O'REILLY®

Машинное обучение Паттерны проектирования

Подготовка данных, создание моделей, внедрение в производство

Валлиаппа Лакшманан,
Сара Робинсон и Майкл Мунн

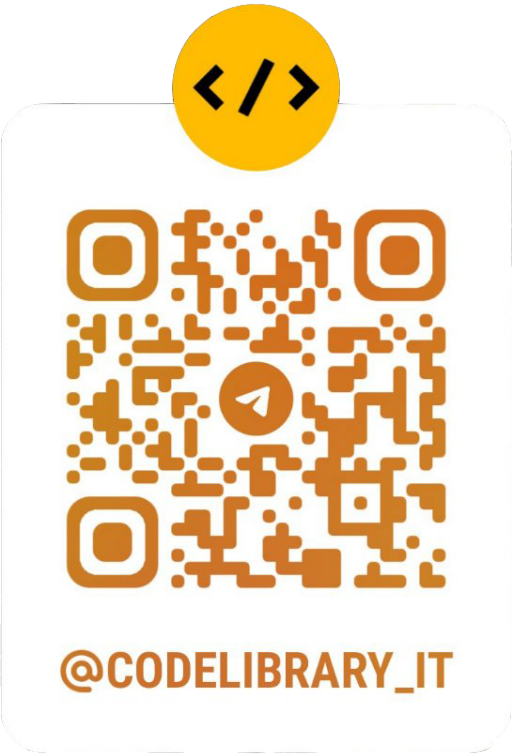
bhv®

Machine Learning Design Patterns

Solutions to Common Challenges in Data Preparation, Model Building, and MLOps

Valliappa Lakshmanan, Sara Robinson, and Michael Munn

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY


@CODELIBRARY_IT

Валлиаппа Лакшманан
Сара Робинсон
Майкл Мунн

Машинное обучение
Паттерны проектирования

Санкт-Петербург
«БХВ-Петербург»
2022

УДК 004.4'236
ББК 32.973.26-018
Л19

Лакшманан, В.

Л19 Машинное обучение. Паттерны проектирования: Пер. с англ. / В. Лакшманан, С. Робинсон, М. Мунн. — СПб.: БХВ-Петербург, 2022. — 448 с.: ил.

ISBN 978-5-9775-6797-8

Приводимые в книге паттерны проектирования отражают лучшие практические подходы к решению типичных задач машинного обучения. Указанные паттерны, реализованные в программном коде, сконцентрировали опыт сотен экспертов в простые и легкодоступные советы. Книга содержит подробный разбор 30 паттернов, служащих для представления данных и задач, тренировки моделей, отказоустойчивого обслуживания, обеспечения воспроизводимости и искусственного интеллекта. Каждый паттерн включает в себя постановку задачи, ряд потенциальных решений и рекомендации по выбору технического приема, наилучшим образом подходящего к данной ситуации.

Для программистов в области машинного обучения

УДК 004.4'236
ББК 32.973.26-018

Научный редактор:

Архитектор решений,
руководитель группы разработки компании КРОК *Дмитрий Бардин*

Группа подготовки издания:

Руководитель проекта	*Евгений Рыбаков*
Зав. редакцией	*Людмила Гауль*
Компьютерная верстка	*Ольги Сергиенко*
Оформление обложки	*Зои Канторович*

Подписано в печать 02.02.22.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 36,12.
Тираж 1200 экз. Заказ № 3478.
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Отпечатано с готового оригинал-макета
ООО "Принт-М", 142300, М.О., г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-1-098-11578-4 (англ.)
ISBN 978-5-9775-6797-8 (рус.)

Оглавление

Об авторах

Валлиаппа (Лак) Лакшманан (Valliappa (Lak) Lakshmanan) — глобальный руководитель отдела аналитики данных и решений в области искусственного интеллекта в Google Cloud. Руководимый им коллектив строит программно-информационные решения для деловых задач, используя продукты Google Cloud для анализа данных и машинного обучения. Он является основателем программы погружения в проблематику машинного обучения в Лаборатории передовых решений компании Google (Google's Advanced Solutions Lab ML). До работы в компании Google Лак был директором по исследованию данных в Climate Corporation и научным исследователем в NOAA.

Сара Робинсон (Sara Robinson) — адвокат разработчиков в коллективе Google Cloud со специализацией в машинном обучении. Она вдохновляет разработчиков и исследователей данных интегрировать машинное обучение в свои приложения с помощью демонстраций, онлайнового контента и мероприятий. Имеет степень бакалавра, полученную в Университете Брандейса (Brandeis University). До работы в компании Google была представителем разработчиков из коллектива Firebase.

Майкл Мунн (Michael Munn) — инженер по техническим решениям в области машинного обучения в Google, в которой он работает с заказчиками инфраструктуры Google Cloud, помогая им разрабатывать и развертывать модели машинного обучения. Также преподает программу погружения в проблематику машинного обучения в Лаборатории передовых решений (Advanced Solutions Lab ML). Имеет докторскую степень по математике, полученную в Университете Нью-Йорка (City University of New York). До прихода в Google он работал профессором-исследователем.

Предисловие

Для кого эта книга предназначена?

Обзорные книги по машинному обучению (machine learning, ML) обычно посвящены ответам на вопросы о том, что и как делается в ML. В них также может быть приведено объяснение математических аспектов новых методов, которые поступают из исследовательских лабораторий искусственного интеллекта (ИИ), и представлены методики, обучающие тому, как использовать фреймворки ИИ для реализации этих методов. В отличие от такой литературы, настоящая книга соединяет в себе наработанный опыт в попытке ответить на вопрос: почему тот или иной метод работает? Указанный вопрос лежит в основе приемов и хитростей, которые опытные практики ML эффективно задействуют, применяя технологию машинного обучения к реальным практическим задачам.

Мы исходим из того, что у вас есть предварительные знания о машинном обучении и обработке данных. Эта книга не является фундаментальным трудом по ML. Напротив, она вам пригодится в случае, если вы являетесь исследователем данных, инженером данных или инженером ML, который ищет вторую книгу по практическому машинному обучению. Если вы уже знаете основы, то наша книга познакомит вас с каталогом идей, отдельные из которых вы (специалист-практик ML) можете распознать, и даст этим идеям название, дабы вы могли уверенно к ним обращаться.

Если вы изучаете информатику и собираетесь работать в IT-индустрии, эта книга дополнит ваши знания и подготовит вас к профессиональному миру. Она поможет вам научиться создавать высококачественные системы ML.

Чего в книге нет

Эта книга в первую очередь предназначена для практикующих инженеров ML, работающих на предприятиях, а не для научных исследователей машинного обучения из академических или отраслевых исследовательских лабораторий.

Мы намеренно не обсуждаем области активных научных исследований — вы найдете здесь очень мало материала, например, об архитектуре ML-моделей (к примеру, о двунаправленных кодировщиках, о механизме внимания или о слоях короткого замыкания), поскольку мы исходим из того, что вы, скорее, будете использовать

уже построенную кем-то модельную архитектуру (например, ResNet-50 или GRUCell), чем писать собственную классификацию изображений или рекуррентную нейронную сеть.

Вот несколько конкретных примеров областей, от которых мы намеренно держимся подальше, потому что считаем, что эти темы больше подходят для курсов колледжей и научных исследователей ML.

- *Алгоритмы машинного обучения.*

  Например, мы не рассматриваем различия между случайными лесами и нейронными сетями. Об этом говорится в учебниках по машинному обучению.

- *Строительные блоки.*

  Мы не рассматриваем различные типы оптимизаторов, построенных на градиентном спуске, или разновидности активационных функций. Мы рекомендуем использовать Adam и ReLU — по нашему опыту, потенциал для улучшения результативности при выборе разных вариантов в таких вещах, как правило, незначителен.

- *Архитектуры моделей машинного обучения.*

  Если вы занимаетесь классификацией изображений, мы рекомендуем вам использовать готовую к работе модель, такую как ResNet либо любую другую, которая окажется самой используемой в то время, когда вы будете читать эту книгу. Предоставьте разработку новых моделей классифицирования изображений или разбора текста научным исследователям, которые специализируются на этой проблематике.

- *Модельные слои.*

  В этой книге вы не найдете ни сверточных, ни рекуррентных нейронных сетей. Они дисквалифицированы вдвойне — во-первых, за то, что являются строительными блоками, а во-вторых, за то, что вы можете использовать их в готовом к работе виде.

- *Сценарии тренировки сетей.*

  Простой вызов `model.fit()` в Keras — это все, что нужно будет сделать специалисту-практику.

Мы попытались включить в эту книгу только распространенные паттерны, которые инженеры машинного обучения будут задействовать в своей повседневной работе на предприятиях.

В качестве аналогии возьмем, к примеру, структуры данных. В обучающем курсе по структурам данных подробно рассматривают реализации разных структур данных. Научный исследователь структур данных должен будет находить способы формально представлять их математические свойства. В отличие от них, специалист-практик будет действовать прагматичнее. Разработчику корпоративного программно-информационного обеспечения просто необходимо знать подходы к эффективной работе с массивами, связными списками, словарями, множествами и де-

ревьями. Эта книга написана для прагматичного специалиста-практика в области машинного обучения.

Примеры исходного кода

Мы предоставляем исходный код для машинного обучения (иногда написанный с использованием Keras/TensorFlow, а иногда с помощью Python-библиотеки scikit-learn или языка BigQuery ML) и для обработки данных (в SQL) как один из путей показать технические приемы реализации обсуждаемых методов на практике. Весь исходный код, на который есть ссылки в книге, является частью нашего репозитория[1] на GitHub, в котором вы найдете полностью работающие модели. Мы настоятельно рекомендуем вам опробовать эти примеры.

Указанный исходный код имеет по отношению к рассматриваемым концепциям и методам второстепенное значение. Наша цель состояла в том, чтобы тема и принципы оставались актуальными независимо от изменений в TensorFlow или Keras и мы могли бы, например, легко обновить репозиторий GitHub, включив в его состав другие ML-фреймворки, оставив при этом текст книги неизменным. И, следовательно, книга останется одинаково информативной в случае, если вашим первичным фреймворком является Python-библиотека PyTorch или даже не-Python-фреймворк, такой как H20.ai или R. Мы, безусловно, приветствуем ваш вклад в репозиторий в виде одного или нескольких приведенных в книге паттернов с использованием вашего любимого фреймворка.

Условные обозначения, принятые в книге

В книге используются следующие типографские условные обозначения:

- *Курсив* указывает новые термины.
- `Моноширинный шрифт` используется для листингов программ, а также внутри абзацев для ссылки на элементы программ, такие как переменные или имена инструкции языка и ключевые слова;
- **`Жирный моноширинный шрифт`** показывает команды либо другой текст, который должен быть напечатан пользователем буквально, а также зарезервированные в языке инструкции и ключевые слова.
- *`Моноширинный шрифт курсивом`* показывает текст, который должен быть заменен значениями пользователя либо значениями, определяемым по контексту, а также комментарии в листингах.

Данный элемент обозначает подсказку или совет.

[1] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns**.

Данный элемент обозначает общее замечание.

Данный элемент обозначает предупреждение или предостережение.

На веб-странице этой книги перечислены ошибки, примеры и любая дополнительная информация. К этой веб-странице можно обратиться по адресу **https://oreil.ly/MLDP**.

Благодарности

Подобного рода книга была бы невозможна без щедрости многочисленных гуглеров, в особенности наших коллег из коллективов по облачному ИИ, инженерии технических решений, профессиональным службам и по связям с разработчиками. Мы благодарны им за то, что они позволили нам наблюдать, анализировать и подвергать сомнению их решения сложных задач, с которыми они сталкивались во время тренировки, совершенствования и промышленного использования моделей ML. Спасибо нашим менеджерам Карлу Вайнмейстеру (Karl Weinmeister), Стиву Челлини (Steve Cellini), Хамиду Диа (Hamidou Dia), Абдулу Разаку (Abdul Razack), Крису Халленбеку (Chris Hallenbeck), Патрику Коулу (Patrick Cole), Луизе Бирн (Louise Byrne) и Рошане Голани (Rochana Golani) за то, что они способствовали развитию в Google духа открытости, дали нам свободу каталогизировать эти модели и опубликовать эту книгу.

Салем Хайкал (Salem Haykal), Бенуа Дерин (Benoit Dherin) и Халид Салама (Khalid Salama) просмотрели каждую модель и каждую главу. Сал указал на упущенные нами нюансы, Бенуа сузил круг наших претензий, а Халид указал нам на соответствующие научные исследования. Эта книга была бы далеко не так хороша без вашего участия. Спасибо! Эми Унру (Amy Unruh), Раджеш Таллам (Rajesh Thallam), Робби Хэртел (Robbie Haertel), Чжитао Ли (Zhitao Li), Ануша Рамеш (Anusha Ramesh), Мин Фан (Ming Fang), Паркер Барнс (Parker Barnes), Эндрю Залдивар (Andrew Zaldivar), Джеймс Векслер (James Wexler), Эндрю Селлергрен (Andrew Sellergren) и Дэвид Кантер (David Kanter) рассмотрели части этой книги, которые соответствуют их областям знаний, и внесли несколько предложений о том, каким образом краткосрочная дорожная карта повлияет на наши рекомендации. Нитин Аггарвал (Nitin Aggarwal) и Мэтью Игер (Matthew Yeager) сделали рукопись привлекательнее для читателя и улучшили ее четкость. Особая благодарность Раджешу Талламу (Rajesh Thallam) за создание прототипа самого последнего рисунка из *главы 8*. Любые оставшиеся ошибки, конечно же, принадлежат только нам.

Издательство O'Reilly является лучшим издателем технической литературы, и профессионализм нашего коллектива это ярко иллюстрирует. Ребекка Новак (Rebecca Novak) сопровождала нас в составлении общего плана, Кристен Браун (Kristen

Brown) с апломбом руководила разработкой всего контента, Корбин Коллинз (Corbin Collins) давал нам полезные указания на каждом этапе, Элизабет Келли (Elizabeth Kelly) была в восторге от работы во время производства, а Чарльз Румелиотис (Charles Roumeliotis) внимательно следил за копирайтингом. Спасибо за вашу помощь!

Майкл: "Спасибо моим родителям за то, что они всегда в меня верили и поощряли мои интересы, как академические, так и другие. Вы сможете оценить их завуалированную защиту так же, как и я. Спасибо Филу за то, что он терпеливо сносил мой невыносимый график во время работы над этой книгой. Теперь же я пойду спать".

Сара: "Джон, ты — главная причина существования этой книги. Спасибо за то, что вдохновил меня заняться ее написанием, за то, что всегда знал, как вызывать у меня смех, ценил мои странности и верил в меня, в особенности когда я переставала верить. Спасибо моим родителям за то, что они были моими самыми большими поклонниками с самого первого дня и поддерживали мою любовь к технологиям и писательству с тех пор, как я себя помню. Элли, Кэти, Рэнди и Софи — спасибо вам за то, что вы являетесь постоянным источником света и смеха в эти смутные времена".

Лак: "Я взялся за эту книгу, думая, что смогу работать над ней, ожидая в аэропортах. COVID-19 сделал так, что бóльшая часть работы была проделана дома. Спасибо Абирами, Сидхарту и Сараде за все ваше терпение, когда я снова приседал на корточки, чтобы что-то записать. Теперь у меня будет больше походов по выходным!"

Мы втроем жертвуем 100% гонорара от этой книги организации Girls Who Code[2], миссия которой состоит в строительстве настоящего конвейера из будущих инженеров-женщин. Разнообразие, объективность и инклюзивность особенно важны в машинном обучении, обеспечивая, чтобы модели ИИ не увековечивали в человеческом обществе существующие искаженные взгляды.

[2] См. **https://girlswhocode.com/**.

ГЛАВА 1

Потребность в паттернах машинного обучения

В инженерных дисциплинах паттерны отражают наилучшие практические приемы и технические решения часто возникающих задач. Они кодифицируют знания и опыт экспертов в советы для специалистов-практиков, которым те могут следовать. Эта книга представляет собой каталог паттернов машинного обучения (machine learning, ML), которые мы наблюдали в ходе работы с сотнями коллективов, специализирующихся на машинном обучении.

Что такое паттерны?

Идея паттернов и каталог проверенных временем паттернов были представлены в области архитектуры Кристофером Александером (Christopher Alexander) и пятью его соавторами в чрезвычайно влиятельной книге "Язык паттернов" (A pattern language. — Oxford University Press, 1977). В своей книге они каталогизируют 253 архитектурных паттерна, представляя их следующим образом.

> "Каждый паттерн описывает задачу, которая в нашей среде возникает многократно, а затем описывает ядро решения этой задачи так, чтобы у вас была возможность использовать это решение миллион раз, всякий раз новым способом.
>
> ...
>
> Каждое решение формулируется таким образом, чтобы оно давало поле связей, необходимых для решения задачи, но в очень общем и абстрактном виде, благодаря чему вы сможете решать задачу по-своему, адаптируя ее к собственным предпочтениям и локальным условиям той проблемы, которую вам нужно устранить".

Например, пара архитектурных паттернов, которые содержат ожидаемые детали при строительстве дома, включают в себя "освещение с двух сторон в каждой комнате" и "шестифутовый балкон". Подумайте о своей любимой комнате и менее любимой комнате в доме. Есть ли в вашей любимой комнате окна на двух стенах? А у менее любимой комнаты? Согласно Александеру:

> "Комнаты, которые освещены с двух сторон естественным светом, создают меньше бликов вокруг людей и предметов; это позволяет нам видеть вещи более замысловато; и самое главное — это позволяет нам подробно различать мельчайшие выражения, которые мелькают на лицах людей..."

Лаконичное название этого паттерна избавляет архитекторов от необходимости всякий раз формулировать этот принцип. Тем не менее, откуда и как вы получаете

два источника света в любом отдельно взятом локальном контексте, зависит от мастерства архитектора. Аналогично при дизайне балкона встает вопрос: каким должен быть его размер? Согласно рекомендации Александера достаточной площадью для двух (неправильно подобранных!) стульев и приставного столика будет 6×6 футов или же 12×12 футов, в случае если вы хотите иметь место в тени и место на солнце.

Эрих Гамма, Ричард Хелм, Ральф Джонсон и Джон Флиссайдс привнесли эту идею в программно-информационное обеспечение, каталогизировав 23 объектно-ориентированных паттерна проектировния в книге "Паттерны проектирования: элементы готового к многоразовому применению объектно-ориентированного программного обеспечения" (Gamma E., Helm R., Johnson R., Vlissides J. Design patterns: elements of reusable object-oriented software. — Addison-Wesley, 1995). Их каталог содержаит такие паттерны, как "Прокси" (Proxy), "Синглтон" (Singleton) и "Декоратор" (Decorator), и оказал большое влияние на сферу объектно-ориентированного программирования. В 2005 году Ассоциация вычислительной техники (Association of Computing Machinery, ACM) присудила авторам ежегодную премию за достижения в области языков программирования, признав бесценный вклад их работы "в практику программирования и дизайн языков программирования".

Создание промышленных моделей машинного обучения становится инженерной практикой, в которой преимущества доказаных в исследовательских условиях методов ML используются и применяются к задачам бизнеса. По мере того как машинное обучение становится все более распространенным, важно, чтобы специалисты-практики использовали испытанные и проверенные временем методы для решения повторяющихся задач.

Одним из преимуществ нашей деятельности в Google Cloud является то, что она позволяет нам контактировать с самыми разнообразными коллективами специалистов в области машинного обучения и науки о данных и отдельными разработчиками со всего мира. В то же время каждый из нас тесно сотрудничает с внутренними коллективами Google, решая передовые задачи машинного обучения. Наконец, нам посчастливилось работать с коллективами TensorFlow, Keras, BigQuery ML, TPU и Cloud AI Platform, которые способствуют демократизации научных исследований в области машинного обучения и его инфраструктуры. Все это позволяет нам рассматривать на нашу деятельность под довольно уникальным углом зрения и каталогизировать лучшие практические приемы, наблюдаемые нами в работе этих коллективов.

Данная книга представляет собой каталог паттернов для решений задач, часто возникающих в инженерии технических решений ML. Например, паттерн "Преобразователь" (Transform; *см. главу 6*) обеспечивает разделения входных переменных, признаков и преобразований и делает преобразования постоянными, упрощая перенос модели ML в результирующий продукт. Точно так же паттерн "Предсказания по ключу" (Keyed Predictions) из *главы 5* обеспечивает крупномасштабное распределение пакетных предсказаний, например, для рекомендательных моделей.

В рамках каждого паттерна мы описываем часто встречающуюся задачу, а затем детально разбираем ряд потенциальных технических решений этой задачи, их ком-

промиссы и рекомендации по выбору между этими решениями. Реализация решений представлена на SQL (неоценимом, если вы выполняете предобработку и другие операции ETL[1] в Spark SQL, BigQuery и т. д.) и на языке Python с использованием библиотеки scikit-learn и/или фреймворка ML Keras с бэкендом TensorFlow.

Как пользоваться этой книгой

Настоящая книга является каталогом паттернов задач, которые мы наблюдали на практике в многочисленных профессиональных коллективах. В некоторых случаях понятия, лежащие в основе решений, известны уже много лет. Мы не претендуем на авторство этих паттернов. Как раз напротив, мы надеемся предоставить специалистам-практикам ML общую систему отсчета и набор инструментов. Мы будем считать нашу задачу успешно выполненной, если эта книга станет словарем понятий, которые вы уже интуитивно включаете в свои проекты ML.

Мы не рассчитываем, что вы будете читать эту книгу последовательно (хотя и можете!). Мы надеемся, что вы будете бегло ее просматривать, читать некоторые ее разделы внимательнее, чем другие, ссылаться на идеи в беседах с коллегами и возвращаться к книге тогда, когда сталкнетесь с задачами, о которых что-то читали. Если вы планируете читать не по порядку, рекомендуем вам перед "погружением" в отдельные паттерны начать с *глав 1* и *8*.

Каждый паттерн содержит краткую постановку задачи, решение, объяснение причины, ответы на вопрос, почему это решение работает, и многосоставное обсуждение компромиссов и альтернатив. Мы рекомендуем вам читать раздел обсуждения, твердо помня о решении, с целью сравнения и противопоставления. Описание паттерна будет включать фрагменты исходного кода, взятые из реализации решения. Полный исходный код можно найти в нашем репозитории[2] на GitHub. Мы настоятельно рекомендуем вам внимательно знакомиться с исходным кодом, когда вы читаете описание паттерна.

Терминология машинного обучения

Поскольку современные специалисты-практики по машинному обучению могут быть компетентными в различных областях — программной инженерии, анализе данных, интеграции программно-информационных продуктов в производство (DevOps) или статистике, могут существовать тонкие различия в том, как разные специалисты-практики используют те или иные термины. В этом разделе мы приведем терминологию, которую используем в книге.

---

[1] То есть стандартные операции извлечения (extract), преобразования (transform) и загрузки (load) данных. — *Прим. перев.*

[2] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns**.

Модели и фреймворки

По своей сути машинное обучение — это процесс строительства моделей, которые усваивают закономерности из данных. Он отличается от традиционного программирования, где мы пишем однозначные правила, которые сообщают программам о том, как себя вести. Модели машинного обучения — это алгоритмы, которые усваивают закономерности из данных. В целях иллюстрации этого момента представьте, что мы являемся компанией, организующей переезды, и должны оценивать затраты на переезд потенциальных клиентов. В традиционном программировании мы могли бы решить эту задачу с помощью инструкции `if`:

```
if num_bedrooms == 2 and num_bathrooms == 2:
    estimate = 1500
elif num_bedrooms == 3 and sq_ft > 2000:
    estimate = 2500
```

Можно только представить, как код быстро усложнится, когда мы добавим больше переменных (число негабаритных предметов мебели, количество предметов одежды, хрупких предметов и т. д.) и попытаемся справиться с крайними случаями. Более того, предварительный запрос всей этой информации у клиентов может испугать их и побудить отказаться от процесса оценивания. Вместо этого мы можем натренировать модель оценивать затраты на переезд, основываясь на прошлых данных о предыдущих домохозяйствах, которым наша компания помогла переехать.

В книге мы используем модели, главным образом основанные на искусственных нейронных сетях прямого распространения (feed-forward neural network). Однако мы также будем ссылаться на линейно-регрессионные модели, модели, основанные на деревьях решений, кластеризационные модели и др. *Искусственные нейронные сети прямого распространения*, которые принято сокращенно именовать просто нейронными сетями, представляют собой тип алгоритма машинного обучения, в котором многочисленные слои, каждый из которых имеет много нейронов, анализируют и обрабатывают информацию, а затем отправляют эту информацию в следующий слой, в результате чего получается конечный слой, который на выходе производит предсказание. Хотя нейронные сети ни в коей мере не идентичны нейронам в нашем мозге, их нередко принято сравнивать из-за связности между узлами и согласно принципам, по которым они способны обобщать и формировать новые предсказания на основе обрабатываемых ими данных. Нейронные сети с более чем одним *скрытым слоем* (слоем, отличающимся от входного и выходного слоев) классифицируются как *глубокое обучение* (рис. 1.1).

Модели машинного обучения независимо от того, как они изображаются визуально, являются математическими функциями и поэтому могут быть реализованы с нуля с помощью численного программно-информационого пакета. Однако инженеры ML, как правило, используют один из нескольких специализированных фреймворков с открытым исходным кодом, которые предоставляют для создания моделей интуитивно понятные API. В большинстве наших примеров будет использоваться TensorFlow — фреймворк машинного обучения с открытым исходным ко-

дом, созданный Google с акцентом на моделях, построенных на основе глубокого обучения. В рамках фреймворка TensorFlow мы в наших примерах будем использовать API Keras, который можно импортировать посредством tensorflow.keras. Keras — это высокоуровневый API для создания нейронных сетей. Указанный API поддерживает многочисленные бэкенды, но мы будем использовать его бэкенд TensorFlow. В других примерах мы будем применять другие популярные фреймворки с открытым исходным кодом, такие как scikit-learn, XGBoost и PyTorch, которые предоставляют утилиты для подготовки данных, а также API для создания линейных и глубоких моделей. Технология машинного обучения становится все более доступной, и этот факт подтверждается применением языка SQL для выражения моделей машинного обучения. В качестве примера мы будем использовать инструмент Big-Query ML, особенно в ситуациях, когда потребуется сочетать предобработку данных и создание модели.

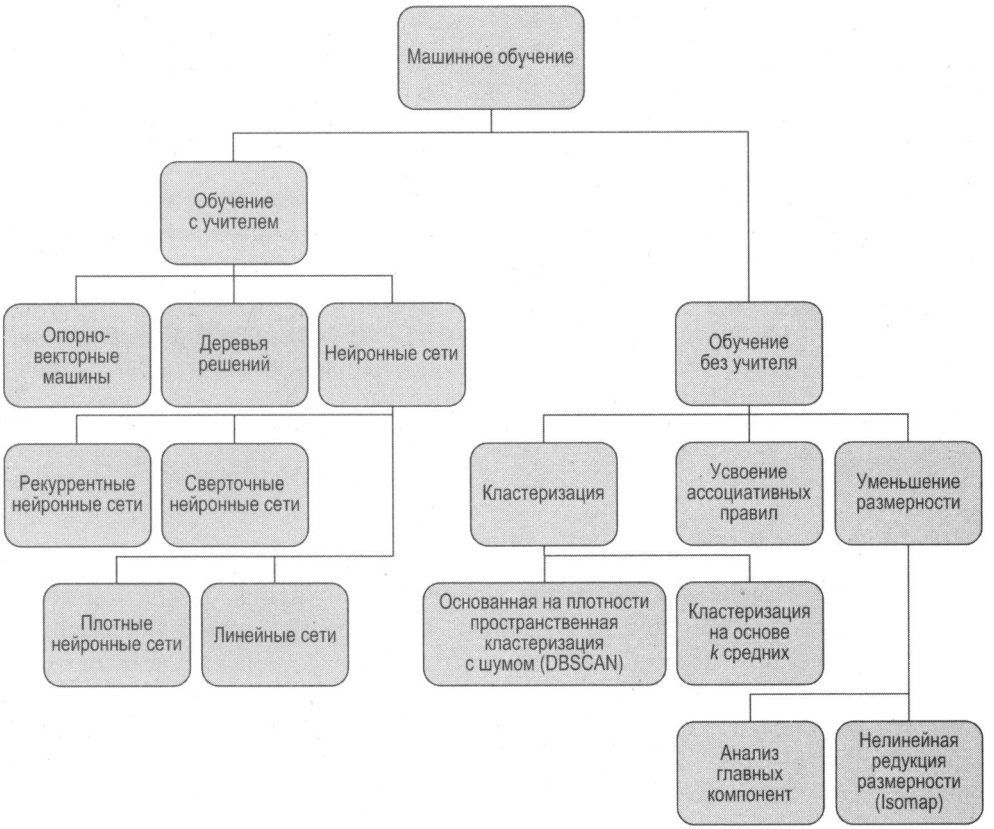


Рис. 1.1. Разбивка на разные типы машинного обучения с несколькими примерами каждого из них. Обратите внимание, что, хотя это и не включено диаграмму, такие нейронные сети, как автокодировщики, можно использовать для обучения без учителя

И наоборот, нейронные сети, имеющие только входной и выходной слои, являются еще одним подмножеством машинного обучения, именуемым *линейными моделями*. Линейные модели выявляют закономерности, которые они усваивают из дан-

ных, используя линейную функцию. *Деревья решений* — это модели машинного обучения, в которых данные используются для создания подмножества путей с различными ветвями. Эти ветви аппроксимируют результаты разных исходов из ваших данных. Наконец, *кластеризационные модели* ищут подобия между разными подмножествами данных и используют эти идентифицированные регулярности для группирования данных в кластеры.

Задачи машинного обучения (см. рис. 1.1) можно разделить на два типа: обучение с учителем и обучение без учителя. *Обучение с учителем* охватывает задачи, в которых метка эмпирического наблюдения в данных известна заранее. Например, сюда может входить назначение снимку метки "кошка" или назначение младенцу метки "2,3 кг при рождении". Вы подаете эти помеченные данные в свою модель в надежде, что она сможет усвоить достаточно знаний для того, чтобы назначать метки новым примерам. В случае же *обучения без учителя* метки в данных заранее не известны, и цель состоит в создании модели, способной отыскивать естественные группировки данных (этот процесс именуется *кластеризацией*), сжимать информационное содержимое (*уменьшение размерности*) или отыскивать ассоциативные правила. Бóльшая часть этой книги будет посвящена обучению с учителем, поскольку подавляющее большинство используемых в производстве моделей машинного обучения являются контролируемыми.

В обучении с учителем задачи обычно делятся на классификационные и регрессионные. *Классификационные модели* назначают входным данным метку (или метки) из заранее заданного множества категорий. Примеры классификационных задач включают определение типа породы домашнего животного на снимке, пометку документов тегами или предсказание о том, что та или иная транзакция является мошеннической. *Регрессионные модели* назначают непрерывные числовые значения входным переменным. Примеры регрессионных моделей включают предсказывание продолжительности велосипедной поездки, будущей выручки компании или цены продукта.

Данные и инженерия признаков

Данные лежат в основе любой задачи машинного обучения. Когда мы говорим о наборах данных, мы имеем в виду данные, используемые для тренировки, валидации и тестирования модели. Основная часть ваших данных будет состоять из *тренировочных данных* — данных, подаваемых в вашу модель в процессе тренировки. *Валидационные данные* — это данные, которые хранятся в тренировочном наборе и используются для оценивания результативности модели после каждой тренировочной эпохи (или прохождения через тренировочные данные). Результативность работы модели на валидационных данных используется для принятия решения о том, когда следует остановить тренировочный прогон, и для выбора *гиперпараметров*, таких как число деревьев в модели, основанной на случайном лесе. *Тестовые данные* — это данные, которые вообще не используются в процессе тренировки и задействуются для оценивания результативности работы натренированной модели. Отчеты о результативности работы модели машинного обучения должны базироваться не на тренировочных или валидационных тестах, а на незави-

симых тестовых данных. Важно, чтобы данные были разбиты таким образом, чтобы все три набора данных (тренировочный, тестовый, валидационный) имели сходные статистические свойства.

Данные для тренировки модели могут принимать разнообразные формы в зависимости от типа модели. Мы подразделяем *структурированные данные* на числовые и категориальные. Числовые данные содержат целочисленные и вещественные значения, а категориальные — данные, которые могут быть разбиты на конечное множество групп, таких как тип автомобиля или уровень образования. Структурированные данные также можно рассматривать как данные, которые обычно можно найти в электронной таблице. Мы будем использовать термин *"табличные данные"* взаимозаменяемо с термином "структурированные данные". *Неструктурированные данные* включают данные, которые невозможно четко представить. Обычно к таким данным относят текст произвольной формы, снимки, видео и аудио.

Числовые данные часто могут подаваться в модель машинного обучения непосредственно, тогда как другие данные, прежде чем они будут готовы к отправке в модель, требуют *предобработки*. Этот шаг, как правило, предусматривает нормализацию числовых значений или конвертирование нечисловых данных в числовой формат, который будет понятен вашей модели. Еще одним термином для предобработки является *инженерия признаков*. Мы будем использовать эти термины взаимозаменяемо.

По мере того как данные проходят процесс инженерии признаков, для их описания используются самые разные термины. Под *входной переменной* имеется в виду один столбец в наборе данных *до* его обработки, а под *признаком* — один столбец *после* его обработки. Например, вашей входной переменной может быть метка времени, а признаком — день недели. В целях конвертирования данных из метки времени в день недели вам понадобится выполнить некоторую предобработку данных. Шаг предобработки также можно называть *преобразованием данных*.

Экземпляр — это элемент, который вы хотите отправить в свою модель для получения предсказания. Экземпляром может быть строка в тестовом наборе данных (без столбца с меткой), снимок, который вы хотите классифицировать, или текстовый документ для отправки в модель анализа настроений. Имея в наличии набор признаков экземпляра, модель вычислит предсказанное значение. Для этого модель сперва тренируется на *тренировочных примерах*, которые увязывают экземпляр с меткой. *Тренировочный пример* ссылается на один экземпляр (строку) в наборе данных, который будет подан в модель. Отталкиваясь от сценария с меткой времени, полный тренировочный пример может включать "день недели", "город" и "тип автомобиля". *Метка* — это выходной столбец в наборе данных, т. е. элемент, который ваша модель предсказывает. Метка может ссылаться как на целевой столбец в наборе данных (так называемую метку *эмпирического наблюдения*[3]), так и на данные, выдаваемые моделью на выходе (так называемое *предсказание*). Меткой об-

[3] Метка эмпирического наблюдения (ground truth label) — это идеальный результат, полученный на основе прямых наблюдений или полевых исследований. — *Прим. перев.*

разца для описанного выше тренировочного примера может быть "продолжительность поездки" — в данном случае вещественное значение, обозначающее минуты.

После того как вы собрали набор данных и определили признаки для модели, следует выполнить *валидацию данных* — процесс, заключающийся в вычислении статистических величин на ваших данных, усвоении схемы и оценивании набора данных с целью выявления таких проблем, как дрейф и асимметрия между тренировкой и обслуживанием. Оценивание различных статистических величин на ваших данных помогает обеспечивать сбалансированное представление каждого признака в наборе данных. В тех случаях, когда собрать больше данных невозможно, понимание сбалансированного соотношения данных поможет вам разработать модель, способную это учитывать. Под усвоением схемы понимается определение по каждому признаку его типа данных и выявление тренировочных примеров, в которых некоторые значения могут быть неправильными или отсутствовать. Наконец, валидация данных помогает выявлять несогласованности, которые могут повлиять на качество тренировочного и тестового наборов. Например, может оказаться так, что бóльшая часть вашего тренировочного набора данных содержит примеры из *будних дней*, в то время как ваш тестовый набор содержит главным образом примеры из *выходных дней*.

Рабочий поток машинного обучения

Первым шагом в типичном рабочем потоке машинного обучения является *тренировка* — процесс передачи тренировочных данных в модель, чтобы та получила возможность учиться распознавать закономерности. После тренировки следующим шагом в этом потоке выступает тестирование уровня результативности вашей модели на данных за пределами тренировочного набора. Этот шаг называется *оцениванием модели*. Вы вполне можете выполнять тренировку и оценивание многократно, генерируя дополнительные признаки и дорабатывая архитектуру модели. После того как вы будете довольны результативностью модели во время ее оценивания, вы, вероятно, захотите превратить свою модель в сервис, чтобы другие имели возможность обращаться к ней и получать предсказания. Мы используем термин *"обслуживание"* (serving) для обозначения приема входящих запросов и отправки предсказаний обратно в результате развертывания модели в качестве сервиса. Инфраструктура обслуживания запросов может находиться в облаке, локально или на периферийном устройстве.

Процесс отправки новых данных в модель и использования результатов ее работы называется *предсказанием*. Допустимо как генерирование предсказаний из локальных моделей, которые еще не были развернуты как сервисы, так и получение предсказаний из развернутых моделей. Во втором случае мы будем ссылаться на предсказание в режиме реального времени или на пакетное предсказание. *Предсказание в режиме реального времени* используется тогда, когда требуется получать предсказания на нескольких примерах почти в реальном времени. В таком предсказании акцент делается на низкой задержке. *Пакетное предсказание* относится к генерированию предсказаний на крупном наборе данных в офлайновом режиме. Задания пакетного предсказания занимают больше времени, чем предсказание в режиме ре-

ального времени, и полезны для предварительного вычисления предсказаний (например, в рекомендательных системах) и анализа предсказаний вашей модели по крупной выборке новых данных.

Слово *"предсказание"* (prediction) уместно использовать тогда, когда речь идет о прогнозировании (forecasting) будущих значений, например о предсказании продолжительности поездки на велосипеде или о возможности оставления тележки покупок покупателем. Это слово будет менее очевидным в случае моделей, занимающихся классифицированием снимков и текста. Если модель анализирует текстовый отзыв о продукте и на выходе выдает, что он позитивный, то это на самом деле не "предсказание" (т. к. тут нет никакого будущего исхода). Поэтому для обозначения предсказаний в книге вы также встретите термин *"вывод"* (inference), который используется в ином смысле, чем в статистике, но на самом деле речь идет не о логическом рассуждении[4].

Нередко процессы сбора тренировочных данных, инженерии признаков, тренировки и оценивания модели выполняются отдельно от производственного конвейера. В этом случае вы станете переоценивать свое техническое решение всякий раз, когда будете понимать, что у вас накопились дополнительные данные в объеме, достаточном для тренировки новой версии вашей модели. В других ситуациях вы, возможно, будете получать новые данные постоянно и обрабатывать их непосредственно перед отправкой в модель для тренировки или для предсказания. Такой подход называется *потоковой передачей данных*. Для манипулирования потоковыми данными вам понадобится многоступенчатое решение с возможностью инженерии признаков, тренировки, оценивания и предсказания. Такие многоступенчатые решения называются *конвейерами ML*.

Инструментарий для работы с данными и моделями

Мы будем ссылаться на разнообразные облачные продукты Google Cloud, которые предоставляют инструментарий для решения задач, связанных с данными и машинным обучением. Эти продукты являются лишь одним из вариантов реализации упомянутых в этой книге паттернов дизайна и не представляют собой исчерпывающий список. Все включенные в книгу продукты являются бессерверными, что позволяет нам больше сосредоточиваться на реализации паттернов дизайна моделей машинного обучения, не тратя время на изучение лежащей в их основе инфраструктуры.

