. aad	2.80	0.6401	06/14/15	1. aad	2.20	0.7479	15
		2. jrl44	3.80	0.6226		2. ideal	3.20	0.7324	
		3. tadej	4.20	0.6456		3. amsl	4.60	0.7158	
2	06/15/15	1. jrl44	1.80	0.4320	11/14/15	1. ideal	2.00	0.5180	35
		2. aad	3.40	0.3529		2. djaj	2.20	0.5142	
		3. mat	4.40	0.3449		3. aad	3.20	0.4977	
3	11/15/15	1. djaj	2.40	0.0901	02/19/16	1. aad	1.80	0.8071	481
		2. NA	NA	NA		2. djaj	2.00	0.7912	
		3. NA	NA	NA		3. ideal	3.80	0.7547	
4	02/20/16	1. aad	2.20	0.3881	05/1/16	1. aad	1.60	0.5238	31
		2. djaj	2.20	0.3841		2. ideal	3.60	0.4998	
		3. marc	2.60	0.3815		3. abhi	5.40	0.4911	
GPU	NA	NA	NA	NA	05/1/16	1. abhi	5.60	0.4913	NA
						2. djaj	6.20	0.4900	
						3. aad	6.20	0.4884	
5	05/1/16	1. aad	1.60	0.5282	NA	NA	NA	NA	NA
		2. djaj	2.60	0.5379					
		3. post	4.60	0.4150					

< *R* > – это средний ранг по всем пяти наборам данных раунда, который использовался для ранжирования участников. < *S* > – средний балл по пяти наборам данных раунда. UP – это прирост производительности в процентах между средней производительностью победителей на этапе AutoML и на финальном этапе того же раунда. В четвертом раунде использовался трек GPU. Названия команд сокращены следующим образом: aad – aad_freiburg, djaj – djajetic, marc – marc.boulle, tadej – tadejs, abhi – abhishek4, ideal – ideal.intel.analytics, mat – matthias.vonrohr, lisheng – lise_sun, asml – amsl.intel.com, jlr44 – backstreet.bayes, post – postech.mlg_exbrain, ref – reference.



(a) Наличие переобучения? (b) Разрыв AutoML/Tweakathon?

Рис. 10.3 ❖ Результаты всех участников в соревновании AutoML 2015/2016. Показаны последние достижения всех участников на всех этапах конкурса 2015/2016 на всех наборах данных из таблиц лидеров конкурса. Символы имеют цветовую маркировку по раундам, как в табл. 10.5. (a) **Переобучение в Tweakathon?** Мы построили график сравнения производительности на финальном тестовом наборе с производительностью на проверочном наборе. Результаты проверки были видны участникам на панели лидеров, пока они настраивали свои модели. Показатели итогового тестового набора были обнародованы только в конце Tweakathon. За исключением нескольких отклонений, большинство участников не превысило показатели в таблице лидеров. (b) **Разрыв между AutoML и Tweakathons?** Мы построили график соотношения производительности Tweakathons и AutoML, чтобы наглядно представить улучшения, полученные благодаря ручной настройке и дополнительным вычислительным ресурсам, доступным в Tweakathons. Точки выше диагонали указывают на такие улучшения



(a) AutoML (тестовый набор) (b) Tweakathon (тестовый набор)

Рис. 10.4 ❖ Распределение производительности на наборах данных задачи 2015/2016 («скрипичный график»). Для каждого набора данных показаны результаты участников в конце этапов AutoML и Tweakathon, выявленные в таблице лидеров. Медиана и квартили представлены горизонтальными отрезками. Профиль распределения (построенный ядерным методом) и его зеркальное отражение представлены по вертикали серой заштрихованной областью. Красным цветом показана медианная производительность по всем наборам данных и соответствующие квартили. (a) **AutoML** (слепое тестирование). Первые пять наборов данных были предоставлены только для целей разработки и не использовались для слепого тестирования на этапе AutoML. В третьем раунде код многих участников не прошел из-за вычислительных ограничений. (b) **Tweakathon** (ручная настройка). Последние пять наборов данных использовались только для финального слепого тестирования, а для Tweakathon данные не раскрывались. Раунд 3 не представлял особой сложности при наличии дополнительных вычислительных мощностей и памяти

10.5.2. Результаты, полученные в конкурсе 2018 года

Конкурс AutoML 2018 длился 4 месяца (с 30 ноября 2017 г. по 31 марта 2018 г.). Как и в предыдущем испытании, были получены и выложены в открытый доступ решения, занявшие первые места. В табл. 10.6 показаны результаты обоих этапов конкурса 2018 г. Напомним, что в этом испытании были этап обратной связи и этап слепого тестирования, и результаты победителей каждого этапа представлены в таблице.

Таблица 10.6. Результаты победителей конкурса AutoML 2018. Каждый этап выполнялся на пяти разных наборах данных. Показаны результаты победителей этапа слепого тестирования в сравнении с их производительностью на этапе обратной связи. Полная таблица доступна по адресу https://competitions.codalab.org/competitions/17767#results

2. Этап AutoML				**1. Этап обратной связи**			
Окончание	**Победители**	**< *R* >**	**< *S* >**	**Окончание**	**Производит.**	**< *R* >**	**< *S* >**
03/31/18	1 .aad_freiburg	2.80	0.4341	03/12/18	aad_freiburg	9.0	0.7422
	2. narnarsO	3.80	0.4180		narnarsO	4.40	0.7324
	3. wlWangl	5.40	0.3857		wlWangl	4.40	0.8029
	3. thanhdng	5.40	0.3874		thanhdng	14.0	0.6845
	3. Malik	5.40	0.3863		Malik	13.8	0.7116

Производительность в этом испытании была несколько ниже, чем в предыдущем. Это объясняется сложностью задач (см. ниже) и тем, что наборы данных на этапе обратной связи включали три обманчивых набора данных (связанных с задачами из предыдущих задач, но не обязательно похожих на наборы данных, использованные на этапе слепого тестирования) из пяти. Мы решили действовать таким образом, чтобы имитировать реалистичные условия AutoML. Хотя это было сложнее, нескольким командам удалось предоставить работы, показавшие результаты выше, чем случайные.

Победителем конкурса стала та же команда, которая победила в конкурсе AutoML 2015/2016, – AAD Freiburg [28]. Конкурс 2018 г. помог постепенно улучшить решение, разработанное этой командой в предыдущей задаче. Интересно, что команда, занявшая второе место, предложила решение, близкое по духу к решению команды-победителя. В этом конкурсе была тройная ничья за третье место, призы были разделены между командами, занявшими призовые места. Среди победителей две команды использовали стартовый набор. Большинство других команд использовало либо стартовый набор, либо решение, обнародованное командой AAD Freiburg в конкурсе AutoML 2015/2016.

10.5.3. Сложность наборов данных/задач

В этом разделе мы оцениваем сложность наборов данных или, скорее, сложность задач, поскольку участники должны были решать задачи прогнозиро-

вания для заданных наборов данных, метрик производительности и ограничений вычислительного времени. Конкурсные задачи содержали различные затруднения, но не в равной степени (табл. 10.2 и 10.3, а также рис. 10.5 и 10.6):

- **категориальные переменные и отсутствующие данные**. В конкурсе 2015/2016 г. лишь немногие наборы данных имели категориальные переменные (ADULT, ALBERT и WALDO), и в этих наборах не очень много переменных были категориальными. Аналогичным образом очень немногие наборы данных имели отсутствующие значения (ADULT и ALBERT), и в тех было лишь несколько отсутствующих значений. Таким образом, ни категориальные переменные, ни отсутствующие данные не представляли реальной трудности в этой задаче, хотя ALBERT оказался одним из самых сложных наборов данных, поскольку он также был одним из самых больших. Ситуация кардинально изменилась в задаче 2018 г., где пять из десяти наборов данных включали категориальные переменные (RL, PM, RI, RH и RM) и отсутствующие значения (GINA, PM, RL, RI и RM). Это были одни из основных аспектов, которые стали причиной низкой производительности большинства методов на этапе слепого тестирования;
- **большое количество классов**. Только один набор данных имел большое количество классов (DIONIS с 355 классами). Этот набор данных оказался сложным для участников, особенно потому, что он также большой и имеет несбалансированные классы. Однако наборы данных с большим количеством классов не очень хорошо представлены в этом испытании. Набор HELENA, занимающий второе место по количеству классов (100 классов), не выделялся как особенно сложный набор данных. Однако в целом многоклассовые задачи оказались более сложными, чем задачи бинарной классификации;
- **регрессия**. У нас было всего четыре регрессионные задачи: CADATA, FLORA, YOLANDA, PABLO;
- **разреженные данные**. Значительное количество наборов данных содержало мало данных (DOROTHEA, FABERT, ALEXIS, WALLIS, GRIGORIS, EVITA, FLORA, TANIA, ARTURO, MARCO). Некоторые из них оказались сложными, особенно ALEXIS, WALLIS и GRIGORIS, представляющие собой большие наборы данных в разреженном формате, что вызвало проблемы с памятью, когда они были представлены в 3-м раунде конкурса 2015/2016. Позже мы увеличили объем памяти на серверах, и аналогичные наборы данных, представленные на последующих этапах, вызывали меньше трудностей;
- **большие наборы данных**. Мы ожидали, что отношение числа N признаков к числу P_{tr} обучающих примеров будет представлять особую сложность (из-за риска переобучения), но современные алгоритмы машинного обучения устойчивы к переобучению. Основная трудность заключалась скорее в произведении N^*P_{tr}. Большинство участников пыталось загрузить весь набор данных в память и преобразовать разреженные матрицы в полные. Это занимало очень много времени и приводило к потере производительности или сбоям в работе программы. К большим наборам данных с $N^*P_{tr} > 20\times10^6$ относятся ALBERT,

