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

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.

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).

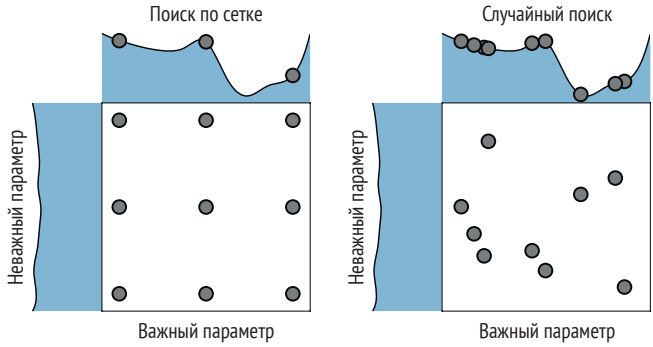

Рис. 1.1 ❖ Сравнение поиска по сетке и случайного поиска для минимизации функции с одним важным и одним неважным параметром. Этот рисунок основан на рис. 1 из работы Бергстры и Бенджио [13]

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.

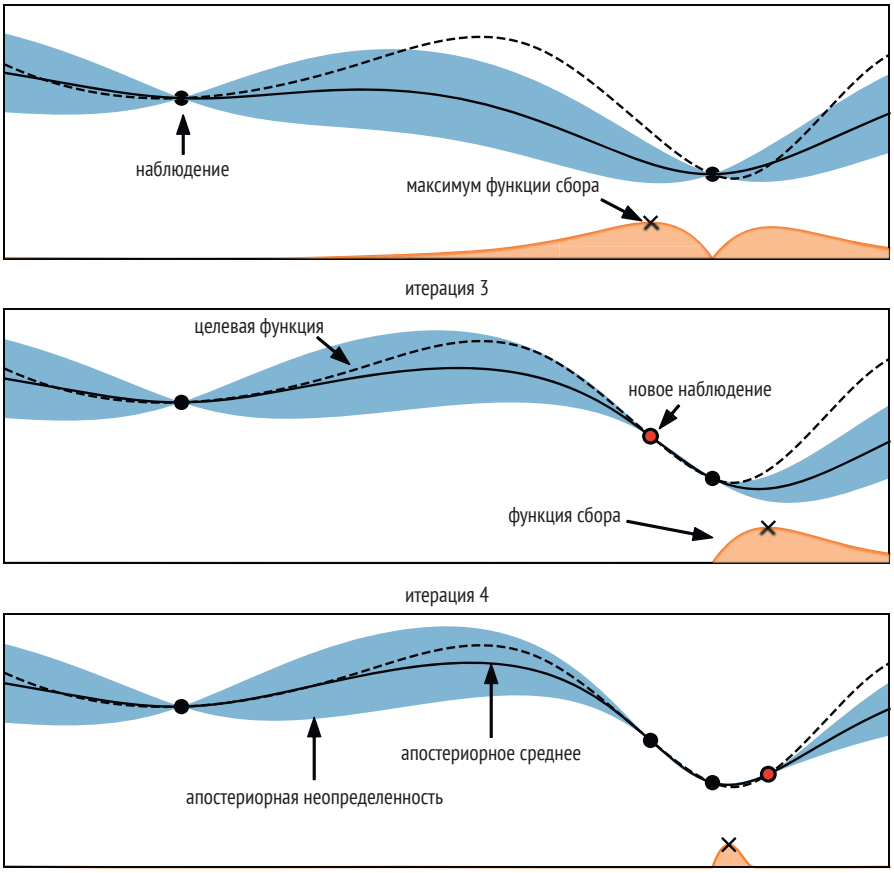

Рис. 1.2 ❖ Иллюстрация байесовской оптимизации одномерной функции. Мы стремимся минимизировать пунктирную линию, используя суррогат гауссова процесса (прогнозы показаны черной линией, а синяя область представляет неопределенность), максимизируя функцию сбора, представленную оранжевой кривой (верхняя 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.

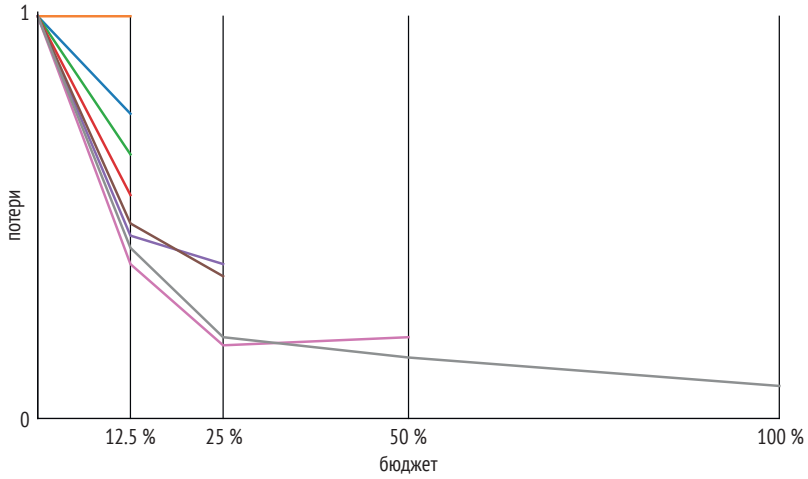

Рис. 1.3 ❖ Иллюстрация последовательного деления пополам для восьми алгоритмов/конфигураций. После оценки всех алгоритмов на 1 от общего бюджета половина из них отбрасывается, а бюджет, выделенный оставшимся алгоритмам, удваивается

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.