BigQuery[5] — это хранилище[6] корпоративных данных уровня всего предприятия, предназначенное для быстрого анализа крупных наборов данных с помощью SQL.

---

[4] В данном случае для термина inference используется перевод "предсказательный вывод" по аналогии с общепринятым термином "статистический вывод" из статистики. — *Прим. перев.*

[5] См. **https://oreil.ly/7PnVj**.

[6] Хранилище данных (data warehouse) — это крупная предметно-ориентированная, интегрированная, зависящая от времени и *неволатильная* коллекция данных для поддержания процесса принятия управленческих решений. Указанный тип хранения данных следует отличать от хранилища данных (data store, резерв данных), которое характеризуется *волатильностью* и меньшим масштабом данных. — *Прим. перев.*

Мы будем использовать BigQuery в примерах для сбора данных и инженерии признаков. Данные в BigQuery организованы в наборы данных (dataset), а набор данных может иметь несколько таблиц. Во многих примерах будут использоваться данные из Google Cloud Public Datasets[7], представляющие собой наборы бесплатных публичных данных, размещенных в BigQuery. Наборы данных Google Cloud Public Datasets состоят из сотен разных наборов данных, включая метеоданные NOAA с 1929 года, вопросы и ответы с веб-сайта Stack Overflow, открытый исходный код из GitHub, данные о рождаемости и многое другое. В целях строительства некоторых моделей в наших примерах мы будем использовать BigQuery Machine Learning[8] (BigQuery ML) — инструмент для строительства моделей из данных, хранящихся в BigQuery. С помощью BigQuery ML мы можем тренировать, оценивать и генерировать предсказания на наших моделях с помощью SQL. Указанный инструмент поддерживает классификационные и регрессионные модели, а также модели, построенные на неконтролируемой кластеризации. Кроме того, существует возможность генерировать предсказания путем импортирования в BigQuery ML ранее натренированных моделей TensorFlow.

Инструментальная платформа Cloud AI Platform[9] объединяет ряд продуктов для тренировки и использования конкретно-прикладных моделей машинного обучения в Google Cloud в качестве служб. В наших примерах мы будем использовать сервис тренировки AI Platform Training и сервис предсказания AI Platform Prediction. Сервис тренировки AI Platform Training предоставляет инфраструктуру для тренировки моделей машинного обучения в инфраструктуре Google Cloud. С помощью службы предсказания AI Platform Prediction вы можете разворачивать свои натренированные модели и генерировать на них предсказания, используя API. Обе службы поддерживают модели TensorFlow, scikit-Learn и XGBoost, а также работу контейнеров для моделей, построенных с помощью других фреймворков. Мы также будем ссылаться на Explainable AI (объяснимый искусственный интеллект, ИИ)[10], инструмент для интерпретирования результатов предсказаний вашей модели. Он предназначен для моделей, развернутых на платформе AI Platform.

Роли

Внутри организации имеется много разных должностных ролей, которые связаны с данными и машинным обучением. Далее представлены определения нескольких наиболее распространенных из них, на которые мы будем часто ссылаться в книге. Настоящая книга предназначена в первую очередь для исследователей данных, инженеров данных и инженеров ML, поэтому давайте начнем с них.

Исследователь данных (data scientist) — это человек, в центре внимания которого находится сбор, интерпретирование и обработка наборов данных. Применительно

---

[7] См. **https://oreil.ly/AbTaJ**.

[8] См. **https://oreil.ly/_VjVz**.

[9] См. **https://oreil.ly/90KLs**.

[10] См. **https://oreil.ly/lDocn**.

к машинному обучению исследователь данных может работать над сбором данных, генерированием признаков, созданием моделей и т. д. Исследователи данных часто работают на Python или R в среде блокнотов, и в организации они обычно являются первыми, кто разрабатывает модели машинного обучения.

Инженер данных (data engineer) сосредоточен на инфраструктуре и процессах, обеспечивающих работу с данными. Они помогают управлять тем, как в компании выполняется приемка данных, как работают конвейеры данных и как данные хранятся и передаются. Инженеры данных реализуют связанные с данными инфраструктуру и конвейеры.

Инженеры машинного обучения (ML engineer) выполняют те же задачи, что и инженеры данных. Они берут модели, разработанные исследователями данных, и занимаются управлением инфраструктурой и операциями, связанными с тренировкой и развертыванием этих моделей. Инженеры ML помогают строить промышленные системы, способные выполнять обновления моделей, управлять версиями моделей и обслуживать конечных пользователей модельными предсказаниями.

Чем меньше коллектив исследователей данных в компании и чем он гибче, тем вероятнее, что один и тот же человек будет играть несколько ролей. Если вы находитесь в такой ситуации, то, вполне возможно, читая три приведенных выше описания, вы видели себя частично во всех трех категориях. Обычно специалист начинает работу над проектом машинного обучения в качестве инженера данных и строит конвейеры данных для приемки данных. Затем он переходит к роли исследователя данных и строит модель (модели) ML. Наконец, он надевает шляпу инженера моделей ML и передает модель в производство. В крупных организациях проекты ML могут проходить одни и те же фазы, но в каждой из них могут участвовать разные коллективы.

Ученые-исследователи (research scientist), аналитики данных и разработчики тоже могут строить и использовать модели ИИ, но указанные должностные роли не являются целевой аудиторией этой книги.

В центре внимания ученых-исследователей в первую очередь находится отыскание и разработка новых алгоритмов для продвижения дисциплины ML. Сюда могут входить различные подобласти машинного обучения, такие как модельная архитектура, обработка естественного языка, компьютерное зрение, гиперпараметрическая настройка, обеспечение интерпретируемости моделей и многое другое. В отличие от обсуждаемых здесь других ролей ученые-исследователи бóльшую часть своего времени тратят не на строительство промышленных систем ML, а на прототипирование и оценивание новых подходов к ML.

Аналитики данных (data analyst) оценивают и собирают информацию из данных, а затем подготавливают ее для других команд внутри своей организации. Они, как правило, работают на языке SQL и в электронных таблицах и используют инструменты аналитики для создания визуализаций данных, чтобы делиться своими находками. Аналитики данных тесно сотрудничают с командами разработчиков, чтобы понять, как их выводы помогают решать бизнес-задачи и создавать ценные продукты. В то время как в центре внимания аналитиков данных находится выявление

трендов в существующих данных и извлечение из них информации, исследователи данных занимаются использованием этих данных для генерирования будущих предсказаний и автоматизирования или масштабирования генерации аналитической информации. По мере демократизации машинного обучения аналитики данных могут повышать свою квалификацию, становясь исследователями данных.

Разработчики (developer) отвечают за строительство промышленных систем, которые обеспечивают конечным пользователям возможность получать доступ к моделям ML. Они нередко участвуют в дизайне API, которые отправляют запросы к модели и возвращают предсказания в удобном для пользователя формате через веб- или мобильное приложение. Здесь могут участвовать модели, размещенные в облаке, либо модели, используемые как сервисы на периферийном устройстве. Разработчики задействуют API, реализованный инженерами моделей, создавая приложения и пользовательские интерфейсы для предоставления пользователям модельных предсказаний в удобной форме.

На рис. 1.2 показано, как разные роли взаимодействуют в процессе разработки модели машинного обучения в организации.

Рис. 1.2. С данными и машинным обучением связано много разных должностных ролей, и эти роли сотрудничают в рабочем потоке процессов ML — от приемки данных до разработки API и интерфейса конечного пользователя. Например, инженер данных работает с приемкой и валидацией данных и тесно сотрудничает с исследователями данных

Распространенные проблемы машинного обучения

Зачем нужна книга о паттернах машинного обучения? Процесс строительства систем ML несет в себе целый ряд уникальных проблем, влияющих на дизайн моделей ML. Понимание этих проблем поможет вам, специалисту-практику, разработать для себя опорный фреймворк, некую систему координат, для приведенных в этой книге технических решений.

Качество данных

Модели машинного обучения надежны лишь настолько, насколько качественны используемые для их тренировки данные. Если вы тренируете ML-модель на неполном наборе данных, на данных с плохо подобранными признаками или на данных, которые при использовании модели неточно представляют популяцию, то предсказания вашей модели будут прямым отражением этих данных. Как следствие, модели машинного обучения нередко называют моделями "мусор вошел, мусор вышел". Здесь мы выделим четыре важных компонента качества данных: точность, полноту, согласованность и своевременность.

Точность данных относится к признакам тренировочных данных и к соответствующим этим признакам меткам эмпирических наблюдений. Осведомленность об источнике данных и понимание любых потенциальных ошибок в процессе сбора данных помогают обеспечивать точность признаков. После сбора данных важно провести обстоятельный анализ с целью выявления опечаток, повторяющихся записей, несогласованностей значений в табличных данных, отсутствующих признаков и любых других ошибок, которые могут повлиять на качество данных. Например, дубликаты в тренировочном наборе данных могут привести к тому, что модель будет неправильно назначать более высокий вес этим точкам данных.

Точные метки данных важны в той же степени, что и точность признаков. Ваша модель опирается исключительно на метки эмпирических наблюдений, которые расположены в тренировочных данных, обновляя свои веса и минимизируя потерю. Неправильно размеченные тренировочные примеры, как следствие, могут привести к ошибочной точности модели. Например, предположим, что вы строите модель анализа настроений и 25% ваших "позитивных" тренировочных примеров были неправильно помечены как "негативные". Ваша модель будет иметь неточную картину того, что следует считать негативным настроением, и это будет непосредственно отражено в ее предсказаниях.

В целях понимания смысла *полноты* данных предположим, что вы тренируете модель на идентификацию породы кошек. Вы тренируете модель на обширном фотонаборе кошек, и результирующая модель в состоянии классифицировать снимки в одну категорию из 10 возможных (бенгальская, сиамская и т. д.) с точностью 99%. Однако, развернув свою модель в реальной среде, вы обнаруживаете, что многие ваши пользователи помимо фотографий кошек для классифицирования закачивают на сервер фотографии собак и разочаровываются результатами работы модели. Поскольку модель была натренирована идентифицировать только 10 разных пород кошек, то это всё, что она умеет делать. Эти 10 категорий пород являются, по сути, исчерпывающей "картиной мира" модели. Независимо от того, что подается в модель, резонно ожидать, что эти данные будут помещены в одну из этих 10 категорий и не более. Модель даже вполне способна классифицировать с высокой уверенностью в случае снимка, на котором объект совсем не похож на кошку. Вдобавок модель никак не сможет вернуть метку "не кошка", если эти данные и метка "не кошка" не были включены в тренировочный набор данных.

Еще одним аспектом полноты данных является обеспечение того, чтобы тренировочные данные содержали разнообразное представление каждой метки. Если

в примере с идентифицированием породы кошек все снимки содержат кошачью мордочку крупным планом, то ваша модель не сможет правильно идентифицировать снимок кошки сбоку или снимок кошки в полный рост. А вот пример с табличными данными: если вы строите модель для предсказания цен на недвижимость в каком-либо городе, но включаете тренировочные примеры домов только площадью более 2000 кв. футов, то результирующая модель будет работать плохо на малых домах.

Третьим аспектом качества данных является *согласованность* данных. В случае крупных наборов данных общепринято поручать работу по сбору и разметке данных группе людей. Наличие набора стандартов для этого процесса помогает обеспечивать согласованность всего набора данных, поскольку каждый участвующий в этом процессе человек неизбежно привнесет в процесс собственные искаженные взгляды. Как и полнота данных, несогласованность данных может обнаруживаться и в признаках данных, и в метках. В качестве примера несогласованных признаков предположим, что вы собираете атмосферные данные с температурных датчиков. Если каждый датчик был откалиброван по разным стандартам, то это приведет к неточным и ненадежным модельным предсказаниям. Несогласованности также могут относиться к формату данных. Если вы собираете данные о местоположении, то некоторые люди могут записывать полный адрес улицы как "*улица* Фруктовая", а другие могут сократить его, записав как "*ул.* Фруктовая". Единицы измерения, такие как мили и километры, тоже могут различаться по всему миру.

Что касается несогласованностей в разметке, то давайте вернемся к примеру с идентифицированием настроения в тексте. В этом случае при разметке тренировочных данных, скорее всего, люди не всегда будут согласны с тем, что считается позитивным и негативным. В целях решения этой проблемы вы можете задействовать несколько человек, каждый из которых будет помечать каждый пример в вашем наборе данных, а затем вы будете брать наиболее часто применяемую метку по каждому объекту. Осведомленность о привносимой разметчиком потенциальной искаженности и реализация систем с ее учетом обеспечат согласованность меток во всем наборе данных. Мы изучим понятие искаженности[11] в *главе 7 (см. разд. "Паттерн 30. Призма объективности")*.

Своевременность данных относится к задержке между моментом возникновения события и моментом его добавления в базу данных. Например, если вы собираете данные из журналов приложений, то для формирования журнала ошибок в базе журналов может потребоваться несколько часов. Для набора данных, записывающего транзакции по кредитным картам, может потребоваться один день с момента совершения транзакции, прежде чем она будет зарегистрирована в вашей системе. В целях обеспечения своевременности полезно записывать как можно больше ин-

[11] В общем смысле английский термин bias означает непропорциональный вес в пользу или против идеи либо предмета, обычно в форме *искаженности*, предвзятости или необъективности, тогда как в узком смысле он означает статистически обусловленную систематическую *смещенность* и по сути аналогичен коэффициенту сдвига по оси *y* в регрессии, который соответствует точке на этой оси, где прямая пересекает эту ось. — *Прим. перев.*

формации об отдельно взятой точке данных и обеспечивать, чтобы эта информация отражалась во время преобразования данных в признаки, предназначенные для модели машинного обучения. В частности, вы можете отслеживать метку времени наступления события и время ее добавления в ваш набор данных. Затем, работая над генерированием признаков, вы можете соответствующим образом учесть эти разницы во времени.

Воспроизводимость

В традиционном программировании результат работы программы воспроизводим и гарантирован. Например, если вы пишете программу на Python, которая переворачивает цепочку символов, то вы знаете, что слово "банан" на входе всегда будет возвращать "нанаб" на выходе. Аналогично, если в вашей программе есть дефект, из-за которого она неправильно переворачивает строковые значения с числами, вы можете отправить программу коллеге и надеяться, что он сможет воспроизвести ошибку, подав в нее те же данные, что и вы (если только дефект не связан с поддержанием программой какого-то неправильного внутреннего состояния, с различиями в архитектуре, такими как прецизионность[12] задания чисел с плавающей точкой, или различиями в исполнении, такими как обработка с использованием нескольких потоков исполнения).

В моделях машинного обучения случайность изначально встроена в качестве их неотъемлемого элемента. Во время тренировки веса ML-модели инициализируются случайными значениями. Эти веса затем постепенно сходятся в процессе тренировки, когда модель итеративно усваивает закономерности из данных. По указанной причине один и тот же модельный исходный код, заданный одними и теми же тренировочными данными, будет давать несколько разные результаты в разных тренировочных прогонах. В итоге возникает проблема с воспроизводимостью. Если вы тренируете модель с точностью 98,1%, то повторный тренировочный прогон не гарантирует достижения того же результата, что затрудняет проведение сравнений между экспериментами.

Для решения указанной проблемы воспроизводимости общепринято устанавливать случайное начальное число (seed), которое ваша модель использует, чтобы обеспечить применение одной и той же случайности, когда вы тренируете модель. В TensorFlow это делается путем выполнения инструкции `tf.random.set_seed(value)` в начале программы.

В scikit-learn многие утилитные функции для перетасовывания данных вдобавок также позволяют устанавливать случайное начальное число:

```
from sklearn.utils import shuffle
data = shuffle(data, random_state=value)
```

[12] Прецизионность (precision) — это мера глубины, в сущности "мелкозернистость" или разрешающая способность результатов измерений, тогда как точность (accuracy) — мера широты, или близости результатов измерений к истинному значению. — *Прим. перев.*

Имейте в виду, что с целью обеспечения повторимых, воспроизводимых результатов в разных экспериментах вам понадобится использовать одни и те же данные и одно и то же случайное начальное число во время тренировки модели.

Тренировка модели нацелена на отлов нескольких артефактов, которые нужно исправить для обеспечения воспроизводимости: используемые данные, механизм разбивки наборов данных на тренировочный и валидационный наборы, подготовка данных и гиперпараметры модели, а также переменные, такие как размер пакета и план скорости обучения (learning rate).

Воспроизводимость также касается зависимостей внутри фреймворка машинного обучения. В дополнение к ручной настройке случайного начального числа элементы случайности также реализованы внутри фреймворков. Указанные элементы исполняются при вызове функции тренировки модели. Если эта опорная реализация меняется от одной версии фреймворка к другой, то повторимость не гарантируется. Возьмем конкретный пример: если одна версия метода `train()` делает 13 вызовов функции `rand()`, а более новая версия того же фреймворка делает 14 вызовов, то использование разных версий между экспериментами приведет к немного разным результатам, даже с одинаковыми данными и исходным кодом модели. Выполнение рабочих нагрузок в контейнерах и стандартизация версий библиотек помогают обеспечивать воспроизводимость. В *главе 6* представлен ряд паттернов, позволяющих делать процессы ML воспроизводимыми.

Наконец, воспроизводимость может касаться тренировочной среды модели. Зачастую из-за крупных наборов данных и сложности тренировка многочисленных моделей требует значительного времени. Эту работу можно ускорить, задействуя распределительные стратегии, такие как параллелизм данных или параллелизм модели (*см. главу 5*). Однако вместе с таким ускорением возникает дополнительная проблема воспроизводимости при повторном выполнении исходного кода, в котором применяется распределенная тренировка.

Смещение данных

В то время как связи в моделях машинного обучения между переменными на входе и результатами на выходе, как правило, являются статическими, данные могут значительно изменяться с течением времени. Смещение данных касается задачи обеспечения актуальности моделей машинного обучения и того, чтобы предсказания моделей оставались точным отражением среды, в которой они используются.

Например, вы тренируете модель классифицировать заголовки новостных статей по таким категориям, как "политика", "бизнес" и "технология". Если вы тренируете и оцениваете свою модель на исторических новостных статьях XX века, то она, скорее всего, не будет работать так же хорошо на современных данных. Сегодня мы знаем, что статья со словом "смартфон" в заголовке, вероятно, посвящена технологиям. Однако модель, натренированная на исторических данных, этого слова знать не будет. В целях решения смещения дрейфа, важно постоянно обновлять тренировочный набор данных, перетренировывать модель и модифицировать вес, который модель назначает тем или иным группам входных данных.

В целях ознакомления с менее очевидным примером смещения давайте посмотрим на набор данных NOAA[13] о сильных штормах, используя BigQuery. Если бы мы тренировали модель предсказывать вероятность сильного шторма в заданном районе, то нам потребовалось бы учитывать траекторию изменений погодных данных во временно́й динамике[14]. На рис. 1.3 видно, что общее число зарегистрированных сильных штормов неуклонно растет с 1950 года.

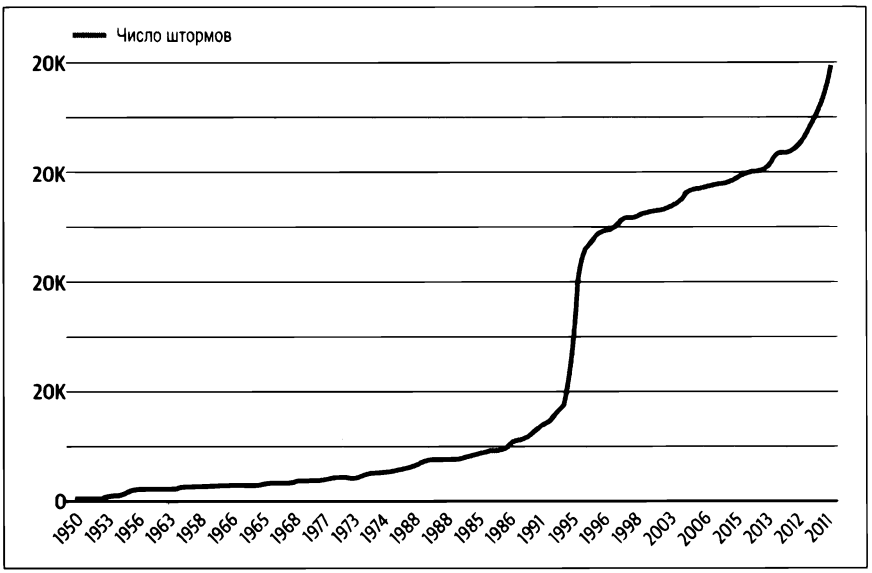


Рис. 1.3. Число сильных штормов, регистрируемых ежегодно с 1950 по 2011 год, по данным NOAA

Исходя из этого тренда, мы видим, что тренировка модели на данных до 2000 года генерировать предсказания на сильных штормах сегодня привела бы к неточным предсказаниям. В дополнение к увеличению суммарного числа зарегистрированных штормов важно учитывать и другие факторы, которые могли повлиять на данные, представленные на рис. 1.3. Например, технология наблюдения за штормами со временем улучшилась, особенно резко с появлением метеорологических радаров в 1990-х годах. В контексте признаков это может означать, что новые данные содержат больше информации о каждом шторме, а признак, имеющийся в современных данных, возможно, не наблюдался в 1950 году. Анализ данных помогает идентифицировать этот тип смещения и может сообщить правильное окно данных для использования во время тренировки. В *разд. "Паттерн 23. Мостовая схема"* представлен подход к манипулированию наборами данных, в которых обеспеченность признаками со временем улучшается.

---

[13] См. **https://oreil.ly/obzvn**.

[14] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/01_need_for_design_patterns/ml_challenges.ipynb**.

Масштаб

Проблема масштабирования присутствует на многих этапах типичного рабочего потока процессов машинного обучения. Скорее всего, вы столкнетесь с проблемами масштабирования во время сбора и предобработки данных, тренировки модели и обработки запросов. Во время приемки и подготовки данных для модели машинного обучения размер набора данных будет обусловливать инструментарий, необходимый для вашего технического решения. Нередко задача инженеров данных заключается в создании конвейеров данных, которые могут масштабироваться с целью манипулирования наборами данных с миллионами строк.

В части тренировки модели инженеры ML отвечают за определение инфраструктуры, необходимой для какого-либо тренировочного задания. В зависимости от типа и размера набора данных тренировка модели бывает трудоемкой и вычислительно дорогостоящей, требующей инфраструктуры (например, графических процессоров), специально предназначенной для рабочих нагрузок ML. Так, модели классифицирования снимков обычно требуют гораздо большей тренировочной инфраструктуры, чем модели, тренируемые исключительно на табличных данных.

В контексте обработки запросов инфраструктура, необходимая для поддержания возможности предоставлять предсказания от модели команде исследователей данных, полностью отличается от инфраструктуры, необходимой для работы модели, получающей миллионы запросов каждый час. Разработчики и инженеры ML, как правило, отвечают за решение задач масштабирования, связанных с развертыванием модели и обслуживанием запросов.

Большинство паттернов ML в этой книге будут полезны без учета организационной зрелости. Однако некоторые паттерны в *главах 6* и *7* решают задачи обеспечения отказоустойчивости и воспроизводимости по-разному, и выбор между ними часто сводится к конкретному варианту использования и способности вашей организации справляться со сложностями.

Несколько целевых установок

Хотя разработкой модели машинного обучения часто заведует всего одна команда, многие подразделения в организации так или иначе используют эту модель. И у них неизбежно могут быть разные представления о том, что считается успешной моделью.

Чтобы понять, как это происходит на практике, предположим, что вы строите модель для идентифицирования дефектных продуктов по снимкам. Как исследователь данных, вы стремитесь минимизировать перекрестно-энтропийную потерю вашей модели. Менеджер по продуктам, возможно, захочет сократить число неправильно классифицируемых и отправляемых клиентам дефектных продуктов. А исполнительный менеджер желает увеличить выручку на 30%. Каждая из этих целей отличается тем, подо что именно выполняется оптимизация, и оптимальное балансирование этих различающихся потребностей внутри организации нередко вырастает в сложную задачу.

Как исследователь данных, вы могли бы перевести потребности бизнеса в контекст вашей модели, сказав, что ложноотрицательные результаты в 5 раз дороже ложноположительных. И поэтому вам следует выполнять оптимизацию не под метрику прецизионности, а под метрику полноты, чтобы удовлетворять ее во время работы над дизайном модели. Тогда вы сможете найти оптимальный баланс между целью бизнеса оптимизировать модель под прецизионность и вашей целью минимизировать потери.

При определении целей для вашей модели важно учитывать потребности разных коллективов организации и то, как потребности каждого коллектива соотносятся с моделью. Проанализировав потребности бизнеса до выстраивания своего технического решения, вы сможете найти области компромисса и оптимально сбалансировать эти разные целевые установки.

Резюме

Паттерны представляют собой способ превратить знания и опыт экспертов в советы, которых могут придерживаться все специалисты-практики. Приводимые в этой книге паттерны отражают лучшие практические приемы и решения задач, часто возникающих во время работы над архитектурой, разработкой и развертыванием систем машинного обучения. Распространенные трудности в машинном обучении, как правило, связаны с качеством данных, воспроизводимостью, смещением данных, масштабированием и необходимостью удовлетворять разным целям.

На различных этапах жизненного цикла машинного обучения мы склонны использовать самые разные паттерны. Некоторые полезны при формулировании задач и оценивании их технической осуществимости. Большинство касается либо разработки, либо развертывания, и довольно много паттернов отражают взаимодействие между этими этапами.

ГЛАВА 2

Паттерны для представления данных

В основе любой модели машинного обучения лежит математическая функция, которая определена для работы только на отдельно взятых типах данных. В то же время реальные практические модели ML должны функционировать на данных, которые, возможно, окажутся непригодными для непосредственной их подстановки в математическую функцию. Математическое ядро дерева решений, например, оперирует булевыми переменными. Обратите внимание, что здесь мы говорим о математическом ядре дерева решений — само же программно-информационное обеспечение машинного обучения, в котором задействуется дерево решений, обычно также содержит функции для обучения оптимального дерева из данных, а также способы чтения и обработки разных типов числовых и категориальных данных. Так или иначе, лежащая в основе дерева решений математическая функция (рис. 2.1) работает на булевых переменных и использует такие операции, как AND (&& на рис. 2.1) и OR (+ на рис. 2.1).

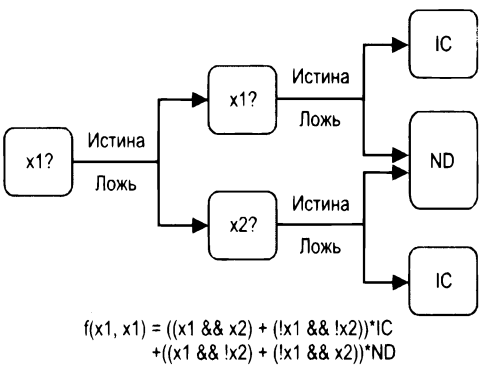


Рис. 2.1. В ядре модели машинного обучения, основанной на дереве решений, генерирующей предсказание о потребности младенца в интенсивной терапии, лежит математическая модель, работающая на булевых переменных

Допустим, что у нас есть дерево решений, которое предсказывает, что младенцу необходимо оказание интенсивной терапии (intensive care, IC) либо что он может быть выписан (normally discharged, ND), и предположим, что дерево решений принимает на входе две переменные — `x1` и `x2`. Натренированная модель может выглядеть примерно так, как показано на рис. 2.1.

Совершенно ясно, что переменные `x1` и `x2` должны быть булевыми для надлежащей работы `f(x1, x2)`. Предположим, что мы хотим, чтобы при классифицировании младенца как нуждающегося или не нуждающегося в интенсивной терапии модель учитывала две части информации: больницу, в которой младенец родился, и вес младенца[1]. Можем ли мы использовать больницу, в которой родился младенец, в качестве перменной на входе в дерево решений? Нет, потому что больница не принимает ни значения "истина", ни значения "ложь" и не может быть подана в оператор && (AND) ввиду математической несовместимости. Конечно же, значение больницы можно "сделать" булевым, выполнив такую операцию, как:

```
x1 = (hospital IN France)
```

вследствие которой переменная `x1` будет истинной, если больница находится во Франции, и ложной в противном случае. Точно так же не получится подать непосредственно в модель и вес младенца, но путем выполнения такой операции, как:

```
x1 = (babyweight < 3 kg)
```

мы сможем использовать больницу или вес младенца в качестве переменной на входе в модель. Это пример того, как входные данные (больница, сложный объект, или вес младенца, число с плавающей точкой) могут быть представлены в ожидаемом моделью (булевом) формате. Под такими манипуляциями подразумевается *представление данных*.

В книге мы будем использовать термин *"входная переменная"* (input) для представления подаваемых в модель реальных данных (например, вес младенца), а термин *"признак"* (feature) для представления преобразованных данных, на которых модель оперирует фактически (например, вес младенца меньше 3 кг). Процесс создания признаков для представления входных данных называется *инженерией признаков*, и поэтому мы можем рассматривать ее как метод отбора представления данных.

Разумеется, вместо жесткого кодирования параметров, таких как порог в 3 кг, мы бы предпочли, чтобы модель машинного обучения сама усваивала то, как создавать каждый узел, отбирая входную переменную и порог. Деревья принятия решений являются примером моделей машинного обучения, которые могут усваивать представление данных[2]. Многие приведенные в этой главе модели будут аналогично обучаться на *представлении данных*.

Паттерн "Векторные вложения" (Embeddings) является каноническим примером представления данных, которое глубокие нейронные сети могут усваивать самостоятельно. В векторном вложении усвоенное представление является плотным и менее размерным, чем у входной переменной, которое может быть разреженным. Алгоритм обучения должен извлекать наиболее важную информацию из входной переменной и представлять ее более сжато в признаке. Процесс обучения призна-

---

[1] С физической точки зрения, конечно, не вес младенца, а масса тела младенца. Однако на бытовом уровне говорят именно "вес младенца". — *Прим. ред.*

[2] Здесь усваиваемое представление данных состоит из веса младенца в качестве входной переменной, оператора "меньше, чем" и порога 3 кг.

ков для представления входных данных называется *извлечением признаков*, и мы можем рассматривать усвоенные представления данных (например, векторные вложения) как автоматически сгенерированные признаки.

Данные не обязательно должны представляться одной входной переменной — например, скошенное дерево решений создает булев признак путем пороговой фильтрации линейной комбинации из двух или более входных переменных. Дерево решений, в котором каждый узел может представлять только одну входную переменную, сводится к ступенчатой линейной функции, тогда как скошенное дерево решений, в котором каждый узел может представлять линейную комбинацию входных переменных, сводится к кусочно-линейной функции (рис. 2.2). Учитывая число шагов, которые придется усваивать, чтобы адекватно представлять линию, кусочно-линейная модель обучается проще и быстрее. Продолжением этой идеи является паттерн "Синтетический признак" (Feature Cross), который упрощает усвоение AND-связей между многозначными категориальными переменными.

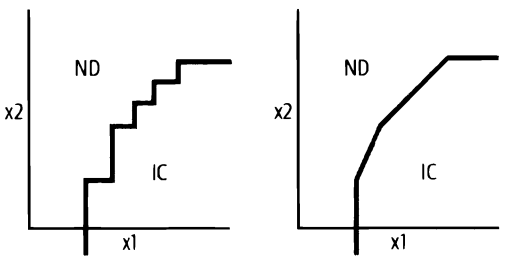


Рис. 2.2. Классификатор, основанный на дереве решений, в котором каждый узел может выполнять пороговую фильтрацию только одной входной переменной (x1 или x2), приведет к ступенчатой линейной граничной функции, тогда как классификатор, основанный на скошенном дереве, в котором узел может выполнять пороговую фильтрацию линейной комбинации входных переменных, приведет к кусочно-линейной граничной функции. Кусочно-линейная функция требует меньшего числа узлов и может обеспечивать бóльшую точность

Представление данных не обязательно должно усваиватся либо быть фиксированным — гибрид тоже возможен. Паттерн "Хешированный признак" (Hashed Feature) является детерминированным, но не требует, чтобы модель знала все потенциальные значения, которые та или иная входная переменная может принимать.

Все представления данных, которые мы рассматривали до сих пор, имеют соотношение "один-к-одному". Хотя мы могли бы представлять входные данные разных типов отдельно или каждый элемент данных только как один признак, бывает выгоднее использовать паттерн "Мультимодальный вход" (Multimodal Input). И это будет четвертый паттерн, который мы рассмотрим в данной главе.

Простые представления данных

Прежде чем мы углубимся в представления данных, синтетические признаки и многое другое, давайте рассмотрим более простые представления данных. Об этих простых представлениях можно думать как о распространенных в машинном обу-

чении *идиомах* — это не совсем паттерны, но тем не менее они задействуются в решениях очень часто.

Числовые входные значения

Большинство современных крупномасштабных моделей машинного обучения (случайные леса, опорно-векторные машины, нейронные сети) работают на числовых значениях, и поэтому, если на входе мы имеем числа, мы можем пропускать их через модель без изменений.

Почему желательно выполнять нормализацию

Поскольку в ML используется оптимизатор, настроенный на работу с числами из промежутка [–1; 1], нередко бывает полезным выполнять нормализацию[3] числовых значений, сводя их к этому интервалу значений.

> **Зачем нормализовать числовые значения из промежутка [–1; 1]?**
>
> Для схождения оптимизаторов, основанных на градиентном спуске, требуется все большее число шагов по мере увеличения кривизны функции потери. Это обусловлено тем, что производные признаков с более крупными относительными магнитудами также будут тяготеть и к более крупным значениям, и поэтому будут приводить к аномальным обновлениям весов. Крупные обновления весов потребуют большего числа шагов для своего схождения и, как следствие, увеличат вычислительную нагрузку.
>
> "Центрирование" данных из промежутка [–1; 1] делает функцию ошибки более сферической. Поэтому модели, натренированные с использованием преобразованных данных, как правило, сходятся быстрее и, следовательно, тренируются быстрее/дешевле. Дополнительно промежуток [–1; 1] обеспечивает наивысшую прецизионность плавающей точки.

Быстрый тест с одним из встроенных в библиотеку scikit-learn наборов данных может доказать этот аргумент (ниже приведена выдержка из репозитория[4] исходного кода этой книги):

```
from sklearn import datasets, linear_model
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
raw = diabetes_X[:, None, 2]
max_raw = max(raw)
```

---

[3] Нормализация (scaling) — это сужение или расширение диапазона данных, обычно для приведения многочисленных переменных к одинаковой шкале измерения. Не следует путать с масштабированием (на англ. тоже scaling), т. е. оптимизацией под возрастающий объем данных и/или вычислений. — *Прим. перев.*

[4] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/simple_data_representation.ipynb**.

```
min_raw = min(raw)
scaled = (2*raw - max_raw - min_raw)/(max_raw - min_raw)

def train_raw():
    linear_model.LinearRegression().fit(raw, diabetes_y)

def train_scaled():
    linear_model.LinearRegression().fit(scaled, diabetes_y)

raw_time = timeit.timeit(train_raw, number=1000)
scaled_time = timeit.timeit(train_scaled, number=1000)
```

Выполнив приведенный выше фрагмент кода, мы получили почти 9%-ное улучшение этой модели, в которой используется только один входной признак. С учетом числа признаков в типичной модели машинного обучения экономия будет прирастать с каждым новым признаком.

Еще одна важная причина нормализации заключается в том, что некоторые алгоритмы и технические приемы машинного обучения очень чувствительны к относительным магнитудам разных признаков. Например, алгоритм кластеризации на основе *k* средних, в котором в качестве меры точности используется евклидово расстояние, будет в итоге сильно опираться на признаки с более крупными магнитудами. Отсутствие нормализации также влияет на эффективность регуляризации L1 или L2, поскольку магнитуда весов у признака зависит от магнитуды значений этого признака, и поэтому различные признаки будут затрагиваться регуляризацией по-разному. Нормализуя все признаки из промежутка [–1; 1], мы обеспечиваем отсутствие большой разницы в относительных магнитудах разных признаков.

Линейная нормализация

Чаще всего задействуются четыре формы нормализации.

- *Минимаксная нормализация (minimax scaling).*

  Числовое значение нормализуется линейно, вследствие чего минимальное значение, которое может принимать входная переменная, равняется –1, а максимально возможное значение равно +1:

  ```
  x1_scaled = (2*x1 - max_x1 - min_x1)/(max_x1 - min_x1)
  ```

  Проблема с минимаксной нормализацией заключается в том, что максимальное и минимальное значения (`max_x1` и `min_x1`) должны вычисляться из тренировочного набора данных, а они часто содержат выбросные значения. Реальные данные нередко сжимаются до очень узкого интервала в полосе [–1; 1].

- *Усечение (clipping, в сочетании с минимаксной нормализацией).*

  Усечение помогает решать проблему выбросов, используя "разумные" значения вместо минимума и максимума, вычисляемых из тренировочного набора данных. Числовое значение нормализуется между этими двумя разумными границами линейно, а затем усекается, чтобы можно было уложиться в промежуток [–1; 1]. Это приводит к тому, что выбросы трактуются как –1 или 1.

- *Нормализация на основе z-оценки (z-score normalization).*

  Z-оценка решает проблему выбросов без необходимости предварительно знать о том, каким является разумный интервал, линейно нормализуя входные данные с использованием среднего значения и стандартного отклонения, вычисляемых над тренировочным набором данных:

  ```
  x1_scaled = (x1 - mean_x1)/stddev_x1
  ```

  Название этого метода отражает тот факт, что нормализуемое значение имеет нулевое среднее значение и нормализуется стандартным отклонением, вследствие чего оно имеет единичную дисперсию над тренировочным набором данных. Нормализуемое значение не ограничено, но в большинстве случаев находится в промежутке [–1; 1] (в нем лежит 67% значений, если опорное распределение является нормальным). Значения вне этого промежутка появляются тем реже, чем крупнее становится их абсолютное значение, но все же присутствуют.

- *Винзоризация (winsorizing).*

  В винзоризации[5] используется эмпирическое распределение данных тренировочного набора, которое усекается до границ, заданных 10-м и 90-м процентилями значений данных (или 5-м и 95-м процентилями и т. д.). Виндзоризованное значение нормализуется с помощью минимаксного метода.

Все рассмотренные до сих пор методы нормализуют данные линейно (в случае усечения и виндзоризации — линейно в пределах типичного интервала). Минимакс и усечение, как правило, лучше всего работают для равномерно распределенных данных, а *z*-оценка — для нормально распределенных данных. Воздействие разных функций нормализации на столбец `mother_age` в примере с предсказанием веса младенца показано на рис. 2.3 (полный исходный код см. в репозитории на Github[6]).

Не выбрасывайте "выбросы"

Обратите внимание, что мы определили процедуру усечения как взятие нормализованных значений меньше –1, трактуя их как –1, и нормализованных значений больше 1, трактуя их как 1. Мы не исключаем такие "выбросы" из набора — мы ожидаем, что модель машинного обучения столкнется с такими выбросами в реальных условиях. Возьмем, к примеру, младенцев, рожденных 50-летними матерями. Поскольку в нашем наборе данных пожилых матерей недостаточно много, усечение в итоге приведет к тому, что все матери (к примеру) старше 45 лет будут трактоваться как 45-летние. Эта же трактовка будет применяться и в реальных условиях, а значит, наша модель сможет учитывать и пожилых матерей. Модель не научилась бы отражать выбросы, если бы мы просто отбросили все тренировочные примеры, в которых младенцы были рождены матерями в возрасте 50+!