ALEXIS, DIONIS, GRIGORIS, WALLIS, EVITA, FLORA, TANIA, MARCO, GINA, GUILLERMO, PM, RH, RI, RICCARDO, RM. Они существенно пересекаются с наборами, содержащими разреженные данные (выделены жирным шрифтом). В конкурсе 2018 г. все наборы данных на финальном этапе превысили этот порог, и это стало причиной того, что код от нескольких команд не уложился в отведенный бюджет времени. Только ALBERT и DIONIS были «по-настоящему» большими (мало признаков, но более 400 000 обучающих записей);

- **наличие закладок**. Некоторые наборы данных содержали определенную долю преднамеренно внесенных отвлекающих признаков или нерелевантных переменных (так называемых *закладок*). Они были получены путем случайной перестановки значений реальных признаков. Две трети наборов данных содержали закладки, а именно ADULT, CADATA, DIGITS, DOROTHEA, CHRISTINE, JASMINE, MADELINE, PHILIPPINE, SYLVINE, ALBERT, DILBERT, FABERT, JANNIS, EVITA, FLORA, YOLANDA, ARTURO, CARLO, PABLO, WALDO. Отчасти это позволило нам сделать наборы данных, находящиеся в открытом доступе, менее узнаваемыми;
- **тип метрики**. Мы использовали шесть метрик, описанных в разделе 10.4.2. Распределение задач, в которых они использовались, было неравномерным: BAC (11), AUC (6), F1 (3) и PAC (6) для классификации и R2 (2) и ABS (2) для регрессии. Это объясняется тем, что не все метрики подходят для всех типов приложений;
- **бюджет времени**. Хотя в раунде 0 мы экспериментировали с предоставлением различных бюджетов времени для разных наборов данных, в итоге мы назначили 1200 с (20 мин) для всех наборов данных во всех остальных раундах. Поскольку наборы данных различались по размеру, это наложило больше ограничений на большие наборы данных;
- **дисбаланс классов**. Эта трудность не встречалась в наборах данных 2015/2016 гг. Однако заметный дисбаланс классов стал основной трудностью для конкурса 2018 г. Дисбаланс меньше или равный 1 к 10 присутствовал в наборах данных RL, PM, RH, RI и RM, а в последнем наборе данных дисбаланс классов достигал 1 к 1000. Это стало причиной низкой результативности команд.

Рисунок 10.4 дает первое представление о сложности наборов данных/заданий для конкурса 2015/2016. На нем схематично показано распределение производительности участников во всех раундах на тестовых данных, как на этапе AutoML, так и на этапе Tweakathon. Видно, что медианная производительность по всем наборам данных улучшается между AutoML и Tweakathon, как и следовало ожидать. Соответственно, средний разброс в производительности уменьшается. Давайте более подробно рассмотрим этапы AutoML. Заметен «провал» 3-го раунда, в котором многие методы потерпели неудачу при слепом тестировании (введение разреженных матриц и больших наборов данных)[1]. Раунд 2 (многоклассовая классификация), похоже, также

[1] Примеры разреженных наборов данных были представлены в раунде 0, но они были меньшего размера.

оказался намного сложнее, чем раунд 1 (бинарная классификация). В 4-м раунде были введены две регрессионные задачи (FLORA и YOLANDA), но не похоже, что регрессия оказалась значительно сложнее, чем многоклассовая классификация. В 5-м раунде не было введено никаких новшеств. Мы можем заметить, что после 3-го раунда медианные оценки набора данных разбросаны вокруг общей медианы. Глядя на соответствующие оценки на этапах Tweakathon, можно заметить, что, как только участники оправились от удивления, раунд 3 не был для них особенно трудным. Раунды 2 и 4 были относительно более сложными.

Если речь идет о наборах данных, использованных в задаче 2018 г., сложность задач была явно связана с крайним дисбалансом классов, включением категориальных переменных и высокой размерностью $N \times P_{tr}$. Однако для наборов данных конкурса 2015/2016 гг. мы обнаружили, что в целом трудно предположить, что делает задачу легкой или трудной, за исключением размера набора данных, что подталкивало участников к границе возможностей оборудования и заставляло их улучшать вычислительную эффективность своих методов. Бинарные задачи классификации (и задачи с несколькими метками) по своей сути «легче», чем многоклассовые задачи, для которых успешное «угадывание» менее вероятно. Это частично объясняет более высокую медианную производительность в раундах 1 и 3, в которых преобладают задачи бинарной и многометочной классификации. Существует недостаточно большое количество наборов данных, иллюстрирующих каждый тип других трудностей, чтобы сделать другие выводы.

Тем не менее мы попытались найти сводную статистику, отражающую общую сложность задач. Если предположить, что данные получены в результате независимых и одинаково распределенных процессов[1]:

$$y = F(\mathbf{x}, noise),$$

где y – целевое значение, $\mathbf{x}$ – входной вектор признаков, F – функция, а *noise* – некоторый случайный шум, взятый из неизвестного распределения, то сложность задачи обучения можно разделить на два аспекта.

1. **Внутренняя сложность**, связанная с количеством шума или отношением сигнал/шум. Если у нас есть бесконечное количество данных и несмещенная обучаемая машина $\hat{F}$, способная определить F, эффективность предсказания не может превышать заданного максимального значения, соответствующего $\hat{F} = F$.
2. **Сложность моделирования**, связанная со смещением и дисперсией оценок $\hat{F}$ в связи с ограниченным количеством обучающих данных и ограниченными вычислительными ресурсами, а также с возможным большим количеством параметров и гиперпараметров для оценки.

Оценить внутреннюю сложность невозможно, если мы не знаем F. Наше лучшее приближение к F – это решение победителей. Поэтому мы используем результаты победителей в качестве оценки наилучшей достижимой производительности. Эта оценка может иметь как смещение, так и дисперсию:

[1] Точнее, независимо и одинаково распределенные выборки процессов.

возможно, она смещена, потому что победители могут недостаточно точно обучить модель; возможно, она имеет дисперсию из-за ограниченного количества тестовых данных. Наличие недообучения трудно проверить. Его признаки могут заключаться в том, что дисперсия или энтропия предсказаний меньше, чем целевые значения.

Оценить сложность моделирования также невозможно, если мы не знаем *F* и класс модели. В отсутствие знаний о классе модели ученые, изучающие данные, часто используют общие модели прогнозирования, не зависящие от процесса генерирования данных. Такие модели варьируются от очень простых моделей, которые сильно смещены в сторону «простоты» и гладкости прогнозов (например, регуляризованные линейные модели), до очень универсальных несмещенных моделей, которые могут выучить любую функцию при достаточном количестве данных (например, ансамбли деревьев решений). Чтобы косвенно оценить сложность моделирования, мы использовали разницу в производительности между методом победителя конкурса и a) лучшей из четырех ненастроенных базовых моделей (взятых из классических методов, представленных в библиотеке scikit-learn [55] с гиперпараметрами по умолчанию) или b) Selective Naive Bayes (SNB) [12, 13] – высоко регуляризованной модели (с уклоном в простоту), обеспечивающей очень надежное и простое исходное решение.

На рис. 10.5 и 10.6 представлены наши оценки внутренней сложности и сложности моделирования для наборов данных 2015/2016 гг. Видно, что наборы данных раунда 0 были одними из самых простых (за исключением, пожалуй, NEWSGROUP). Это были относительно небольшие (и хорошо известные) наборы данных. Удивительно, но наборы данных 3-го раунда также оказались довольно легкими, несмотря на то что большинство участников не справилось с ними (в основном из-за ограничений памяти: алгоритмы scikit-learn не были оптимизированы для разреженных наборов данных, и было невозможно уместить в памяти матрицу данных, преобразованную в плотную матрицу). Два набора данных имеют небольшую внутреннюю сложность, но большую сложность моделирования: MADELINE и DILBERT. MADELINE – это искусственный набор данных, который является очень нелинейным (кластеры или два класса расположены на вершинах гиперкуба в 5-мерном пространстве) и поэтому очень сложен для наивного байесовского алгоритма. DILBERT – это набор данных для распознавания изображений объектов, повернутых во всевозможных положениях, также очень сложный для наивного байесовского алгоритма. Наборы данных последних двух этапов, кажется, имеют большую внутреннюю сложность по сравнению со сложностью моделирования. Но это может быть обманчивое впечатление, поскольку наборы данных оказались относительно новыми для участников конкурса, и результаты победителей могут быть еще далеки от наилучших достижимых показателей.