---

[5] Винзоризация (winsorizing) — это преобразование статистической величины путем лимитирования экстремальных значений в статистических данных с целью уменьшения влияния возможно мнимых выбросов. Данный метод назван в честь инженера-биостатистика Чарльза П. Уинзора (1895–1951). Эффект винзоризации тот же, что и при амплитудном ограничении в обработке сигналов. — *Прим. перев.*

[6] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/simple_data_representation.ipynb**.

На это можно посмотреть и с другой стороны. Хотя совершенно допустимо выбрасывать невалидные данные, недопустимо выбрасывать *валидные данные*. Так, вполне оправданно выбрасывать строки данных, в которых поле возраста матери (`mother_age`) является отрицательным, потому что такое значение, вероятно, является ошибкой ввода данных. В реальных условиях валидация входной формы будет обеспечивать, чтобы принимающий сотрудник вносил возраст матери повторно. Однако мы не вправе выбрасывать строки, в которых поле `mother_age` равно 50, потому что 50 является совершенно валидным входным значением, и мы ожидаем, что столкнемся с 50-летними матерями после развертывания модели в промышленной среде.

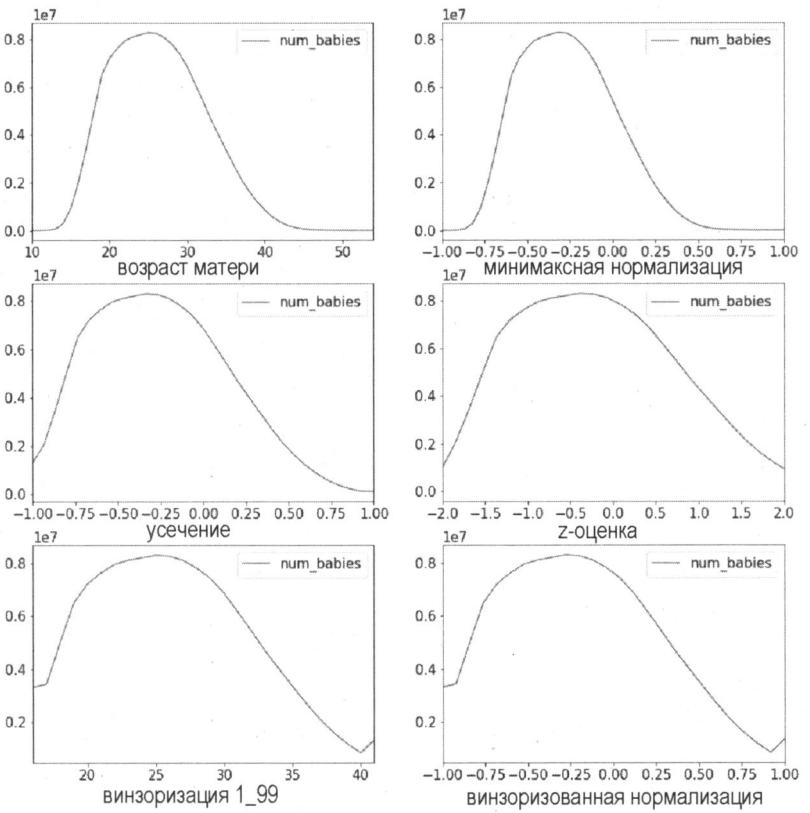


Рис. 2.3. Гистограмма поля `mother_age` (возраст матери) в примере с предсказанием веса младенца показана на левом верхнем графике, а разные функции нормализации (см. метку оси *x*) показаны на остальных графиках

На рис. 2.3 обратите внимание на то, что `minmax_scaled` помещает значения `x` в желательный промежуток [–1; 1], но продолжает удерживать значения на крайних концах распределения, где примеров недостаточно. Усечение сужает многие проблемные значения, но требует получения строго правильных порогов усечения — здесь медленное снижение числа детей вместе с возрастом матерей старше 40 лет создает

проблемы в установлении жесткого порога. Винзоризация, подобно усечению, требует получения строго правильных процентильных пороговых значений. Нормализация на основе *z*-оценки улучшает интервал (но не ограничивает значения промежутком [–1; 1]) и выталкивает проблемные значения дальше наружу. Из этих трех методов нулевое нормирование (т. е. *z*-оценивание) лучше всего работает для поля `mother_age`, потому что сырые возрастные значения напоминают колоколообразную кривую. Для других задач лучше использовать минимаксную нормализацию, усечение или винзоризацию.

Нелинейные преобразования

Что делать, если наши данные асимметричны и распределены ни равномерно, ни как колоколообразная кривая? В этом случае перед нормализацией входных данных лучше применить к ним нелинейное преобразование. Одна из распространенных хитростей заключается в том, чтобы брать логарифм входного значения перед его нормализацией. Другие распространенные преобразования включают сигмоидное и полиномиальное разложения (квадрат, квадратный корень, куб, кубический корень и т. д.). Мы будем знать о том, что получили хорошую функцию преобразования по распределению преобразованного значения, которое должно стать равномерным или нормально распределенным.

Предположим, что мы строим модель для предсказания продаж научно-популярной книги. Одной из переменных на входе в модель является популярность страницы Википедии, которая соответствует теме книги. Число просмотров страниц в Википедии, однако, сильно асимметрично и занимает большой динамический диапазон (см. левую панель рис. 2.4: асимметричное распределение сильно искажено в сторону редко просматриваемых страниц, но наиболее распространенные страницы просматриваются десятки миллионов раз). Взяв логарифм просмотров, затем взяв корень четвертой степени из этого логарифмического значения и линейно пронормализировав результат, мы получим нечто, находящееся в желательном диапазоне и несколько колоколообразное. Подробная информация об исходном коде запроса данных Википедии, применении этих преобразований и генерировании приведенного на рис. 2.4 графика находится в репозитории[7] на GitHub этой книги.

Довольно трудно разработать линеаризирующую функцию, которая делает распределение похожим на колоколообразную кривую. Более простой подход заключается в группировании числа просмотров в корзины, выбирая границы корзин так, чтобы те укладывались в желательное распределение выходных данных. Принципиальным подходом к выбору этих корзин является гистограммное выравнивание, при котором корзины гистограммы выбираются на основе квантилей сырого распределения (см. третью гистограмму на рис. 2.4). В идеальной ситуации гистограммное выравнивание приводит к равномерному распределению (хотя и не в этом случае из-за повторяющихся значений в квантилях).

---

[7] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/simple_data_representation.ipynb**.

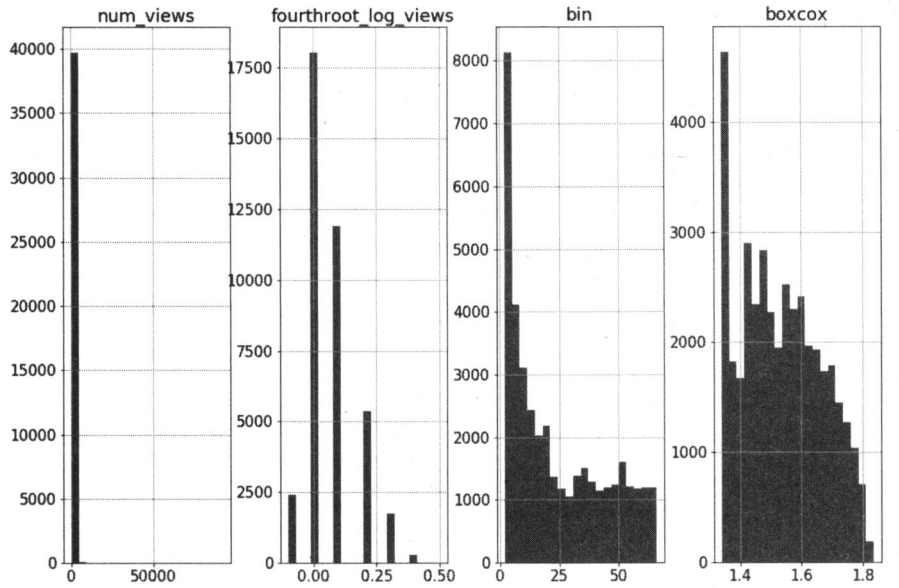


Рис. 2.4. На гистограмме слева распределение числа просмотров страниц Википедии имеет сильную асимметрию и занимает большой динамический диапазон. Вторая гистограмма демонстрирует, что проблемы могут быть решены путем последовательного преобразования числа просмотров с использованием логарифма, степенной функции и линейной нормализации. Третья гистограмма показывает эффект выравнивания, а четвертая — эффект преобразования Бокса — Кокса

В целях выполнения гистограммного выравнивания в BigQuery можно сделать следующее:

```
ML.BUCKETIZE(num_views, bins) AS bin
```

где корзины получаются из:

```
APPROX_QUANTILES(num_views, 100) AS bins
```

Более подробная информация находится в блокноте репозитория[8] исходного кода этой книги.

Еще один метод манипулирования асимметричными распределениями состоит в использовании технического приема, именуемого параметрическим преобразованием, такого как *преобразование Бокса — Кокса* (Box — Cox transform). Преобразование Бокса — Кокса выбирает свой единственный параметр, лямбда, управляющий гетероскедастичностью[9], вследствие которой дисперсия больше не зависит от магнитуды. Здесь дисперсия между редко просматриваемыми страницами Википедии будет намного меньше, чем дисперсия между часто просматриваемыми страницами, и преобразование Бокса — Кокса пытается уравнять дисперсию во всех

[8] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/simple_data_representation.ipynb**.

[9] Гетероскедастичность (heteroskedasticity) — это ситуация, когда некоторые диапазоны результата показывают более высокую дисперсию. — *Прим. перев.*

диапазонах числа просмотров. Это можно сделать с помощью пакета SciPy языка Python:

```
traindf['boxcox'], est_lambda = (
    scipy.stats.boxcox(traindf['num_views']))
```

Параметр, вычисленный над тренировочным набором данных (`est_lambda`), затем используется для преобразования других значений:

```
evaldf['boxcox'] = scipy.stats.boxcox(evaldf['num_views'], est_lambda)
```

Массив чисел

Иногда входные данные представляют собой массив чисел. Если массив имеет фиксированную длину, то представление данных может быть довольно простым: сгладить массив и рассматривать каждую позицию как отдельный признак. Но нередко массив будет иметь переменную длину. Например, одной из переменных на входе в модель предсказания продаж научно-популярной книги может быть объем продаж всех предыдущих книг по данной теме. Примером входной переменной может быть:

```
[2100, 15200, 230000, 1200, 300, 532100]
```

Очевидно, что длина этого массива будет варьироваться в каждой строке, потому что количество книг, опубликованных на различные темы, разное.

Распространенные идиомы манипулирования массивами чисел включают следующие:

- Представление входного массива в терминах его совокупных статистических величин. Например, мы могли бы использовать длину (т. е. количество предыдущих книг по теме), среднее, медиану, минимум, максимум и т. д.
- Представление входного массива в терминах его эмпирического распределения, т. е. по 10-му/20-му/... процентилю и т. д.
- Представление входного массива последними тремя или каким-либо иным фиксированным числом элементов, в случае если массив каким-то образом упорядочен (например, по времени или по размеру). Для массивов из менее трех элементов признак дополняется отсутствующими значениями до длины три.

Все они в итоге представляют массив переменной длины как признак фиксированной длины. Мы могли бы также сформулировать эту задачу как задачу прогнозирования на основе временно́го ряда, как задачу предсказания продаж следующей книги по теме на основе временно́й истории продаж предыдущих книг. Трактуя продажи предыдущих книг как входную переменную в виде массива, мы исходим из того, что наиболее важными факторами в предсказании продаж книги являются не временная непрерывность объемов продаж, а характеристики самой книги (автор, издатель, отзывы и т. д.).

Категориальные входные значения

Поскольку большинство современных крупномасштабных моделей машинного обучения (случайные леса, опорно-векторные машины, нейронные сети) оперируют числовыми значениями, категориальные входные значения должны представляться числами.

Простое перечисление возможных значений и их соотнесение с порядковой шкалой будет работать слабо. Предположим, что одной из переменных на входе в модель, которая предсказывает продажи научно-популярной книги, является язык, на котором она написана. Мы не сможем просто создать таблицу соответствий (табл. 2.1).

Таблица 2.1. *Таблица соответствий на основе одного признака*

Категориальное входное значение	Числовой признак
Английский	1.0
Китайский	2.0
Немецкий	3.0

Это обусловлено тем, что модель машинного обучения позже попытается интерполировать данные между популярностью книг на немецком и английском языках, получив популярность книги на китайском! Поскольку между языками нет порядковых отношений, нам нужно использовать категориальное числовое соотнесение, которое позволяет модели самостоятельно усваивать рынок книг, написанных на этих языках.

Кодирование с одним активным состоянием

Простейшим методом соотнесения категориальных переменных при обеспечении независимости переменных является *кодирование с одним активным состоянием*[10]. В нашем примере категориальная входная переменная будет преобразована в трехэлементный признаковый вектор с использованием соотнесения в таблице соответствия (табл. 2.2).

Таблица 2.2. *Таблица соответствий на основе трех признаков*

Категориальный вход	Числовой признак
Английский	[1.0, 0.0, 0.0]
Китайский	[0.0, 1.0, 0.0]
Немецкий	[0.0, 0.0, 1.0]

[10] Словосочетание "кодирование с одним активным состоянием" (one-hot encoding) пришло из терминологии цифровых интегральных микросхем, в которой оно описывает конфигурацию микросхемы, допускающую, чтобы только один бит был положительным (активным). — *Прим. перев.*

Кодирование с одним активным состоянием требует, чтобы мы знали *словарь* категориальных входных данных заранее. Здесь словарь состоит из трех лексем (английский, китайский и немецкий), а длина результирующего признака равна размеру этого словаря.

Кодирование с использованием фиктивных переменных или кодирование с одним активным состоянием?

Вообще, двухэлементного признакового вектора достаточно, чтобы обеспечить уникальное соответствие для словаря размером 3 (табл. 2.3).

Таблица 2.3. *Таблица соответствий на основе двух признаков*

Категориальный вход	Числовой признак
Английский	`[0.0, 0.0]`
Китайский	`[1.0, 0.0]`
Немецкий	`[0.0, 1.0]`

Это называется *кодированием с использованием фиктивных переменных*[11]. Поскольку представление с использованием фиктивных переменных более компактное, оно предпочтительно используется в статистических моделях, которые работают лучше, когда входные данные линейно независимы.

Однако в современных алгоритмах машинного обучения не требуется, чтобы поступающие на их вход данные были линейно независимыми, и в них используются такие методы, как регуляризация L1, служащие для подрезания избыточных входных данных. Дополнительная степень свободы позволяет каркасу прозрачно обращаться с недостающим входным значением в производстве как с серией одних нулей (табл. 2.4).

Таблица 2.4. *Таблица соответствий четырех входных категорий трем признакам*

Категориальный вход	Числовой признак
Английский	`[1.0, 0.0, 0.0]`
Китайский	`[0.0, 1.0, 0.0]`
Немецкий	`[0.0, 0.0, 1.0]`
(отсутствует)	`[0.0, 0.0, 0.0]`

Поэтому многие фреймворки машинного обучения нередко поддерживают только кодирование с одним активным состоянием.

[11] При кодировании с использованием фиктивных переменных (dummy encoding) для передачи всей необходимой информации используются только нули и единицы. В общем случае при наличии k атрибутов или полей будет $k - 1$ кодированных переменных. Каждая фиктивная переменная использует одну степень свободы, поэтому у k атрибутов будет $k - 1$ степеней свободы. — *Прим. перев.*

В некоторых случаях бывает полезно трактовать числовое входное значение как категориальное и ставить ему в соответствие столбец, кодированный с одним активным состоянием.

- *Когда числовая входная переменная является индексом.*

  Например, если мы пытаемся предсказать уровень трафика и одной из входных переменных является день недели, то мы можем трактовать день недели как числовой (1, 2, 3, ..., 7), но полезно осознавать, что день недели здесь не отражает непрерывную шкалу, а является просто индексом. Лучше трактовать его как категориальный (воскресенье, понедельник, ..., суббота), потому что индексация является произвольной. В какой день должна начинаться неделя? В воскресенье (как в США), в понедельник (как во Франции) или же в субботу (как в Египте)?

- *Когда связь между входной переменной и меткой не является непрерывной.*

  Склонить чашу весов в сторону трактовки дня недели как категориального признака должен тот факт, что уровень трафика в пятницу не зависит от уровня трафика в четверг и субботу.

- *Когда есть преимущества в группировке числовой переменной в корзины.*

  В большинстве городов уровень трафика зависит от того, является ли день недели выходным либо будним, и это может варьироваться в зависимости от географического местоположения (в большинстве стран мира выходными являются суббота и воскресенье, хотя в некоторых исламских странах это четверг и пятница). Было бы полезно тогда трактовать день недели как булев признак (выходной либо будний день). Соотнесение, при котором число разных входных значений (здесь семь) больше числа отличимых значений признака (здесь два), называется группировкой в корзины[12]. Обычно группировка в корзины выполняется в терминах интервалов: например, мы можем задать группировку для поля материнского возраста `mother_age` в интервалы, которые заканчиватся на 20, 25, 30 и т. д., и трактовать каждую корзину как категориальную, но следует понимать, что при этом теряется порядковая природа переменной `mother_age`.

- *Когда мы хотим трактовать разные значения числовой входной переменной как независимые в том, что касается их влияния на метку.*

  Например, вес младенца зависит от многоплодности[13] беременности, т. к. двойняшки и тройняшки, как правило, весят меньше, чем одноплодный младенец. Таким образом, если младенец с меньшим весом входит в состав тройни, то он может быть здоровее, чем близнец с тем же весом. В этом случае мы могли бы соотнести многоплодность с категориальной переменной, поскольку категориальная переменная позволяет модели усваивать независимые настраиваемые параметры для разных значений многоплодности. Разумеется, мы можем сделать

---

[12] Группировка данных в корзины (bucketing, binning) представляет собой один из способов обобщения, при котором последовательность более или менее непрерывных значений группируется в меньшее число интервальных корзин. — *Прим. перев.*

[13] Если близнецы, то многоплодность (plurality) равна 2. Если тройня, то многоплодность равна 3.

это только в том случае, если в нашем наборе данных достаточно примеров двойняшек и тройняшек.

Массив категориальных переменных

Иногда входные данные представляют собой массив категорий. Если массив имеет фиксированную длину, то мы можем трактовать каждую позицию массива как отдельный признак. Но часто массив будет иметь варьирующуюся длину. Например, одной из переменных на входе в модель рождаемости может быть тип предыдущих родов у этой матери:

```
[Индуцированный, Индуцированный, Естественный, Кесарев]
```

Очевидно, что длина этого массива будет варьироваться в каждой строке, потому что для каждого младенца существуют разное количество старших братьев или сестер.

Общепринятые идиомы манипулирования массивами категориальных переменных включают следующие.

- *Подсчет числа* появлений каждого элемента словаря. Так, в нашем примере данные будут представлены массивом `[2, 1, 1]` исходя из того, что словарь состоит из лексем "Индуцированный", "Естественный" и "Кесарев" (в указанном порядке). Теперь он представляет собой числовой массив фиксированной длины, который можно сгладить и использовать в позиционном порядке. Если у нас массив, в котором элемент может встречаться только один раз (например, массив из языков, на которых человек говорит), или признак просто указывает на присутствие, а не количество (например, была ли у матери когда-либо операция кесарева сечения или нет), то количество в каждой позиции будет равно 0 или 1, и это называется *кодированием с несколькими активными состояниями* (multi-hot encoding).
- Во избежание крупных чисел можно использовать *относительную частоту* вместо количества. В нашем примере данные будут представлены массивом `[0.5, 0.25, 0.25]` вместо `[2, 1, 1]`. Пустые массивы (первенцы без предыдущих братьев или сестер) представляются как `[0, 0, 0]`. В технологии обработки естественного языка относительная частота слова в целом нормализуется относительной частотой документов, содержащих это слово, давая показатель *частоты термина — обратной частоты документа*[14] (term frequency — inverse document frequency, TF-IDF). Показатель TF-IDF отражает степень уникальности слова для документа.
- Представление входного массива последними тремя элементами, в случае если массив каким-то образом упорядочен (например, в порядке времени). Массивы короче трех элементов заполняются отсутствующими значениями.
- Представление массива совокупными статистическими величинами, например длиной массива, модой (наиболее распространенной записью), медианой, 10-м/20-м/... процентилем и т. д.

[14] См. **https://oreil.ly/kNYHr**.

Из приведенных выше идиом наиболее распространены идиома подсчета числа появлений и идиома относительной частоты. Обратите внимание, что они обе являются обобщением кодирования с одним активным состоянием: если бы у младенца не было старших братьев или сестер, то данные были бы представлены массивом `[0, 0, 0]`, а если бы у младенца был один старший брат или сестра, родившиеся естественным путем, то данные были бы представлены массивом `[0, 1, 0]`.

Ознакомившись с простыми представлениями данных, давайте обсудим паттерны, которые помогают представлять данные.

ПАТТЕРН 1. Хешированный признак

Паттерн "Хешированный признак" (Hashed Feature) решает три возможные проблемы, связанные с категориальными признаками: неполный словарь, размер модели вследствие кардинальности и холодный пуск. Указанный паттерн делает это путем группирования категориальных признаков и принятия компромисса между коллизиями в представлении данных.

Постановка задачи

Кодирование категориальной входной переменной с использованием одного активного состояния предполагает заблаговременное знание словаря. Это не проблема, в случае если входная переменная является чем-то вроде языка, на котором написана книга, или дня недели, когда предсказывается уровень трафика.

Что делать, если рассматриваемая категориальная переменная является чем-то вроде ID больницы (`hospital_id`), где младенец родился, или ID медика (`physician_id`), принимающего роды? Подобные категориальные переменные создают несколько проблем.

- Знание словаря лексем предполагает его извлечение из тренировочных данных. Из-за случайности отбора вполне возможно, что тренировочные данные не будут содержать всех возможных больниц или врачей. Словарь может оказаться *неполным*.
- Категориальные переменные имеют *высокую кардинальность (мощность)*. Помимо признаковых векторов с тремя языками или семью днями бывают признаковые векторы, длина которых исчисляется тысячами или миллионами. Такие признаковые векторы на практике создают ряд проблем. Они предусматривают участие настолько большого числа весов, что тренировочных данных может оказаться недостаточно. Даже если мы и сможем натренировать модель, натренированная модель потребует много места для хранения, потому что во время обработки запросов моделью потребуется весь словарь. Следовательно, мы не сможем развернуть модель на малоемких устройствах.
- После внедрения модели в производственной среде могут быть введены в строй новые больницы и наняты новые врачи. Для них модель не сможет делать пред-

сказания, и поэтому для решения таких проблем с холодным пуском потребуется отдельная *инфраструктура модельного обслуживания*[15].

Даже в случае простых представлений, таких как кодирование с одним активным состоянием, стоит предвидеть проблему холодного пуска и в явной форме резервировать нули для входных значений, находящихся за пределами словаря.

В качестве конкретного примера возьмем задачу предсказания задержки прибытия авиарейса. Одной из переменных на входе в модель является аэропорт вылета. На момент сбора данных в Соединенных Штатах насчитывалось 347 аэропортов:

```
SELECT
  DISTINCT(departure_airport)
FROM `bigquery-samples.airline_ontime_data.flights`
```

В некоторых аэропортах было всего 1–3 рейса за весь период времени, и поэтому мы ожидаем, что словарь тренировочных данных будет неполным. Число 347 достаточно велико, и указанный признак окажется довольно разряженным, а мы, безусловно, имеем как раз тот случай, когда будут построены новые аэропорты. Все три проблемы (неполный словарь, высокая кардинальность, холодный пуск) будут существовать, если мы закодируем аэропорт вылета с использованием одного активного состояния.

Набор данных об авиакомпаниях, как и набор данных о раждаемости и почти все другие наборы данных, которые мы используем в этой книге для иллюстрации, является публичным набором данных, входящим в состав BigQuery[16], поэтому вы можете попробовать выполнить запрос. На момент написания этой книги запросы в объеме 1 Тбайт в месяц являются бесплатными и доступна песочница[17], вследствие чего вы можете использовать BigQuery вплоть до указанного лимита, не привлекая средства своей банковской карты. Мы рекомендуем вам добавить наш репозиторий GitHub в закладки. Например, полный исходный код можно найти в блокноте репозитория на GitHub[18].

Решение

Паттерн "Хешированный признак" представляет категориальную входную переменную, выполняя следующее.

---

[15] Холодный пуск (cold start) — это потенциальная проблема в системе ML, касающаяся неспособности системы делать какие-либо предсказательные выводы в отношении предметов, о которых она еще не собрала достаточной информации. — *Прим. перев.*

[16] См. **https://oreil.ly/lgcKA**.

[17] Песочница (sandbox) — это изолированная среда тестирования, которая позволяет пользователям выполнять программы или файлы, не влияя на приложение, систему или платформу, на которой они работают. — *Прим. перев.*

[18] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/hashed_feature.ipynb**.

1. Конвертируя категориальное входное значение в уникальную строку. Для аэропорта вылета мы можем использовать трехбуквенный код в формате IATA[19].
2. Вызывая детерминированный (без случайных начальных чисел или соли[20]) и переносимый (вследствие чего один и тот же алгоритм может использоваться как во время тренировки модели, так и во время обработки запросов) алгоритм хеширования на строковом значении.
3. Беря остаток, когда результат хеширования делится на желательное число корзин. В типичной ситуации алгоритм хеширования возвращает целое число, которое может быть отрицательным, а деление по модулю отрицательного целого числа является отрицательным. Поэтому берется абсолютное значение результата.

В BigQuery SQL эти шаги выполняются следующим образом:

```
ABS(MOD(FARM_FINGERPRINT(airport), numbuckets))
```

В функции `FARM_FINGERPRINT` используется семейство алгоритмов хеширования FarmHash, которое является детерминированным, хорошо распределенным[21] и для которого имеются реализации[22] на ряде языков программирования.

В TensorFlow эти шаги реализованы в функции `feature_column`:

```
tf.feature_column.categorical_column_with_hash_bucket(
    airport, num_buckets, dtype=tf.dtypes.string)
```

Например, в табл. 2.5 показано значение хеша FarmHash некоторых кодов аэропортов в формате IATA при хешировании в 3, 10 и 1000 корзин.

Таблица 2.5. *Хеш FarmHash некоторых кодов аэропортов в формате IATA при хешировании в разные числа корзин*

Строка	departure_airport	hash3	hash10	hash1000
1	DTW	1	3	543
2	LBB	2	9	709
3	SNA	2	7	587
4	MSO	2	7	737
5	ANC	0	8	508
6	PIT	1	7	267
7	PWM	1	9	309
8	BNA	1	4	744
9	SAF	1	2	892
10	IPL	2	1	591

[19] См. **https://oreil.ly/B8nLw**.

[20] В криптографии соль (salt) — это случайные данные, которые используются в качестве дополнительных на входе в одностороннюю функцию, хеширующую пароль или парольную фразу. — *Прим. перев.*

[21] См. **https://github.com/google/farmhash/blob/master/Understanding_Hash_Functions**.

[22] См. **https://github.com/google/farmhash**.

Почему это работает

Допустим, что мы решили хешировать код аэропорта с использованием 10 корзин (hash10 в табл. 2.5). Как это решает выявленные нами проблемы?

Входные значения за пределами словаря

Даже если аэропорт с горсткой рейсов не входит в состав тренировочного набора данных, его хешированное признаковое значение будет находиться в промежутке [0; 9]. Следовательно, во время обработки запросов моделью не будет никакой проблемы с отказоустойчивостью — неизвестный аэропорт получит предсказания, соответствующие другим аэропортам в хеш-корзине. Модель не будет ошибаться.

Если у нас 347 аэропортов, то в среднем 35 аэропортов получат один и тот же код хеш-корзины при условии, что мы хешируем их в 10 корзин. Аэропорт, отсутствующий в тренировочном наборе данных, будет "заимствовать" свои характеристики у других аналогичных ~35 аэропортов в хеш-корзине. Разумеется, предсказание для отсутствующего аэропорта не будет точным (неразумно ожидать точных предсказаний для неизвестных входных значений), но он будет лежать в правильном интервале.

Число хеш-корзин следует выбирать, уравновешивая необходимость разумно манипулировать входными значениями за пределами словаря с необходимостью точного отражения моделью категориального входного значения. При наличии 10 хеш-корзин смешивается ~35 аэропортов. Хорошее эмпирическое правило состоит в том, чтобы выбирать такое число хеш-корзин, чтобы каждая корзина получала около пяти записей. В данном случае это означало бы, что 70 хеш-корзин будет хорошим компромиссом.

Высокая кардинальность

Легко увидеть, что проблема высокой кардинальности не возникает до тех пор, пока мы выбираем достаточно малое число хеш-корзин. Даже если у нас будут миллионы аэропортов, больниц или врачей, мы сможем хешировать их в несколько сотен корзин, тем самым удерживая требования к памяти системы и к размеру модели практически осуществимыми.

Нам не нужно хранить словарь, потому что код преобразования не зависит от фактического значения данных, а ядро модели имеет дело не с полным словарем, а только с входными значениями `num_buckets`.

Стоит признать, что хеширование допускает потери — поскольку у нас 347 аэропортов, то в среднем 35 аэропортов будут получать один и тот же код хеш-корзины в случае их хеширования в 10 корзин. Однако, когда альтернативой является отказ от переменной, поскольку она слишком широка по охвату, кодирование с потерями — приемлемый компромисс.

Холодный пуск

Ситуация с холодным пуском аналогична ситуации со значениями за пределами словаря. Если в систему будет добавлен новый аэропорт, он изначально станет по-

лучать предсказания, соответствующие другим аэропортам в хеш-корзине. По мере того как аэропорт будет становиться все популярнее, начнет расти число рейсов из этого аэропорта. При условии, что мы будем периодически тренировать модель заново, ее предсказания начнут отражать задержки прибытия из нового аэропорта. Этот вопрос подробнее обсуждается в *разд. "Паттерн 18. Непрерывное оценивание модели" главы 5*.

Выбирая число хеш-корзин таким, чтобы каждая корзина получала около пяти записей, мы сможем обеспечить, чтобы любая корзина имела разумные начальные результаты.

Компромиссы и альтернативы

Большинство паттернов предполагают какой-то компромисс, и паттерн "Хешированный признак" не исключение. Ключевой компромисс здесь заключается в том, что мы теряем точность модели.

Коллизия корзин

Реализации хешированного признака в части взятия модуля числа является операцией с потерями. Выбирая размер хеш-корзины равным 100, мы делаем выбор в пользу того, чтобы 3–4 аэропорта пользовались одной корзиной. Мы в явной форме идем на уступки в отношении способности точно представлять данные (имея фиксированный словарь и кодирование с одним активным состоянием), для того чтобы справляться с проблемами входных значений за пределами словаря, ограничений по кардинальности/размеру модели и холодного пуска. Этот "обед" не бесплатен. Не следует выбирать паттерн "Хешированный признак", если вы знаете словарь заранее, если размер словаря относительно мал (размер из тысяч лексем для набора данных с миллионами примеров будет приемлемым) и холодный пуск не вызывает беспокойства.

Обратите внимание, что мы не можем просто увеличить количество корзин до чрезвычайно большого числа, надеясь вообще избежать коллизий. Даже если мы увеличим корзины до 100 тыс., имея только 347 аэропортов, вероятность того, что по крайней мере два аэропорта будут пользоваться одной и той же хеш-корзиной, составит 45% — что неприемлемо много (табл. 2.6). Поэтому мы должны исполь-

Таблица 2.6. *Ожидаемое число записей в расчете на корзину и вероятность хотя бы одной коллизии при хешировании кодов аэропортов IATA в разные числа корзин*

num_hash_buckets	collision_prob	entries_per_bucket
3	115.666667	1.000000
10	34.700000	1.000000
100	3.470000	1.000000
1000	0.347000	1.000000
10000	0.034700	0.997697
100000	0.003470	0.451739

зовать хешированные признаки, только если готовы мириться с тем, что многочисленные категориальные входные значения будут пользоваться одним и тем же значением хеш-корзины.

Асимметрия

Потеря точности становится особенно острой, когда распределение категориальных входных значений является сильно асимметричным. Рассмотрим случай с хеш-корзиной, содержащей ORD (один из самых загруженных аэропортов в мире — Чикагский аропорт). Мы можем найти это, использовав следующие инструкции:

```
CREATE TEMPORARY FUNCTION hashed(airport STRING, numbuckets INT64)
    AS (ABS(MOD(FARM_FINGERPRINT(airport), numbuckets))
);

WITH airports AS (
SELECT
    departure_airport, COUNT(1) AS num_flights
FROM `bigquery-samples.airline_ontime_data.flights`
GROUP BY departure_airport
)

SELECT
    departure_airport, num_flights
FROM airports
WHERE hashed(departure_airport, 100) = hashed('ORD', 100)
```

Результат показывает, что в то время как из ORD есть ~3,6 млн рейсов, из BTV (Берлингтон, штат Вермонт) есть только ~67 тыс. рейсов (табл. 2.7).

Таблица 2.7. *Результат запроса*

departure_airport	num_flights
ORD	3610491
BTV	66555
MCI	597761

Всё указывает на то, что при любом раскладе модель будет вменять присущие аэропорту Чикаго длительное время ожидания такси и продолжительные задержки по метеоусловиям муниципальному аэропорту города Берлингтон! Точность модели для BTV и MCI (аэропорта г. Канзас-Сити) будет довольно низкой по причине того, что из Чикаго столь много рейсов.

Агрегатный признак

Когда распределение категориальной переменной является асимметричным или когда число корзин настолько мало, что коллизии корзин происходят часто, воз-

можно, полезным окажется добавление на входе в модель агрегатного признака. Например, в тренировочном наборе данных по каждому аэропорту мы могли бы найти вероятность рейсов по расписанию и добавить ее в модель в качестве признака. Это позволит нам избегать потери информации, связанной с отдельными аэропортами, когда мы хешируем коды аэропортов. В некоторых случаях мы могли бы вообще отказаться от использования названия аэропорта, поскольку относительная частота рейсов по расписанию может быть достаточной.

Гиперпараметрическая настройка

Из-за компромиссов с частотой коллизий корзин бывает трудно выбрать число. Очень часто это зависит от самой задачи. Поэтому мы рекомендуем рассматривать число корзин как настраиваемый гиперпараметр:

```
- parameterName: nbuckets
      type: INTEGER
      minValue: 10
      maxValue: 20
      scaleType: UNIT_LINEAR_SCALE
```

Следует обеспечивать, чтобы число корзин оставалось в пределах разумного интервала кардинальности хешируемой категориальной переменной.

Криптографический хеш

Причина потери информации в паттерне "Хешированный признак" кроется в той части его реализации, где берется модуль числа. А если вообще отказаться от модуля? В конце концов, отпечаток алгоритма хеширования Farm Fingerprint[23] имеет фиксированную длину (INT64 составляет 64 бита), и поэтому он может быть представлен 64 признаковыми значениями, каждое из которых равно 0 или 1. Это называется *двоичным кодированием*.

Однако двоичное кодирование не решает проблему входных значений за пределами словаря или холодного пуска (решая только проблему высокой кардинальности). На самом деле побитовое кодирование является отвлекающим маневром. Если мы не будем делить по модулю, то сможем получить уникальное представление, просто кодируя три символа, которые образуют код IATA (и поэтому используя признак длиной $3 \cdot 26 = 78$). Проблема с этим представлением проявляется сразу же: аэропорты, названия которых начинаются с буквы О, не имеют ничего общего, когда дело касается их характеристик задержки рейса — кодировка создала *мнимую корреляцию* между аэропортами, которые начинаются с одной и той же буквы. Та

[23] В информатике хеш-функции, также именуемые функциями цифровых отпечатков (fingerprinting functions), часто используются для уменьшения произвольной длины данных до фиксированной битовой длины. Их результаты называются хешем или отпечатком. В частности, алгоритм хеширования Farm Fingerprint (дословно — цифровой отпечаток фермы) — это процедура, которая соотносит произвольно крупный элемент данных (например, компьютерный файл) с гораздо более короткой битовой строкой, ее отпечатком, который однозначно идентифицирует исходные данные во всех отношениях. — *Прим. перев.*

же самая картина соблюдается и в бинарном пространстве. По этой причине мы не рекомендуем применять двоичное кодирование значений цифровых отпечатков.

Двоичное кодирование алгоритмом хеширования MD5 не будет страдать от указанной выше проблемы ложной корреляции, потому что результат на выходе из хеш-алгоритма MD5 равномерно распределен, и значит, результирующие биты будут равномерно распределены. Однако, в отличие от хеш-алгоритма Farm Fingerprint, хеш-алгоритм MD5 не детерминирован и не уникален — это односторонний алгоритм хеширования, который будет иметь много неожиданных коллизий.

В паттерне "Хешированный признак" мы вынуждены использовать не криптографический хеш-алгоритм, а хеш-алгоритм цифрового отпечатка. Это обусловлено тем, что цель хеш-функции цифрового отпечатка — получение детерминированного и уникального значения. Если вдуматься, то это является ключевым требованием функций предобработки в машинном обучении, поскольку нам нужно применять одну и ту же функцию во время вызова модели и получать одно и то же хешированное значение. Функция цифрового отпечатка не производит равномерно распределенного результата, тогда как криптографические хеш-алгоритмы, такие как MD5 или SHA1, действительно создают распределенный результат, но не являются детерминированными и целенаправленно делаются вычислительно дорогостоящими. Следовательно, криптографический хеш не может использоваться в контексте инженерии признаков, где хешированное значение, вычисленное для конкретного входного признака во время предсказания, должно быть таким же, что и хеш, вычисленный во время тренировки, и где хеш-функция не должна замедлять работу модели машинного обучения.

Причина, по которой алгоритм MD5 не является детерминированным, заключается в том, что в хешируемую строку обычно добавляется соль. Соль — это случайное строковое значение, добавляемое в каждый пароль[24] для обеспечения того, чтобы хешированное значение в базе данных отличалось, даже если два пользователя применяют один и тот же пароль. Это необходимо для предотвращения атак, основанных на "радужных таблицах", т. е. атак, основанных на словарях часто выбираемых парольных слов и сравнивающих хеш известного пароля с хешами в базе данных. По мере увеличения вычислительной мощи становится возможным проводить атаку на любую соль методом грубой силы[25], и поэтому современные криптографические реализации выполняют хеширование в цикле с целью увеличения вычислительных затрат. Даже если бы мы отключили соль и сократили число итераций до одной, MD5-хеш был бы лишь одним из путей. Он не будет уникальным.

Таким образом, мы просто вынуждены использовать хеш-алгоритм цифрового отпечатка и выполнять деление по модулю результирующего хеша.

---

[24] См. **https://oreil.ly/cv7PS**.

[25] То есть путем исчерпывающего перебора всех значений. — *Прим. перев.*

Порядок операций

Обратите внимание, что сначала мы вычисляем остаток от деления (MOD), а затем берем абсолютное значение (ABS):

```
CREATE TEMPORARY FUNCTION hashed(airport STRING, numbuckets INT64) AS (
    ABS(MOD(FARM_FINGERPRINT(airport), numbuckets))
);
```

Порядок операций ABS, MOD и FARM_FINGERPRINT в приведенном выше фрагменте кода важен, поскольку интервал типа INT64 не является симметричным. В частности, его значения лежат между –9 223 372 036 854 775 808 и 9 223 372 036 854 775 807 (оба значения включительно). И поэтому если бы мы выполнили:

```
ABS(FARM_FINGERPRINT(airport))
```

то столкнулись бы с редкой и, вероятно, невоспроизводимой ошибкой переполнения, если операция FARM_FINGERPRINT случайно вернула бы значение –9 223 372 036 854 775 808, т. к. его абсолютное значение невозможно представить с использованием типа INT64!

Пустые хеш-корзины

Хотя и маловероятно, но есть шанс, что одна из хеш-корзин окажется пустой, даже если для представления 347 аэропортов мы выберем 10 хеш-корзин. Поэтому при использовании хешированных признаковых столбцов бывает выгодно[26] также использовать регуляризацию L2, благодаря которой веса, ассоциированные с пустой корзиной, будут близки к нулю. И если аэропорт за пределами словаря и вправду попадет в пустую корзину, то это не приведет к численной нестабильности модели.

ПАТТЕРН 2. Векторные вложения

Векторное вложение (Embedding) — это усваиваемое моделью представление данных, которое соотносит данные высокой кардинальности с низкоразмерным пространством таким образом, что информация, относящаяся к задаче обучения, сохраняется. Векторное вложение находится в ядре современного машинного обучения и имеет самые разные воплощения по всей области.

Постановка задачи

Модели машинного обучения систематически отыскивают в данных закономерности, которые улавливают степень соотнесенности свойств входных признаков модели с выходной меткой. Как следствие, представление входных признаков напрямую влияет на качество окончательной модели. Хотя манипулировать структурированными числовыми значениями на входе относительно просто, однако данные, необходимые для тренировки модели машинного обучения, поступают в бесчис-

[26] См. **https://oreil.ly/xlwAH**.

ленном многообразии, как, например, категориальные признаки, текст, снимки, аудио, временные ряды и многие другие. Для этих представлений данных нам нужно содержательное числовое значение, которое можно было бы подавать в модель машинного обучения, предоставляя возможность признакам вписываться в типичную парадигму тренировки. Векторное вложение обеспечивает подход к манипулированию некоторыми разрозненными типами данных таким образом, чтобы они сохраняли подобие между элементами и, как следствие, улучшали способность модели усваивать существенные закономерности[27].

Кодирование с одним активным состоянием является распространенным подходом к представлению категориальных входных переменных. Например, возьмем входную переменную `plurality` (многоплодность) в наборе данных о рождаемости[28]. Указанная входная переменная является категориальной и имеет шесть возможных значений: `['Одноплодный(1)', 'Многоплодные(2+)', 'Двойняшки(2)', 'Тройняшки(3)', 'Четверняшки(4)', 'Пятерняшки(5)']`. Мы можем манипулировать этой категориальной переменной, используя кодирование с одним активным состоянием, которое соотносит каждое потенциальное входное строковое значение с единичным вектором в R^6[29] (табл. 2.8).

***Таблица 2.8.** Пример кодирования категориальной входной переменной с использованием одного активного состояния для набора натальных данных*

Многоплодность	Кодирование с одним активным состоянием
Одноплодный(1)	[1,0,0,0,0,0]
Многоплодные(2+)	[0,1,0,0,0,0]
Двойняшки(2)	[0,0,1,0,0,0]
Тройняшки(3)	[0,0,0,1,0,0]
Четверняшки(4)	[0,0,0,0,1,0]
Пятерняшки(5)	[0,0,0,0,0,1]

При таком кодировании нам нужно иметь шесть размерностей, чтобы представить каждую отдельную категорию. Возможно, иметь шесть размерностей не так уж и плохо, но что делать, если бы нам пришлось рассматривать гораздо больше категорий?

Например, что делать, если бы наш набор данных состоял из истории просмотров видеобазы клиентами, а наша задача заключалась бы в том, чтобы предлагать список новых видео с учетом предыдущих просмотров клиентов? В этом случае поле `customer_id` может содержать миллионы уникальных записей. Аналогично `video_id`

---

[27] Векторное вложение — это соотнесение дискретной переменной в вектор непрерывных чисел путем создания плотного вектора для каждого тренировочного примера, выбираемого таким образом, чтобы он был похож на векторы, которые появляются в схожем контексте. — *Прим. перев.*

[28] Этот набор данных доступен в BigQuery: bigquery-public-data.samples.natality.

[29] То есть с размерностью 6. — *Прим. перев.*

ранее просмотренных видео тоже может содержать тысячи записей. Кодирование категориальных признаков *высокой кардинальности*, таких как идентификаторы `video_id` или `customer_id`, с использованием одного активного состояния в качестве значений, подаваемых на вход в модель машинного обучения, приводит к разреженной матрице, которая плохо подходит для ряда алгоритмов машинного обучения.

Вторая проблема с таким кодированием заключается в том, что оно рассматривает категориальные переменные как *независимые*. Однако представление данных для двойняшек должно быть близким к представлению данных для тройняшек и довольно далеким от представления данных для пятерняшек. Многоплодные младенцы, скорее всего, относятся к двойняшкам, но могут относиться и к тройняшкам. В качестве примера в табл. 2.9 показано альтернативное представление столбца многоплодности в более низкой размерности, которая отражает это отношение близости.

Таблица 2.9. *Использование векторных вложений с более низкой размерностью для представления столбца многоплодности в наборе данных о рождаемости*

Многоплодность	Кодирование кандидата
`Одноплодный(1)`	`[1.0,0.0]`
`Многоплодные(2+)`	`[0.0,0.6]`
`Двойняшки(2)`	`[0.0,0.5]`
`Тройняшки(3)`	`[0.0,0.7]`
`Четверняшки(4)`	`[0.0,0.8]`
`Пятерняшки(5)`	`[0.0,0.9]`

Эти числа, конечно, произвольны. Но существует ли возможность усваивать наилучшее возможное представление столбца многоплодности, используя для задачи о рождаемости только две размерности? Именно эту задачу решает паттерн "Векторное вложение".

Та же самая проблема высокой кардинальности и зависимых данных возникает и во время работы со снимками и текстом. Снимки состоят из тысяч пикселов, которые не являются независимыми друг от друга. Текст на естественном языке черпается из словаря, состоящего из десятков тысяч слов, и такое слово, как "идти", ближе к слову "бежать", чем к слову "книга".

Решение

Паттерн "Векторные вложения" решает задачу плотного представления данных высокой кардинальности в более низкой размерности путем пропускания входных данных через слой векторного вложения, имеющий тренируемые веса. Указанный слой соотносит высокоразмерную, категориальную входную переменную с вещественным вектором в некотором низкоразмерном пространстве. Веса для создания

плотного представления устанавливаются для модели в рамках ее оптимизации (рис. 2.5). На практике эти векторные вложения в итоге улавливают отношения близости во входных данных.

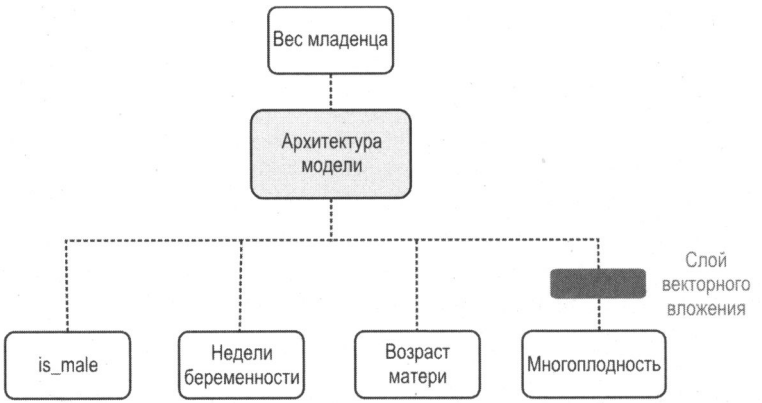


Рис. 2.5. Веса слоя векторного вложения корректируются моделью во время тренировки

Поскольку векторные вложения улавливают отношения близости во входных данных в низкоразмерном представлении, мы можем использовать слой векторного вложения в качестве замены методов кластеризации (например, методов сегментации клиентов) и методов редукции размерности, таких как анализ главных компонент (principal components analysis, PCA). Веса векторного вложения определяются в главном цикле тренировки модели, что избавляет от необходимости предварительной кластеризации или выполнения PCA.

Веса в слое векторного вложения корректируются в рамках процедуры градиентного спуска во время тренировки модели натальности.

В конце тренировки веса слоя векторного вложения могут быть такими, как показанная в табл. 2.10 кодировка категориальных переменных.

Таблица 2.10. *Кодировка с одним активным состоянием и усвоенная кодировка для столбца многоплодности в наборе натальных данных*

Многоплодность	Кодирование с одним активным состоянием	Усвоенная кодировка
Одноплодный(1)	[1,0,0,0,0,0]	[0.4, 0.6]
Многоплодные(2+)	[0,1,0,0,0,0]	[0.1, 0.5]
Двойняшки(2)	[0,0,1,0,0,0]	[-0.1, 0.3]
Тройняшки(3)	[0,0,0,1,0,0]	[-0.2, 0.5]
Четверняшки(4)	[0,0,0,0,1,0]	[-0.4, 0.3]
Пятерняшки(5)	[0,0,0,0,0,1]	[-0.6, 0.5]

Векторное вложение соотносит разреженный, кодированный с одним активным состоянием, вектор в плотный вектор в R^2.

В TensorFlow мы сначала строим для этого признака категориальный признаковый столбец, а затем обертываем его в признаковый столбец векторного вложения. Например, для нашего признака многоплодности мы бы написали следущий фрагмент кода:

```
plurality = tf.feature_column.categorical_column_with_vocabulary_list(
            'plurality', ['Single(1)', 'Multiple(2+)', 'Twins(2)',
                          'Triplets(3)', 'Quadruplets(4)', 'Quintuplets(5)'])
plurality_embed = tf.feature_column.embedding_column(plurality, dimension=2)
```

Результирующий признаковый столбец (`plurality_embed`) используется в качестве входа для нижестоящих узлов нейронной сети вместо признакового столбца, кодированного с одним активным состоянием (`plurality`).

Векторные вложения текста

Текст обеспечивает естественную среду, в которой выгодно использовать слой векторного вложения. Учитывая кардинальность словаря (часто порядка десятков тысяч слов), кодировать каждое слово с использованием одного активного состояния непрактично. Это создало бы невероятно крупную (высокоразмерную) и разреженную матрицу для тренировки. Кроме того, мы хотели бы, чтобы схожие слова имели векторные вложения, которые располагались бы в пространстве векторных вложений рядом, а неродственные слова находились бы в нем далеко. Поэтому мы используем плотное векторное вложение слов для векторизации дискретного текста перед тем, как передавать его на вход в модель.

В целях реализации векторного вложения текста в Keras мы сначала выполняем лексимизацию путем создания лексемы (токена) для каждого слова в словаре (рис. 2.6). Затем используем результат этой лексемизации для соотнесения со слоем векторного вложения аналогично тому, как это было сделано для столбца многоплодности.

Рис. 2.6. Лексемизатор создает справочную таблицу, которая соотносит каждое слово с индексом

Результатом лексемизации является справочная (просмотровая) таблица, в которой каждое слово в словаре соотнесено с индексом и может рассматриваться как кодирование каждого слова в кодировке с одним активным состоянием, в котором лексемизированный индекс является местоположением ненулевого элемента, кодированного с одним активном состоянием. Такое кодирование требует полного прохода по набору данных (предположим, что он состоит из заголовков статей[30]),

[30] Этот набор данных доступен в BigQuery: bigquery-public-data.hacker_news.stories.

в результате которого создается справочная таблица, и может быть выполнено в Keras. Полный исходный код можно найти в репозитории[31] этой книги:

```
from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer()
tokenizer.fit_on_texts(titles_df.title)
```

Здесь мы можем использовать класс `Tokenizer` из библиотеки keras.preprocessing.text. Вызов метода `fit_on_texts()` создает справочную таблицу, которая соотносит каждое найденное в наших заголовках слово с индексом. Вызвав метод `tokenizer.index_word`, мы можем проинспектировать эту справочную таблицу напрямую

```
tokenizer.index_word
{1: 'the',
 2: 'a',
 3: 'to',
 4: 'for',
 5: 'in',
 6: 'of',
 7: 'and',
 8: 's',
 9: 'on',
10: 'with',
11: 'show',
...
```

Затем мы можем вызвать эту таблицу соответствий с помощью метода `texts_to_sequences()` лексемизатора. В результате каждая последовательность слов в подаваемом на вход тексте (здесь мы исходим из того, что имеем дело с заголовками статей) будет соотнесена с последовательностью лексем, соответствующих каждому слову (рис. 2.7):

```
integerized_titles = tokenizer.texts_to_sequences(titles_df.title)
```

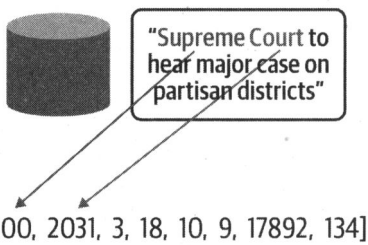


Рис. 2.7. С помощью лексемизатора каждый заголовок соотносится с последовательностью целочисленных значений индекса[32]

[31] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/embeddings.ipynb**.

[32] Надпись на рисунке: "Верховный суд заслушает особо важное дело о партийных участках". — *Прим. перев.*

Лексемизатор содержит другую релевантную информацию, которой мы воспользуемся позже для создания слоя векторного вложения. В частности, VOCAB_SIZE фиксирует число элементов индексной справочной таблицы, а MAX_LEN содержит максимальную длину текстовых строк в наборе данных:

```
VOCAB_SIZE = len(tokenizer.index_word)
MAX_LEN = max(len(sequence) for sequence in integerized_titles)
```

Перед созданием модели необходимо выполнить предобработку заголовков в наборе данных. Нам понадобится дополнить элементы заголовка для подачи в модель. Для этого в Keras имеется вспомогательная функция pad_sequence в верхней части методов лексемизатора. Функция create_sequences на входе принимает заголовки и максимальную длину предложения, а на выходе возвращает соответствующий нашим лексемам список целых чисел, дополненный до максимальной длины предложения:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

def create_sequences(texts, max_len=MAX_LEN):
    sequences = tokenizer.texts_to_sequences(texts)
    padded_sequences = pad_sequences(sequences,
                                     max_len,
                                     padding='post')
    return padded_sequences
```

Далее мы построим модель, основанную на глубокой нейронной сети (deep neural network, DNN), в которой реализуется простой слой векторного вложения, преобразующий словарные целые числа в плотные векторы. Слой Embedding в Keras можно рассматривать как отображение из целочисленных индексов отдельно взятых слов в плотные векторы (векторные вложения этих слов). Размерность векторного вложения определяется аргументом output_dim. Аргумент input_dim задает размер словаря, а input_shape — длину входных последовательностей. Поскольку здесь перед передачей заголовков в модель мы их дополнили, мы задали input_shape=[MAX_LEN]:

```
model = models.Sequential([layers.Embedding(input_dim=VOCAB_SIZE + 1,
                                            output_dim=embed_dim,
                                            input_shape=[MAX_LEN]),
                           layers.Lambda(lambda x: tf.reduce_mean(x,axis=1)),
                           layers.Dense(N_CLASSES, activation='softmax')])
```

Обратите внимание, что нам нужно поместить прикладной слой Lambda фреймворка Keras между слоем векторного вложения и плотным слоем с функцией мягкого максимума softmax, чтобы усреднить словарные векторы, возвращаемые слоем векторного вложения. Это усредненное значение подается в плотный softmax-слой. При этом мы создаем простую модель, которая теряет информацию о порядке слов, создавая модель, которая видит предложения как *мешок слов*.

Векторные вложения снимков

В то время как текст представляет собой очень разряженные входные данные, другие типы данных, такие как снимки или аудио, состоят из плотных выскоразмерных векторов, обычно с несколькими каналами, которые содержат сырую пиксельную или частотную информацию. В этом случае векторное вложение улавливает релевантное, низкоразмерное представление входных данных.

В случае векторных вложений снимков сначала тренируется сложная сверточная нейронная сеть, такая как Inception или ResNet. Это делается на крупном наборе, таком как ImageNet, содержащем миллионы снимков и тысячи возможных классификационных меток. Затем из модели удаляется последний softmax-слой. Без завершающего softmax-классифицирующего слоя модель может использоваться для извлечения признакового вектора для заданной входной переменной. Этот признаковый вектор содержит всю релевантную информацию о снимке, поэтому он по существу является низкоразмерным векторным вложением входного снимка.

Давайте рассмотрим задачу создания надписи, т. е. генерирования текстовой подписи к заданному снимку (рис. 2.8).

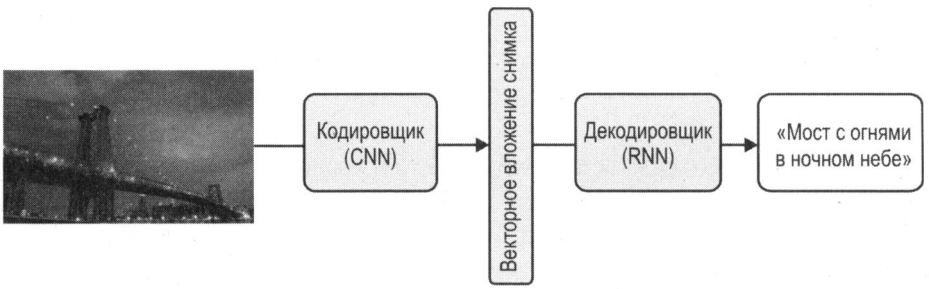


Рис. 2.8. В случае задачи трансляции снимка в текст кодировщик создает низкоразмерное представление снимка в формате векторного вложения

В ходе тренировки этой архитектуры на массивном наборе пар снимок/подпись, кодировщик усваивает эффективное для снимков векторное представление. Декодировщик учится транслировать этот вектор в текстовую надпись. В этом смысле кодировщик становится машиной векторных вложений Image2Vec.

Почему это работает

Слой векторного вложения — это просто еще один скрытый слой нейронной сети. Веса ассоциируются с каждой размерностью высокой кардинальности, а результат на выходе из него пропускается через остальную часть сети. Следовательно, необходимые для создания векторного вложения веса усваиваются точно так же, как и любые другие веса в нейронной сети, т. е. в процессе градиентного спуска. Это означает, что результирующие векторные вложения порождают наиболее эффективное низкоразмерное представление этих признаковых значений по отношению к задаче обучения.

Хотя это улучшенное векторное вложение в конечном счете оказывает поддержку модели, сами векторные вложения имеют собственную ценность и позволяют нам получать дополнительную скрытую информацию о наборе данных.

Еще раз возьмем набор данных о клиентских видео. Если использовать только кодирование с одним активным состоянием, то два любых отдельных пользователя `user_i` и `user_j` будут иметь одинаковую меру подобия. Аналогично точечное произведение или косинусное подобие двух любых отличимых шестимерных кодировок многоплодности родов с использованием одного активного состояния будет иметь нулевое подобие. И это не лишено смысла, т. к. кодирование с одним активным состоянием, по сути, говорит нашей модели о том, что два любых разных значения многоплодности беременности следует трактовать как отдельные и несвязанные. В случае набора данных о клиентах и просмотрах видео мы теряем всякое представление о подобии между клиентами или видео. Но это кажется не совсем правильным. Два разных клиента или видео, скорее всего, имеют между собой сходства. То же самое относится и к многоплодности беременности. Появление четверни и пятерни, вероятно, влияет на вес при рождении статистически сходным образом, в отличие от веса при рождении у одноплодного младенца (рис. 2.9).

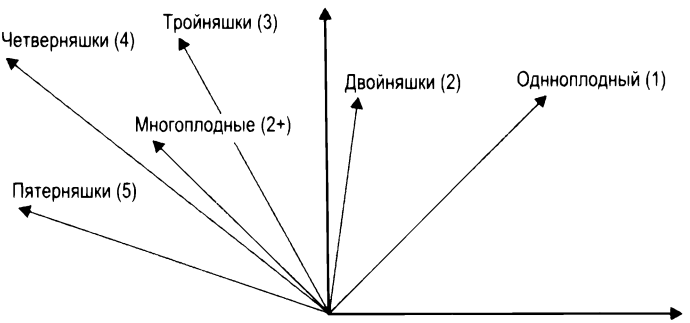


Рис. 2.9. Помещая категориальную переменную в низкоразмерное пространство векторных вложений, мы также можем усваивать связи между разными категориями

При вычислении подобия категорий многоплодности в формате векторов в кодировке с одним активным состоянием мы получаем матрицу идентичности, поскольку каждая категория трактуется как отличимый признак (табл. 2.11).

Таблица 2.11. *Когда признаки кодируются с использованием одного активного состояния, матрица подобия представляет собой просто матрицу идентичности*

	Одноплодный (1)	Многоплодные (2+)	Двойняшки (2)	Тройняшки (3)	Четверняшки (4)	Пятерняшки (5)
Одноплодный (1)	1	0	0	0	0	0
Многоплодные (2+)	–	1	0	0	0	0
Двойняшки (2)	–	–	1	0	0	0
Тройняшки (3)	–	–	–	1	0	0

***Таблица 2.11** (окончание)*

	Одноплодный (1)	Многоплодные (2+)	Двойняшки (2)	Тройняшки (3)	Четверняшки (4)	Пятерняшки (5)
Четверняшки (4)	–	–	–	–	1	0
Пятерняшки (5)	–	–	–	–	–	1

Однако, как только многоплодность вкладывается в две размерности, мера подобия становится нетривиальной и появляются важные связи между разными категориями (табл. 2.12).

***Таблица 2.12.** Когда признаки вкладываются в две размерности, матрица подобия дает нам больше информации*

	Одноплодный (1)	Многоплодные (2+)	Двойняшки (2)	Тройняшки (3)	Четверняшки (4)	Пятерняшки (5)
Одноплодный (1)	1	0.92	0.61	0.57	0.06	0.1
Многоплодные (2+)	–	1	0.86	0.83	0.43	0.48
Двойняшки (2)	–		1	0.99	0.82	0.85
Тройняшки (3)	–			1	0.85	0.88
Четверняшки (4)	–				1	0.99
Пятерняшки (5)	–	–	–	–	–	1

Таким образом, векторное вложение позволяет нам извлекать внутренне присущее подобие между двумя отдельными категориями, и с учетом представления в формате числового вектора мы получаем возможность четко выражать подобие между двумя категориальными признаками количественно.

Рис. 2.10. Усваивая низкоразмерный, плотный вектор вложения для каждого клиента и видео, модель, основанная на векторных вложениях, способна неплохо обобщать с меньшей косвенной нагрузкой, связанной с ручной инженерией признаков

Это легко визуализировать с помощью набора данных о рождаемости, но тот же принцип применим и при работе с идентификаторами `customer_id` и их векторным вложением в 20-мерное пространство. Векторные вложения, при их применении к набору данных о клиентах, позволяют нам извлекать похожих клиентов по заданному идетификатору `customer_id` и давать рекомендации, основываясь на подобии, например, в отношении того, какие видео клиент, вероятно, будет смотреть (рис. 2.10). Более того, эти векторные вложения пользователей и предметов могут сочетаться с другими признаками во время тренировки отдельной модели машинного обучения. Использование в ML-модели предварительно натренированных вложений называется *передачей модели обучения* или *трансферным обучением*.

Компромиссы и альтернативы

Главным компромиссом при использовании векторного вложения является ослабленное представление данных. При переходе от представления с высокой кардинальностью к более низкоразмерному представлению неминуема потеря информации. Взамен мы получаем информацию о близости и контексте предметов.

Выбор размерности вложения

Будучи специалистами-практиками, мы нацелены выбрать точную размерность пространства векторных вложений. И насколько же большую или малую размерность векторного вложения нам следует выбрать? Конечно же, как и в большинстве случаев, которые происходят в машинном обучении, всегда есть компромисс. Потеря представленности контролируется за счет размера слоя векторного вложения. При выборе очень малой выходной размерности слоя векторного вложения слишком много информации втискивается в малое векторное пространство и контекст может быть потерян. С другой стороны, когда размерность векторного вложения слишком велика, оно теряет усвоенную контекстуальную важность признаков. В крайнем случае мы возвращаемся к проблеме, возникшей при кодировании с использованием одного активного состояния. Оптимальная размерность векторного вложения нередко отыскивается экспериментально, подобно выбору числа нейронов в слое глубокой нейронной сети.

Если мы торопимся, то нам следует вспомнить одно эмпирическое правило о том, чтобы использовать корень четвертый степени[33] из суммарного числа уникальных категориальных элементов, а другое — что размерность вложения должна быть примерно в 1,6 раза больше квадратного корня[34] из числа уникальных элементов в категории и не менее 600. Например, предположим, что мы хотим использовать слой векторного вложения для кодирования признака, имеющего 625 уникальных значений. Используя первое эмпирическое правило, мы выбираем размерность вложения 5, а воспользовавшись вторым эмпирическим правилом, мы выбираем 40.

---

[33] См. **https://oreil.ly/ywFco**.

[34] См. **https://oreil.ly/github-fastai-2-blob-fastai-2-tabular-model-py**.

Если мы выполняем гиперпараметрическую настройку, то, возможно, стоит поискать внутри этого диапазона.

Автокодировщики

Тренировка векторных вложений в контролируемом режиме несет в себе трудности, связанные с тем, что для этого требуется много размеченных данных. Для того чтобы модель классифицирования снимков, такая как Inception, могла продуцировать полезные векторные вложения снимков, она тренируется на базе данных ImageNet, которая имеет 14 млн размеченных снимков. Автокодировщики предоставляют один из подходов, позволяющих обойти эту потребность в массивном размеченном наборе данных.

Типичная автокодировочная архитектура, показанная на рис. 2.11, состоит из узкого (bottleneck) слоя, который является слоем векторного вложения. Часть сети перед узким слоем ("кодировщик") отображает высокоразмерные входные данные в низкоразмерный слой векторного вложения, в то время как вторая сеть после него ("декодировщик") отображает это представление обратно в более высокую размерность, обычно ту же самую размерность, что и исходная. Модель, как правило, тренируется на некотором варианте ошибки восстановления, что вынуждает выходные модельные данные быть как можно более похожими на входные.

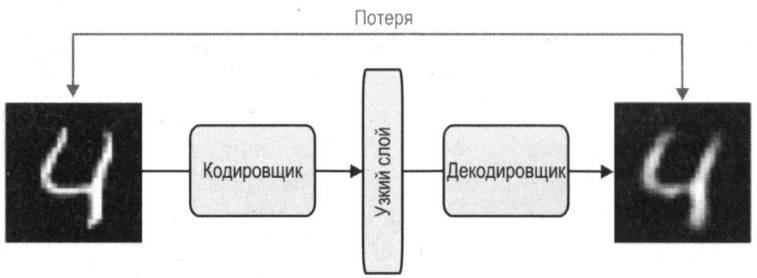


Рис. 2.11. Во время тренировки автокодировщика признак и метка одинаковы, а потерей является ошибка восстановления. Это позволяет автокодировщику добиваться нелинейного уменьшения размерности

Поскольку входные данные совпадают с выходными, никаких дополнительных меток не требуется. Кодировщик определяет оптимальное нелинейное уменьшение размерности входных данных. Узкий слой автокодировщика способен получать нелинейное уменьшение размерности посредством векторного вложения подобно тому, как PCA достигает линейного уменьшения размерности.

Это позволяет разбивать трудную задачу машинного обучения на две части. Сначала мы используем все неразмеченые данные, которые у нас есть, чтобы перейти от высокой кардинальности к более низкой, используя автокодировщики в качестве *вспомогательной задачи обучения*. Затем мы решаем фактическую задачу классифицирования снимков, для которой мы обычно имеем гораздо меньше размеченных данных, используя векторное вложение, произведенное вспомогательной задачей автокодировщика. Как следствие, повышается вероятность ускоренной работы

модели, потому что теперь модель должна корректировать веса только для низкоразмерного окружения (т. е. она должна работать с меньшим количеством весов).

В дополнение к автокодировщикам снимков недавняя научно-исследовательская работа[35] была посвящена применению методов глубокого обучения для структурированных данных. TabNet — это глубокая нейронная сеть, специально разработанная для обучения на основе информации из табличных данных, и она поддается тренировке в режиме обучение без учителя. Если модифицировать модель так, чтобы она имела структуру "кодировщик — декодировщик", то TabNet будет работать как автокодировщик на табличных данных, что позволяет модели усваивать векторные вложения из структурированных данных посредством преобразователя признаков.

Контекстно-языковые модели

Существует ли вспомогательная задача обучения, которая работает для текста? Контекстные языковые модели, такие как Word2Vec, и маскированные языковые модели, такие как двунаправленно-кодированные представления из трансформаторов (Bidirectional encoding representations from transformers, BERT), меняют задачу обучения на такую, в которой больше нет дефицита меток.

Word2Vec — это хорошо известный метод конструирования векторного вложения с использованием мелких нейронных сетей и сочетания двух технических приемов — непрерывного мешка слов (Continuous Bag of Words, CBOW) и skip-граммной модели — применительно к крупному корпусу текста, такому как Википедия. В то время как цель обеих моделей состоит в том, чтобы усваивать контекст слова путем соотнесения входного слова (слов) с целевым словом (словами) с помощью промежуточного слоя векторного вложения, достигается вспомогательная цель — усвоение низкоразмерных векторных вложений с наилучшим улавливанием контекста слов. Результирующие векторные вложения слов, усвоенные с помощью Word2Vec, улавливают семантические связи между словами таким образом, что векторные представления в пространстве векторных вложений сохраняют содержательную дистанцию и направленность (рис. 2.12).

Рис. 2.12. Векторные вложения слов улавливают семантические связи

[35] См. **https://oreil.ly/ywFco**.

Тренировка модели BERT выполняется с использованием маскированной языковой модели и предсказания следующего предложения. В маскированной языковой модели слова в тексте случайным образом закрываются маской и модель угадывает пропущенные слова. Предсказание следующего предложения — это классификационная задача, в которой модель делает предсказание о том, что в изначальном тексте два предложения следовали друг за другом. Таким образом, в качестве помеченного набора данных подходит любой корпус текста. Модель BERT была первоначально натренирована на всей английской Википедии и книжном корпусе BooksCorpus. Несмотря на то что BERT и Word2Vec учились на этих вспомогательных задачах, усвоенные ими векторные вложения оказались очень мощными при использовании на других нижестоящих задачах обучения. Векторные вложения слов, усвоенные моделью Word2Vec, являются одинаковыми независимо от предложения, в котором это слово появляется. Однако векторные вложения слов модели BERT являются контекстуальными, т. е. вектор вложения отличается в зависимости от контекста использования слова.

Предварительно натренированное векторное вложение текста, такое как Word2Vec, NNLM, GLoVE или BERT, можно добавлять в модель машинного обучения для обработки текстовых признаков в сочетании со структурированными входными данными и другими усвоенными векторными вложениями из нашего набора данных о клиентах и видео (рис. 2.13).

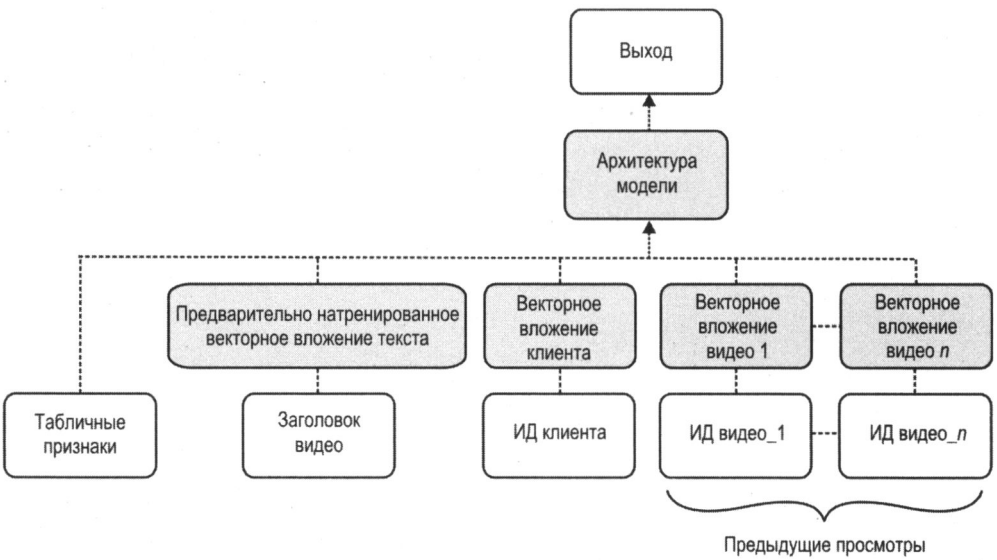


Рис. 2.13. Предварительно натренированное векторное вложение текста можно добавлять в модель для обработки текстовых признаков

В конечном счете векторные вложения учатся сохранять информацию, относящуюся к предписанной задаче обучения. В случае добавления подписей к снимкам задача состоит в том, чтобы научиться понимать, каким образом контекст элементов снимка соотносится с текстом. В автокодировочной архитектуре метка совпадает

с признаком, поэтому уменьшение размерности узкого слоя помогает обучить абсолютно все без учета какого-либо конкретного контекста того, что представляет важность.

Векторные вложения на хранилище данных

Машинное обучение на структурированных данных лучше всего выполнять непосредственно с помощью языка SQL на хранилище данных. Это позволяет избегать необходимости экспортировать данные из хранилища и уменьшает проблемы с приватностью и безопасностью данных.

Однако многие задачи требуют наличия смеси из структурированных данных и текстовых данных на естественном языке или снимковых данных. В хранилищах данных текст на естественном языке (например, отзывы) хранится непосредственно в виде столбцов, а снимки обычно хранятся в виде URL-адресов файлов в облачном хранилище. В этих случаях упрощается работа последующего машинного обучения по дополнительному хранению векторных вложений текстовых столбцов или снимков в качестве столбцов с типом "массив". При таком подходе обеспечивается возможность легкого встраивания подобных неструктурированных данных в модели машинного обучения.

В целях создания векторных вложений текста можно загрузить предварительно натренированную модель, такую как Swivel, из хаба TensorFlow Hub в BigQuery. Полный исходный код находится в репозитории на GitHub[36]:

```
CREATE OR REPLACE MODEL advdata.swivel_text_embed
OPTIONS(model_type='tensorflow', model_path='gs://BUCKET/swivel/*')
```

Затем применить модель для преобразования столбца с текстом на естественном языке в массив векторного вложения и сохранить справочную таблицу векторного вложения в новой таблице:

```
CREATE OR REPLACE TABLE advdata.comments_embedding AS
SELECT
  output_0 as comments_embedding,
  comments
FROM ML.PREDICT(MODEL advdata.swivel_text_embed,(
  SELECT comments, LOWER(comments) AS sentences
  FROM `bigquery-public-data.noaa_preliminary_severe_storms.wind_reports`
))
```

Теперь можно выполнить операцию соединения с этой таблицей, чтобы получить векторное вложение текста для любого комментария. В случае векторных вложений снимков мы можем подобным же образом преобразовать URL-адреса снимков в векторные вложения и загрузить их в хранилище данных.

Такое предварительное вычисление признаков является примером из *разд. "Паттерн 26. Хранилище признаков" (см. главу 6)*.

---

[36] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/text_embeddings.ipynb**.

ПАТТЕРН 3. Синтетический признак

Паттерн "Синтетический признак" (Feature Cross) помогает моделям быстрее устанавливать связи между входными данными, в явной форме делая каждое сочетание входных значений отдельным признаком.

Постановка задачи

Рассмотрим набор данных на рис. 2.14 и задачу создания двоичного классификатора, который разделяет метки + и −.

Если использовать только координаты `x_1` и `x_2`, то нет никакой возможности отыскать линейную границу, которая разделяет классы + и −.

Из этого следует, что для решения указанной задачи мы должны сделать модель сложнее, возможно, добавив в нее больше слоев. Однако существует более простое техническое решение.

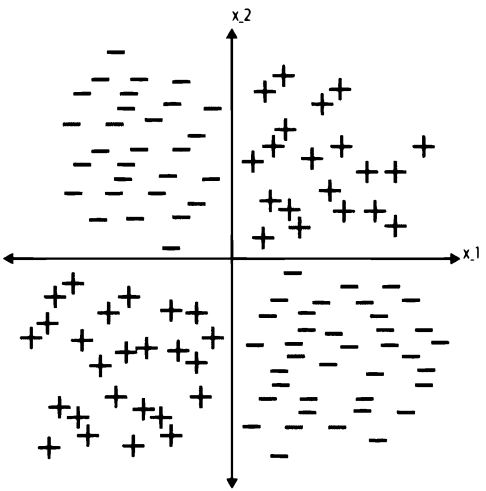


Рис. 2.14. Этот набор данных не является линейно разделимым, если использовать в качестве входных переменных только `x_1` и `x_2`

Решение

В машинном обучении инженерия признаков представляет собой процесс использования знаний предметной области для создания новых признаков, которые помогают процессу машинного обучения и повышают предсказательную способность модели. Одним из широко используемых технических приемов инженерии признаков является генерация признаков[37].

[37] Синтетический признак (гибрид признаков) получается путем скрещивания (умножения) двух или более признаков. — *Прим. перев.*

Синтетический признак — признак или гибрид признаков, образованный путем конкатенирования двух или более категориальных признаков с целью улавливания взаимодействия между ними. Соединив таким образом два признака, можно закодировать в модель нелинейность, которая обеспечит предсказательные способности, выходящие за рамки того, что каждый из признаков мог бы обеспечить индивидуально. Синтетические признаки обеспечивают подход, благодаря которому модели быстрее усваивают связи между признаками. В то время как более сложные модели, такие как нейронные сети и деревья, могут самостоятельно изучать синтетические признаки, использование таких признаков в явной форме позволяет нам избегать тренировки только линейной модели. И поэтому синтетические признаки могут ускорять тренировку модели (делая ее менее дорогостоящей) и снижать сложность модели (требуя меньше тренировочных данных).

В целях создания признакового столбца для приведенного выше набора данных мы можем сгруппировать `x_1` и `x_2` в две корзины в зависимости от их знака. Как следствие, `x_1` и `x_2` будут конвертированы в категориальные признаки. Обозначим через *A* корзину, где `x_1 >= 0`, и через *B* корзину, где `x_1 < 0`. Обозначим через *C* корзину, где `x_2 >= 0`, и через *D* корзину, где `x_2 < 0` (рис. 2.15).

x_1× x_2	Метка
AC	+
BC	-
AD	+
BD	-

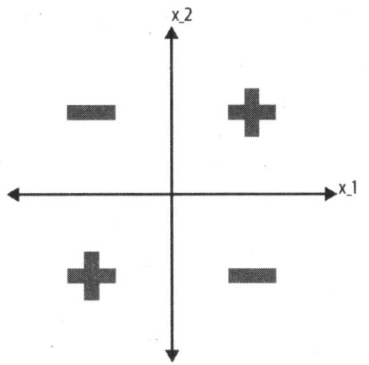


Рис. 2.15. Синтетический признак вводит четыре новых булевых признака

Синтетический признак этих сгруппированных в корзины признаков вставляет в модель четыре новых булевых признака:

- *AC*, где `x_1 >= 0` и `x_2 >= 0`
- *BC*, где `x_1 < 0` и `x_2 >= 0`
- *AD*, где `x_1 >= 0` и `x_2 < 0`
- *BD*, где `x_1 < 0` и `x_2 < 0`

Каждый из этих четырех булевых признаков (*AC*, *BC*, *AD* и *BD*) получит собственный вес во время тренировки модели. Это означает, что мы можем рассматривать каждый квадрант как самостоятельный признак. Поскольку изначальный набор данных был идеально разделен созданными нами интервальными корзинами, соединение признаков *A* и *B* способно разделить набор данных линейно.

Но это всего лишь иллюстрация. А что делать с реальными практическими данными? Рассмотрим публичный набор данных о поездках в нью-йоркском желтом такси (табл. 2.13)[38].

Таблица 2.13. *Обзор публичного набора данных о поездках нью-йоркском такси в BigQuery*

pickup_datetime (дата_время посадки)	Pickuplon (долгота посадки)	Pickuplat (широта посадки)	Dropofflon (долгота высадки)	Dropofflat (широта высадки)	Passengers (пассажиры)	fare_amount (уплаченная сумма)
2014-05-17 15:15:00 UTC	-73.99955	40.7606	-73.99965	40.72522	1	31
2013-12-09 15:03:00 UTC	-73.99095	40.749772	-73.870807	40.77407	1	34.33
2013-04-18 08:48:00 UTC	-73.973102	40.785075	-74.011462	40.708307	1	29
2009-11-05 06:47:00 UTC	-73.980313	40.744282	-74.015285	40.711458	1	14.9
2009-05-21 09:47:06 UTC	-73.901887	40.764021	-73.901795	40.763612	1	12.8

Приведенный выше набор данных содержит информацию о поездках в нью-йоркском такси с такими признаками, как временна́я метка посадки, широта и долгота посадки и высадки, а также количество пассажиров. Меткой здесь является уплаченная сумма (`fare_amount`), т. е. стоимость проезда в такси. Какие синтетические признаки могут быть релевантными для этого набора данных?

Их может быть много. Давайте рассмотрим дату и время посадки (`pickup_datetime`). Основываясь на этом признаке, мы можем воспользоваться информацией о часе и дне недели поездки. Каждая из них является категориальной переменной, и, безусловно, обе содержат предсказательную способность в определении стоимости проезда в такси. Для этого набора данных имеет смысл рассмотреть соединение признаков дня недели (`day_of_week`) и часа дня (`hour_of_day`), поскольку разумно предположить, что поездки в такси в понедельник вечером в 5 часов должны рассматриваться иначе, чем поездки в такси в пятницу в то же самое время (табл. 2.14).

Таблица 2.14. *Предварительный анализ данных, которые мы используем для создания соединения признаков: столбцы дней недели и часов дня*

day_of_week	hour_of_day
Воскресенье	00
Воскресенье	01
...	...
Суббота	23

[38] Блокнот feature_cross.ipynb в репозитории этой книги (**https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/feature_cross.ipynb**) поможет вам проследить за изложением темы.

Соединение этих двух признаков будет представлять собой 168-мерный вектор с одним активным состоянием (24 часа × 7 дней = 168), где пример "Понедельник в 5 часов вечера" будет занимать один индекс, обозначая "день недели `day_of_week`, равный понедельнику, конкатенируется с часом дня `hour_of_day`, равным 17".

Хотя эти два признака важны сами по себе, допустив возможность соединения признаков `hour_of_day` и `day_of_week`, мы тем самым помогаем модели предсказания стоимости проезда в такси понять, что час пик в конце недели влияет на продолжительность проезда в такси и, следовательно, на стоимость поездки.

Соединения признаков в BigQuery ML

В целях создания синтетических признаков в BigQuery мы можем воспользоваться функцией `ML.FEATURE_CROSS`, передав в нее структуру `STRUCT`, состоящую из признаков `day_of_week` и `hour_of_day`:

```
ML.FEATURE_CROSS(STRUCT(day_of_week,hour_of_week)) AS day_X_hour
```

Спецификатор `STRUCT` создает упорядоченную пару признаков. Если наш фреймворк не поддерживает функцию соединенения признаков, мы можем получить тот же эффект, используя строковую конкатенацию:

```
CONCAT(CAST(day_of_week AS STRING),
       CAST(hour_of_week AS STRING)) AS day_X_hour
```

Полный тренировочный пример для задачи о рождаемости показан ниже, с соединением признаковых столбцов `is_male` (мальчик?) и `plurality` (многоплодность), используемым в качестве признаков; обратитесь к полному исходному коду в репозитории[39] этой книги:

```
CREATE OR REPLACE MODEL babyweight.natality_model_feat_eng
TRANSFORM(weight_pounds,
    is_male,
    plurality,
    gestation_weeks,
    mother_age,
    CAST(mother_race AS string) AS mother_race,
    ML.FEATURE_CROSS(
            STRUCT(
                is_male,
                plurality)
       ) AS gender_X_plurality)
OPTIONS
  (MODEL_TYPE='linear_reg',
   INPUT_LABEL_COLS=['weight_pounds'],
   DATA_SPLIT_METHOD="NO_SPLIT") AS
```

[39] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/feature_cross.ipynb**.