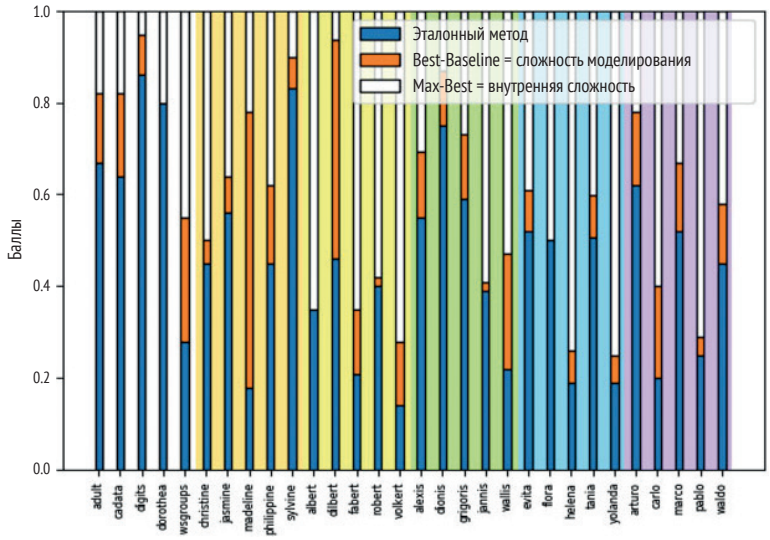


Рис. 10.5 ❖ Сложность задач в конкурсе AutoML 2015/2016. Мы рассматриваем два показателя сложности задачи: внутренняя сложность (оценивается по производительности победителей) и сложность моделирования (разница между производительностью победителя и базового метода, здесь Selective Naive Bayes (SNB)). Лучшие задачи должны иметь относительно низкую внутреннюю сложность и высокую сложность моделирования, чтобы хорошо разделить участников

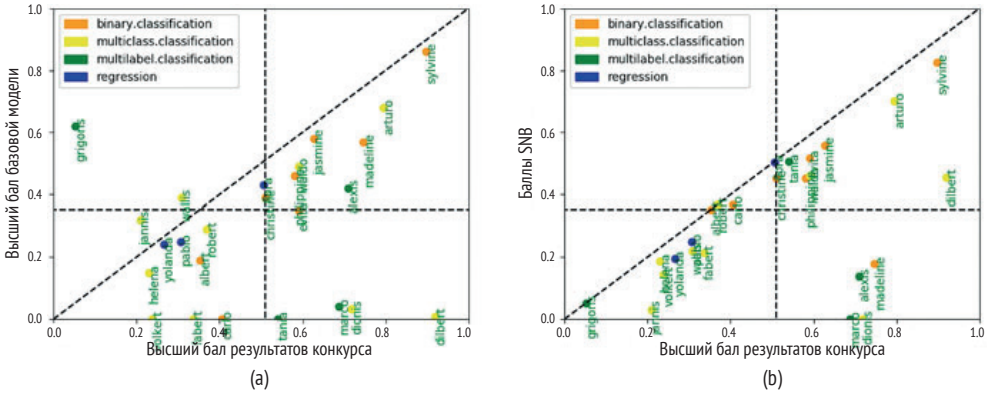


Рис. 10.6 ❖ Сложность моделирования в сравнении с внутренней сложностью. Для этапов AutoML 2015/2016 гг. мы построили график зависимости показателя сложности моделирования от показателя внутренней сложности наборов данных (наивысший балл в таблице лидеров). (a) Сложность моделирования оценивается по баллам лучшей ненастроенной модели (по KNN, NaiveBayes, RandomForest и SGD (LINEAR)). (b) Сложность моделирования оценивается по баллам стандартной модели Selective Naive Bayes (SNB). Во всех случаях более высокие оценки лучше, а отрицательные оценки и NaN заменяются нулем. Горизонтальные и вертикальные разделительные линии представляют медианы. В правом нижнем квадранте представлены наборы данных с низкой внутренней сложностью и высокой сложностью моделирования: это лучшие наборы данных для целей бенчмаркинга

Мы попытались предсказать внутреннюю сложность набора по метапризнакам, используемым AAD Freiburg для метаобучения [25], которое является частью OpenML [67], с помощью классификатора Random Forest и ранжировали метапризнаки в порядке важности (наиболее часто выбираемые случайным лесом). Список метапризнаков представлен в онлайн-приложении. Следующие три метапризнака лучше всего предсказывают сложность набора данных (рис. 10.7):

- LandmarkDecisionTree: производительность классификатора на основе дерева решений;
- Landmark1NN: производительность классификатора на основе метода *k*-ближайших соседей;
- SkewnessMin: минимальное значение *перекоса* (skewness) всех признаков. Перекос служит мерой симметрии распределения. Положительное значение перекоса означает, что в левом хвосте распределения больше веса.

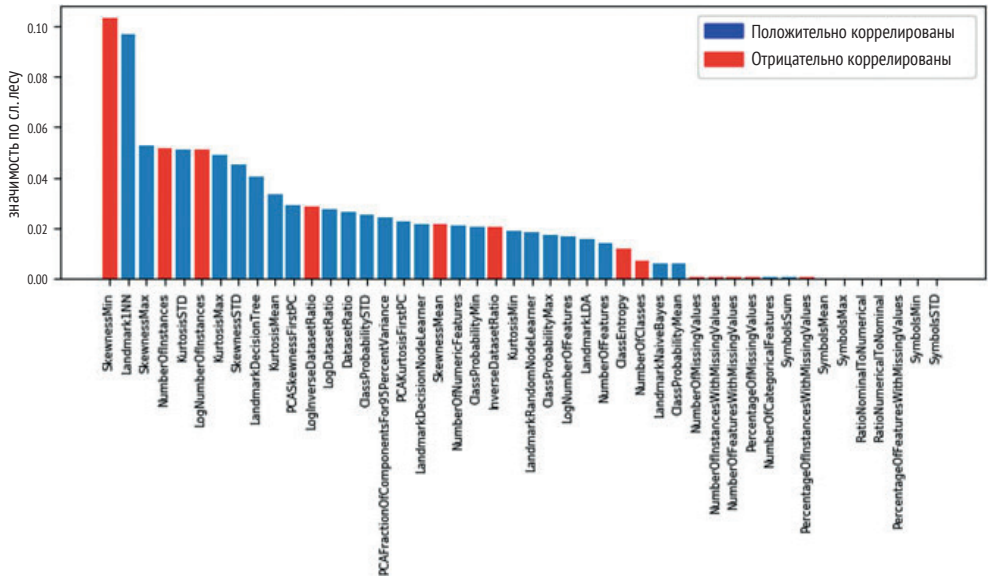


Рис. 10.7 ❖ Метапризнаки, наиболее точно предсказывающие внутреннюю сложность набора данных (на примере данных конкурса AutoML 2015/2016 гг.). Значимости метапризнаков GINI вычисляются регрессором случайного леса, обученным прогнозировать наивысший балл участника в таблице лидеров с использованием метапризнаков наборов данных. Описание этих метапризнаков можно найти в табл. 1 дополнительного материала [25]. Синий и красный цвета соответствуют положительным и отрицательным корреляциям (корреляции Пирсона между метапризнаками и медианами оценок)

10.5.4. Оптимизация гиперпараметров

Многие участники использовали пакет scikit-learn (sklearn), включая победившую группу AAD Freiburg, которая разработала программное обеспече-

ние auto-sklearn. Мы использовали API auto-sklearn для проведения систематических исследований эффективности оптимизации гиперпараметров. Мы сравнили результаты, полученные с настройками гиперпараметров по умолчанию в scikit-learn и с гиперпараметрами, оптимизированными с помощью auto-sklearn[1], в рамках бюджета времени, установленного в ходе конкурса, для четырех репрезентативных базовых методов: *k*-ближайших соседей (KNN), наивного байесовского алгоритма (NB), случайного леса (RF) и линейной модели, обученной с помощью стохастического градиентного спуска (SGD-linear)[2]. Результаты показаны на рис. 10.8. Мы видим, что оптимизация гиперпараметров обычно улучшает производительность, но не всегда. Преимущество настройки гиперпараметров в основном связано с гибкостью переключения метрики оптимизации на ту, которая назначена в конкурсе, и с поиском гиперпараметров, которые хорошо работают с учетом текущего набора данных и метрики. Однако в некоторых случаях из-за размера набора данных (score ⩽ 0) не удалось провести оптимизацию гиперпараметров в рамках бюджета времени. Следовательно, предстоит дальнейшая работа по тщательной настройке гиперпараметров в условиях жестких временных ограничений и огромных наборов данных (рис. 10.8).