```
SELECT
  *
FROM
    babyweight.babyweight_data_train
```

Здесь при генерировании признаков для модели рождаемости используется паттерн "Преобразователь" (Transform, *см. главу 6*). Он также позволяет модели "помнить" о том, что нужно создать соединение полей входных данных во время предсказания.

При наличии достаточного объема данных паттерн "Синтетический признак" (Feature Cross) позволяет моделям становиться проще. В наборе данных о рождаемости показатель RMSE для оценочного набора в линейной модели с паттерном "Синтетический признак" составляет 1,056. С другой стороны, тренировка глубокой нейронной сети в BigQuery ML на том же наборе данных без соединения признаков дает RMSE 1,074. Несмотря на использование гораздо более простой линейной модели, наша результативность немного улучшилась и время тренировки тоже резко сократилось.

Соединение признаков в TensorFlow

В целях реализации синтетического признака с использованием признаков `is_male` и `plurality` в TensorFlow мы применяем метод `tf.feature_column.crossed_column`. Метод `crossed_column` принимает два аргумента: список ключей скрещиваемых признаков и размер хеш-корзины `hash_bucket_size`. Скрещенные признаки будут хешироваться в соответствии с размером хеш-корзины, поэтому он должен быть достаточно большим, чтобы удобным образом уменьшить вероятность коллизий. Поскольку входной признак `is_male` может принимать 3 значения (истина, ложь, неизвестно), а входной признак `plurality` может принимать 6 значений (`Одноплодный(1)`, `Двойняшки(2)`, `Тройняшки(3)`, `Четверняшки(4)`, `Пятерняшки(5)`, `Многоплодные(2)`), существует 18 возможных пар признаков (`is_male`, `plurality`). Если установить размер хеш-корзины `hash_bucket_size` равным 1000, то мы сможем быть уверены в отсутствии коллизий на 85%.

Наконец, для использования скрещенного столбца в модели, основанной на глубокой нейросети (DNN), нам нужно обернуть его либо в индикаторный столбец `indicator_column`, либо в столбец векторного вложения `embedding_column` в зависимости от того, чего мы хотим добиться: закодировать его с одним активным состоянием либо представить в более низкой размерности (*см. "Паттерн 2. Векторные вложения" ранее в этой главе*):

```
gender_x_plurality = fc.crossed_column(["is_male", "plurality"],
                                       hash_bucket_size=1000)
crossed_feature = fc.embedding_column(gender_x_plurality, dimension=2)
```

либо

```
gender_x_plurality = fc.crossed_column(["is_male", "plurality"],
                                       hash_bucket_size=1000)
crossed_feature = fc.indicator_column(gender_x_plurality)
```

Почему это работает

Синтетические признаки являются ценным средством инженерии признаков. Они обеспечивают более высокую сложность и выразительность, а также придают больше способностей простым моделям. Подумайте еще раз о соединении признаков `is_male` и `plurality` в наборе данных о рождаемости. Паттерн "Синтетический признак" позволяет модели рассматривать мальчиков-близнецов отдельно от девочек-близнецов, отдельно от мальчиков-тройняшек, отдельно от одноплодных девочек и т. д. Когда мы используем столбец `indicator_column`, модель может рассматривать каждое результирующее соединение как независимую переменную, по существу добавляя в модель 18 дополнительных двоичных категориальных признаков (рис. 2.16).

Синтетические признаки хорошо масштабируются до массивных данных. Хотя внесение дополнительных слоев в глубокую нейронную сеть потенциально может обеспечивать достаточный объем нелинейности для обучения характера поведения пар (`is_male`, `plurality`), этот шаг резко увеличивает время тренировки. Оперируя набором данных о рождаемости, мы наблюдали, что результативность линейной модели с соединением признаков, натренированной в BigQuery ML, сравнима с глубокой нейросетью (DNN), натренированной без соединения признаков. Однако тренировка линейной модели проходит значительно быстрее.

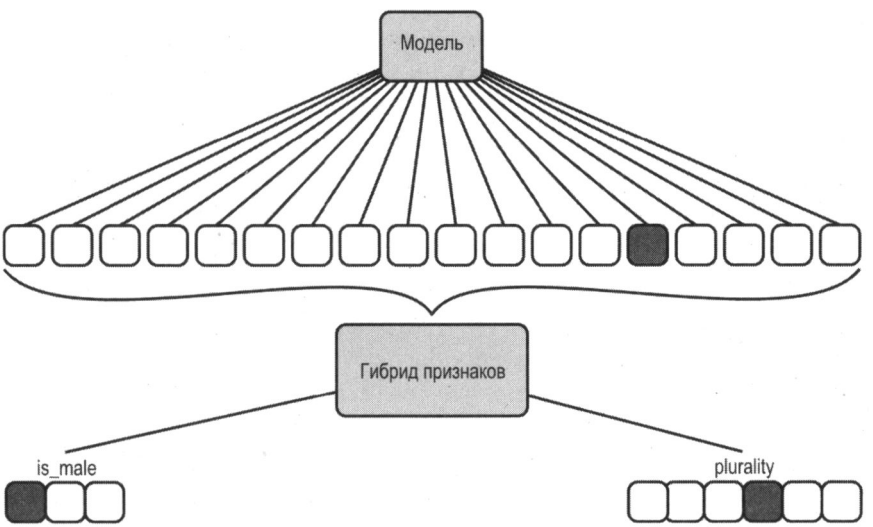


Рис. 2.16. Синтетический признак из `is_male` и `plurality` создает в ML-модели дополнительные 18 двоичных признаков

В табл. 2.15 сравниваются время тренировки в BigQuery ML и оценочная потеря как для линейной модели с синтетическим признаком (`is_male`, `plurality`), так и для глубокой нейронной сети без скрещивания каких-либо признаков.

Простая линейная регрессия достигает сопоставимой ошибки на оценочном наборе, но тренируется в 100 раз быстрее. Сочетание синтетических признаков с массив-

ными данными является альтернативной стратегией, призванной усваивать сложные связи в тренировочных данных.

Таблица 2.15. *Сравнение тренировочных метрик BigQuery ML для моделей с ситетическими признаками и без них*

Тип модели	Включая ситетический признак	Время тренировки, мин	Оценочная потеря (RMSE)
Линейная	Да	0,42	1,05
Глубокая нейросетевая (DNN)	Нет	48	1,07

Компромиссы и альтернативы

Хотя мы обсуждали синтетические признаки как подход к манипулированию категориальными переменными, они, с некоторой предобработкой, могут применяться и к числовым признакам. Синтетические признаки приводят к разреженности в моделях и часто используются вместе с техническими приемами, которые противодействуют этой разреженности.

Манипулирование числовыми признаками

Мы бы ни за что не захотели скрещивать признаки с непрерывными входными данными. Следует всегда помнить, что если одна входная переменная принимает m возможных значений, а другая входная переменная принимает n возможных значений, то скрещивание этих двух признаков приведет к $m \times n$ элементам. Числовые входные данные являются плотными, т. к. представляют собой континуум значений. Было бы невозможно перечислить все возможные значения в скрещивании/соединении непрерывных входных данных.

Вместо этого если наши данные являются непрерывными, то перед тем, как применять срещивание признаков, мы можем сгруппировать данные в корзины, сделав их категориальными. Например, широта и долгота являются непрерывными входными переменными, и интуитивно понятно, что синтетический признак с использованием этих входных переменных имеет смысл, поскольку местоположение определяется упорядоченной парой широты и долготы. Однако вместо того, чтобы создавать признак с использованием сырой широты и долготы, мы бы разместили эти непрерывные значения в интервальные корзины и скрестили сгруппированную широту `binned_latitude` и сгруппированную долготу `binned_longitude`:

```
import tensorflow.feature_column as fc

# Создать столбец сгруппированного в корзины признака для широты.
latitude_as_numeric = fc.numeric_column("latitude")
lat_bucketized = fc.bucketized_column(latitude_as_numeric,
                                      lat_boundaries)
```

```
# Создать столбец сгруппированного в корзины признака для долготы.
longitude_as_numeric = fc.numeric_column("longitude")
lon_bucketized = fc.bucketized_column(longitude_as_numeric,
                                      lon_boundaries)

# Создать синтетический признак широты и долготы.
lat_x_lon = fc.crossed_column([lat_bucketized, lon_bucketized],
                              hash_bucket_size=nbuckets**4)

crossed_feature = fc.indicator_column(lat_x_lon)
```

Манипулирование высокой кардинальностью

Поскольку кардинальность категорий, получающаяся из скрещивания признаков, многократно возрастает по отношению к кардинальности входных признаков, синтетические признаки приводят к разреженности данных на входе в модель. Даже с соединением признаков `day_of_week` и `hour_of_day` гибрид будет разреженным вектором размерности 168 (рис. 2.17).

Бывает полезно пропустить синтетический признак через слой векторного вложения (*см. разд. "Паттерн 2. Векторные вложения" ранее в этой главе*), чтобы создать более низкоразмерное представление (рис. 2.18).

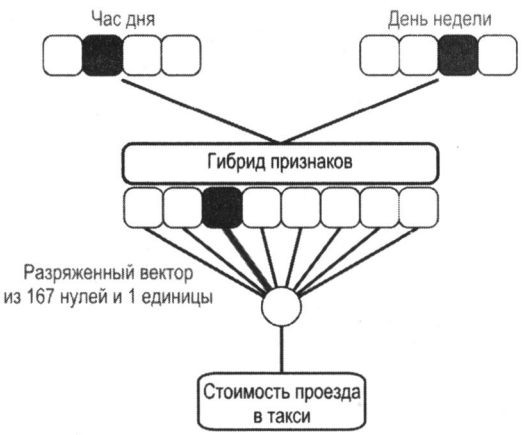


Рис. 2.17. Синтетический признак `day_of_week` и `hour_of_day` создает разреженный вектор размерности 168

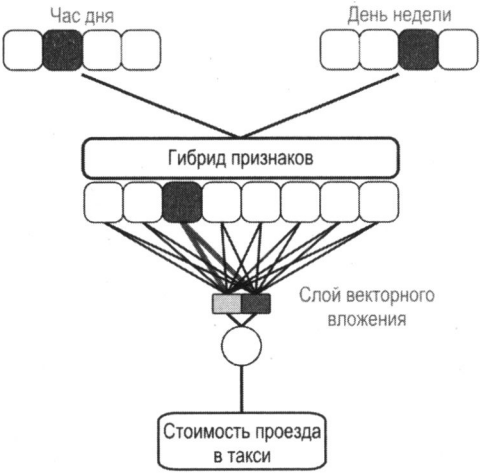


Рис. 2.18. Слой векторного вложения является полезным подходом к решению проблемы разреженности результата скрещивания признаков

Поскольку паттерн "Векторные вложения" позволяет нам улавливать отношения подобия, пропускание синтетического признака через слой векторного вложения позволяет модели делать обобщения в отношении того, насколько некоторые скрещивания признаков, поступающие из пар сочетаний часов дня и дней недели, влияют на результат работы модели. В приведенном выше примере с широтой и

долготой мы могли бы использовать признаковый столбец векторного вложения вместо индикаторного столбца:

```
crossed_feature = fc.embedding_column(lat_x_lon, dimension=2)
```

Потребность в регуляризации

При скрещивании двух категориальных признаков, обоих с большой кардинальностью, мы получаем синтетический признак с мультипликативной кардинальностью. Естественно, что при наличии большего числа категорий для отдельного признака число категорий в синтетическом признаке может резко увеличиться. Если оно дойдет до того, что отдельные корзины будут иметь слишком мало элементов, то это будет препятствовать способности модели выполнять обобщения. Вспомните пример с широтой и долготой. Если бы для широты и долготы мы взяли очень мелкие корзины, то синтетический признак был бы настолько детальным, что позволил бы модели запомнить каждую точку на карте. Однако если бы это запоминание было основано только на горстке примеров, то такое запоминание на самом деле было бы переобучением[40].

В целях иллюстрации возьмем пример предсказания стоимости поездки в нью-йоркском такси с учетом места посадки и высадки и времени посадки[41]:

```
CREATE OR REPLACE MODEL mlpatterns.taxi_l2reg
TRANSFORM(
  fare_amount
, ML.FEATURE_CROSS(STRUCT(CAST(EXTRACT(DAYOFWEEK FROM pickup_datetime)
                    AS STRING) AS dayofweek,
                          CAST(EXTRACT(HOUR FROM pickup_datetime)
                    AS STRING) AS hourofday), 2) AS day_hr
, CONCAT(
    ML.BUCKETIZE(pickuplon, GENERATE_ARRAY(-78, -70, 0.01)),
    ML.BUCKETIZE(pickuplat, GENERATE_ARRAY(37, 45, 0.01)),
    ML.BUCKETIZE(dropofflon, GENERATE_ARRAY(-78, -70, 0.01)),
    ML.BUCKETIZE(dropofflat, GENERATE_ARRAY(37, 45, 0.01))
  ) AS pickup_and_dropoff
)
OPTIONS(input_label_cols=['fare_amount'],
        model_type='linear_reg', l2_reg=0.1)
AS
SELECT * FROM mlpatterns.taxi_data
```

Здесь имеются два синтетических признака: один во времени (день недели и час дня), а другой в пространстве (места посадки и высадки). Месторасположение,

---

[40] То есть чересчур плотно прилегало бы к данным. — *Прим. перев.*

[41] Полный исходный код находится в блокноте 02_data_representation/feature_cross.ipynb репозитория исходного кода этой книги.

в частности, имеет очень высокую кардинальность, и вполне вероятно, что некоторые из корзин будут иметь очень мало примеров.

По этой причине целесообразно сочетать синтетические признаки с регуляризацией L1, которая способствует разреженности признаков, либо регуляризацией L2, которая ограничивает переобучение. Это позволяет нашей модели игнорировать посторонние шумы, генерируемые многими синтетическими признаками, и бороться с переобучением. Действительно, на этом наборе данных регуляризация слегка улучшает RMSE, на 0,3%.

В качестве смежного замечания, выбирая признаки для сочетания в гибриде, мы не хотели бы скрещивать два признака, которые сильно коррелируют. Синтетический признак (или гибрид признаков) можно рассматривать как сочетание двух признаков для создания упорядоченной пары. На самом деле термин "гибрид" в гибриде признаков относится к декартову произведению. Если два признака сильно коррелированы, то "размах" их гибридизации не приносит в модель никакой новой информации. В качестве крайнего примера предположим, что у нас есть два признака, `x_1` и `x_2`, где $x\_2 = 5 \cdot x\_1$. Группировка значений признаков `x_1` и `x_2` в корзины по их знаку и создание синтетического признака все равно приведут к появлению четырех новых булевых признаков. Однако из-за зависимости `x_1` и `x_2` два из этих четырех признаков фактически являются пустыми, а два других — это в точности те две корзины, которые были созданы для `x_1`.

ПАТТЕРН 4. Мультимодальный вход

Паттерн "Мультимодальный вход" (Multimodal Input) решает задачу представления разных типов данных или данных, которые могут выражаться сложным образом, путем конкатенирования всех имеющихся представлений данных.

Постановка задачи

В типичной ситуации значение на входе в модель может представляться в виде числа либо категории, снимка либо текста произвольной формы. Многие готовые к работе модели определяются только для отдельно взятых типов входных данных — стандартная модель классифицирования снимков, такая как Resnet-50, например, не имеет возможности манипулировать входными данными, отличными от снимков.

Для того чтобы понять, зачем необходимо мультимодальное представление на входе, предположим, что у нас есть камера, которая ведет съемку на перекрестке с целью выявления нарушений правил дорожного движения. Мы хотим, чтобы наша модель манипулировала и снимковыми данными (кадрами с камеры), и некоторыми метаданными, касающимися условий, когда этот снимок был сделан (временем суток, днем недели, погодой и т. д.), как показано на рис. 2.19.

Та же задача возникает во время тренировки модели, принимающей структурированные данные, в которой одной из переменных на входе является текст произвольной формы. В отличие от числовых данных, снимки и текст не могут подаваться

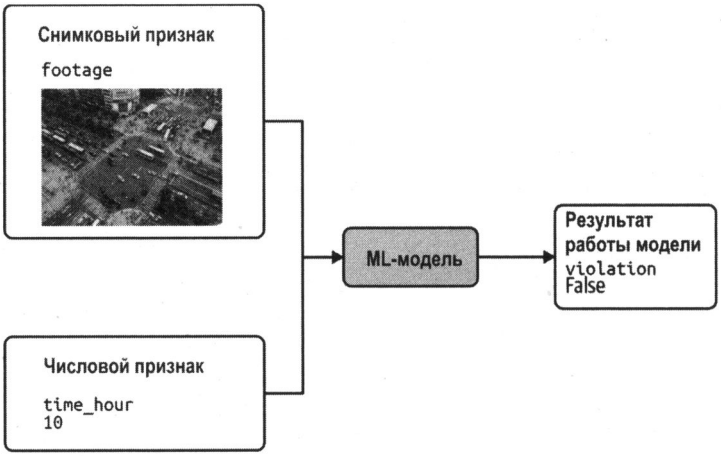


Рис. 2.19. Модель, сочетающая снимковые и числовые признаки, позволяет предсказывать нарушение правил дорожного движения на кадре видеозаписи перекрестка

в модель непосредственно. Значит, нам понадобится представить снимковые и текстовые входные переменные таким образом, чтобы наша модель могла их понимать (обычно с помощью паттерна "Векторные вложения"), а затем объединить эти входные данные с другими табличными[42] признаками. Например, у нас вполне может возникнуть потребность в предсказывании оценки ресторана его посетителем,

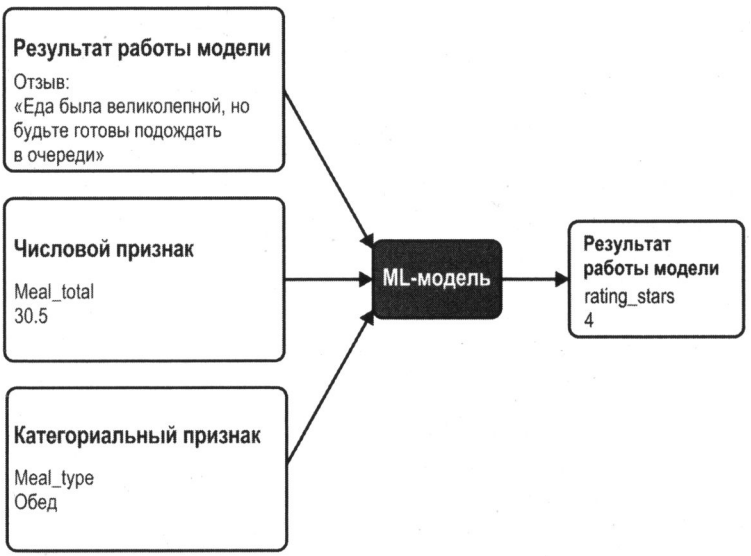


Рис. 2.20. Модель, сочетающая входной текст произвольной формы с табличными данными для предсказания оценки ресторана посетителем

[42] Мы используем термин "табличные данные" для обозначения числовых и категориальных входных данных, но не текста произвольной формы. Табличные данные можно рассматривать как что-то, что обычно можно найти в электронной таблице. Например, такие значения, как возраст, тип автомобиля, цена или число отработанных часов. Табличные данные не включают текст произвольной формы, такой как описания или отзывы.

основываясь на тексте его отзыва и других атрибутах, таких как оплаченный счет и тип заказанных им блюд: обед или ужин (рис. 2.20).

Решение

Для начала возьмем приведенный выше пример с текстом из отзыва о ресторане в сочетании с табличными метаданными о еде, на которые имеются ссылки в отзыве. Сначала мы объединим числовые и категориальные признаки. Для признака типа еды (`meal_type`) есть три возможных варианта, и поэтому мы можем превратить их в кодировку с одним активным состоянием и представить ужин как `[0, 0, 1]`. Представив этот категориальный признак массивом, мы теперь можем объединить его с признаком общей суммы (`meal_total`), добавив стоимость еды в качестве четвертого элемента массива: `[0, 0, 1, 30.5]`.

При кодировании текста для моделей машинного обучения широко принято использовать подход на основе паттерна "Векторные вложения". Если бы наша модель имела только текст, то мы могли бы представить его как слой векторного вложения, используя следующий ниже исходный код tf.keras:

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Embedding

model = Sequential()
model.add(Embedding(batch_size, 64, input_length=30))
```

Здесь нам нужно сгладить векторное вложение[43], чтобы выполнить конкатенацию `meal_type` с `meal_total`:

```
model.add(Flatten())
```

Далее мы могли бы применить серию плотных (`Dense`) слоев для преобразования этого очень крупного массива[44] в меньшие, в итоге на выходе получив результат, который представляет собой массив, скажем, из трех чисел:

```
model.add(Dense(3, activation="relu"))
```

Теперь нам нужно конкатенировать эти три числа, которые образуют векторное вложение предложения в отзыве с более ранними входными данными: `[0, 0, 1, 30.5, 0.75, -0.82, 0.45]`.

Для этого мы воспользуемся функциональным API Keras и применим те же шаги. Слои, выстраиваемые с помощью функционального API, могут вызываться, что позволяет нам связывать их в цепочку, начиная с входного слоя[45]. Для того чтобы

---

[43] Когда мы передаем кодированный массив из 30 слов в модель, слой Keras трансформирует его в представление в виде 64-размерного векторного вложения, поэтому отзыв у нас будет представлен матрицей 64×30.

[44] Отправной точкой является массив, состоящий из 1920 чисел.

[45] Полный исходный код модели см. в блокноте 02_data_representation/mixed_representation.ipynb репозитория исходного кода этой книги.

использовать их, мы сначала определим слои векторного вложения и табличные слои:

```
embedding_input = Input(shape=(30,))
embedding_layer = Embedding(batch_size, 64)(embedding_input)
embedding_layer = Flatten()(embedding_layer)
embedding_layer = Dense(3, activation='relu')(embedding_layer)

tabular_input = Input(shape=(4,))
tabular_layer = Dense(32, activation='relu')(tabular_input)
```

Обратите внимание, что мы определили входные (`Input`) части обоих этих слоев как самостоятельные переменные. Это обусловлено тем, что нам потребуется передать входные слои, когда мы будем строить модель с помощью функционального API. Затем мы создадим конкатенированный слой, подадим его в выходной слой и, наконец, создадим модель, передав изначальные входные слои `Input`, которые мы определили выше:

```
merged_input = keras.layers.concatenate([embedding_layer, tabular_layer])
merged_dense = Dense(16)(merged_input)
output = Dense(1)(merged_dense)

model = Model(inputs=[embedding_input, tabular_input], outputs=output)
merged_dense = Dense(16, activation='relu')(merged_input)
output = Dense(1)(merged_dense)

model = Model(inputs=[embedding_input, tabular_input], outputs=output)
```

Теперь у нас есть единая модель, которая на входе принимает мультимодальные данные.

Компромиссы и альтернативы

Как мы только что убедились, паттерн "Мультимодальный вход" разведывает подходы к представлению *разных форматов входных данных* в одной и той же модели. В дополнение к смешиванию разных *типов* данных, возможно, мы также захотим представлять *одни и те же данные по-разному* с целью помочь нашей модели выявлять закономерности. Например, мы можем иметь поле оценок, которое ассоциировано со шкалой от 1 до 5 звезд, и рассматривать это поле оценок одновременно как числовое и как категориальное. Здесь мы ссылаемся на *мультимодальное представление* данных на входе как на одно из двух:

- сочетание разных типов данных, таких как снимки + метаданные;
- представление сложных данных по-разному.

Мы начнем с того, что рассмотрим приемы представления табличных данных самыми разными способами, а затем обратимся к текстовым и снимковым данным.

Табличные данные самыми разными способами

Давайте ознакомимся со способами представления табличных данных по-разному для одной и той же модели, а для этого вернемся к примеру с отзывом посетителя о ресторане. Пусть теперь входной переменной для нашей модели является оценка, и мы пытаемся предсказать полезность отзыва (т. е. количество людей, которым отзыв понравился). В качестве входной модельной переменной оценка может быть представлена как целочисленное значение в диапазоне от 1 до 5 и как категориальный признак. В целях представления оценки категориально мы можем сгруппировать ее в корзины. Принцип, по которому мы группируем данные в корзины, зависит от нас, а также от набора данных и варианта использования. Для простоты допустим, что мы хотим создать две корзины: "хорошую" и "плохую". "Хорошая" корзина содержит оценки 4 и 5, а "плохая" — 3 и ниже. Затем мы можем создать булево значение, кодирующее корзины с оценками, и конкатенировать целочисленное и булево значения в один массив (полный исходный код[46] находится в репозитории на GitHub).

Вот как это может выглядеть для небольшого набора данных с тремя точками данных:

```
rating_data = [2, 3, 5]

def good_or_bad(rating):
    if rating > 3:
        return 1
    else:
        return 0

rating_processed = []

for i in rating_data:
    rating_processed.append([i, good_or_bad(i)])
```

Результирующий признак представляет собой двухэлементный массив, состоящий из целочисленной оценки и его булева представления:

```
[[2, 0], [3, 0], [5, 1]]
```

Если вместо этого мы решили бы создать более двух корзин, то закодировали бы каждое входное значение с использованием одного активного состояния и присоединили бы этот кодированный массив к целочисленному представлению.

Причина, по которой полезно представлять оценку двумя способами, заключается в том, что значение оценки, измеряемое от 1 до 5 звезд, необязательно увеличивается линейно. Оценки 4 и 5 очень похожи, а оценки от 1 до 3, скорее всего, указывают на то, что автор отзыва не был удовлетворен. Оценивание того, что вам не нравится, одной, двумя или тремя звездами часто связано не с самим отзывом, а

[46] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/mixed_representation.ipynb**.

с вашей тенденциозностью как автора отзыва. Несмотря на это, все же полезно сохранить более гранулярную информацию, присутствующую в звездной оценке, поэтому мы кодируем ее двумя способами.

Вдобавок, следует подумать и о признаках с более крупным интервалом, чем от 1 до 5, наподобие расстояния между домом автора отзыва и рестораном. Если для того, чтобы попасть в ресторан, кому-то приходится ехать два часа, то его отзыв может быть более важным, чем отзыв того, кому для этого нужно лишь перейти улицу. В этом случае мы могли бы иметь более высокие выбросные значения, и поэтому имело бы смысл ограничить числовое представление расстояния порогом на уровне примерно 50 км и включить отдельное категориальное представление расстояния. Категориальный признак можно сгруппировать в корзины "внутри штата", "внутри страны" и "за границей".

Мультимодальное представление текста

И текст, и снимки не структурированы и требуют больше преобразований, чем табличные данные. Представляя их в различных форматах, мы помогаем нашим моделям извлекать больше закономерностей. Здесь мы будем отталкиваться от обсуждения темы текстовых моделей, начатого в предыдущем разделе, рассмотрев разные подходы к представлению текстовых данных. Затем мы введем снимки и углубимся в несколько вариантов представления снимковых данных в ML-моделях.

Текстовые данные самыми разными способами. Учитывая сложную природу текстовых данных, можно утверждать, что существует масса подходов к извлечению из них смысла. Паттерн "Векторные вложения" обеспечивает модель возможностью группировать похожие слова вместе, выявлять связи между словами и понимать синтаксические элементы текста. В то время как представление текста посредством векторного вложения слов наиболее точно отражает то, как люди интуитивно понимают язык, существуют и дополнительные представления текста, которые могут максимизировать способность нашей модели выполнять конкретную задачу предсказания. В этом разделе мы рассмотрим подход к представлению текста в формате мешка слов, а также извлечение табличных признаков из текста.

В целях демонстрации представления текстовых данных мы будем ссылаться на набор данных, содержащий текст миллионов вопросов и ответов с веб-сайта Stack Overflow[47], а также метаданные о каждом посте. Например, следующий ниже запрос даст нам подмножество вопросов, помеченных тегами "keras", "matplotlib" или "pandas", а также число полученных ответов на каждый вопрос:

```
SELECT
  title,
  answer_count,
  REPLACE(tags, "|", ",") as tags
FROM
  `bigquery-public-data.stackoverflow.posts_questions`
```

[47] Этот набор данных доступен в BigQuery: bigquery-public-data.stackoverflow.posts_questions.

```
WHERE
  REGEXP_CONTAINS( tags, r"(?:keras|matplotlib|pandas)")
```

Выполнив этот запрос, мы получим результат, как в табл. 2.16.

Таблица 2.16. *Результат выполнения запроса*

Строка	title	answer_count	tags
1	Building a new column in a pandas dataframe by matching string values in a list	6	python,python-2.7,pandas,replace,nested-loops
2	Extracting specific selected columns to new DataFrame as a copy	6	python,pandas,chained-assignment
3	Where do I call the BatchNormalization function in Keras?	7	python,keras,neural-network,datascience, batch-normalization
4	Using Excel like solver in Python or SQL	8	python,sql,numpy,pandas,solver

При представлении текста с использованием подхода на основе мешка слов (Bag of words, BOW) мы воображаем, что каждый вводимый в нашу модель текст представляет собой мешок костяшек для игры в слова "Скраббл", в которой каждая костяшка содержит не одну букву, а целое слово. BOW не сохраняет порядок текста, но он обнаруживает присутствие или отсутствие тех или иных слов в каждой части текста, которую мы посылаем в нашу модель. Этот подход представляет собой разновидность кодирования с несколькими активными состояниями, при котором каждый подаваемый в модель текст конвертируется в массив из единиц и нулей. Каждый индекс в этом BOW-кодированном массиве соответствует слову из нашего словаря.

Как работает мешок слов

Первым шагом в кодировании в формате мешка слов (BOW) является выбор размера словаря, который будет включать в себя верхние N наиболее часто встречающихся слов в нашем текстовом корпусе. Теоретически размер словаря может быть равен числу уникальных слов во всем наборе данных. Однако это привело бы к очень большим входным массивам, состоящим в основном из нулей, поскольку многие слова могут быть присущи только одному вопросу. Вместо этого мы хотим выбрать достаточной малый размер словаря, чтобы тот содержал ключевые, повторяющиеся слова, которые передают смысл для нашей задачи предсказания, но при этом он должен быть достаточно большим, чтобы словарь не ограничивался словами, встречающимися почти в каждом вопросе (к примеру, такими как the, is, and и т. д.).

Каждое значение, подаваемое на вход модели, будет представлять собой массив размером с наш словарь. Следовательно, это представление в формате BOW полностью игнорирует слова, которые не входят в словарь. Для выбора размера словаря нет никакого магического числа или процента — полезно попробовать несколько из них и посмотреть то, которое работает на модели лучше всего.

В целях понимания кодировки в формате BOW давайте сначала рассмотрим упрощенный пример. Для примера, предположим, что мы предсказываем тег вопроса Stack Overflow из списка, состоящего из трех возможных тегов: "pandas", "keras" и "matplotlib". Для простоты предположим, что наш словарь состоит только из 10 слов, перечисленных ниже:

```
dataframe
layer
series
graph
column
plot
color
axes
read_csv
activation
```

Этот список является нашим словарным индексом, и каждое подаваемое нами на вход в модель значение будет представлять собой 10-элементный массив, в котором каждый индекс соответствует одному из перечисленных выше слов. Например, 1 в первом индексе входного массива означает, что отдельно взятый вопрос содержит слово dataframe. Для понимания кодировки в формате BOW с точки зрения нашей модели представьте, что мы учим новый язык и приведенные выше 10 слов — это единственные слова, которые мы знаем. Каждое выполняемое нами "предсказание" будет основываться исключительно на наличии или отсутствии этих 10 слов и будет игнорировать любые слова за пределами этого списка.

Допустим, есть заголовок фразы "How to plot dataframe bar graph" (Как построить гистограмму кадра данных). Встает вопрос: как его преобразовать в представление в формате BOW? Сначала, давайте отметим в этом предложении слова, которые появляются в словаре: plot, dataframe и graph. Подход на основе мешка слов проигнорирует другие слова. Используя приведенный выше словарный индекс, это предложение примет следующий вид:

```
[ 1 0 0 1 0 1 0 0 0 0 ]
```

Обратите внимание, что единицы в этом массиве соответствуют индексам слов dataframe, graph и plot в порядке их появления в списке. На рис. 2.21 показан результат преобразования входных данных из сырого текста в BOW-кодированный массив на основе нашего словаря.

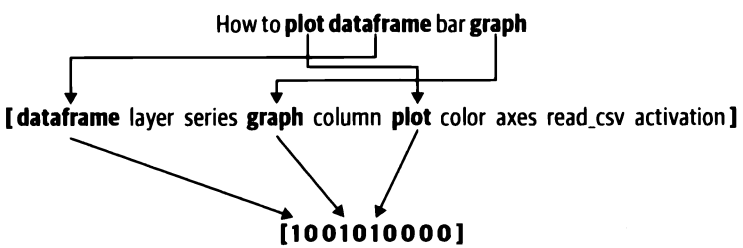


Рис. 2.21. Сырой входной текст → выявление присутствующих в этом тексте слов с точки зрения нашего словаря → преобразование в кодировку в формате BOW с несколькими активными состояниями

> В Keras есть несколько утилитных методов кодирования текста в формате BOW, поэтому нам не нужно писать исходный код для выявления слов вверху нашего текстового корпуса и кодирования с нуля сырого текста в массивы с несколькими активными состояниями.

Учитывая, что существуют два разных подхода к представлению текста (векторное вложение и BOW), возникает вопрос: какой подход мы должны выбрать для поставленной задачи? Как и во многих аспектах машинного обучения, все зависит от набора данных, природы задачи предсказания и типа модели, которую мы планируем использовать.

Векторные вложения вносят в нашу модель дополнительный слой и обеспечивают дополнительной информацией о смысле слова, которая недоступна из кодировки в формате BOW. Однако векторные вложения требуют тренировки модели (если только наша задача не предусматривает использование предварительно натренированного векторного вложения). В то время как модель на основе глубокого обучения может достигать более высокой точности, мы также можем попробовать использовать кодирование в формате BOW в линейно регрессионной модели или в модели, основанной на дереве решений, используя такие вычислительные фреймворки, как scikit-learn или XGBoost. Использовать кодирование в формате BOW полезно с более простым типом модели, в целях быстрого прототипирования или проверки того, что выбранная нами задача предсказания будет работать на нашем наборе данных. В отличие от векторных вложений, в BOW не учитывается порядок или смысл слов в текстовом документе. Если любой из этих аспектов важен для задачи предсказания, то векторные вложения будут наилучшим подходом.