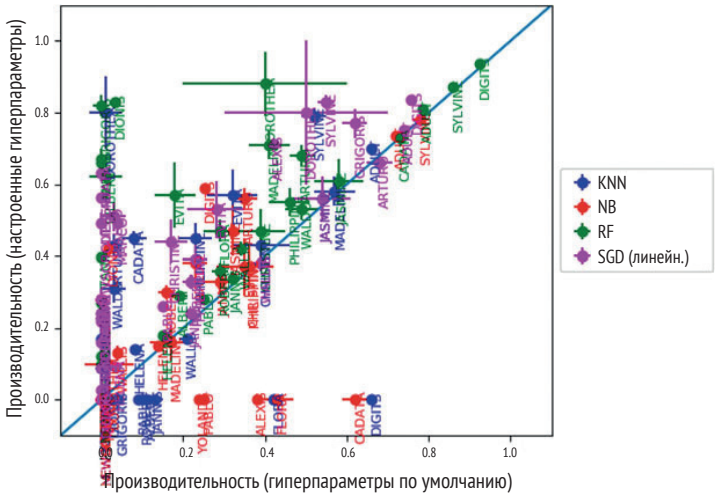


Рис. 10.8 ❖ Настройка гиперпараметров (данные конкурса AutoML 2015/2016). Мы сравниваем показатели, полученные с гиперпараметрами по умолчанию, и показатели с гиперпараметрами, оптимизированными с помощью auto-sklearn, в рамках тех же временных бюджетов, которые были заданы во время конкурса. Показатели предикторов, которые не дали результатов за отведенное время, заменяются нулем

[1] Мы используем sklearn 0.16.1 и auto-sklearn 0.4.0, чтобы имитировать условия задачи.

[2] В scikit-learn для этих экспериментов мы установили потерю SGD равной ‘log’.

Мы также сравнили результаты, полученные с помощью различных метрик оценки (рис. 10.9). Базовые методы не дают выбора метрик для оптимизации, но auto-sklearn дообучается на метриках задач конкурса. Следовательно, когда используются «общие метрики» (*BAC* и R^2), метод победителей конкурса, не оптимизированный для *BAC*/R^2, обычно не превосходит базовые методы. И наоборот, когда используются метрики конкурса, часто наблюдается явный разрыв между базовыми методами и победителями, но не всегда (RF-auto обычно показывает сравнимую производительность, иногда даже превосходит победителей).

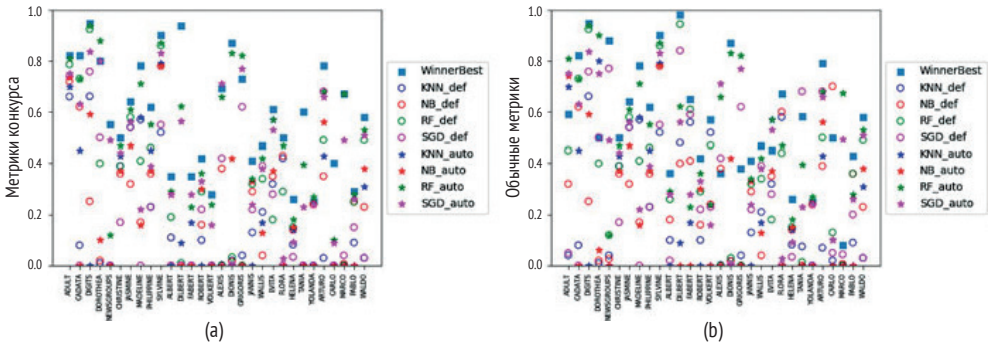


Рис. 10.9 ❖ Сравнение метрик (конкурс AutoML 2015/2016). (a) Использованы метрики конкурса. (b) Использована нормализованная сбалансированная точность для всех задач классификации и метрика R^2 для задач регрессии. Сравнивая два рисунка, мы видим, что победитель остается на первом месте в большинстве случаев независимо от метрики. Не существует базового метода, который бы доминировал над всеми остальными. Хотя RF-auto (случайный лес с оптимизированным HP) очень силен, иногда его опережают другие методы. Простая линейная модель SGD_def иногда выигрывает, когда используются общие метрики, но победители лучше работают с метриками конкурса. В целом техника победителей оказалась эффективной

10.5.5. Метаобучение

Один из принципиальных вопросов – возможно ли метаобучение [14], т. е. можно ли научить модель предсказывать, будет ли данный классификатор хорошо работать на будущих наборах данных (без фактического обучения), основываясь только на его прошлых результатах на других наборах данных. Мы исследовали, можно ли предсказать, какой базовый метод будет работать лучше, основываясь на метапризнаках auto-sklearn (см. онлайн-приложение). Мы удалили из набора метапризнаков признаки Landmark, поскольку они являются показателями базовых предикторов (хотя и довольно плохих, с большим количеством отсутствующих значений), что привело бы к «утечке данных».

Мы использовали четыре базовых предиктора:

- NB: наивный байесовский;

- линейный SGD: линейная модель (обученная с помощью стохастического градиентного спуска);
- KNN: *k*-ближайшие соседи;
- RF: случайный лес.

Мы использовали также реализацию библиотеки scikit-learn с настройками гиперпараметров по умолчанию. На рис. 10.10 показаны два первых компонента *линейного дискриминантного анализа* (linear discriminant analysis, LDA) при обучении классификатора LDA на метапризнаках, чтобы предсказать, какой базовый классификатор будет работать лучше. Методы разделяются на три отдельных кластера, один из которых объединяет нелинейные методы, которые плохо разделяются (KNN и RF), а два других – NB и линейный SGD.

Наиболее значимыми в прогностическом отношении признаками являются ClassProbability и PercentageOfMissingValues. Это говорит о том, что дисбаланс классов и/или большое количество классов (в многоклассовой задаче) и процент пропущенных значений могут быть важны, но существует высокая вероятность переобучения, о чем свидетельствует неустойчивое ранжирование лучших признаков при повторной выборке обучающих данных.

10.5.6. Методы, использованные в конкурсах

Краткое описание методов, использованных в обоих конкурсах, представлено в онлайн-приложении, а также в результатах опроса о методах, который мы провели после решения задач конкурсантами. В свете обзора из раздела 10.2 и результатов, представленных в предыдущем разделе, может возникнуть вопрос, появилась ли доминирующая методология для решения задачи AutoML и применялись ли конкретные общепринятые технические решения. В этом разделе мы называем *пространством моделей* множество всех рассматриваемых моделей. Мы называем *базовыми моделями* (также называемыми в других местах «простыми моделями», «индивидуальными моделями», «базовыми учениками») компоненты библиотеки моделей, из которых строятся наши гипермодели ансамблей моделей.

Ансамблирование: борьба с переобучением и сквозное обучение

Ансамблирование – главный победитель серии конкурсов AutoML, поскольку его используют более 80 % от общего числа участников и все участники, занявшие призовые места. Если несколько лет назад самым острым вопросом при выборе моделей и оптимизации гиперпараметров была борьба с переобучением, то в настоящее время, похоже, этой проблемы удалось избежать благодаря использованию методов ансамблирования. В конкурсе AutoML 2015/2016 мы варьировали отношение числа обучающих примеров к числу переменных (P_{tr}/N) на несколько порядков. В пяти наборах данных отношение P_{tr}/N было меньше единицы (dorothea, newsgroup, grigoris, wallis и flora), что представляет собой случаи, особенно склонные к переобучению. Хотя P_{tr}/N является наиболее предсказуемой переменной для медианной производительности участников, нет никаких признаков того, что наборы

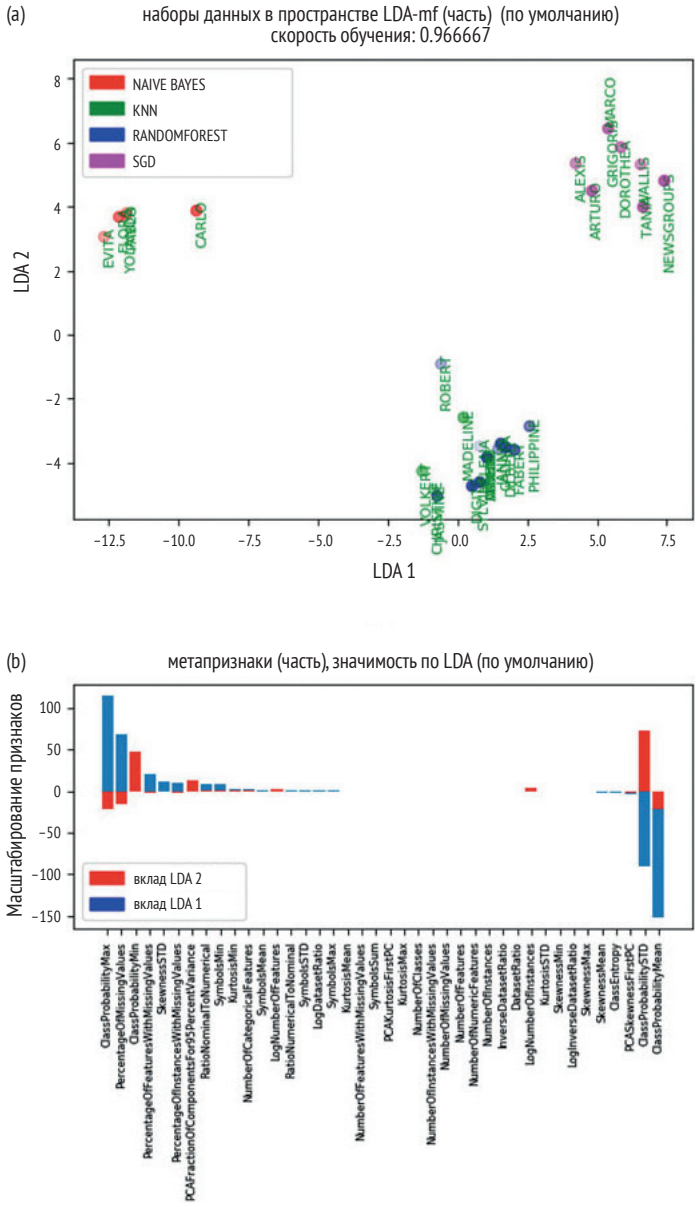


Рис. 10.10 ❖ Линейный дискриминантный анализ. (a) Диаграмма разброса данных по главным осям. Мы обучили LDA, используя X = метапризнаки, кроме ориентиров; y = какая модель победила из четырех базовых моделей (NB, линейная SGD, KNN, RF). Производительность базовых моделей измеряется с помощью общепринятых метрик. Модели были обучены с гиперпараметрами по умолчанию. В пространстве двух первых компонент LDA каждая точка представляет один набор данных. Цвета обозначают победившие базовые модели. (b) Значимость метапризнаков, вычисленная как масштабные коэффициенты каждого компонента LDA

данных с $P_{tr}/N < 1$ оказались для них особенно трудными (рис. 10.5). Ансамбли предикторов имеют дополнительное преимущество, заключающееся в простом решении проблемы переобучения путем постепенного увеличения ансамбля предикторов, улучшая производительность с течением времени. В ансамбль обычно включаются все обученные предикторы. Например, если используется перекрестная проверка, предикторы всех подвыборок непосредственно включаются в ансамбль, что экономит вычислительное время на повторное обучение одной модели на лучших отобранных HP и может дать более надежные решения (хотя и немного более смещенные из-за меньшего размера выборки). Подходы различаются тем, как они взвешивают вклад различных предикторов. Некоторые методы используют один и тот же вес для всех предикторов (это относится к методам бэггинга, таким как случайный лес, и к байесовским методам, которые выбирают предикторы в соответствии с их апостериорной вероятностью в пространстве моделей). Некоторые методы оценивают веса предикторов в процессе обучения (это относится, например, к методам бустинга). Один из простых и эффективных методов создания ансамблей разнородных моделей был предложен в [16]. Он был успешно использован в нескольких прошлых конкурсах, например [52], и именно этот метод был реализован командой aad_freibug, одним из сильнейших участников обоих конкурсов [25]. Метод состоит в том, чтобы несколько раз перебрать все обученные модели и включить в ансамбль в каждом цикле ту модель, которая наиболее улучшает производительность ансамбля. Модели голосуют с весом 1, но они могут быть включены несколько раз, что де-факто приводит к их взвешиванию. Этот метод позволяет очень быстро пересчитать веса моделей, если сохраняются предсказания, проверенные перекрестным путем. Более того, данный метод позволяет оптимизировать ансамбль для любой метрики путем подгонки прогнозов ансамбля к желаемой метрике (аспект, который был важен в данной задаче).

Оценка моделей: перекрестная или простая проверка

Оценка точности прогнозов, выдаваемых моделями, является критически важным и необходимым элементом любого метода выбора модели для ансамбля. Критерии выбора модели, вычисляемые на основе прогностической точности базовых моделей, оцененных по обучающим данным (путем однократного обучения на всех обучающих данных и, возможно, за счет незначительных дополнительных вычислений), такие как границы производительности, не использовались вообще, как это уже было в предыдущих организованных нами конкурсах [35]. Широко использовалась перекрестная проверка, особенно k-кратная. Однако для базовых моделей часто применяли «дешевое» оценивание только на одной подвыборке, чтобы быстро отбросить неперспективные области пространства моделей. Эта техника все чаще используется для ускорения поиска. Еще одна стратегия ускорения – обучение на подмножестве обучающих примеров и наблюдение за кривой обучения. Стратегия «замораживания-размораживания» [64] приостанавливает обучение моделей, которые не выглядят перспективными на основе кривой обучения, но может возобновить их обучение в более поздний момент. Эта стратегия была использована, например, в [48] в конурсе AutoML 2015/2016.

Пространство моделей: однородное или неоднородное

Нерешенным остается вопрос о том, следует ли искать в большом или малом пространстве моделей. Итоги конкурса не позволили нам дать однозначный ответ на этот вопрос. Большинство участников предпочло поиск в относительно большом пространстве, включая широкий спектр моделей, найденных в библиотеке scikit-learn. Тем не менее один из самых сильных участников (команда Intel) представил результаты, полученные просто с помощью расширенного дерева решений (т. е. состоящего из однородного набора слабых обучателей / базовых моделей). Очевидно, что достаточно использовать всего один подход машинного обучения, который является универсальным аппроксиматором, чтобы иметь возможность научиться чему угодно, обладая достаточным количеством обучающих данных. Так зачем же использовать несколько? Это вопрос *скорости сходимости* – как быстро мы поднимаемся по кривой обучения. Включение в ансамбль более сильных базовых моделей служит одним из способов быстрее подняться по кривой обучения. Наши эксперименты после конкурса (рис. 10.9) показывают, что версия случайного леса от scikit-learn (ансамбль однородных базовых моделей-деревьев решений) обычно работает не так хорошо, как версия победителей, что намекает на то, что в решении Intel, которое также основано на ансамблях деревьев решений, есть много ноу-хау, которые не присущи базовому ансамблю деревьев решений, такому как случайный лес. Мы надеемся, что в будущем будут проведены более углубленные исследования на эту тему.

Стратегии поиска: фильтр, обертка и встроенные методы

При наличии мощных наборов инструментов машинного обучения, таких как scikit-learn (на котором был основан стартовый набор), велик соблазн реализовать методы «полной обертки» для решения задачи CASH (или задачи полного выбора модели). Действительно большинство участников пошло по этому пути. Хотя было опубликовано несколько способов оптимизации гиперпараметров с помощью встроенных методов для нескольких базовых классификаторов [35], каждый из них требует изменения реализации базовых методов, что отнимает много времени и чревато ошибками по сравнению с использованием уже отлаженной и хорошо оптимизированной библиотечной версии методов. Поэтому прикладные специалисты неохотно тратят время на разработку альтернативных реализаций встроенных методов. Заметным исключением является программное обеспечение marc.boulle, которое предлагает автономное решение без гиперпараметров на основе наивного байесовского подхода, включающее перекодировку переменных (группировка или дискретизация) и выбор переменных.

Многоуровневая оптимизация

Другой интересный вопрос заключается в том, следует ли рассматривать несколько уровней гиперпараметров по соображениям вычислительной эффективности или во избежание преобучения. В байесовской схеме, например, вполне реально рассмотреть иерархию параметров/гиперпараметров и несколько уровней априорных/гипераприорных распределений. Однако

похоже, что по практическим соображениям в задачах AutoML участники используют неглубокую организацию пространства гиперпараметров и избегают вложенных циклов перекрестной проверки.

Управление временем: компромисс между исследованием и использованием

В условиях ограниченного бюджета времени необходимо внедрять эффективные стратегии поиска, чтобы обеспечить компромисс между исследованием и использованием. Для сравнения стратегий мы приводим в онлайн-приложении кривые обучения двух участников, занявших первые места в рейтинге, которые использовали совершенно разные методы: Abhishek и aad_freiburg. Первая команда использует эвристические методы, основанные на предыдущем экспертном опыте человека, а вторая инициализирует поиск с моделями, предсказанными как наиболее подходящие с помощью метаобучения, а затем выполняет байесовскую оптимизацию гиперпараметров. Abhishek, похоже, часто начинает с лучшего решения, но ищет менее эффективно. Напротив, aad_freiburg начинает с более низкого уровня, но часто заканчивает лучшим решением. Некоторый элемент случайности в поиске полезен для получения лучших решений.

Предварительная обработка и выбор признаков

Наборы данных имели внутренние недостатки, которые частично можно было устранить с помощью предварительной обработки или специальных модификаций алгоритмов: разреженность, пропущенные значения, категориальные и нерелевантные переменные. Тем не менее похоже, что среди участников, занявших первые места, предварительная обработка не стояла в центре внимания. Они полагались на простые эвристики, предоставленные в стартовом наборе: замену отсутствующих значений медианой и добавление переменной-индикатора отсутствия, а также позиционное кодирование категориальных переменных. Использовалась простая нормализация. Нерелевантные переменные проигнорировали 2/3 участников, а участники, занявшие первые места, не использовали отбор признаков. Используемые методы, включающие ансамблирование, по-видимому, внутренне устойчивы к нерелевантным переменным. Более подробную информацию из информационных бюллетеней можно найти в онлайновом приложении.