Также бывает полезно построить модель на основе глубокого обучения, которая сочетает в себе представления как в формате BOW, так и в формате векторного вложения текста, чтобы извлекать больше закономерностей из данных. Для этого мы можем применить паттерн "Мультимодальный вход", за исключением того, что вместо конкатенирования текстовых и табличных признаков мы будем конкатенировать представления в формате векторного вложения и BOW (см. исходный код[48] на GitHub). Здесь форма входного слоя `Input` будет размером словаря BOW-представления. Далее перечислены некоторые выгоды от представления текста самыми разными способами.

- BOW-кодирование обеспечивает сильные сигналы для наиболее значимых слов в словаре, в то время как векторное вложение может идентифицировать связи между словами в гораздо более крупном словаре.
- Если у нас есть многоязыковый текст, мы можем построить векторные вложения (или BOW-кодировки) для каждого из них и конкатенировать их.
- Векторные вложения могут кодировать частоту слов в тексте, тогда как BOW будет трактовать присутствие каждого слова как булево значение. Оба представления имеют ценность.

---

[48] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/mixed_representation.ipynb**.

- BOW-кодирование может идентифицировать закономерности среди отзывов, которые содержат слово "amazing" (удивительно), в то время как векторное вложение может научиться соотносить словосочетание "not amazing" (не удивительно) с отзывом ниже среднего. Опять же, оба эти представления имеют ценность.

Извлечение табличных признаков из текста. Помимо кодирования сырых текстовых данных есть и другие характеристики текста, которые могут быть представлены в качестве табличных признаков. Предположим, что мы строим модель, чтобы предсказывать возможность получения ответа на вопрос Stack Overflow. Релевантными для тренировки модели на этой задаче могут быть самые разные факторы, относящиеся к тексту, но не связанные с самими словами. Например, возможно, на вероятность ответа на вопрос влияет длина вопроса или наличие вопросительного знака. Однако, когда мы создаем векторное вложение, мы обычно усекаем слова до определенной длины. И в этом представлении данных фактическая длина вопроса теряется. Точно так же нередко убираются и знаки препинания. Мы можем воспользоваться паттерном "Мультимодальный вход", чтобы вернуть эту потерянную информацию в модель.

В следующем ниже запросе мы извлекаем несколько табличных признаков из поля `title` набора данных Stack Overflow для предсказания возможности получения ответа на вопрос:

```
SELECT
  LENGTH(title) AS title_len,
  ARRAY_LENGTH(SPLIT(title, " ")) AS word_count,
  ENDS_WITH(title, "?") AS ends_with_q_mark,
IF
  (answer_count > 0,
    1,
    0) AS is_answered,
FROM
  `bigquery-public-data.stackoverflow.posts_questions`
```

В итоге будет получен результат, как в табл. 2.17.

Таблца 2.17. *Результат запроса для предсказания возможности получения ответа*

Строка	title_len	word_count	ends_with_q_mark	is_answered
1	84	14	true	0
2	104	16	false	0
3	85	19	true	1
4	88	14	false	1
5	17	3	false	1

В дополнение к этим признакам, непосредственно извлекаемым из заголовка вопроса, мы также могли бы представить метаданные о вопросе в качестве признаков.

Например, мы могли бы добавить признаки, представляющие число имевшихся в вопросе тегов и день недели, когда он был опубликован. Затем мы могли бы совместить эти табличные признаки с нашим кодированным текстом и передать оба представления в модель, используя конкатенатный слой Keras, чтобы совместить текстовый массив, кодированный в формате мешка слов, с табличными метаданными, описывающими текст.

Мультимодальное представление снимков

Аналогично нашему анализу векторных вложений и кодирования текста в формате мешка слов существует масса подходов к представлению снимковых данных при их подготовке для модели ML. Как и сырой текст, снимки нельзя подавать в модель непосредственно и нужно преобразовывать в понятный для модели числовой формат. Мы начнем с обсуждения некоторых распространенных подходов к представлению снимковых данных: в качестве пиксельных значений, в качестве наборов плиток и в качестве наборов оконных последовательностей. Паттерн "Мультимодальный вход" позволяет использовать более одного представления изображения в нашей модели.

Снимки в качестве пиксельных значений. Снимки представляют собой массивы пиксельных значений. Например, черно-белое изображение содержит пиксельные значения в диапазоне от 0 до 255. И, следовательно, черно-белый снимок размером 28×28 пикселов можно представить в модели в формате массива размером 28×28 с целыми значениями в диапазоне от 0 до 255. В этом разделе мы будем ссылаться на набор данных MNIST — популярный набор данных ML, который содержит изображения рукописных цифр.

С помощью API Sequential[49] мы можем представить пиксельные значения снимков MNIST, используя слой `Flatten`, который сглаживает снимок в одномерный 784-элементый (28×28) массив:

```
layers.Flatten(input_shape=(28, 28))
```

В случае цветных изображений задача усложняется. Каждый пиксел в цветном снимке в RGB-кодировке имеет три значения — по одному для красного, зеленого и синего цветов. Если бы наши снимки в приведенном выше примере были цветными, мы бы добавили в `input_shape` модели третью размерность, в результате чего инструкция была бы следующей:

```
layers.Flatten(input_shape=(28, 28, 3))
```

Хотя представление снимков в качестве массивов пиксельных значений хорошо работает для простых снимков, таких как изображения в градациях серого из набора данных MNIST, оно начинает разрушаться при вводе снимков с большим числом краев и контуров. Когда в сеть подаются все пикселы снимка одновременно, ей трудно сфокусироваться на более мелких участках смежных пикселов, которые содержат важную информацию.

---

[49] API Sequential в Keras служит для послойного строительства моделей. — *Прим. перев.*

Снимки в качестве плиточных структур. Нам нужен подход к представлению более сложных, реальных снимков, который позволит модели извлекать содержательные детали и понимать закономерности. Если мы будем подавать в сеть только малые части снимка за один раз, то с большей вероятностью сможем идентифицировать, например, пространственные градиенты и края, присутствующие в соседних пикселах.

Распространенной модельной архитектурой для достижения этой цели является сверточная нейронная сеть (convolutional neural network, CNN).

Сверточная нейронная сеть

Взгляните на рис. 2.22. В этом примере у нас есть сетка 4×4, в которой каждый квадрат представляет пиксельные значения на нашем снимке. Будем использовать процедуру сведения на основе максимума (max pooling)[50], чтобы взять наибольшее значение из каждого фрагмента и сгенерировать меньшую результирующую матрицу. Теперь, после разбиения снимка на сетку из плиток, наша модель способна извлекать ключевые идеи из каждого участка изображения на разных уровнях гранулярности.

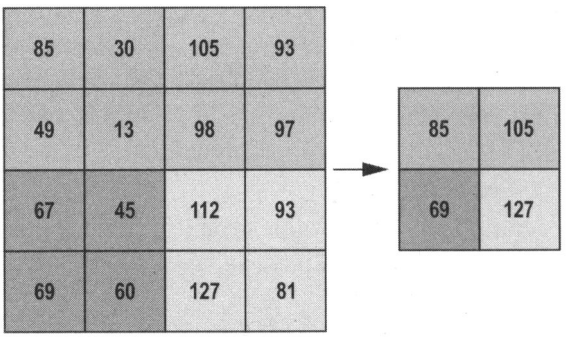


Рис. 2.22. Сведение на основе максимума на одном срезе размером 4×4 пиксельных данных

На рис. 2.22 используется ядро размером (2, 2). Каждый фрагмент ядра относится к соответствующему фрагменту снимка. Число пропускаемых ячеек, на которые фильтр перемещается перед созданием своего следующего блока, также именуемое сдвиговым шагом (stride), равно 2. Поскольку шаг равен размеру ядра, создаваемые блоки не перекрываются.

Хотя этот метод разложения на плитки сохраняет больше деталей, чем представление снимков в качестве массивов пиксельных значений, довольно много информации теряется после каждого шага сведения. На приведенном рисунке следующий шаг сведения произведет скалярное значение 127, сведя нашу матрицу размером 4×4 к одному-единственному значению всего за два шага. Остается только гадать,

[50] Сведение на основе максимума (max pooling), или редукция на основе функции max, означает взятие максимального значения участка окрестных признаков и тем самым уменьшение числа активаций с целью обеспечения инвариантности сдвига на снимке. Помимо max, могут использоваться и другие функции. — *Прим. перев.*

как это может исказить модель на реальном снимке, побудив ее фокусироваться на участках с доминирующими значениями пикселов, теряя при этом важные детали, которые могут окружать эти участки.

Каким образом можно оттолкнуться от идеи разбиения снимков на меньшие блоки, в то же время сохранив важные детали снимков? Это делается путем наложения блоков друг на друга. Если бы в примере рис. 2.22 вместо сдвигового шага 2 использовался сдвиговый шаг 1, то результатом была бы матрица 3×3 (рис. 2.23).

Затем мы могли бы преобразовать ее в сетку 2×2 (рис. 2.24).

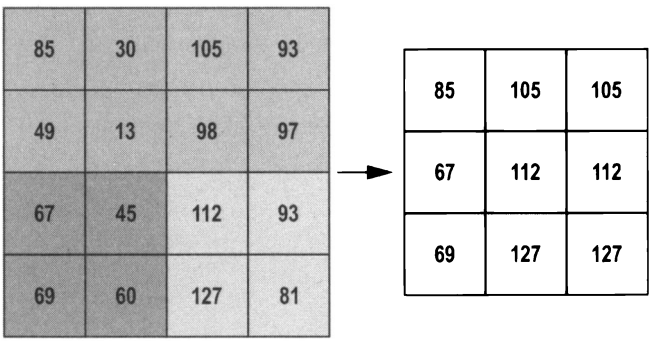


Рис. 2.23. Использование накладывающихся окон для сведения на основе максимума, выполняемого на сетке размером 4×4 пиксела

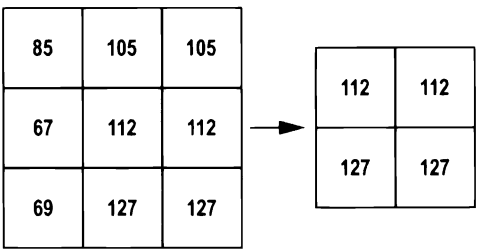


Рис. 2.24. Преобразование сетки 3×3 в сетку 2×2 со скользящими окнами и сведением на основе максимума

Мы приходим к окончательному скалярному значению 127. И хотя оно то же самое, видно, что промежуточные шаги сохранили больше деталей из первоначальной матрицы.

Фреймворк Keras предоставляет сверточные слои для строительства моделей, которые разбивают снимки на меньшие оконные блоки. Допустим, мы создаем модель для классифицирования цветных снимков размером 28×28 с собаками и кошками. Раз уж эти снимки цветные, каждый из них будет представлен как 28×28×3-размерный массив, поскольку каждый пиксел имеет три цветовых канала. Вот как мы бы определили входные данные для этой модели, используя сверточный слой и API Sequential:

```
Conv2D(filters=16, kernel_size=3, activation='relu', input_shape=(28,28,3))
```

В этом примере мы разбиваем входные снимки на блоки размером 3×3, перед тем как пропустить их через слой сведения на основе максимума. Архитектура модели, разбивающая снимки на блоки скользящих окон, позволяет нашей модели распознавать более мелкие детали снимка, такие как края и контуры.

Сочетание разных представлений снимков. В добавление к этому, как и в случае с мешком слов и векторным вложением текста, бывает полезно представлять одни и те же снимковые данные самыми разными способами. Это можно сделать с помощью функционального API Keras.

Вот как мы бы совместили наши пиксельные значения с представлением в виде скользящего окна с помощью конкатенатного слоя Keras:

```
# Определить входной слой для вводимого снимка (одинаковой формы
# как для пиксельного, так и для плиточного представления)
image_input = Input(shape=(28,28,3))

# Определить пиксельное представление
pixel_layer = Flatten()(image_input)

# Определить плиточное представление
tiled_layer = Conv2D(filters=16, kernel_size=3,
                     activation='relu')(image_input)
tiled_layer = MaxPooling2D()(tiled_layer)
tiled_layer = tf.keras.layers.Flatten()(tiled_layer)

# Конкатенировать в единый слой
merged_image_layers = keras.layers.concatenate([pixel_layer, tiled_layer])
```

В целях определения модели, которая принимает это представление в качестве мультимодального признака, мы можем подать конкатенированный слой в выходной слой:

```
merged_dense = Dense(16, activation='relu')(merged_image_layers)
merged_output = Dense(1)(merged_dense)

model = Model(inputs=image_input, outputs=merged_output)
```

Выбор варианта представления изображений для использования в своей модели, включая мультимодальное представление, во многом зависит от типа снимковых данных, с которыми мы работаем. Чем детальнее наши снимки, тем больше вероятность того, что мы захотим представить их в виде плиток либо скользящих окон, состоящих из плиток. Для набора данных MNIST будет достаточно представления снимков только в качестве пиксельных значений. С другой стороны, имея дело со сложными медицинскими снимками, мы повысим точность, сочетая несколько представлений. Зачем сочетать несколько представлений снимков? Представляя снимки в виде пиксельных значений, мы обеспечиваем модель возможностью выявлять на снимке более высокоуровневые точки фокусировки, например доминирующие высококонтрастные объекты. Плиточные представления помогают моделям выявлять более мелкие, менее контрастные края и контуры.

Использование снимков с метаданными. Ранее мы обсуждали разные типы метаданных, которые можно ассоциировать с текстом, а также способы извлечения и представления этих метаданных в виде табличных признаков для модели. Мы можем применить эту концепцию и к снимкам. Для этого давайте вернемся к приведенному на рис. 2.19 примеру модели, в которой используются кадры перекрестка для предсказания на нем нарушения правила дорожного движения. Наша модель может самостоятельно извлекать целый ряд закономерностей из снимков дорожного движения, но могут иметься и другие данные, способные повысить точность нашей модели. Например, то или иное поведение (правый поворот на красный) не разрешено в час пик, но является нормальным в другое время суток. Или же, например, водители чаще нарушают правила дорожного движения в плохую погоду. Если мы собираем снимковые данные с многочисленных перекрестков, то осведомленность нашей модели о местоположении снимка также может быть полезной.

Сейчас мы выявили три дополнительных табличных признака, которые могли бы улучшить нашу снимковую модель:

- время суток;
- погода;
- местоположение.

Давайте подумаем о возможных представлениях для каждого из этих признаков. Мы могли бы представить время как целое число, обозначающее час дня. Оно поможет нам выявлять закономерности, связанные с интенсивностью движения, например, в часы пик. В контексте указанной модели может оказаться более полезным знание о том, был ли получен снимок в темное время суток. В этом случае мы могли бы представить время как булев признак.

Погоду тоже может представлять разнообразными способами, как числовыми, так и категориальными значениями. Мы могли бы включить в качестве признака температуру, но в этом случае видимость может быть полезнее. Еще один вариант представления погоды заключается в категориальной переменной, обозначающей наличие дождя или снега.

Если мы собираем данные из многих мест, то, скорее всего, также захотим закодировать в качестве признака и местоположение. Это было бы наиболее разумно сделать в качестве категориального признака и, возможно, даже в качестве нескольких признаков (город, страна, штат и т. д.) в зависимости от числа мест, из которых мы собираем кадры.

Предположим, что в этом примере мы хотели бы использовать следующие табличные признаки:

- время (time) как час дня — целочисленный признак;
- видимость (visibility) — вещественный признак;
- ненастная погода (inclement weather) — категориальный признак (дождь, снег, нет);
- ID местоположения (location) — категориальный признак с пятью возможными местоположениями.

Вот как может выглядеть подмножество этого набора данных для последнего примера:

```
data = {
    'time': [9,10,2],
    'visibility': [0.2, 0.5, 0.1],
    'inclement_weather': [[0,0,1], [0,0,1], [1,0,0]],
    'location': [[0,1,0,0,0], [0,0,0,1,0], [1,0,0,0,0]]
}
```

Тепрь мы можем совместить эти табличные признаки в единый массив по каждому примеру, вследствие чего массив на входе в нашу модель будет содержать 10 элементов. Входной массив для первого примера будет выглядеть так:

```
[9, 0.2, 0, 0, 1, 0, 1, 0, 0, 0]
```

Мы могли бы подать эти данные на вход плотного (`Dense`) полносвязаного слоя, а на выходе из модели было бы одно-единственное значение между 0 и 1, указывающее на наличие либо отсутствие в экземпляре нарушения правила дорожного движения. В целях совмещения всего изложенного с нашими снимковыми данными мы будем использовать подход, аналогичный тому, который мы обсуждали для текстовых моделей. Сначала мы определим сверточный слой для манипулирования снимковыми данными, затем плотный слой для манипулирования табличными данными и, наконец, конкатенируем оба слоя в один выходной слой.

Этот подход представлен на рис. 2.25.

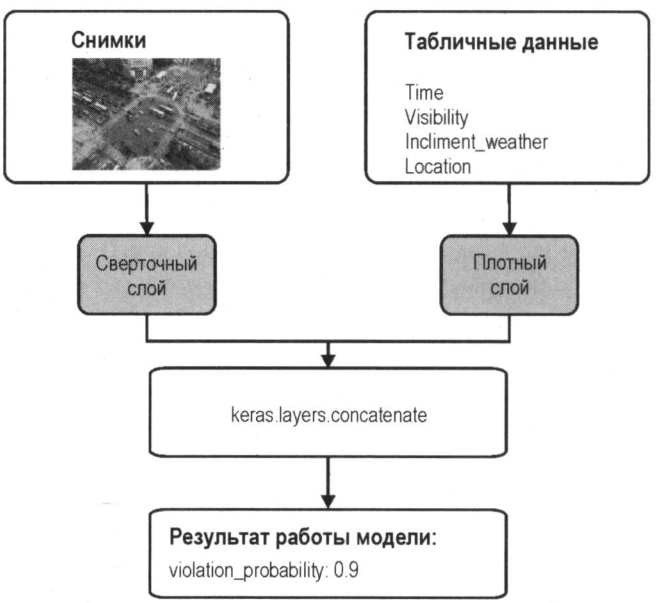


Рис. 2.25. Конкатенирование слоев с целью манипулирования признаками со снимковыми и табличными метаданными

Мультимодальные представления признаков и интерпретируемость модели

Модели на основе глубокого обучения по самой своей сути труднообъяснимы. Если мы построим модель, которая имеет точность 99%, мы все равно не будем знать конкретно, как именно наша модель делает предсказания и, следовательно, является ли путь, которым она приходит к этим предсказаниям, правильным. Например, предположим, что мы тренируем модель на взятых в лаборатории снимках чашек Петри, которая достигает высокого балла точности. Эти снимки также содержат комментарии ученого, который их взял. Мы знаем о содержимом чашек Петри, но не знаем о том, что модель неправильно использует комментарии для генерирования предсказаний.

Существует несколько технических приемов, которые служат для объяснения снимковых моделей, например, выделяя пикселы, сигнализировавшие о сделанном моделью предсказании. Однако, когда в одной модели мы совмещаем многочисленные представления данных, эти признаки начинают зависеть друг от друга. В результате бывает трудно объяснить принцип, по которому модель делает предсказания. Проблематика обеспечения объяснимости рассматривается в *главе 7*.

Резюме

В этой главе мы узнали о разных подходах к представлению модельных данных. Мы начали с рассмотрения приемов манипулирования числовыми входными данными и нормализации этих входных данных, которые помогают ускорять время тренировки модели и повышать ее точность. Затем мы рассмотрели приемы инженерии признаков на категориальных входных данных, в частности, с помощью кодирования с одним активным состоянием и использования массивов категориальных значений.

В остальной части главы мы обсудили четыре паттерна для представления данных. Первым был "Хешированный признак" (Hashed Feature), который предусматривает кодирование категориальных входных данных в виде уникальных строковых значений. Мы рассмотрели несколько разных подходов к хешированию с использованием набора данных об аэропортах из BigQuery. Вторым рассмотренным в этой главе был паттерн "Векторные вложения" (Embeddings) — технический прием представления входных данных с высокой кардинальностью, таких как данные с многочисленными возможными категориями или текст. Векторные вложения представляют данные в многомерном пространстве, в котором размерность зависит от данных и задачи предсказания. Затем мы рассмотрели паттерн "Синтетический признак" (Feature Cross) — подход, который соединяет два признака для извлечения связей, трудноуловимых путем кодирования признаков по отдельности. Наконец, мы рассмотрели представления с использованием паттерна "Мультимодальный вход" (Multimodal Input), обратившись к приемам объединения входных данных разных типов в одной и той же модели и приемам представления одного признака самыми разными способами.

Эта глава была посвящена подготовке входных данных для моделей. В следующей главе мы сосредоточимся на разультатах работы модели, углубившись в разные подходы к представлению задачи предсказания.

ГЛАВА 3

Паттерны для представления задачи

В *главе 2* рассматривались паттерны, каталогизирующие мириады приемов, которыми могут представляться входные переменные, подаваемые в модели машинного обучения. В этой главе рассматриваются разные типы задач машинного обучения и анализируются варианты архитектур моделей в зависимости от конкретной задачи.

Типы данных на входе и на выходе — два ключевых фактора, влияющих на архитектуру модели. Например, выходные данные в задачах машинного обучения с учителем могут варьироваться в зависимости от характера решаемой задачи: классификационной или регрессионной. Для тех или иных типов входных данных существуют специальные нейросетевые слои: сверточные слои для снимков, речи, текста и других данных с пространственно-временной корреляцией, рекуррентные слои для последовательных данных и т. д. По проблематике этих типов слоев возникла колоссальная по объему литература, которая сконцентрирована вокруг специальных технических приемов, таких как сведение на основе максимума, внимание и т. д. Кроме того, были разработаны специальные классы технических решений для распространенных задач, таких как рекомендации (например, матричная факторизация) или прогнозирование на основе временного ряда (например, ARIMA). Наконец, группа более простых моделей вместе с распространенными идиомами может использоваться для решения более сложных задач, например генерация текста часто предусматривает классификационную модель, данные на выходе из которой постобрабатываются с помощью алгоритма лучевого поиска.

В целях ограничения объема обсуждаемого материала, при этом держась подальше от областей активных научных исследований, мы будем игнорировать паттерны и идиомы, связанные со специализированными областями машинного обучения. Вместо этого мы сосредоточимся на регрессии и классификации и рассмотрим паттерны представления задач только в этих двух типах моделей ML.

Паттерн "Переформулировка" (Reframing) берет решение, которое интуитивно воспринимается как регрессионная задача, и представляет его как классификационную задачу (и наоборот). Паттерт "Мультиметка" (Multilabel) хорошо работает в случае, когда тренировочные примеры могут принадлежать более чем одному классу. Паттерн "Каскад" (Cascade) предназначен для ситуаций, когда задача машинного обучения может быть выгодно подразделена на серию (или каскад) задач. Паттерн "Ансамбли" (Ensembles) решает задачу путем тренировки нескольких моделей и агрегирования их ответов. Паттерн "Нейтральный класс" (Neutral Class) помогает

в том, как обращаться с ситуациями, когда эксперты расходятся во мнениях. Паттерн "Перебалансировка" (Rebalancing) рекомендует подходы к манипулированию сильно искаженными или несбалансированными данными.

ПАТТЕРН 5. Переформулировка

Паттерн "Переформулировка" (Reframing) связан с изменением представления данных на выходе из задачи машинного обучения. Например, мы могли бы взять нечно интуитивно воспринимаемое как регрессионная задача и вместо этого представить как классификационную задачу (и наоборот).

Постановка задачи

Первый шаг создания любого технического решения по машинному обучению состоит в формулировке задачи. О каком классе задач идет речь? Принадлежит ли она классу задач обучения с учителем? Или же обучения без учителя? Какие признаки в ней используются? Если это задача обучения с учителем, то какие в ней используются метки? Какая величина ошибки будет допустимой? Разумеется, ответы на эти вопросы должны рассматриваться в контексте тренировочных данных, решаемой задачи и метрик успеха.

Например, предположим, что мы хотим построить модель машинного обучения, чтобы предсказывать количество осадков в некотором месте. Возникает вопрос о том, как какому типу относить эту задачу: регрессии или же классификации? Что ж, поскольку мы пытаемся предсказать количество осадков (например, 0,3 см), имеет смысл рассматривать ее как задачу прогнозирования на основе временно́го ряда: какое количество осадков с учетом текущей и исторической климатической и погодной закономерностей мы должны ожидать в данной области ближайшие 15 минут? С другой стороны, поскольку метка (количество осадков) является вещественным числом, мы могли бы построить регрессионную модель. Когда мы начнем разрабатывать и тренировать нашу модель, то обнаружим (возможно, догадываясь), что предсказывать погоду труднее, чем кажется поначалу. Наши предсказанные количества осадков все время сбиваются, потому что для одного и того же набора признаков иногда идет дождь, который можно оценить как 0,3 см осадков, а иногда — как 0,5 см. Что следует сделать для улучшения предсказаний? Нужно ли добавить в сеть больше слоев? Или же сгенерировать больше признаков? Вероятно, поможет больше данных? А может, нам нужна другая функция потери?

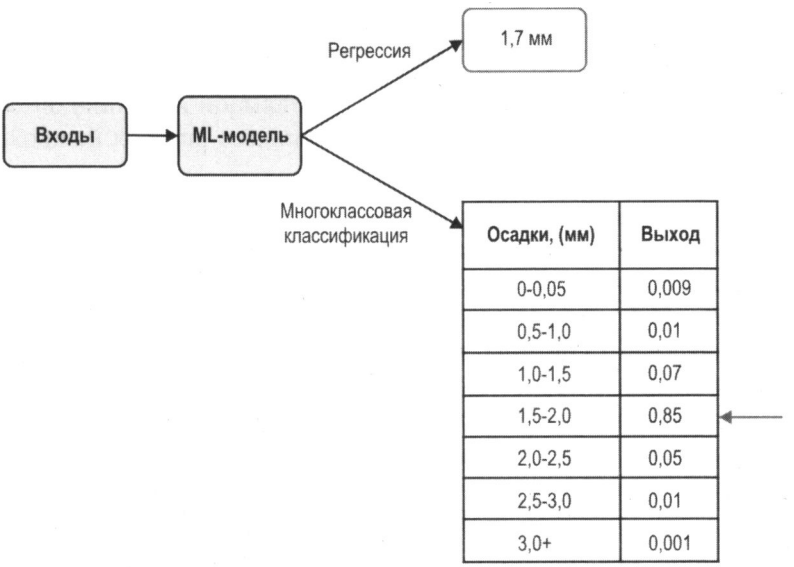
Любая из этих корректировок могла бы улучшить нашу модель. Но постойте. Разве регрессия является единственным способом формулировки этой задачи? По всей видимости, мы сможем переформулировать нашу целевую установку машинного обучения, что позволит повысить результативность нашей задачи.

Решение

Основная трудность здесь заключается в том, что осадки вероятностны. Для одного и того же набора признаков иногда идет дождь с осадками 0,3 см, а иногда —

с 0,5 см. Тем не менее даже если бы регрессионная модель смогла усвоить два возможных количества осадков, она ограничена предсказанием только одного числа.

Вместо того чтобы пытаться предсказывать количество осадков как регрессионную задачу, мы можем переформулировать нашу целевую установку как классификационную задачу. Этого можно достичь за счет привлечения разных подходов. Один из них заключается в моделировании дискретного распределения вероятностей (рис. 3.1). Вместо того чтобы предсказывать количество осадков в качестве вещественного результата, мы моделируем результат как многоклассовую классификацию, продуцирующую вероятность того, что количество осадков в ближайшие 15 минут будет находиться в определенном диапазоне количества осадков.



Осадки, (мм)	Выход
0-0,05	0,009
0,5-1,0	0,01
1,0-1,5	0,07
1,5-2,0	0,85
2,0-2,5	0,05
2,5-3,0	0,01
3,0+	0,001

Рис. 3.1. Вместо того чтобы предсказывать осадки как результат регрессии, мы можем моделировать дискретное распределение вероятностей, используя многоклассовую классификацию

И регрессионный подход, и новый переформулированный подход как классификация дают предсказание осадков на следующие 15 минут. Однако классификационный подход позволяет модели не выбирать среднее значение распределения осадков, а улавливать распределение вероятностей осадков в различных количественных величинах. Моделирование распределения, таким образом, выгодно тем, что осадки не демонстрируют типичную колоколообразную кривую нормального распределения, а вместо этого подчиняются распределению Твиди (Tweedie)[1], которое допускает преобладание точек на нуле. И действительно, это как раз тот подход, который был принят в научно-исследовательской работе, опубликованной в базе данных Google Research[2]. В ней предсказывалось количество осадков в данном месте с использованием категориального распределения на основе 512 способов. Дру-

[1] См. **https://oreil.ly/C8JfK**.

[2] См. **https://oreil.ly/PGAEw**.

гие причины, по которым моделирование распределения может быть выгодным, возникают, когда распределение является бимодальным или даже нормальным, но с большой дисперсией. Недавняя работа, которая превосходит все эталонные критерии в предсказании структуры сворачивания белка[3], тоже предсказывает расстояние между аминокислотами 64 способами как классификационную задачу, в которой расстояния сгруппированы в 64 корзины.

Еще одна причина переформулировать задачу возникает, когда целевая установка оказывается лучше в модели другого типа. Например, предположим, что мы пытаемся построить систему рекомендаций видео. Естественным подходом к формулировке этой задачи является классификационная задача предсказания вероятности того, что пользователь будет смотреть то или иное видео. Однако эта формулировка может привести к рекомендательной системе, которая во главу угла будет ставить клик-наживку[4]. И, возможно, было бы лучше переформулировать ее в регрессионную задачу предсказания части видео, которая будет просмотрена.

Почему это работает

Изменение контекста и переформулировка задачи помогают во время создания технического решения по машинному обучению. Вместо того чтобы запоминать одно действительное число, мы ослабляем цель предсказания, превращая ее в дискретное распределение вероятностей. Мы теряем немного прецизионности из-за группировки значений в корзины, но получаем выразительность полной функции плотности распределения вероятностей (probability density function, PDF). Дискретизированные предсказания, предоставляемые классификационной моделью, лучше подходят для изучения сложной цели, чем более жесткая регрессионная модель.

Дополнительным преимуществом такой классификационной постановки задачи является то, что мы получаем апостериорное распределение вероятностей предсказываемых значений, что обеспечивает более подробную информацию. Например, предположим, что усвоенное распределение является бимодальным. Моделируя классификацию как дискретное распределение вероятностей, модель способна улавливать бимодальную структуру предсказаний (рис. 3.2); а если бы предсказывалось только одно-единственное числовое значение, то эта информация была бы утеряна. В зависимости от варианта использования такая постановка могла бы облегчить усвоение задачи и сделать ее предпочтительней.

Улавливание неопределенности

Давайте еще раз обратимся к набору данных о рождаемости и к задаче о предсказании веса младенца. Поскольку вес младенца является положительным вещественным значением, эта задача интуитивно воспринимается как регрессионная. Однако обратите внимание, что в том или ином наборе входных данных метка `weight_`

---

[3] См. **https://oreil.ly/-Hi3k**.

[4] То есть назойливые рекламные сообщения и посты на веб-странице, которые заставляют по ним щелкать. — *Прим. перев.*

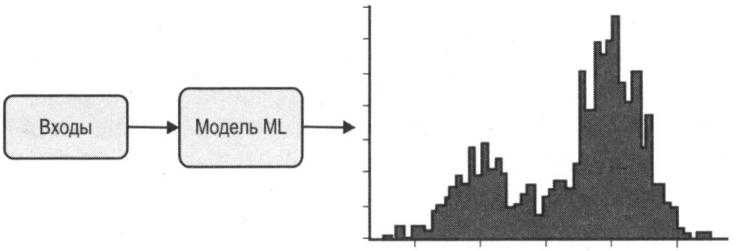


Рис. 3.2. Переформулировка задачи в классификационную с моделированием распределения вероятностей позволяет предсказаниям улавливать бимодальный результат. Предсказание не ограничивается одним значением, как в регрессии

pounds (вес в фунтах) может принимать множество разных значений. Мы видим, что распределение веса младенцев для конкретного набора входных значений (младенцы-мальчики, рожденные 25-летними матерями на 38-й неделе беременности) приблизительно подчиняется нормальному распределению с центром около 7,5 фунта (3,4 кг). Исходный код для создания графика рис. 3.3 можно найти в репозитории[5] этой книги.

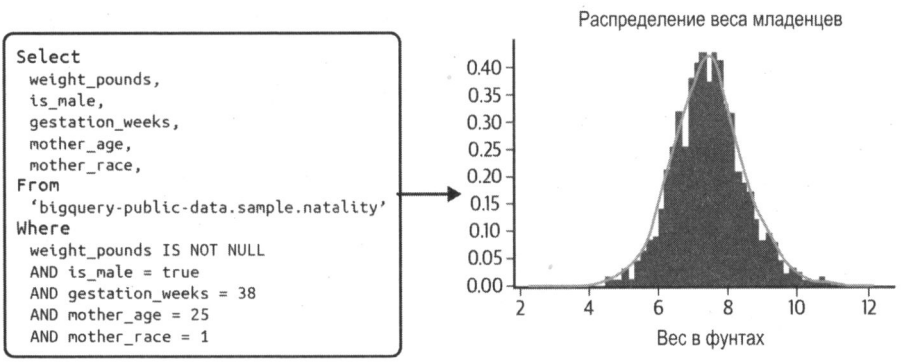


Рис. 3.3. С учетом того или иного набора входных данных (например, младенцы-мальчики, рожденные 25-летними матерями на 38-й неделе беременности) переменная weight_pounds принимает диапазон значений, приблизительно подчиняющийся нормальному распределению с центром в 7,5 фунта

Однако тут следует обратить внимание на ширину распределения — даже если распределение достигает пика 7,5 фунта, существует нетривиальная вероятность (на самом деле 33%) того, что данный младенец будет весить меньше 6,5 фунта (2,95 кг) или больше 8,5 фунта (около 3,9 кг)! Ширина этого распределения указывает на неустранимую ошибку, присущую задаче предсказания веса младенца. Действительно, наилучший корень из среднеквадратической ошибки (root mean square error, RMSE), который мы можем получить в этой задаче, если сформулировать ее как регрессионную задачу, равен стандартному отклонению распределения (рис. 3.3).

[5] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/03_problem_representation/reframing.ipynb**.

Если бы мы сформулировали эту задачу как регрессионную, нам пришлось бы установить результат предсказания с погрешностью как 7,5±1,0 (каково бы ни было стандартное отклонение). Тем не менее ширина распределения будет варьироваться для разных комбинаций входных данных, и поэтому усвоение ширины — это еще одна задача машинного обучения. Например, на 36-й неделе беременности для матерей того же возраста стандартное отклонение составляет 1,16 фунта (0,53 кг). *Квантильная регрессия*, которую мы рассмотрим позже при обсуждении данного паттерна, пытается сделать именно это, но непараметрически.

Если бы распределение было мультимодальным (с несколькими пиками), то аргументы в пользу переформулировки задачи как классификационной были бы еще убедительнее. Однако полезно понимать, что из-за закона больших чисел, пока мы улавливаем все релевантные входные данные, многие распределения, с которыми мы столкнемся в крупных наборах данных, будут иметь колоколообразную форму, хотя другие распределения тоже возможны. Чем шире колоколообразная кривая и чем больше эта ширина варьируется при различных значениях входных данных, тем важнее улавливать неопределенность и тем более весомы аргументы в пользу переформулировки регрессионной задачи в классификационную.

Переформулировав задачу, мы тренируем модель как многоклассовую классификацию, которая усваивает дискретное распределение вероятностей для заданных тренировочных примеров. Эти дискретизированные предсказания более гибкие с точки зрения улавливания неопределенности и способны лучше аппроксимировать сложную цель, чем регрессионная модель. Во время предсказательного вывода модель предсказывает коллекцию вероятностей, соответствующих этим потенциальным результатам. То есть мы получаем дискретную функцию плотности распределения вероятностей (probability density function, PDF), дающую относительную вероятность любого отдельно взятого веса. Конечно же, здесь следует соблюдать осторожность — классификационные модели могут быть чрезвычайно некалиброванными (например, модель будет чересчур уверенной и ошибочной).

Изменение целевой установки

В некоторых сценариях переформулировка классификационной задачи в качестве регрессионной могла бы оказаться полезной. Например, предположим, что у нас есть крупная база данных фильмов с клиентскими оценками по шкале от 1 до 5 для всех фильмов, которые зритель смотрел и оценивал. Наша задача — построить модель, которая будет использоваться для рекомендаций пользователям.

Глядя на задачу как на классификационную, мы могли бы подумать о создании модели, которая на входе принимает ID пользователя (user_id) наряду с предыдущими видеопросмотрами и оценками этого пользователя и предсказывает фильм из нашей базы данных, который следует рекомендовать к следующему просмотру. Однако существует возможность сформулировать эту задачу по-новому как регрессионную. Вместо категориального результата работы модели, соответствующего фильму в нашей базе данных, наша модель могла бы выполнять многозадачное обучение, при котором она запоминает ряд ключевых характеристик (таких как доход, потребительский сегмент и т. д.) пользователей, которые, вероятно, будут смотреть данный фильм.

Модель, переформулированная как регрессионная задача, теперь предсказывает представление пользовательского пространства для данного фильма. Для предоставления рекомендаций мы выбираем набор фильмов, наиболее подходящих под известные характеристики пользователя. Таким образом, вместо модели, обеспечивающей вероятность того, что пользователю понравится фильм, как в случае классификации, мы получим кластер фильмов, просмотренных зртелями, похожими на этого пользователя.

Переосмысливая классификационную задачу рекомендации фильмов как регрессию пользовательских характеристик, мы получаем возможность легко адаптировать нашу рекомендательную модель для рекомендаций трендовых видео, классических или документальных фильмов без необходимости каждый раз тренировать отдельную классификационную модель.

Этот тип подхода также полезен, когда числовое представление имеет интуитивно понятную интерпретацию; например, вместо предсказаний городских районов можно использовать широтно-долготную пару. Предположим, мы хотим предсказывать город, в котором произойдет следующая вирусная вспышка, или район Нью-Йорка, где резко повысятся цены на недвижимость. Было бы проще предсказывать широту и долготу и выбирать ближайший к этому местоположению город или район, чем предсказывать сам город или район.

Компромиссы и альтернативы

Редко существует только один способ сформулировать задачу, и полезно знать о любых компромиссах или альтернативах для данной реализации. Например, группировка выходных значений регрессии в корзины является общепринятым подходом к переформулировке задачи как классификационной. Еще один подход — это многозадачное обучение, которое объединяет обе задачи (классификационную и регрессионную) в модель с использованием нескольких направлений (leads) предсказания. При любом подходе к переформулировке очень важна осведомленность о присущих вашим данным пределах или риске внесения искаженности в метку[6] технического решения.

Сгруппированные результаты

Типичный подход к переформулировке регрессионной задачи как классификационной состоит в том, чтобы сгруппировать выходные значения в корзины. Например, если наша модель будет применяться для определения ситуаций, когда новорожденному потребуется неотложная помощь, то приведенных в табл. 3.1 категорий может быть достаточно.