Обучение без учителя

Несмотря на большой интерес к обучению без учителя, вызванный успехами сообщества исследователей глубокого обучения, в серии конкурсов AutoML обучение без учителя не нашло широкого применения, за исключением использования классических методов уменьшения размерности пространства, таких как ICA и PCA. Более подробную информацию см. в онлайн-приложении.

Трансферное обучение и метаобучение

Насколько нам известно, только команда aad_freiburg использовала метаобучение для инициализации поиска гиперпараметров. Для этого она

использовала наборы данных OpenML[1]. Количество доступных наборов данных и разнообразие задач не позволили участникам провести эффективное трансферное обучение или метаобучение.

Глубокое обучение

Тип вычислительных ресурсов, доступных на этапах AutoML, исключал использование глубокого обучения, за исключением трека GPU. Однако даже в этом треке методы глубокого обучения не вырвались вперед. Исключением является команда aad_freiburg, которая использовала глубокое обучение в третьем и четвертом раундах Tweakathon и нашла его его полезным для наборов данных Alexis, Tania и Yolanda.

Оптимизация задач и метрик

Было представлено четыре типа задач (регрессия, бинарная классификация, многоклассовая классификация и классификация по нескольким меткам) и шесть метрик оценки (R^2, ABS, BAC, AUC, F1 и PAC). Более того, баланс классов и их количество сильно варьировались в зависимости от задачи. К разработке методов, оптимизирующих конкретные метрики, было приложено мало усилий. Чаще использовались общие методы, а обучение под целевые метрики выполняли путем перекрестной проверки или с помощью ансамблирования.

Инженерный подход к результатам

Один из главных уроков серии конкурсов AutoML заключается в том, что большинство методов не может гарантированно вернуть результат во всех случаях – не обязательно даже «хороший», а хоть какой-то разумный результат. Основными причинами неудач являются нехватка времени и/или памяти или различные другие сбои (например, численная неустойчивость). Мы еще очень далеки от того, чтобы иметь базовые модели, которые работают на всех наборах данных. Одна из сильных сторон автоматизированного обучения заключается в том, что оно игнорирует те модели, которые не работают, и обычно находит хотя бы одну, которая дает результат.

Параллелизм

Доступные компьютеры имели несколько ядер, поэтому, в принципе, участники могли использовать параллелизм. Одной из распространенных стратегий было просто полагаться на численные библиотеки, которые внутри автоматически используют такой параллелизм. Команда aad_freiburg использовала разные ядра для параллельного поиска моделей для разных наборов данных (поскольку каждый раунд включал пять наборов данных). Этот подход к использованию вычислительных ресурсов проявляется на кривых обучения (см. онлайн-приложение).

[1] https://www.openml.org/.

10.6. Обсуждение

Здесь мы кратко суммируем основные вопросы, которые мы задавали себе, и основные выводы, которые сделали.

1. **Достаточно ли предоставленного бюджета времени для выполнения задач конкурса?** Мы построили кривые обучения в зависимости от времени для победившего решения aad_freiburg (auto-sklearn, см. онлайн-приложение). Это показало, что для большинства наборов данных производительность продолжала улучшаться далеко за пределами лимита времени, установленного организаторами. Хотя примерно для половины наборов данных улучшение скромное (не более 20 % от оценки, полученной в конце установленного лимита времени), для некоторых наборов данных улучшение было очень большим (более двух баллов от первоначальной оценки). Улучшения обычно происходят постепенно, но случаются и внезапные улучшения производительности. Например, для набора Wallis результат внезапно удвоился на третий минуте времени, установленного для конкурса. Авторы пакета auto-sklearn [25] тоже отмечают, что он медленно стартует, но в долгосрочной перспективе достигает показателей, близких к лучшему методу.
2. **Есть ли задачи, которые оказались значительно сложнее других для участников?** Да, диапазон сложности заданий был очень широк, о чем свидетельствует разброс участников по среднему (медиана) и вариативности (третий квартиль) их оценок. Синтетический набор данных Madeline с очень нелинейной задачей оказался очень трудным для многих участников. Среди других трудностей, которые привели к невозможности найти решение, были большие требования к памяти (особенно для методов, которые пытались преобразовать разреженные матрицы в полные) и короткие бюджеты времени для наборов данных с большим количеством обучающих экземпляров и/или признаков или с большим количеством классов или меток.
3. **Существуют ли метапризнаки наборов данных и методов, позволяющие рекомендовать определенные методы для определенных типов наборов данных?** Команда aad_freiburg использовала подмножество из 53 метапризнаков (надмножество простых статистических данных, предоставленных вместе с наборами данных задачи) для измерения сходства между наборами данных. Это позволило им более эффективно проводить поиск гиперпараметров, инициализируя поиск с настройками, идентичными тем, которые были выбраны для аналогичных наборов данных, обработанных ранее (разновидность метаобучения). Наш собственный анализ показал, что очень сложно предсказать эффективность предикторов по метапризнакам, но можно относительно точно предсказать, какой базовый метод будет работать лучше. С помощью LDA мы смогли визуализировать, как наборы данных восстанавливаются в двух измерениях, и показать четкое разделение между наборами данных, «предпочитающими» наивный байесовский подход, линейный SGD, метод *k*-ближайших соседей или случайный лес. Этот вопрос заслуживает дальнейшего изучения.

4. **Действительно ли оптимизация гиперпараметров улучшает производительность по сравнению с использованием значений по умолчанию?** Проведенное нами сравнение показывает, что оптимизация гиперпараметров вместо выбора значений по умолчанию для набора из четырех основных прогностических моделей (*k*-ближайшие соседи, случайный лес, линейный SGD и наивный байесовский подход) в целом дает положительный результат. В большинстве случаев (но не всегда) оптимизация гиперпараметров (hyper-opt) приводит к лучшим показателям, чем значения по умолчанию. Иногда hyper-opt не срабатывает из-за ограничений по времени или памяти, что оставляет возможность для улучшения.
5. **Как решения победителя выглядят в сравнении с базовыми моделями scikit-learn?** Они выглядят достаточно успешными. Например, результаты базовых моделей, параметры которых были оптимизированы, в целом не дают таких хороших результатов, как auto-sklearn. Однако базовая модель с параметрами по умолчанию иногда превосходит эту же модель, настроенную с помощью auto-sklearn.

10.7. Заключение

Мы проанализировали результаты нескольких раундов конкурса AutoML.

Наш проект первого конкурса AutoML (2015/2016) был удовлетворительным во многих отношениях. В частности, мы привлекли большое количество участников (более 600), получили результаты, которые являются статистически значимыми, и повысили уровень развития автоматизации машинного обучения. В результате проделанной работы появились общедоступные библиотеки, в том числе auto-sklearn.

В частности, мы разработали бенчмарк с большим количеством разнообразных наборов данных, с достаточно большими тестовыми наборами, чтобы выявить участников, занявших призовые места. Трудно предугадать размер необходимых тестовых наборов, поскольку планки ошибок зависят от результатов, достигнутых участниками, поэтому мы довольны тем, что сделали разумные предположения. Наше простое правило $N = 50/E$, где N – количество тестовых экземпляров, а E – коэффициент ошибок наименьшего класса, зарекомендовало себя как широко применимое на практике. Мы убедились, что наборы данных не были ни слишком легкими, ни слишком трудными. Это важно для того, чтобы иметь возможность разделить участников по рейтингу. Чтобы получить количественные оценки рейтинга, мы ввели понятия «внутренняя сложность» и «сложность моделирования». Внутренняя сложность может быть оценена по производительности лучшего метода (как суррогат наилучшей достижимой производительности, т. е. коэффициент Байеса для задач классификации). Сложность моделирования можно определить по разбросу в производительности между методами. Наши лучшие задачи имеют относительно низкую внутреннюю сложность и высокую сложность моделирования. Однако разнообразие 30 наборов данных наше-

го первого конкурса 2015/2016 гг. является одновременно и достоинством, и проклятием: оно позволяет нам проверить устойчивость программного обеспечения в различных ситуациях, но делает метаобучение очень сложным, если не невозможным. Следовательно, для метаобучения необходимо использовать внешние (внеконкурсные) данные. Именно такой стратегии придерживалась команда AAD Freiburg, которая использовала данные OpenML для метаобучения. Аналогичным образом к каждому набору данных мы приложили различные метрики. Это способствовало тому, что задачи стали более реалистичными и сложными, но также усложнило метаобучение. Во втором конкурсе 2018 г. мы уменьшили разнообразие наборов данных и использовали единую метрику.

Что касается построения конкурса, мы убедились, что дьявол кроется в мелочах. Участники конкурса настолько концентрируются на предложенных задачах, что их решение может оказаться неадаптируемым к, казалось бы, похожим сценариям. В случае с конкурсом AutoML мы размышляли над тем, должна ли метрикой задачи быть площадь под кривой обучения или одна точка на кривой обучения (производительность, полученная после истечения фиксированного максимального времени вычислений). В итоге мы предпочли второе решение из практических соображений. Изучив после испытания кривые обучения некоторых участников, мы поняли, что эти два подхода радикально отличаются, особенно в отношении стратегий, смягчающих компромисс между исследованием и использованием. Это заставило нас задуматься о различиях между обучением «с фиксированным временем» (участники заранее знают лимит времени и оцениваются только по решению, полученному по истечении этого времени) и «обучением в неопределенное время» (участников могут остановить в любой момент и попросить вернуть решение). Оба сценария полезны: первый – когда модели должны предоставляться непрерывно в быстром темпе, например для маркетинговых приложений; второй – в условиях, когда вычислительные ресурсы ненадежны и возможны перерывы (например, люди, работающие удаленно через ненадежное соединение). Это понимание будет учтено при организации следующих конкурсов.

Две версии конкурса AutoML, который мы проводили, различаются по сложности трансферного обучения. В конкурсе 2015/2016 гг. в раунде 0 была представлена выборка всех типов данных и трудностей (типы целей, разреженные данные или нет, отсутствующие данные или нет, категориальные переменные или нет, больше записей, чем признаков или нет). Затем в каждом раунде сложность повышалась. Наборы данных раунда 0 были относительно легкими. Затем в каждом раунде код участников проверялся вслепую на данных, которые были на одну ступень сложнее, чем в предыдущем раунде. Таким образом, перенос знаний от этапа к этапу был существенно усложнен. В конкурсе 2018 г. у нас было два этапа, каждый из которых состоял из пяти наборов данных схожей сложности, а наборы данных первого этапа были сопоставлены с одним соответствующим набором данных по схожей задаче. В результате перенос знаний был упрощен.

Что касается стартового набора и базовых методов, мы предоставили код, который в итоге стал основой решения большинства участников (за замет-

ными исключениями из промышленности, такими как Intel и Orange, которые использовали свои собственные пакеты «in house»). Таким образом, мы можем задаться вопросом о том, является ли предоставленное программное обеспечение в некоторой мере навязанным по отношению к принятым подходам. Действительно, все участники использовали ту или иную форму ансамблевого обучения аналогично стратегии, использованной в стартовом наборе. Однако можно утверждать, что это естественная стратегия для данной проблемы. Но в целом вопрос предоставления участникам достаточного количества стартового материала без смещения задачи в определенную сторону остается весьма деликатным.

С точки зрения разработки протокола конкурса мы поняли, что трудно удерживать команды сфокусированными в течение длительного периода времени и проводить их через множество этапов. Мы достигли большого числа участников (более 600) за все время проведения конкурса AutoML, который длился более года (2015/2016) и сопровождался несколькими мероприятиями (например, хакатонами). Однако, возможно, предпочтительнее организовывать ежегодные мероприятия, разделяемые семинарами. Это естественный способ сбалансировать конкуренцию и сотрудничество, поскольку семинары – это место обмена опытом. Участники, естественно, вознаграждаются признанием, которое они получают через систему научных публикаций. В подтверждение этой гипотезы второй этап конкурса AutoML (2017/2018), длившийся всего четыре месяца, собрал около 300 участников.

Одним из важных новшеств нашего конкурса было представление кода. Выполнение кода участников на одной платформе в строго одинаковых условиях является большим шагом к справедливости и воспроизводимости, а также обеспечивает жизнеспособность решения с вычислительной точки зрения. В качестве условия получения приза мы потребовали от победителей опубликовать свой код под лицензией с открытым исходным кодом. Это оказалось достаточно хорошим стимулом, чтобы получить несколько публикаций в качестве своего рода «продукта» организованных нами конкурсов. В нашем втором конкурсе (AutoML 2018) мы использовали Docker. Распространение образов Docker позволяет любому, кто скачает код участников, легко воспроизвести результаты, не сталкиваясь с проблемами установки и развертывания из-за несоответствия компьютерных сред и библиотек. Тем не менее аппаратное обеспечение может быть разным, и мы обнаружили, что при оценке результатов после конкурса смена компьютеров может дать значительные различия в результатах. Надеемся, что с распространением доступных облачных вычислений эта проблема станет менее актуальной.

Серия конкурсов AutoML только начинается. В настоящее время изучается несколько новых направлений. Например, мы готовим задачу Life Long Machine Learning, где участникам будут представлены данные, распределение которых медленно дрейфует со временем. Мы также рассматриваем задачу автоматизированного машинного обучения, в которой мы сосредоточимся на переносе данных из схожих областей.

Благодарности. Компания Microsoft поддержала организацию этой задачи и предоставила призы и время облачных вычислений на Azure. Этот проект получил дополнительную поддержку от Лаборатории фундаментальной информатики (LIF, UMR

CNRS 7279) Университета Экс-Марселя, Франция, через программу LabeX Archimede, Лаборатории исследований в области информатики Университета Paris Sud и INRIA-Saclay в рамках проекта TIMCO, а также поддержку Центра науки о данных Париж-Сакле (CDS). Дополнительные компьютерные ресурсы были щедро предоставлены Й. Бухманом, ETH Zürich. Эта работа была частично поддержана испанским проектом TIN2016-74946-P (MINECO/FEDER, UE) и программой CERCA/Generalitat de Catalunya. Конкурсные наборы данных были выбраны из 72 наборов, предоставленных (или отформатированных с использованием общедоступных данных) соавторами и авторами, среди которых: Ю. Афиньянапонгс, О. Шапель, З. Ифтихар Малхи, В. Лемер, С. Лин, М. Мадани, Г. Столовицкий, Х.-Дж. Тизен, и И. Цамардинос. Многие члены сообщества машинного обучения предоставили свои отзывы о ранних вариантах протокола и/или протестировали платформу вызова, в том числе: К. Беннетт, К. Каппони, Г. Коули, Р. Каруана, Г. Дрор, Т. К. Но, Б. Кёгль, Х. Ларошель, В. Лемаре, С. Лин, В. Понке-Лопез, Н. Масиа, С. Мерсер, Ф. Попеску, Д. Сильвер, С. Трежо и И. Цамардинос. Среди разработчиков программного обеспечения, внесших вклад в реализацию платформы Codalab и образца кода, – Э. Камайкл, И. Чаабане, И. Джадсон, К. Пулен, П. Лян, А. Песах, Л. Ромашко, Х. Баро Соле, Э. Уотсон, Ф. Жингри, М. Зисковски. Часть анализа результатов конкурсов была выполнена И. Шаабаном, Дж. Ллойдом, Н. Масиа и А. Такуром. Катарина Эггенспергер, Сайед Мохсин Али и Маттиас Фойрер помогли в организации конкурса Beat AutoSKLearn. Маттиас Фойрер также участвовал в моделировании работы auto-sklearn на наборах данных 2015–2016 гг.

10.8. Литература

1. Alamdari, A.R.S.A., Guyon, I.: Quick start guide for CLOP. Tech. rep., Graz University of Technology and Clopinet (May 2006).
2. Andrieu, C., Freitas, N.D., Doucet, A.: Sequential MCMC for Bayesian model selection. In: IEEE Signal Processing Workshop on Higher-Order Statistics. pp. 130–134 (1999).
3. Assunção, F., Lourenço, N., Machado, P., Ribeiro, B.: Denser: Deep evolutionary network structured representation. arXiv preprint arXiv:1801.01563 (2018).
4. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167 (2016).
5. Bardenet, R., Brendel, M., Kégl, B., Sebag, M.: Collaborative hyperparameter tuning. In: 30th International Conference on Machine Learning. vol. 28, pp. 199–207. JMLR Workshop and Conference Proceedings (May 2013).
6. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence 35(8), 1798–1828 (2013).
7. Bennett, K.P., Kunapuli, G., Jing Hu, J.S.P.: Bilevel optimization and machine learning. In: Computational Intelligence: Research Frontiers, Lecture Notes in Computer Science, vol. 5050, pp. 25–47. Springer (2008).
8. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research 13(Feb), 281–305 (2012).
9. Bergstra, J., Yamins, D., Cox, D.D.: Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision archi-

tectures. In: 30th International Conference on Machine Learning. vol. 28, pp. 115–123 (2013).
10. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems. pp. 2546–2554 (2011).
11. Blum, A.L., Langley, P.: Selection of relevant features and examples in machine learning. Artificial Intelligence 97(1–2), 273–324 (December 1997).
12. Boullé, M.: Compression-based averaging of selective naive bayes classifiers. Journal of Machine Learning Research 8, 1659–1685 (2007), http://dl.acm.org/citation.cfm?id=1314554.
13. Boullé, M.: A parameter-free classification method for large scale learning. Journal of Machine Learning Research 10, 1367–1385 (2009), https://doi.org/10.1145/1577069.1755829.
14. Brazdil, P., Carrier, C.G., Soares, C., Vilalta, R.: Metalearning: Applications to data mining. Springer Science & Business Media (2008).
15. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001).
16. Caruana, R., Niculescu-Mizil, A., Crew, G., Ksikes, A.: Ensemble selection from libraries of models. In: 21st International Conference on Machine Learning. pp. 18–. ACM (2004).
17. Cawley, G.C., Talbot, N.L.C.: Preventing over-fitting during model selection via Bayesian regularisation of the hyper-parameters. Journal of Machine Learning Research 8, 841–861 (April 2007).
18. Colson, B., Marcotte, P., Savard, G.: An overview of bilevel programming. Annals of Operations Research 153, 235–256 (2007).
19. Dempe, S.: Foundations of bilevel programming. Kluwer Academic Publishers (2002).
20. Dietterich, T.G.: Approximate statistical test for comparing supervised classification learning algorithms. Neural Computation 10(7), 1895–1923 (1998).
21. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley, 2nd edn. (2001).
22. Efron, B.: Estimating the error rate of a prediction rule: Improvement on cross-validation. Journal of the American Statistical Association 78(382), 316–331 (1983).
23. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., Leyton-Brown, K.: Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In: NIPS workshop on Bayesian Optimization in Theory and Practice (2013).
24. Escalante, H.J., Montes, M., Sucar, L.E.: Particle swarm model selection. Journal of Machine Learning Research 10, 405–440 (2009).
25. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Proceedings of the Neural Information Processing Systems, pp. 2962–2970 (2015), https://github.com/AutoML/auto-sklearn.
26. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Methods for improving bayesian optimization for AutoML. In: Proceedings of the International Conference on Machine Learning 2015, Workshop on Automatic Machine Learning (2015).

27. Feurer, M., Springenberg, J., Hutter, F.: Initializing bayesian hyperparameter optimization via meta-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. pp. 1128– 1135 (2015).
28. Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., Hutter, F.: Practical automated machine learning for the AutoML challenge 2018. In: International Workshop on Automatic Machine Learning at ICML (2018), https://sites.google.com/site/AutoML2018icml/.
29. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. The Annals of Statistics 29(5), 1189–1232 (2001).
30. Ghahramani, Z.: Unsupervised learning. In: Advanced Lectures on Machine Learning. Lecture Notes in Computer Science, vol. 3176, pp. 72–112. Springer Berlin Heidelberg (2004).
31. Guyon, I.: Challenges in Machine Learning book series. Microtome (2011–2016), http://www.mtome.com/Publications/CiML/ciml.html.
32. Guyon, I., Bennett, K., Cawley, G., Escalante, H.J., Escalera, S., Ho, T.K., Macià, N., Ray, B., Saeed, M., Statnikov, A., Viegas, E.: AutoML challenge 2015: Design and first results. In: Proc. of AutoML 2015@ICML (2015), https://drive.google.com/file/d/0BzRGLkqgrI-qWkpzcGw4bFpBMUk/view.
33. Guyon, I., Bennett, K., Cawley, G., Escalante, H.J., Escalera, S., Ho, T.K., Macià, N., Ray, B., Saeed, M., Statnikov, A., Viegas, E.: Design of the 2015 ChaLearn AutoML challenge. In: International Joint Conference on Neural Networks (2015), http://www.causality.inf.ethz.ch/AutoML/AutoML_ijcnn15.pdf.
34. Guyon, I., Chaabane, I., Escalante, H.J., Escalera, S., Jajetic, D., Lloyd, J.R., Macía, N., Ray, B., Romaszko, L., Sebag, M., Statnikov, A., Treguer, S., Viegas, E.: A brief review of the ChaLearn AutoML challenge. In: Proc. of AutoML 2016@ICML (2016), https://docs.google.com/a/chalearn.org/viewer?a=v&pid=sites&srcid=Y2hhbGVhcm4ub3JnfGF1dG9tbHxneDoyYThjZjhhNzRjMzI3MTg4.
35. Guyon, I., Alamdari, A.R.S.A., Dror, G., Buhmann, J.: Performance prediction challenge. In: the International Joint Conference on Neural Networks. pp. 1649–1656 (2006).
36. Guyon, I., Bennett, K., Cawley, G., Escalante, H.J., Escalera, S., Ho, T.K., Ray, B., Saeed, M., Statnikov, A., Viegas, E.: AutoML challenge 2015: Design and first results (2015).
37. Guyon, I., Cawley, G., Dror, G.: Hands-On Pattern Recognition: Challenges in Machine Learning, Volume 1. Microtome Publishing, USA (2011).
38. Guyon, I., Gunn, S., Nikravesh, M., Zadeh, L. (eds.): Feature extraction, foundations and applications. Studies in Fuzziness and Soft Computing, Physica-Verlag, Springer (2006).
39. Hastie, T., Rosset, S., Tibshirani, R., Zhu, J.: The entire regularization path for the support vector machine. Journal of Machine Learning Research 5, 1391–1415 (2004).
40. Hastie, T., Tibshirani, R., Friedman, J.: The elements of statistical learning: Data mining, inference, and prediction. Springer, 2nd edn. (2001).
41. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proceedings of the conference on Learning and Intelligent OptimizatioN (LION 5) (2011).

42. Ioannidis, J.P.A.: Why most published research findings are false. PLoS Medicine 2(8), e124 (August 2005).
43. Jordan, M.I.: On statistics, computation and scalability. Bernoulli 19(4), 1378–1390 (September 2013).
44. Keerthi, S.S., Sindhwani, V., Chapelle, O.: An efficient method for gradient-based adaptation of hyperparameters in SVM models. In: Advances in Neural Information Processing Systems (2007).
45. Klein, A., Falkner, S., Bartels, S., Hennig, P., Hutter, F.: Fast bayesian hyperparameter optimization on large datasets. In: Electronic Journal of Statistics. vol. 11 (2017).
46. Kohavi, R., John, G.H.: Wrappers for feature selection. Artificial Intelligence 97(1–2), 273– 324 (December 1997).
47. Langford, J.: Clever methods of overfitting (2005), blog post at http://hunch.net/?p=22.
48. Lloyd, J.: Freeze Thaw Ensemble Construction. https://github.com/james-robertlloyd/AutoML-phase-2 (2016).
49. Momma, M., Bennett, K.P.: A pattern search method for model selection of support vector regression. In: In Proceedings of the SIAM International Conference on Data Mining. SIAM (2002).
50. Moore, G., Bergeron, C., Bennett, K.P.: Model selection for primal SVM. Machine Learning 85(1–2), 175–208 (October 2011).
51. Moore, G.M., Bergeron, C., Bennett, K.P.: Nonsmooth bilevel programming for hyperparameter selection. In: IEEE International Conference on Data Mining Workshops. pp. 374–381 (2009).
52. Niculescu-Mizil, A., Perlich, C., Swirszcz, G., Sindhwani, V., Liu, Y., Melville, P., Wang, D., Xiao, J., Hu, J., Singh, M., et al.: Winning the kdd cup orange challenge with ensemble selection. In: Proceedings of the 2009 International Conference on KDD-Cup 2009-Volume 7. pp. 23–34. JMLR. org (2009).
53. Opper, M., Winther, O.: Gaussian processes and SVM: Mean field results and leave-one-out, pp. 43–65. MIT (10 2000), massachusetts Institute of Technology Press (MIT Press) Available on Google Books.
54. Park, M.Y., Hastie, T.: L1-regularization path algorithm for generalized linear models. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 69(4), 659–677 (2007).
55. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830 (2011).
56. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268 (2018).
57. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Le, Q., Kurakin, A.: Large-scale evolution of image classifiers. arXiv preprint arXiv:1703.01041 (2017).
58. Ricci, F., Rokach, L., Shapira, B., Kantor, P.B. (eds.): Recommender Systems Handbook. Springer (2011).
59. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press (2001).

60. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems 25, pp. 2951–2959 (2012).
61. Statnikov, A., Wang, L., Aliferis, C.F.: A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification. BMC Bioinformatics 9(1) (2008).
62. Sun, Q., Pfahringer, B., Mayo, M.: Full model selection in the space of data mining operators. In: Genetic and Evolutionary Computation Conference. pp. 1503–1504 (2012).
63. Swersky, K., Snoek, J., Adams, R.P.: Multi-task Bayesian optimization. In: Advances in Neural Information Processing Systems 26. pp. 2004–2012 (2013).
64. Swersky, K., Snoek, J., Adams, R.P.: Freeze-thaw bayesian optimization. arXiv preprint arXiv:1406.3896 (2014).
65. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms. CoRR abs/1208.3719 (2012).
66. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 847–855. ACM (2013).
67. Vanschoren, J., Van Rijn, J.N., Bischl, B., Torgo, L.: Openml: networked science in machine learning. ACM SIGKDD Explorations Newsletter 15(2), 49–60 (2014).
68. Vapnik, V., Chapelle, O.: Bounds on error expectation for support vector machines. Neural computation 12(9), 2013–2036 (2000).
69. Weston, J., Elisseeff, A., BakIr, G., Sinz, F.: Spider (2007), http://mloss.org/software/view/29/.
70. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016).

Предметный указатель

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@alians-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/**.

Франк Хуттер, Ларс Коттхофф, Хоакин Ваншорен

Введение в автоматизированное машинное обучение (AutoML)

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Зам. главного редактора *Сенченкова Е. А.*
Перевод *Яценков В. С.*
Корректор *Абросимова Л. А.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Гарнитура PT Serif. Печать цифровая.
Усл. печ. л. 20,8. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**