Наша регрессионная модель теперь становится многоклассовой классификацией. В интуитивном плане легче предсказывать один из четырех возможных категориальных случаев, чем предсказывать одно значение из континуума вещественных

[6] Искаженность в метке возникает, когда набор помеченных данных не полностью представляет весь универсум потенциальных меток. — *Прим. перев.*

Таблица 3.1. Сгруппированные результаты для веса младенца

Категория	Описание
Большой вес при рождении	Более 8,8 фунта (более 4 кг)
Средний вес при рождении	От 5,5 до 8,8 фунта (от 2,5 до 4 кг)
Малый вес при рождении	От 3,31 до 5,5 фунта (от 1,5 до 2,5 кг)
Очень малый вес при рождении	Менее 3,31 фунта (менее 1,5 кг)

чисел. Точно так же было бы легче предсказывать цель, состоящую из двоичного нуля и единицы, например для цели "вес ниже нормы?" (is_underweight) вместо четырех отдельных категорий "большой вес" (high_weight), "средний вес" (avg_weight), "малый вес" (low_weight) и "очень малый вес" (very_low_weight). Опираясь на категориальные результаты, наша модель меньше мотивируется на получение значения, произвольно близкого к фактическому выходному значению, поскольку мы существенно поменяли выходную метку на диапазон значений вместо одного-единственного вещественного числа.

В прилагаемом к этому разделу блокноте[7] мы тренируем как регрессионную, так и многоклассовую классификационную модели. Регрессионная модель достигает RMSE = 1,3 на валидационном наборе данных, в то время как классификационная модель имеет точность 67%. Сравнивать эти две модели трудно по той причине, что одной метрикой оценивания является RMSE, а другой — точность. В конце концов, выбор варианта дизайна регулируется вариантом использования. Если медицинские решения основываются на значениях, сгруппированных в корзины, то наша модель должна быть классификационной и использовать эти корзины. Однако если требуется более четкое предсказание веса младенца, то имеет смысл использовать регрессионную модель.

Другие подходы к улавливанию неопределенности

Есть и другие подходы к улавливанию неопределенности в регрессии. Простой подход заключается в выполнении квантильной регрессии. Например, вместо того чтобы предсказывать только среднее значение, мы можем оценивать условный 10, 20, 30, ..., 90-й процентиль того, что должно быть предсказано. Квантильная регрессия является продолжением линейной регрессии. С другой стороны, переформулировка задачи может работать с более сложными моделями машинного обучения.

Еще один, более сложный подход заключается в использовании фреймворка, наподобие TensorFlow Probability[8], для вычисления регрессии. Однако мы должны моделировать распределение результатов на выходе в явной форме. Например, если

---

[7] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/03_problem_representation/reframing.ipynb**.

[8] См. **https://oreil.ly/AEtLG**.

ожидается, что результаты будут нормально распределены вокруг среднего значения, завися от данных на входе, то выходной слой модели будет таким:

```
tfp.layers.DistributionLambda(lambda t: tfd.Normal(loc=t, scale=1))
```

С другой стороны, если мы знаем, что дисперсия увеличивается вместе со средним значением, мы могли бы смоделировать результат с помощью лямбда-функции. Переформулировка задачи, в свою очередь, не требует от нас моделирования апостериорного распределения вероятностей.

Во время тренировки любой модели машинного обучения принципиально большое значение имеют данные. Более сложные связи, как правило, требуют большего числа тренировочных данных, чтобы найти эти неуловимые закономерности. С учетом этого факта важно учитывать то, как потребности в данных соотносятся с регрессионной и/или классификационной моделями. Общее эмпирическое правило классификационных задач состоит в том, что для каждой категории меток мы должны иметь в 10 раз больше модельных признаков. Эмпирическое правило для регрессионной модели состоит в 50-кратном превышении числа модельных признаков. Конечно же, эти цифры — всего лишь грубая и нечеткая эвристика. Однако интуиция подсказывает, что регрессионные задачи обычно требуют большего числа тренировочных примеров. Вдобавок по мере усложнения задачи эта потребность в массивных данных только возрастает. Следовательно, могут появиться пределы данных, которые следует учитывать при рассмотрении типа используемой модели или, в случае классификации, числа категорий меток.

Прецизионность предсказаний

Задумываясь о переформулировке регрессионной модели как многоклассовой классификации, важно помнить о том, что прецизионность классификационной модели определяется шириной корзин для выходной метки. Если бы нам в нашем примере с весом младенца требовалась более прецизионная информация из дискретной функции плотности распределения вероятностей, то нам пришлось бы увеличить число корзин категориальной модели. На рис. 3.4 показано, как дискретные распределения вероятностей будут выглядеть в качестве классификации с 4 или 10 категориями.

Заостренность функции плотности распределения вероятностей (probability density function, PDF) указывает на прецизионность задачи как регрессионной. Более заостренная PDF указывает на меньшее стандартное отклонение выходного распределения, в то время как более широкая PDF — на большее стандартное отклонение и, следовательно, бóльшую дисперсию. Для получения очень заостренной функции плотности лучше придерживаться регрессионной модели (рис. 3.5).

Ограничение предсказательного диапазона

Еще одна причина переформулировать задачу заключается в необходимости ограничивать диапазон предсказания на выходе. Допустим, например, что реалистичные выходные значения для регрессионной задачи находятся в промежутке [3; 20]. Если мы натренируем регрессионную модель, в которой выходной слой представлен линейной активационной функцией, то всегда существует вероятность того,

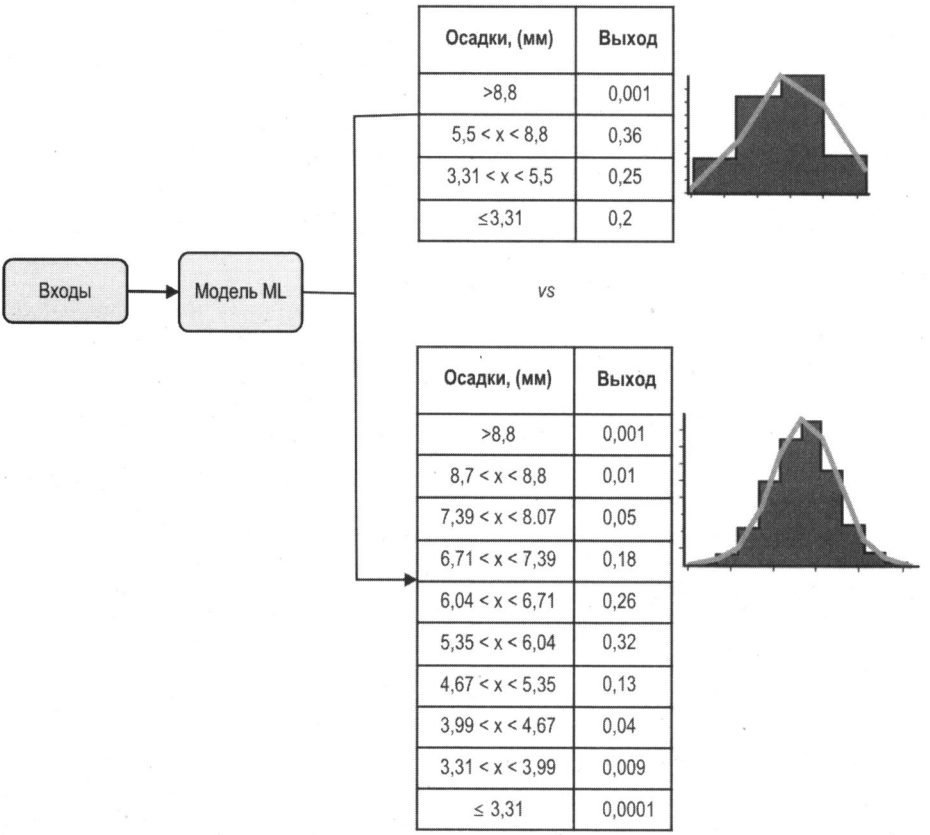


Рис. 3.4. Прецизионность мультиклассовой классификации контролируется шириной корзин для метки

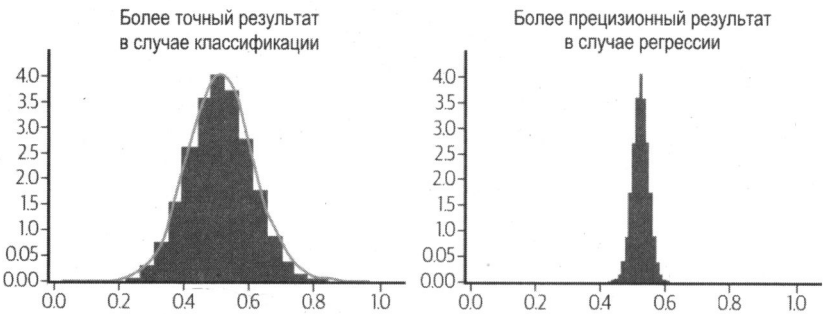


Рис. 3.5. Прецизионность регрессии определяется заостренностью функции плотности распределения вероятностей для фиксированного набора входных значений

что модельные предсказания будут выходить за пределы этого интервала. Один из подходов к ограничиванию диапазона значений на выходе состоит в переформулировке задачи.

Оно состоит в том, чтобы сделать активационную функцию предпоследнего слоя сигмоидной функцией (которая обычно ассоциируется с классификацией), вследствие чего она будет находиться в промежутке [0; 1], и в последнем слое выполнить нормализацию этих значений в желательный интервал:

```
MIN_Y = 3
MAX_Y = 20
input_size = 10
inputs = keras.layers.Input(shape=(input_size,))
h1 = keras.layers.Dense(20, 'relu')(inputs)
h2 = keras.layers.Dense(1, 'sigmoid')(h1) # интервал 0-1
output = keras.layers.Lambda(
            lambda y : (y*(MAX_Y-MIN_Y) + MIN_Y))(h2) # пронормализовать
model = keras.Model(inputs, output)
```

Мы можем убедиться (полный исходный код см. в блокноте[9] в репозитории на GitHub), что теперь эта модель эмитирует числа в промежутке [3; 20]. Обратите внимание на то, что, поскольку выходное значение является сигмоидным, модель никогда не достигнет минимума и максимума интервала, а будет к нему только приближаться. Натренировав приведенную выше модель на случайных данных, мы получили значения в промежутке [3,03; 19,99].

Искаженность в метке

Рекомендательные системы, наподобие матричной факторизации, можно переформулировать в контексте нейронных сетей как регрессионные и как классификационные. Одним из преимуществ такого изменения контекста является то, что нейронная сеть, переформулированная как регрессионная или классификационная модель, может включать в себя гораздо больше дополнительных признаков, помимо векторных вложений пользователя и предметов, усвоенных в матричной факторизации. Поэтому такая альтернатива бывает привлекательной.

Однако во время переформулировки задачи важно учитывать природу целевой метки. Например, предположим, что мы переформулировали рекомендательную модель в классификационную задачу, которая предсказывает вероятность того, что пользователь нажмет на тот или иной значок видео. Такая переформулировка задачи кажется вполне разумной, поскольку наша цель состоит в том, чтобы доставлять контент, который пользователь будет выбирать и смотреть. Но следует соблюдать осторожность. Это изменение метки на самом деле не соответствует нашей задаче предсказания. Выполняя оптимизацию, заточенную под щелчки мышью пользова-

[9] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/03_problem_representation/reframing.ipynb**.

телей, наша модель непреднамеренно будет продвигать клик-наживку и фактически не рекомендовать пользователю контент для использования.

Вместо этого более выгодной меткой было бы время просмотра видео, если бы мы переформулировали нашу рекомендательную модель как регрессионную. У нас вполне может возникнуть потребность модифицировать целевой критерий классифицирования и предсказывать вероятность того, что пользователь будет смотреть по крайней мере половину видеоклипа. Нередко существует несколько предпочтительных подходов, и при формулировании решения важно рассматривать задачу целостно.

При изменении метки и задачи тренировки модели машинного обучения следует соблюдать осторожность, т. к. это изменение может непреднамеренно внести искаженность в метку вашего технического решения. Еще раз обратитесь к примеру с рекомендацией видео для просмотра, который мы обсуждали в *разд. "Почему это работает"*.

Многозадачное обучение

Одной из альтернатив переформулировке задачи является многозадачное обучение. Делать и то и другое, вместо того чтобы пытаться выбирать между регрессией или классификацией! Вообще многозадачное обучение относится к любой модели машинного обучения, в которой оптимизируется более одной функции потери. Это достигается самыми разными способами, но две наиболее распространенные формы многозадачного обучения в нейронных сетях — это жесткое и мягкое совместное использование параметров.

Совместное использование параметров (parameter sharing) относится к параметрам нейронной сети, совместно используемым между разными выходными задачами, такими как регрессия и классификация. Жесткое совместное использование параметров происходит, когда скрытые слои модели совместно используются всеми выходными задачами. При мягком использовании параметров каждая метка имеет собственную нейронную сеть со своими параметрами и происходит поощрение параметров разных моделей, нацеленное на то, чтобы параметры были похожими после воздействия на них некоторой формы регуляризации. На рис. 3.6 показана типичная архитектура жесткого и мягкого использования параметров.

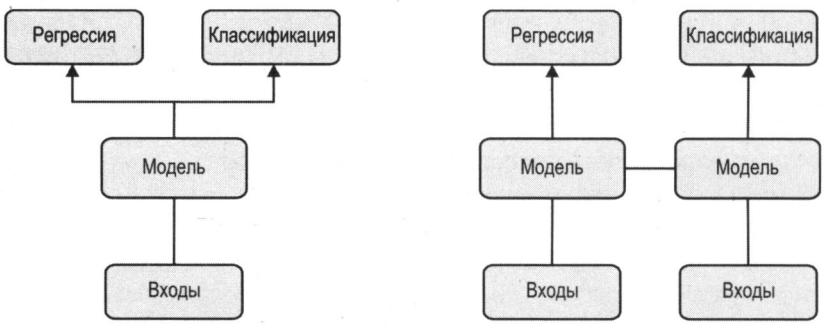


Рис. 3.6. Две распространенные реализации многозадачного обучения — это жесткое и мягкое совместное использование параметров

В указанном контексте в нашей модели мы могли бы иметь два направления: одно для предсказания регрессионного результата, а другое для предсказания классификационного результата. Например, в одной научно-исследовательской работе[10] продемонстрирована тренировка модели для компьютерного зрения с применением классификационного результата softmax-вероятностей вместе с регрессионным результатом для предсказания ограничительных рамок. В ней объясняется, что такой подход обеспечивает более высокую результативность, чем смежная исследовательская работа, в которой сети тренируются отдельно для классификационной и локализационной задач. Идея состоит в том, что посредством совместного использования параметров задачи обучаются одновременно, а обновления градиента из двух функций потерь информируют оба результата и приводят к более обобщаемой модели.

ПАТТЕРН 6. Мультиметка

Паттерн "Мультиметка" (Multilabel) относится к задачам, в которых мы можем назначать конкретному тренировочному примеру *несколько меток*. В нейронных сетях такой подход требует изменения активационной функции, используемой в завершающем выходном слое модели, и принятия решения о том, как наше приложение будет выполнять разбор данных на выходе из модели. Обратите внимание, что он отличается от *многоклассовых* классификационных задач, в которых одному примеру назначается ровно одна метка из группы многих (более 1) возможных классов. Вы также можете встретить указанный паттерн еще под одним названием — *мультиметочная многоклассовая классификация*, поскольку он предусматривает выбор нескольких меток из группы, состоящей из более чем одного возможного класса. Обсуждая этот паттерн, мы в первую очередь сосредоточимся на нейронных сетях.

Постановка задачи

Нередко задачи модельного предсказания предусматривают применение одной классификации к данному тренировочному примеру. Это предсказание определяется из N возможных классов, где $N > 1$. В этом случае в качестве активационной функции для выходного слоя обычно используется функция мягкого максимума softmax. Пройдя обработку функцией softmax, данные на выходе из модели представляют собой N-элементный массив, в котором сумма всех значений равна 1. Каждое значение указывает на вероятность того, что конкретный тренировочный пример ассоциирован с классом в этом индексе.

Например, если наша модель классифицирует изображения как фото кошек, собак или кроликов, то для конкретного снимка результат на выходе из softmax может выглядеть следующим образом: `[.89, .02, .09]`. Он означает, что наша модель предсказывает с 89%-ной вероятностью то, что на снимке изображена кошка, с 2%-ной вероятностью — собака и 9%-ной вероятностью — кролик. Поскольку в этом сце-

[10] См. **https://oreil.ly/sIjsF**.

нарии каждый снимок может иметь только одну возможную метку, мы можем взять argmax (индекс наибольшей вероятности), чтобы определить предсказанный нашей моделью класс. Менее распространенный сценарий заключается в том, что каждому тренировочному примеру может быть назначено более одной метки, и как раз этот сценарий указанный паттерн и решает.

Паттерн "Мультиметка" (Multilabel) существует для моделей, тренируемых на всех вариантах данных. В случае классифицирования снимков в приведенном выше примере с кошками, собаками, кроликами мы могли бы использовать тренировочные снимки, каждый из которых изображал бы несколько животных и, следовательно, мог бы иметь несколько меток. В случае текстовых моделей мы можем представить несколько сценариев, в которых текст может помечаться несколькими тегами. Используя набор данных вопросов Stack Overflow в BigQuery в качестве примера, мы могли бы построить модель для предсказания тегов, ассоциированных с тем или иным вопросом. Например, вопрос: "How do I plot a pandas DataFrame?" (Как построить график кадра данных pandas?) может быть помечен метками "Python", "pandas" и "визуализация". Еще одним примером мультиметочного классифицирования текста является модель, которая выявляет едкие комментарии. В такой модели мы могли бы помечать комментарии несколькими метками едкости. И поэтому комментарий может иметь метку "ненавистный" и метку "непристойный".

Указанный паттерн также можно применять к табличным наборам данных. Представьте себе набор медицинских данных с различными физическими характеристиками по каждому пациенту, такими как рост, вес, возраст, артериальное давление и многое другое. Эти данные можно использовать для предсказания нескольких состояний здоровья. Например, у пациента может быть риск как сердечно-сосудистых заболеваний, так и сахарного диабета.

Решение

Техническое решение для создания моделей, которые могут назначать *более одной метки* для заданного тренировочного примера, заключается в использовании сигмоидной активационной функции в завершающем выходном слое. Вместо того чтобы генерировать массив, в котором все значения в сумме составляют 1 (как в softmax), каждое *отдельное* значение в сигмоидном массиве является вещественным в интервале между 0 и 1. Другими словами, при реализации паттерна "Мультиметка" наша метка должна быть кодирована с использованием нескольких активных состояний. Длина массива с использованием нескольких активных состояний соответствует числу классов в модели, и каждый результат в этом массиве меток будет значением сигмоиды.

Отталкиваясь от приведенного выше примера со снимками, предположим, что наш тренировочный набор данных включал снимки с более чем одним животным. Сигмоидный результат для снимка, содержащего кошку и собаку, но не кролика, может выглядеть следующим образом: `[.92, .85, .11]`. Этот результат означает, что модель на 92% уверена в том, что снимок содержит кошку, на 85% — что он содержит собаку, и на 11% — что он содержит кролика.

Версия этой модели для 28×28-пиксельных снимков с сигмоидным результатом может выглядеть следующим образом (использован API Sequential в Keras):

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(3, activation='sigmoid')
])
```

Главное различие в результатах приведенной выше сигмоидной модели и softmax-примера из *разд. "Постановка задачи"* состоит в том, что softmax-массив содержит три значения, которые гарантированно в сумме составляют 1, тогда как сигмоидный результат будет содержать три значения, каждое от 0 до 1.

Сигмоидная активация против активации с функцией мягкого максимума

Сигмоида — это нелинейная, непрерывная и дифференцируемая активационная функция, которая принимает результаты каждого нейрона в предыдущем слое модели ML и сжимает эти результаты до значений из промежутка между 0 и 1. На рис. 3.7 показан внешний вид сигмоидной функции.

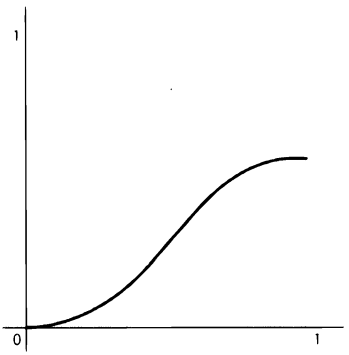


Рис. 3.7. Сигмоидная функция

В то время как сигмоида принимает одно-единственное значение на входе и предоставляет одно-единственное значение на выходе, softmax принимает массив значений на входе и преобразовывает его в массив вероятностей, сумма которых составляет 1. Значение на входе в функцию softmax может состоять из значений на выходе из N сигмоид.

В многоклассовой классификационной задаче, в которой каждый пример может иметь только одну метку, softmax используется в качестве последнего слоя, чтобы можено было получать распределение вероятностей. В паттерне "Мультиметка" допустимо, чтобы сумма значений выходного массива не была равна 1, поскольку мы оцениваем вероятность каждой отдельной метки.

Ниже приведены примеры выходных сигмоидного и softmax-массивов:

```
sigmoid = [.8, .9, .2, .5]
softmax = [.7, .1, .15, .05]
```

Компромиссы и альтернативы

При использовании паттерна "Мультиметка" и сигмоидного результата следует учитывать несколько особых случаев. Далее мы изучим вопрос структурирования моделей, которые имеют два возможных класса меток, вопрос интерпретирования сигмоидных результатов и приведем другие важные соображения для мультиметочных моделей.

Сигмоидный результат для моделей с двумя классами

Есть два типа моделей, в которых результат может принадлежать к двум возможным классам.

- Каждому тренировочному примеру может быть назначен *только один класс*. Этот тип также называется *двоичной классификацией* и представляет собой особый тип многоклассовой классификационной задачи.
- Некоторые тренировочные примеры могут принадлежать к *обоим классам*. Этот тип является разновидностью *мультиметочной классификационной* задачи.

На рис. 3.8 показано отличие между этими классификациями.

Рис. 3.8. Понимание отличия между многоклассовой, мультиметочной и двоичной классификационными задачами

Первый случай (двоичная классификация) уникален тем, что это единственный тип однометочной классификационной задачи, в которой в качестве активационной функции мы рассматривали бы использование сигмоиды. Для любой другой многоклассовой классификационной задачи (например, классифицирования текста на одну из пяти возможных категорий) мы использовали бы функцию softmax. Однако, когда есть только два класса, функция softmax становится избыточной. Возьмем, к примеру, модель, которая предсказывает, что та или иная транзакция является мошеннической. Если бы в этом примере мы использовали softmax-результат, то вот как могло бы выглядеть модельное предсказание мошенничества:

```
[.02, .98]
```

В этом примере первое значение относится к варианту "не мошенническая", а второе — к варианту "мошенническая". Результат избыточен, потому что мы могли бы также представить это с помощью одного скалярного значения и, таким образом, использовать сигмоидный результат. То же самое предсказание можно было бы представить просто как `.98`. Поскольку каждому входному значению может быть назначен только один класс, из этого результата можно заключить, что модель предсказала 98%-ную вероятность мошенничества и 2%-ную вероятность отсутствия мошенничества.

Поэтому для двоичных классификационных моделей вполне оптимально на выходе использовать форму с одноэлементным результатом с сигмоидной активационной функцией. Модели с одним выходным узлом к тому же более эффективны, поскольку у них будет меньше тренируемых параметров и, скорее всего, они будут тренироваться быстрее. Вот как выглядел бы выходной слой двоичной классификационной модели:

```
keras.layers.Dense(1, activation='sigmoid')
```

Во втором случае, когда тренировочный пример может принадлежать к обоим возможным классам и вписывается в паттерн "Мультиметка", мы также захотим использовать сигмоиду, на этот раз с двухэлементным результатом на выходе:

```
keras.layers.Dense(2, activation='sigmoid')
```

Какую функцию потери следует использовать?

Теперь, когда мы знаем, когда использовать сигмоиду в качестве активационной функции в нашей модели, остается ответить на вопрос: какой вид функции потери выбрать для использования с ней? Для случая двоичной классификации, когда модель имеет одноэлементный выход, используется двоичная перекрестно-энтропийная потеря. В Keras мы задаем функцию потери, когда компилируем модель:

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Интересно, что мы также используем двоичную перекрестно-энтропийную потерю для мультиметочных моделей с сигмоидным результатом. Это обусловлено тем, что мультиметочная задача с тремя классами по существу является тремя меньшими двоичными классификационными задачами (рис. 3.9).

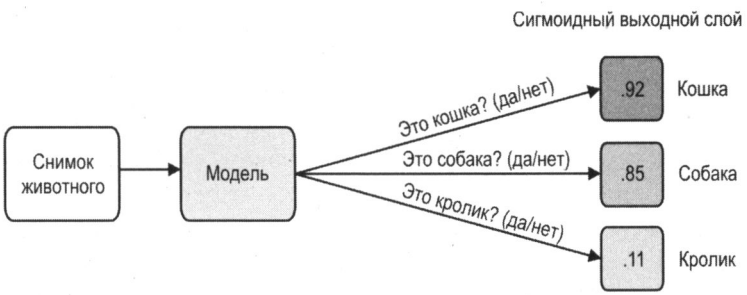


Рис. 3.9. Объяснение паттерна "Мультиметка" путем разбиения задачи на меньшие двоичные классификационные задачи

Разбор сигмоидных результатов

В целях извлечения предсказанной метки для модели с softmax-результатом на выходе просто берут argmax (индекс наибольшего значения) выходного массива, получая предсказанный класс. Разбор сигмоидных выходных данных менее прямолинеен. Вместо того чтобы брать класс с наивысшей предсказанной вероятностью, необходимо оценить вероятность каждого класса в выходном слое и учесть *порог* вероятности для нашего варианта использования. Оба подхода в значительной степени зависят от приложения конечного пользователя, в котором наша модель задействуется.

Под порогом мы подразумеваем вероятность, с которой нам удобно подтверждать принадлежность входного значения к отдельно взятому классу. Например, если мы строим модель для классифицирования разных типов животных на снимках, то нам, возможно, будет удобно говорить, что снимок содержит кошку, даже если модель в этом уверена только на 80%. С другой стороны, если мы строим модель, которая делает предсказания в области здравоохранения, то мы, вероятно, захотим, чтобы уверенность модели была ближе к 99%, прежде чем мы сможем подтвердить наличие или отсутствие какого-то заболевания. Хотя пороговое ограничение придется учитывать для любого типа классификационной модели, оно имеет особенную актуальность для паттерна "Мультиметка", поскольку нам понадобится определять пороговые значения для каждого класса, а они могут быть разными.

Обратимся к конкретному примеру: возьмем набор данных Stack Overflow из BigQuery и применим его для создания модели, которая предсказывает теги, связанные с вопросом Stack Overflow, имея его заголовок. Для простоты ограничим наш набор данных вопросами только с пятью тегами:

```
SELECT
  title,
  REPLACE(tags, "|", ",") AS tags
FROM
  `bigquery-public-data.stackoverflow.posts_questions`
WHERE
  REGEXP_CONTAINS(tags,
r"(?:keras|tensorflow|matplotlib|pandas|scikit-learn)")
```

Выходной слой модели будет выглядеть следующим образом (полный исходный код этого раздела доступен в репозитории[11] на GitHub):

```
keras.layers.Dense(5, activation='sigmoid')
```

Давайте возьмем вопрос Stack Overflow "What is the definition of a non-trainable parameter?" (Каково определение понятия нетренируемого параметра?) в качестве входного примера. Исходя из того, что наши выходные индексы соответствуют порядку тегов в запросе, выходное значение для этого вопроса может выглядеть следующим образом:

```
[.95, .83, .02, .08, .65]
```

[11] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/03_problem_representation/multilabel.ipynb**.

Наша модель на 95% уверена в том, что этот вопрос должен быть помечен тегом Keras, и на 83% в том, что он должен быть помечен тегом TensorFlow. Во время оценивания модельных предсказаний нам понадобится просматривать каждый элемент в выходном массиве и определяться с тем, как мы хотим показывать эти результаты нашим конечным пользователям. Если наш порог для всех тегов составляет 80%, то мы будем показывать ассоциированные с этим вопросом теги Keras и TensorFlow. С другой стороны, возможно, мы захотим, чтобы пользователи добавляли как можно больше тегов, а показывать будем варианты для любого тега с предсказательной уверенностью выше 50%.

Для подобного рода примеров, в которых цель состоит в том, чтобы в первую очередь предлагать возможные теги, делая меньший акцент на получение *строго* правильного тега, типичным эмпирическим правилом является использование частного от `n_specific_tag/n_total_examples` в качестве порога для каждого класса. Здесь `n_specific_tag` — это число примеров с одним тегом в наборе данных (например, "pandas"), а `n_total_examples` — суммарное число примеров в тренировочном наборе по всем тегам. За счет этого обеспечивается более качественная работа модели по сравнению с угадыванием какой-либо метки на основе ее появления в тренировочном наборе данных.

Более строгий подход к подбору порогового значения состоит в использовании метода S-Cut (score-Cut) или оптимизировании под модельную *F*-меру. Подробнее об этом можно прочитать в специальной статье[12]. Также нередко бывает полезно выполнять калибровку вероятностей по каждой метке, в особенности когда имеются тысячи меток и вы хотите учитывать верхние *K* из них (такая калибровка часто встречается в задачах поиска и ранжирования).

Как вы уже убедились, мультиметочные модели обеспечивают бóльшую гибкость в том, как мы выполняем разбор предсказаний, и требуют от нас старательного обдумывания результата на выходе из каждого класса.

Соображения в отношении наборов данных

Во время работы с мультиметочными классификационными задачами мы можем обеспечить сбалансированность набора данных, стремясь к относительно равному числу тренировочных примеров по каждому классу. Построение сбалансированного набора данных имеет свои нюансы для паттерна "Мультиметка".

Возьмем, к примеру, набор данных Stack Overflow. В нем, скорее всего, будет много вопросов, одновременно связанных и с TensorFlow, и с Keras. Но будут и вопросы о Keras, которые не имеют ничего общего с TensorFlow. Аналогично мы можем увидеть вопросы о построении графиков данных, помеченных одновременно как matplotlib и pandas, а также вопросы о предобработке данных, помеченных одновременно и как pandas, и как scikit-learn. Для того чтобы модель усваивала уникальные сведения из каждого тега, следует обеспечить, чтобы тренировочный набор данных состоял из разнообразных комбинаций каждого тега. Если большинст-

[12] См. **https://oreil.ly/oyR57**.

во вопросов о matplotlib в нашем наборе данных также помечены тегом pandas, то модель не научится классифицировать matplotlib в отдельности. Для учета этой ситуации следует проанализировать разные связи между метками, которые могут присутствовать в модели, и подсчитать число тренировочных примеров, принадлежащих каждой перекрывающейся комбинации меток.

При проведении разведывательного анализа связей между метками в наборе данных мы также можем столкнуться с иерархическими метками. Популярный набор данных для классифицирования изображений ImageNet[13] содержит тысячи помеченных снимков и часто используется в качестве отправной точки для трансферного обучения на снимковых моделях. Все метки, используемые в ImageNet, являются иерархическими, т. е. у всех снимков по меньшей мере одна метка, а многие снимки имеют более специфические метки, которые являются частью иерархии. Вот пример одной иерархии меток в ImageNet:

животные → беспозвоночные → членистоногие → арахнид → паук.

В зависимости от размера и природы набора данных существуют два распространенных подхода к манипулированию иерархическими метками.

- Использовать плоский подход и помещать каждую метку в один и тот же выходной массив независимо от иерархии, обеспечив, чтобы у вас было достаточно примеров каждой метки уровня "листового узла".
- Использовать паттерн "Каскад" (Cascade). Построить одну модель для выявления меток более высокого уровня. Основываясь на высокоуровневой классификации, отправлять пример в отдельную модель для более специфической классификационной задачи. Например, у нас может быть исходная модель, которая помечает снимки как "Растение", "Животное" или "Человек". В зависимости от того, какие метки применила первая модель, мы будем отправлять снимок в разные модели, чтобы применять более детализированные метки, такие как "суккулент" (сочное растение) или "берберийский лев".

Плоский подход проще, чем паттерн "Каскад", поскольку он требует только одной модели. Однако он может привести к тому, что модель потеряет информацию о более подробных классах меток, поскольку в наборе данных, естественно, будет больше тренировочных примеров с метками более высокого уровня.

Входные данные с перекрывающимися метками

Паттерн "Мультиметка" полезен и в тех случаях, когда входные данные иногда имеют перекрывающиеся метки. В качестве примера давайте возьмем модель, которая классифицирует предметы одежды для каталога. Если у нас несколько человек помечают каждый снимок в тренировочном наборе данных, то один разметчик может обозначить снимок юбки как "юбка макси", в то время как другой идентифицировать его как "плиссированная юбка". И то и другое верно. Однако если мы построим мультиклассовую классификационную модель на этих данных и переда-

[13] См. **https://oreil.ly/0VXtc**.

дим ей несколько примеров одного и того же снимка с разными метками, то, скорее всего, столкнемся с ситуациями, когда при генерировании предсказаний модель будет помечать похожие снимки по-разному. В идеале нам нужна модель, которая иногда не будет помечать снимок как "макси-юбка", а иногда как "плиссированная юбка", а будет предсказывать предмет одежды, обозначая его и тем и другим тегом (рис. 3.10).

Рис. 3.10. Использование входных значений от нескольких разметчиков для создания перекрывающихся меток в тех случаях, когда многочисленные описания элемента являются правильными

Паттерн "Мультиметка" решает эту задачу, позволяя нам ассоциировать обе перекрывающиеся метки со снимком. В случаях с перекрывающимися метками, где каждый снимок тренировочного набора данных оценивается несколькими разметчиками, мы можем выбирать максимальное число меток, которые мы хотели бы, чтобы разметчики назначали данному снимку, а затем брать наиболее часто выбираемые теги, чтобы связать их со снимком во время тренировки. Порог для "наиболее часто выбираемых меток" будет зависеть от задачи предсказания и количества имеющихся у нас разметчиков. Например, если каждый снимок оценивается 5 разметчиками и у нас 20 возможных тегов для каждого снимка, мы можем предложить разметчикам назначать каждому снимку 3 тега. Из этого списка из 15 меточных "голосов" в расчете на снимок затем мы могли бы выбирать от 2 до 3 с наибольшим числом голосов от разметчиков. Оценивая эту модель, мы должны принимать во внимание среднюю предсказательную уверенность, которую модель возвращает для каждой метки, и использовать ее для итеративного улучшения набора данных и качества меток.

Один против всех

Еще один технический прием манипулирования мультиметочной классификацией заключается в тренировке нескольких двоичных классификаторов вместо одной мультиметочной модели. Этот подход называется *"один против всех"*. В случае примера с набором данных Stack Overflow, в котором мы хотим помечать вопросы как TensorFlow, Python и pandas, мы бы натренировали отдельный классификатор для каждого из этих трех тегов: Python или нет, TensorFlow или нет и т. д. Затем мы

бы выбрали порог уверенности и пометили изначальный входной вопрос тегами из каждого двоичного классификатора выше некоторого порога.

Выгода от подхода "один против всех" заключается в том, что мы можем использовать его с архитектурами, которые способны выполнять только двоичную классификацию, например SVM. Он также помогает с редкими категориями, поскольку модель будет выполнять только по одной классификационной задаче на каждом входном значении и есть возможность применять паттерн "Перебалансировка" (Rebalancing). Недостатком этого подхода является дополнительная сложность тренировки большого числа разных классификаторов, требующая от нас вести разработки таким образом, чтобы предсказания генерировались каждой такой моделью в противовес всего лишь одному предсказанию.

Таким образом, паттерн "Мультиметка" следует использовать тогда, когда ваши данные попадают в любой из следующих сценариев классификации:

- один тренировочный пример может быть ассоциирован с взаимоисключающими метками;
- один тренировочный пример может иметь много иерархических меток;
- разметчики описывают один и тот же предмет по-разному, и каждая интерпретация является точной.

Во время реализации модели с множественными метками следует обеспечивать хорошую представленность комбинаций перекрывающихся меток в вашем наборе данных и учитывать пороговые значения, которые вы готовы принимать для каждой возможной метки в вашей модели. Использование сигмоидного выходного слоя является наиболее распространенным подходом для разработки моделей, которые реализуют мультиметочную классификацию. В дополнение к этому сигмоидный результат также может применяться к двоичным классификационным задачам, в которых тренировочный пример может иметь только одну из двух возможных меток.

ПАТТЕРН 7. Ансамбли

Паттерн "Ансамбли" (Ensembles) демонстрирует технические приемы машинного обучения, которые совмещают несколько ML-моделей и агрегируют их результаты для генерирования предсказаний. Ансамбли могут быть эффективным средством повышения результативности и продуцирования предсказаний, более качественных, чем любая отдельная модель.

Постановка задачи

Предположим, мы натренировали модель предсказания веса младенца, сгенерировав специальные признаки и внеся в нейронную сеть дополнительные слои, вследствие чего ошибка на тренировочном наборе почти равна нулю. "Превосходно!" — скажете вы. Однако, когда мы попытаемся использовать эту модель в производственных условиях в больнице или попробуем оценить результативность на отло-

женном тестовом наборе, наши предсказания окажутся полностью ошибочными. В чём же дело? И, что еще важнее, как это можно исправить?

Ни одна модель машинного обучения не идеальна. Для того чтобы понять, где и как наша модель становится ошибочной, ошибку ML-модели можно разбить на три части: неустранимую ошибку, ошибку из-за смещенности и ошибку из-за дисперсии. *Неустранимая ошибка* — это внутренне присущая ошибка в модели, возникающая из-за шума в наборе данных, формулировки задачи или плохих тренировочных примеров, таких как ошибки измерений или посторонние (вмешивающиеся) факторы. Как и следует из названия, мы мало что можем сделать с неустранимой ошибкой.

Две другие, смещенность и дисперсия, называются *устранимой ошибкой*, и именно здесь мы можем повлиять на результативность модели. Иными словами, смещенность — это неспособность модели усваивать достаточно сведений о связи между модельными признаками и метками, в то время как дисперсия отражает неспособность модели обобщать на новых, ранее не встречавшихся примерах. Модель с высокой смещенностью чрезмерно упрощает связи и считается *недоподогнанной*. Модель с высокой дисперсией слишком много усвоила о тренировочных данных и считается *переобученной*. Конечно, в любой ML-модели преследуется цель получить низкие смещенность и дисперсию, но на практике достичь и того и другого очень трудно. Этот феномен известен как компромисс между смещенностью и дисперсией. Невозможно иметь одновременно и то и другое. Например, увеличение сложности модели уменьшает смещенность, но при этом увеличивает дисперсию, в то время как уменьшение сложности модели уменьшает дисперсию, но вносит больше смещенности[14].

В одной научно-исследовательской работе[15] показано, что при использовании современных методов машинного обучения, таких как большие нейронные сети с высокой емкостью, подобное поведение действует только до определенного момента. В наблюдаемых экспериментах существует "интерполяционный порог", за пределами которого модели с очень высокой емкостью способны достигать нулевой ошибки тренировки, а также низкой ошибки на ранее не встречавшихся данных. Конечно же, во избежание переобучения на моделях большой емкости нам нужны гораздо более крупные наборы данных.

Существует ли технический прием, позволяющий смягчить этот компромисс между смещенностью и дисперсией на мало- и среднемасштабных задачах?

Решение

Ансамблевые методы — это метаалгоритмы, которые сочетают в себе несколько моделей машинного обучения и выступают в качестве технического приема, позволяющего уменьшать смещенность и/или дисперсию и улучшать результативность

[14] Для справки: если строго исходить из смысла слов bias и variance, то указанн компромисс может переводиться как "компромисс между искаженностью и вариативностью" модели. — *Прим. перев.*

[15] См. **https://oreil.ly/PxUvs**.

модели. В сущности, идея заключается в том, что сочетанием нескольких моделей можно улучшать результаты машинного обучения. Построив несколько моделей с разными индуктивными смещенностями и агрегировав их результаты, мы надеемся получить модель с более высокой результативностью. В этом разделе мы обсудим несколько часто используемых ансамблевых методов, включая бэггинг (модельное усреднение), бустинг (модельное усиление) и стэкинг (модельное наложение).

Бэггинг

Бэггинг, или модельное усреднение, — это разновидность параллельного ансамблирования, которая используется для урегулирования проблем, связанных с высокой дисперсией в моделях машинного обучения. Бутстраповская часть бэггинга относится к наборам данных, используемым для тренировки членов ансамбля. В частности, если существует k подмоделей, то для тренировки каждой подмодели ансамбля используется k отдельных наборов данных. Каждый набор данных строится путем случайного отбора экземпляров из изначального тренировочного набора данных (с возвратом)[16]. Это означает, что существует высокая вероятность того, что в любом из k наборов данных будут отсутствовать некоторые тренировочные примеры, но и любой набор данных, вероятно, будет иметь повторяющиеся тренировочные примеры. Агрегирование выполняется на результате работы многочисленных членов ансамблевой модели — берется либо среднее значение в случае регрессионной задачи, либо большинство голосов в случае классификационной.

Хорошим примером метода бутстрап-агрегированного ансамбля является случайный лес: многочисленные деревья решений тренируются на случайно отобранных подмножествах всех тренировочных данных, а затем предсказания деревьев агрегируются, продуцируя предсказания (рис. 3.11).

Популярные библиотеки машинного обучения имеют реализации методов бэггинга. Например, в целях реализации регрессии, основанной на случайном лесе в scikit-learn, чтобы предсказывать вес младенца из нашего набора данных о рождаемости, мы используем следующий фрагмент исходного кода:

```
from sklearn.ensemble import RandomForestRegressor

# Создать модель с 50 деревьями
RF_model = RandomForestRegressor(n_estimators=50,
                                 max_features='sqrt',
                                 n_jobs=-1, verbose = 1)

# Выполнить подгонку на тренировочных данных
RF_model.fit(X_train, Y_train))
```

[16] Бэггинг (bagging), или агрегирование бутстраповских выборок (bootstrap aggregating), предусматривает извлечение бутстраповской выборки, или бутстрапа (bootstrap sample), которое можно концептуально представить, как многократное взятие экземпляров с их возвратом в набор данных с целью получения синтетической выборки. — *Прим. перев.*

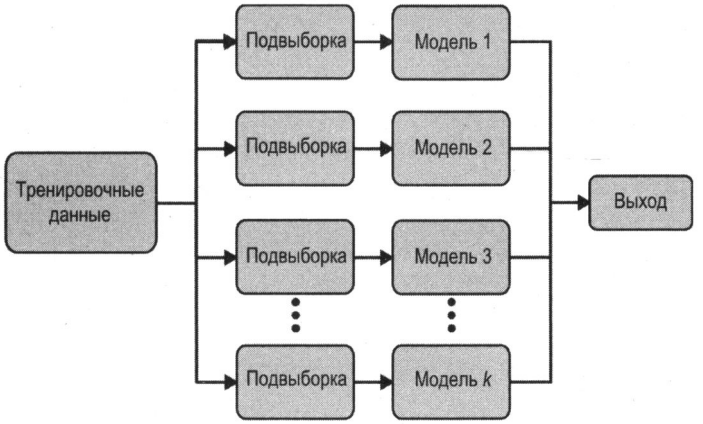


Рис. 3.11. Бэггинг хорошо подходит для уменьшения дисперсии на выходе из модели машинного обучения

Модельное усреднение является мощным и надежным методом уменьшения дисперсии модели. Как мы увидим, различные ансамблевые методы совмещают несколько подмоделей самыми разными способами, иногда используя разные модели, разные алгоритмы или даже разные целевые функции. В бэггинге модель и алгоритмы одинаковы. Например, в случайном лесе все подмодели представляют собой низкорослые деревья решений.

Бустинг

Бустинг, или модельное усиление, — это еще один ансамблевый технический прием. Однако, в отличие от бэггинга, бустинг строит ансамблевую модель с *большей* емкостью, чем модели отдельных его членов. По этой причине бустинг обеспечивает более эффективное средство уменьшения смещенности, чем дисперсия. Идея бустинга заключается в итеративном строительстве ансамбля моделей, в котором каждая последующая модель фокусируется на усвоении примеров, в которых предыдущая модель ошиблась. Иными словами, бустинг итеративно улучшает последовательность, состоящую из слабых учеников, беря средневзвешенное значение, чтобы в итоге получить сильного ученика.

В начале процедуры бустинга выбирается простая базовая модель `f_0`. В регрессионной задаче базовой моделью может быть просто среднее целевое значение: `f_0 = np.mean(Y_train)`. На первом итерационном шаге остатки `delta_1` измеряются и аппроксимируются посредством отдельной модели. Эта остаточная модель может быть чем угодно, но обычно она не очень сложная; часто используют слабого ученика, такого как дерево решений. Аппроксимация, обеспечиваемая остаточной моделью, добавляется в текущее предсказание, и процесс продолжается.

После многочисленных итераций остатки стремятся к нулю, а предсказание постепенно становится все лучше в моделировании изначального тренировочного набора данных. Обратите внимание, что на рис. 3.12 остатки для каждого элемента набора данных уменьшаются с каждой последующей итерацией.

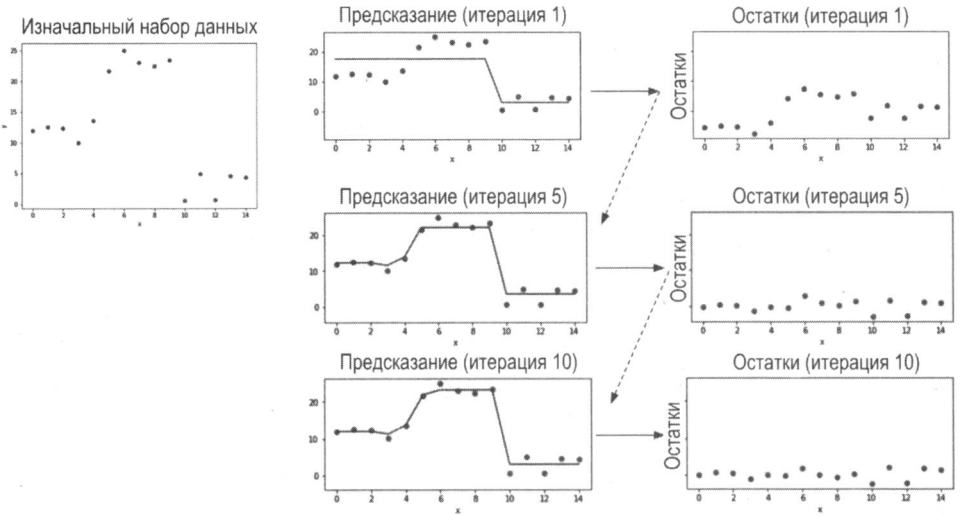


Рис. 3.12. Бустинг конвертирует слабых учеников в сильных, итеративно улучшая модельное предсказание

Вот несколько наиболее известных алгоритмов бустинга: AdaBoost, машины градиентного бустинга (gradient boosting machines)[17] и XGBoost. Все они имеют простые в использовании реализации в популярных фреймворках машинного обучения, таких как scikit-learn или TensorFlow.

Реализация в scikit-learn также является прямолинейной:

```
from sklearn.ensemble import GradientBoostingRegressor

# Создать регрессор, основанный на градиентном бустинге
GB_model = GradientBoostingRegressor(n_estimators=1,
                                     max_depth=1,
                                     learning_rate=1,
                                     criterion='mse')

# Выполнить подгонку на тренировочных данных
GB_model.fit(X_train, Y_train)
```

Стэкинг

Стэкинг, или модельное наложение, — это ансамблевый метод, который совмещает результаты на выходе из коллекции моделей для генерирования предсказания. Первоначальные модели, которые обычно относятся к разным типам моделей, тренируются до завершения на полном тренировочном наборе данных. Затем вторичная метамодель тренируется, используя первоначальные модельные результаты в каче-

[17] Градиентный бустинг (gradient boosting) — более общая форма бустинга, которая создана с точки зрения минимизирования функции стоимости. — *Прим. перев.*

стве признаков[18]. Эта вторая метамодель учится оптимально совмещать результаты первоначальных моделей с целью уменьшения ошибки тренировки и может быть любой моделью машинного обучения.

В целях реализации стэкингового ансамбля мы сначала тренируем всех членов ансамбля на тренировочном наборе данных. Следующий фрагмент исходного кода вызывает функцию `fit_model()`, которая принимает в качестве аргументов модель и данные тренировочного набора `X_train`, и метку `Y_train`. При таком подходе *участники* представляют собой список, содержащий все натренированные модели в нашем ансамбле. Полный код этого примера можно найти в репозитории[19] исходного кода книги:

```
members = [model_1, model_2, model_3]

# Выполнить подгонку моделей и сохранить их
n_members = len(members)

for i in range(n_members):
    # выполнить подгонку модели
    model = fit_model(members[i])
    # сохранить модель
    filename = 'models/model_' + str(i + 1) + '.h5'
    model.save(filename, save_format='tf')
    print('Сохранено в {}\n'.format(filename))
```

Эти подмодели встраиваются в более крупную стэкинговую ансамблевую модель в качестве индивидуальных входных переменных. Поскольку эти входные модели тренируются наряду со вторичной ансамблевой моделью, мы фиксируем веса этих входных моделей. Это можно сделать, установив `layer.trainable` равным `False` для моделей — членов ансамбля:

```
for i in range(n_members):
    model = members[i]
    for layer in model.layers:
        # сделать нетренируемым
        layer.trainable = False
        # переименовать во избежание проблемы с 'уникальным именем слоя'
        layer._name = 'ensemble_' + str(i+1) + '_' + layer.name
```

Мы создаем ансамблевую модель, сшивая компоненты вместе с помощью функционального API Keras:

```
member_inputs = [model.input for model in members]

# Конкатенировать результат из каждой модели
member_outputs = [model.output for model in members]
```

---

[18] Принято говорить, что эта последняя модель накладывается поверх других, отсюда и название "модельное наложение". — *Прим. перев.*

[19] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/03_problem_representation/ensemble_methods.ipynb**.

```
merge = layers.concatenate(member_outputs)
hidden = layers.Dense(10, activation='relu')(merge)
ensemble_output = layers.Dense(1, activation='relu')(hidden)
ensemble_model = Model(inputs=member_inputs, outputs=ensemble_output)

# Построить график ансамбля
tf.keras.utils.plot_model(ensemble_model, show_shapes=True,
                          to_file='ensemble_graph.png')

# Скомпилировать
ensemble_model.compile(loss='mse', optimizer='adam', metrics=['mse'])
```

В этом примере вторичная модель представляет собой плотную нейронную сеть с двумя скрытыми слоями. Посредством тренировки эта сеть учится совмещать результаты участников ансамбля наилучшим образом при генерировании предсказаний.

Почему это работает

Методы модельного усреднения, такие как бэггинг, работают потому, что, как правило, не все составляющие ансамблевую модель отдельные модели будут делать одинаковые ошибки на тестовом наборе. В идеальной ситуации каждая отдельная модель отклоняется на случайную величину, поэтому, когда их результаты усредняются, случайные ошибки взаимно аннулируются, и предсказание становится ближе к правильному ответу. Одним словом, в толпе есть мудрость.

Бустинг хорошо работает, потому что модель наказывается все больше и больше в соответствии с остатками на каждом итерационном шаге. С каждой итерацией ансамблевая модель вынуждена становиться все лучше в предсказании этих труднопредсказуемых примеров. Стэкинг работает, потому что сочетает в себе лучшее из бэггинга и бустинга. Вторичную модель можно рассматривать как более сложную версию модельного усреднения.

Бэггинг

Предположим, что мы натренировали k нейросетевых регрессионных моделей и усреднили их результаты, создав ансамблевую модель. Если каждая модель имеет ошибку `error_i` на каждом примере, где `error_i` берется из многомерного нормального распределения с нулевым средним значением, дисперсией `var` и ковариацией `cov`, то ансамблевый предсказатель будет иметь ошибку:

```
ensemble_error = 1./k * np.sum([error_1, error_2,..., error_k])
```

Если ошибки `error_i` идеально коррелированы так, что `cov = var`, то корень из среднеквадратической ошибки (RMSE) ансамблевой модели сводится к `var`. В этом случае модельное усреднение совсем не помогает. Если же ошибки `error_i` совершенно не коррелированы, то `cov = 0` и корень из среднеквадратической ошибки ансамблевой модели равен `var/k`. Поэтому ожидаемая квадратическая ошибка линей-

но уменьшается вместе с числом k моделей в ансамбле[20]. Таким образом, в среднем ансамбль будет показывать по меньшей мере такую же хорошую результативность, как и любая из отдельных моделей в ансамбле. Более того, если модели в ансамбле делают независимые ошибки (например, `cov = 0`), то ансамбль будет работать значительно лучше. В конечном счете ключом к успеху с бэггингом является разнообразие моделей.

Это также объясняет причину, почему бэггинг обычно менее эффективен для более стабильных учеников, таких как k ближайших соседей (k-nearest neighbors, kNN), наивный Байес, линейные модели или опорно-векторные машины (support vector machines, SVM), поскольку размер тренировочного набора уменьшается за счет бутстрапирования. Даже при использовании одних и тех же тренировочных данных нейронные сети ввиду случайных инициализаций весов, или случайного отбора мини-пакетов, или разных гиперпараметров могут демонстрировать разнообразные решения, создавая модели, ошибки которых частично независимы. Следовательно, модельное усреднение может даже приносить пользу нейронным сетям, натренированным на одном и том же наборе данных. По сути, одними из рекомендуемых решений для устранения высокой дисперсии в нейронных сетях являются тренировка многочисленных моделей и агрегирование их результатов.

Бустинг

Алгоритм бустинга работает путем итеративного совершенствования модели с целью уменьшения ошибки предсказания. Каждый новый слабый ученик исправляет ошибки предыдущего предсказания, моделируя остатки `delta_i` каждого шага. Окончательное предсказание представляет собой сумму результатов на выходе из базового ученика и каждого последующего слабого ученика (рис. 3.13).

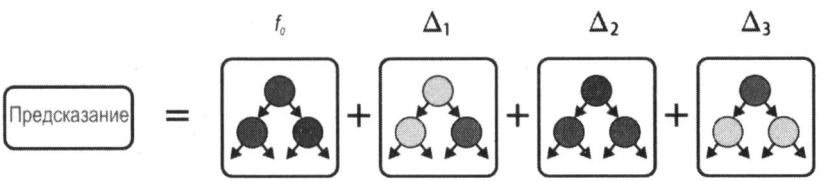


Рис. 3.13. Бустинг итеративно строит сильного ученика из последовательности слабых учеников, которые моделируют остаточную ошибку предыдущей итерации

Таким образом, результирующая ансамблевая модель последовательно становится всё сложнее, обладая большей емкостью, чем любой из ее членов. Это также объясняет, почему бустинг в особенности хорош для борьбы с высокой смещенностью. Напомним, что смещенность связана с тенденцией модели становиться недоподогнанной. Итеративно фокусируясь на труднопредсказуемых примерах, бустинг эффективно уменьшает смещенность результирующей модели.

---

[20] Непосредственное вычисление этих значений можно найти в гл. 7 книги "Глубокое обучение" (Goodfellow I., Bengio Y., Courville A. Deep Learning. — Cambridge, MA: MIT Press, 2016. — Ch. 7).

Стэкинг

Стэкинг можно рассматривать как расширение простого модельного усреднения, в котором мы тренируем k моделей до завершения на тренировочном наборе данных, а затем усредняем результаты для определения предсказания. Простое модельное усреднение похоже на стэкинг, но модели в ансамбле могут быть разных типов, в то время как в случае бэггинга модели имеют один тип. В более общем случае мы могли бы модифицировать шаг усреднения и брать средневзвешенное значение, например, чтобы придавать больший вес в нашем ансамбле одной модели по сравнению с другими (рис. 3.14).

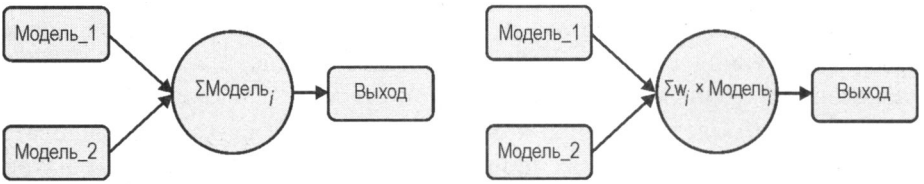


Рис. 3.14. Простейшая форма модельного усреднения усредняет результаты двух или более разных моделей машинного обучения. Среднее значение можно заменить средневзвешенным, где вес может основываться на относительной точности моделей

Стэкинг можно рассматривать как более продвинутую версию модельного усреднения, в котором вместо того чтобы брать среднее или средневзвешенное значение, мы тренируем вторую модель машинного обучения на результатах, чтобы научиться совмещать результаты моделей в нашем ансамбле, производя предсказание (рис. 3.15). За счет этого обеспечиваются все преимущества уменьшения дисперсии, как в случае с приемами бэггинга, но и осуществляется контроль высокой смещенности.

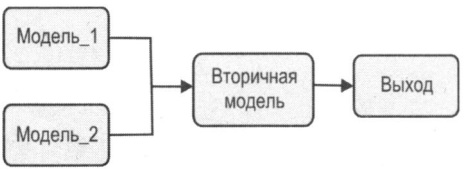


Рис. 3.15. Стэкинг — это технический прием ансамблевого обучения, который объединяет результаты из нескольких разных ML-моделей, передавая их на вход во вторичную ML-модель, которая делает предсказания

Компромиссы и альтернативы

В современном машинном обучении ансамблевые методы стали довольно популярными и сыграли большую роль в получении наград в хорошо известных конкурсах, возможно, в первую очередь в Премии Netflix[21]. Кроме того, многочисленные теоретические доказательства подтверждают успех, продемонстрированный на этих реальных конкурсах.

[21] См. **https://oreil.ly/ybZ28**.

Увеличенное время тренировки и проектирования

Одним из недостатков ансамблевого обучения является увеличение времени тренировки и проектирования. Например, для стэкинговой ансамблевой модели подбор моделей — членов ансамбля может потребовать отдельного уровня компетенции и ставит свои вопросы в отношении возможности многократного использования одних и тех же архитектур или поощрения разнообразия. Если мы используем разные архитектуры, то какие из них мы должны использовать? И сколько? Вместо разработки одной модели ML (что само по себе является большой работой!) мы теперь разрабатываем k моделей. Мы внесли дополнительный объем накладных расходов в разработку, не говоря уже о техническом сопровождении, сложности предсказательного вывода и использовании ресурсов, если предполагается внедрение ансамблевой модели. По мере увеличения числа моделей в ансамбле этот подход может быстро стать непрактичным.

Популярные библиотеки машинного обучения, например scikit-learn и TensorFlow, предоставляют простые в использовании реализации для многих распространенных методов бэггинга и бустинга, таких как случайный лес, AdaBoost, градиентный бустинг и XGBoost. Однако мы должны старательно продумывать вопрос о том, стоит ли увеличивать накладные расходы, связанные с ансамблевым методом. Всегда следует сравнивать точность и использование ресурсов с линейной или глубокой нейросетевой моделью. Обратите внимание, что перегонка (*см. разд. "Паттерн 11. Полезное переобучение" главы 4*) ансамбля нейронных сетей нередко способна снизить сложность и повысить результативность.

Отсев в качестве бэггинга

Такие приемы, как отсев, обеспечивают мощную и эффективную альтернативу. *Отсевом* называется технический прием регуляризации в глубоком обучении, но под ним также понимается аппроксимация бэггинга. Отсев в нейронной сети случайно (с заданной вероятностью) "выключает" нейроны сети для каждого тренировочного мини-пакета, по существу оценивая бэггинговый ансамбль экспоненциально большого числа нейронных сетей. Тем не менее тренировка нейронной сети с отсевом — это не совсем то же самое, что и бэггинг. Есть два заметных отличия. Во-первых, в случае бэггинга модели независимы, в то время как во время тренировки с отсевом модели используют параметры коллективно. Во-вторых, в бэггинге модели тренируются до схождения на соответствующем им тренировочном наборе. Однако во время тренировки с отсевом модели членов ансамбля будут тренироваться только в течение одного тренировочного шага, поскольку на каждой итерации цикла тренировки отсеиваются разные узлы.

Снижение модельной интерпретируемости

Есть еще один момент, который следует иметь в виду, и это модельная интерпретируемость. Даже при использовании одной модели в глубоком обучении бывает трудно дать внятное объяснение причины, почему наша модель делает именно такие предсказания. Указанная проблема усугубляется ансамблевыми моделями.

Возьмем, к примеру, деревья решений в сравнении со случайным лесом. Дерево решений усваивает граничные значения для каждого признака, которые ведут отдельный экземпляр к окончательному модельному предсказанию. Легко объяснить, почему дерево решений делает именно эти предсказания. Случайный лес, будучи ансамблем многочисленных деревьев решений, этот уровень локальной интерпретируемости теряет.

Выбор правильного инструмента для задачи

Также важно иметь в виду компромисс между смещенностью и дисперсией. Некоторые ансамблевые технические приемы лучше справляются со смещенностью или дисперсией, чем другие (табл. 3.2). В частности, бустинг адаптирован для устранения высокой смещенности, в то время как бэггинг полезен для исправления высокой дисперсии. Тем не менее, как мы видели в *разд. "Бэггинг" ранее в этой главе*, сочетание двух моделей с сильно коррелированными ошибками не сделает ничего, чтобы снизить дисперсию. И поэтому использование неправильного ансамблевого метода для решаемой задачи не обязательно улучшит результативность; оно просто добавит ненужные накладные расходы.

Таблица 3.2. *Резюме компромисса между смещенностью и дисперсией*

Задача	Ансамблевое решение
Высокая смещенность (недоподгонка)	Бустинг
Высокая дисперсия (переобучение)	Бэггинг

Другие ансамблевые методы

Мы обсудили несколько наиболее распространенных ансамблевых методов ML. Рассмотренный выше список ни в коем случае не является исчерпывающим, и в эти категории вписываются разные алгоритмы. Существуют и другие ансамблевые технические приемы, в том числе многие из них включают байесов подход или сочетают поиск нейронной архитектуры и обучение с подкреплением[22], например это такие технологии, как Google AdaNet или AutoML. Одним словом, паттерн "Ансамбли" охватывает технические приемы, которые совмещают многочисленные модели машинного обучения с целью повышения совокупной модельной результативности и могут быть особенно полезны при устранении распространенных трудностей, возникающих во время тренировки, таких как высокая смещенность или высокая дисперсия.

---

[22] Обучение с подкреплением (reinforcement learning) — гедонистическая самообучающаяся система обучения путем проб и ошибок, которая адаптирует свое поведение с целью максимизировать подкрепление окружающей среды. Под подкреплением здесь понимается стимул, т. е. действие извне, заставляющее ученика приспосабливать свое поведение в ней. Говоря проще, здесь уместна аналогия с ослом и морковкой, где учеником является осел, а морковкой — стимул. — *Прим. перев.*

ПАТТЕРН 8. Каскад

Паттерн "Каскад" (Cascade) затрагивает ситуации, в которых задача машинного обучения может быть выгодно разбита на ряд задач. Такой каскад часто требует продуманного планирования эксперимента.

Постановка задачи

Что произойдет, если нам нужно предсказывать значение как обычной, так и необычной активности? Модель научится игнорировать необычную активность, потому что она редко происходит. Если необычная активность также ассоциируется с аномальными значениями, то страдает тренируемость.

Например, предположим, что мы пытаемся натренировать модель на предсказание вероятности того, что клиент вернет купленный товар. Если мы натренируем одну-единственную модель, то будет потеряно поведение ресейлеров (перепродавцов), потому что есть миллионы розничных покупателей (и розничных транзакций) и только несколько тысяч ресейлеров. В то время когда совершается покупка, мы не знаем, кем является клиент: розничным покупателем или ресейлером. Однако, наблюдая за другими рынками, мы выяснили: когда купленные у нас товары впоследствии перепродаются, то наш тренировочный набор данных имеет метку, которая выявляет покупку как совершенную ресейлером.

Один из подходов к решению этой задачи заключается в придании экземплярам ресейлеров большего веса во время тренировки модели. Этот подход является субоптимальным, потому что нам нужно понять более распространенный сценарий розничного покупателя как можно правильно. Мы не хотим выменивать более низкую точность в сценарии розничного покупателя на более высокую точность в сценарии ресейлера. Однако розничные покупатели и ресейлеры ведут себя совершенно по-разному; например, в то время как розничные покупатели возвращают товары в течение недели или около того, ресейлеры возвращают товары только в том случае, если они не могут их продать, и поэтому возврат может произойти через несколько месяцев. Деловое решение о товарных запасах будет отличаться для вероятных возвратов от розничных покупателей по сравнению с возвратами от ресейлеров. Поэтому необходимо понять оба типа возвратов как можно точнее. Простое придание дополнительного веса экземплярам ресейлеров не сработает.

Интуитивно понятный подход к решению этой задачи состоит в использовании паттерна "Каскад". Мы разбиваем задачу на четыре части:

1. Предсказание того, что отдельно взятая транзакция будет совершена ресейлером.
2. Тренировка одной модели на продажах розничным покупателям.
3. Тренировка второй модели на продажах ресейлерам.
4. Совмещение результатов трех отдельных моделей в производственных условиях для предсказания возможности возврата каждого приобретенного товара и вероятности того, что сделка совершена ресейлером.

Это позволяет принимать разные решения о товарах, которые могут быть возвращены в зависимости от типа покупателя, и обеспечивает максимальную точность моделей на своем сегменте тренировочных данных в шагах 2 и 3. Каждая из этих моделей тренируется относительно просто. Первая — это всего-навсего классификатор, и если необычная активность чрезвычайно редка, то мы можем использовать паттерн "Перебалансировка" с целью улаживания такой ситуации. Следующие две модели по существу являются классификационными моделями, натренированными на разных сегментах тренировочных данных. Эта комбинация является детерминированной, поскольку мы выбираем выполняемую модель исходя из того, кому принадлежала активность: ресейлеру или кому-то еще.

Проблема возникает во время предсказания, когда у нас нет истинных меток, а есть только результат на выходе из первой классификационной модели. Основываясь на результате первой модели, нужно определиться с тем, какую из двух моделей продаж мы используем. Проблема в том, что мы тренируем на метках, но во время предсказательного вывода нам придется принимать решения, основываясь на предсказаниях. А предсказания имеют ошибки. Поэтому вторая и третья модели должны будут делать предсказания на данных, которые они, возможно, никогда не видели во время тренировки.

В качестве крайнего примера предположим, что адрес, который предоставляют ресейлеры, всегда находится в промзоне города, тогда как розничные покупатели могут жить где угодно. Если первая (классификационная) модель ошибается и розничный покупатель ошибочно идентифицируется как ресейлер, то вызываемая предсказательная модель аннулирования не будет иметь района проживания клиента в своем словаре.

Как натренировать каскад моделей, в котором результат на выходе из одной модели является входом в следующую модель или который определяет подборку последующих моделей?

Решение

Любая задача машинного обучения, в которой результат на выходе из одной модели является входом в следующую модель или определяет подборку последующих моделей, называется *каскадом*. Во время тренировки каскада ML-моделей следует проявлять особую осторожность.

Например, задачу машинного обучения, которая иногда предусматривает наличие необычных обстоятельств, решают, рассматривая ее как каскад из четырех задач машинного обучения:

1. Классификационная модель для выявления данного обстоятельства.
2. Отдельная модель, натренированная на необычных обстоятельствах.
3. Отдельная модель, натренированная на типичных обстоятельствах.
4. Модель для объединения результатов на выходах из двух отдельных моделей, поскольку результат представляет собой вероятностную комбинацию двух других результатов.

На первый взгляд, этот подход выглядит как частный случай паттерна "Ансамбль", но рассматривается отдельно из-за особого плана экспериментов, необходимого при выполнении каскада.

Допустим, что для оценивания стоимости запаса велосипедов на станциях мы хотим предсказывать расстояние между станциями проката и местами возврата велосипедов в Сан-Франциско. Другими словами, цель модели состоит в том, чтобы предсказывать расстояние, необходимое для транспортировки велосипеда обратно к месту проката, учитывая такие признаки, как время начала проката, места выдачи велосипеда в прокат, принадлежность арендатора к подписчикам и т. д. Проблема в том, что продолжительность проката более 4 часов предусматривает совершенно иное поведение арендатора, чем более кратковременный прокат, и алгоритм обеспечения запаса требует обоих результатов (вероятности того, что прокат будет длиться более 4 часов, и возможного расстояния, на которое велосипед должен быть перевезен). Однако только очень небольшая часть осуществленных прокатов связана с такими ненормальными поездками.

Один из подходов к решению этой задачи состоит в тренировке классификационной модели сначала на классификацию поездки на основе их продолжительности (`Long`) и типичности (`Typical`) (полный код[23] находится в репозитории исходного кода этой книги):

```
CREATE OR REPLACE MODEL mlpatterns.classify_trips
TRANSFORM(
  trip_type,
  EXTRACT (HOUR FROM start_date) AS start_hour,
  EXTRACT (DAYOFWEEK FROM start_date) AS day_of_week,
  start_station_name,
  subscriber_type,
  ...
)
OPTIONS(model_type='logistic_reg',
        auto_class_weights=True,
        input_label_cols=['trip_type']) AS
SELECT
  start_date, start_station_name, subscriber_type, ...
  IF(duration_sec > 3600*4, 'Long', 'Typical') AS trip_type
FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
```

Может возникнуть соблазн просто разбить тренировочный набор данных на две части в зависимости от фактической продолжительности проката и натренировать следующие две модели: одну на продолжительных прокатах, а другую на типичных прокатах. Проблема в том, что только что рассмотренная классификационная модель будет иметь ошибки. И действительно, оценивание модели на отложенном срезе данных о велосипедах Сан-Франциско показывает, что точность модели

[23] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/03_problem_representation/cascade.ipynb**.

Рис. 3.16. Точность классификационной модели в предсказании атипичного поведения вряд ли составит 100%

составляет всего около 75% (рис. 3.16). Учитывая это обстоятельство, тренировка модели на идеальном срезе данных вызовет слезы на глазах.

Вместо этого после тренировки указанной классификационной модели нам нужно использовать предсказания этой модели для создания тренировочного набора данных для следующего набора моделей. Например, мы могли бы создать тренировочный набор данных, чтобы модель предсказывала расстояние типичных прокатов, используя:

```
CREATE OR REPLACE TABLE mlpatterns.Typical_trips AS
SELECT
  * EXCEPT(predicted_trip_type_probs, predicted_trip_type)
FROM
ML.PREDICT(MODEL mlpatterns.classify_trips,
  (SELECT
  start_date, start_station_name, subscriber_type, ...,
  ST_Distance(start_station_geom, end_station_geom) AS distance
  FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`)
)
WHERE predicted_trip_type = 'Typical' AND distance IS NOT NULL
```

Затем следует применить этот набор данных для тренировки модели предсказывать расстояния:

```
CREATE OR REPLACE MODEL mlpatterns.predict_distance_Typical
TRANSFORM(
  distance,
  EXTRACT (HOUR FROM start_date) AS start_hour,
  EXTRACT (DAYOFWEEK FROM start_date) AS day_of_week,
  start_station_name,
  subscriber_type,
  ...
)
OPTIONS(model_type='linear_reg', input_label_cols=['distance']) AS
```

```
SELECT
  *
FROM
  mlpatterns.Typical_trips
```

Наконец, наше оценивание, предсказание и другие действия должны учитывать, что вместо только одной модели нам нужно будет использовать три натренированные модели. И это реализовано в паттерне "Каскад" (Cascade).

На практике уследить за "Каскадным" рабочим потоком становится трудно. Вместо того чтобы тренировать модели по отдельности, лучше автоматизировать весь рабочий поток целиком, используя паттерн "Конвейер рабочего потока" (Workflow Pipelines; *см. главу 6*), как показано на рис. 3.17. Главное — обеспечивать, чтобы тренировочные наборы данных для двух нижестоящих моделей создавались всякий раз, когда эксперимент выполняется, основываясь на предсказаниях вышестоящих моделей.

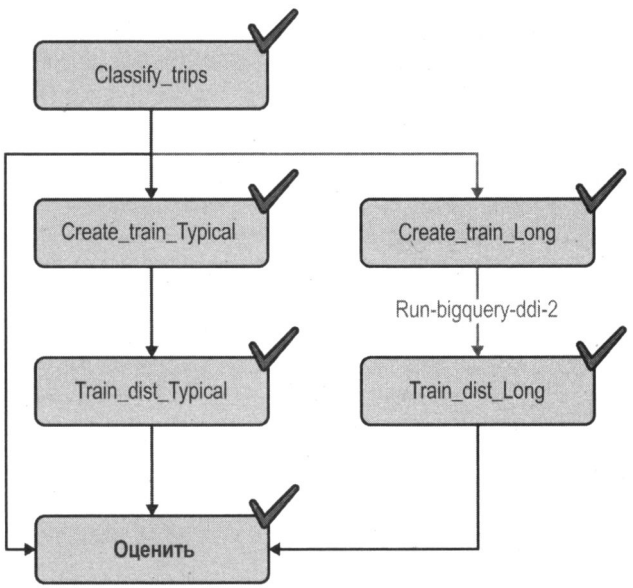


Рис. 3.17. Конвейер для тренировки каскада моделей как единого задания

Хотя мы ввели паттерн "Каскад" как подход к предсказанию значения во время обычной и необычной активности, техническое решение способно справляться с более общей ситуацией. Конвейерный каркас позволяет нам манипулировать любой ситуацией, когда задача машинного обучения может быть выгодно подразделена на серию (или каскад) задач. Всякий раз, когда результат на выходе из модели машинного обучения должен подаваться на вход другой модели, вторая модель должна быть натренирована на предсказаниях первой модели. Во всех таких ситуациях будет полезен конвейер.

Платформа для строительства и развертывания рабочих процессов машинного обучения Kubeflow Pipeplines обеспечивает такой каркас. Поскольку она работает

с контейнерами, опорные модели машинного обучения и склеивающий код могут быть написаны практически на любом языке программирования или языка написания скриптов. Здесь мы обернем приведенные выше модели SQL BigQuery в функции Python с помощью клиентской библиотеки BigQuery. Для реализации отдельных компонентов мы могли бы использовать TensorFlow, или scikit-learn, или даже R.

Конвейерный код, использующий платформу Kubeflow Pipeplines, можно выразить довольно просто, как показано ниже (полный код[24] можно найти в репозитории исходного кода этой книги):

```
@dsl.pipeline(
    name='Каскадный конвейер на велопрокате в Сан-Франциско',
    description='Каскадный конвейер на велопрокате в Сан-Франциско'
)

def cascade_pipeline(
    project_id = PROJECT_ID
):
    ddlop = comp.func_to_container_op(run_bigquery_ddl,
                    packages_to_install=['google-cloud-bigquery'])
    c1 = train_classification_model(ddlop, PROJECT_ID)
    c1_model_name = c1.outputs['created_table']

    c2a_input = create_training_data(ddlop,
                          PROJECT_ID, c1_model_name, 'Typical')
    c2b_input = create_training_data(ddlop,
                          PROJECT_ID, c1_model_name, 'Long')

    c3a_model = train_distance_model(ddlop,
                          PROJECT_ID, c2a_input.outputs['created_table'],
'Typical')
    c3b_model = train_distance_model(ddlop,
                          PROJECT_ID, c2b_input.outputs['created_table'], 'Long')

    ...
```

Весь конвейер целиком может быть отправлен на выполнение, а разные запуски эксперимента отслеживаются с помощью платформы Kubeflow Pipeplines.

Если в качестве конвейерного каркаса мы используем TFX (мы можем выполнять TFX на платформе Kubeflow Pipeplines), то нет необходимости развертывать вышестоящие модели, для того чтобы использовать их выходные предсказания в нижестоящих моделях. Вместо этого мы можем применить метод `tft.apply_saved_model` библиотеки TensorFlow Transform в рамках наших операций предобработки. Паттерн "Преобразователь" (Transform) обсуждается в *главе 6*.

[24] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/03_problem_representation/cascade.ipynb**.

Использование конвейера настоятельно рекомендуется всякий раз, когда ML-модели соединены в цепочку. Такой каркас обеспечивает, чтобы нисходящие модели перетренировывались всякий раз, когда вышестоящие модели пересматриваются, и наличие у нас истории всех предыдущих тренировочных прогонов.

Компромиссы и альтернативы

С паттерном "Каскад" не следует перебарщивать: в отличие от многих паттернов, которые мы рассматриваем в этой книге, "Каскад" не обязательно наилучшее практическое решение. Он вносит значительную долю сложности в рабочий поток процессов машинного обучения и может привести к снижению производительности. Обратите внимание, что конвейер является передовой практикой, но, насколько это возможно, постарайтесь ограничить конвейер одной задачей машинного обучения (приемка данных, их предобработка, валидация, преобразование, тренировка модели, ее оценивание и развертывание). В одном конвейере следует избегать наличия нескольких моделей машинного обучения, как в паттерне "Каскад".

Детерминированные входные данные

Идея подразделять задачи ML обычно является неудачной, поскольку модель ML может/должна усваивать комбинации многочисленных факторов. Приведем примеры.

- Если условие можно узнать детерминированно из входной переменной (праздничные покупки против покупок в будние дни), мы должны добавить в модель указанное условие в качестве еще одной входной переменной.
- Если условие предусматривает экстремумы только в одной входной переменной (некоторые клиенты живут поблизости, тогда как другие живут далеко, и смысл слов "близко"/"далеко" необходимо усвоить из данных), то для манипулирования таким условием мы можем использовать смешанное представление входных данных.

Паттерн "Каскад" обращается к необычному сценарию, для которого у нас нет категориальной входной переменной и для которого экстремальные значения должны усваиваться из нескольких входных переменных.

Одиночная модель

Паттерн "Каскад" не следует использовать для распространенных сценариев, в которых достаточно одной-единственной модели. Например, предположим, что мы пытаемся узнать склонность клиента к покупке. Мы можем подумать, что нам нужно усвоить разные модели для людей, которые делали сравнительные покупки в сопоставлении с теми, кто этого не делал. Мы действительно не знаем, кто делал сравнительные покупки, но мы можем высказать обоснованную догадку, основываясь на числе посещений, продолжительности нахождения товара в корзине и т. д. Эта задача не нуждается в паттерне "Каскад", потому что она распространена настолько (бóльшая часть клиентов будет делать сравнительные покупки), что модель

машинного обучения должна быть в состоянии усваивать это неявно в ходе тренировки. В распространенных сценариях следует тренировать одну-единственную модель.

Внутренняя согласованность

"Каскад" необходим, когда нам нужно поддерживать внутреннюю согласованность между предсказаниями многочисленных моделей. Обратите внимание, что мы пытаемся сделать больше, чем просто предсказывать необычную активность. Мы пытаемся предсказывать возвраты, учитывая, что также будет иметься некоторая активность со стороны ресейлеров. Если задача состоит только в том, чтобы предсказывать, что продажа будет совершена ресейлером, то мы бы использовали паттерн "Перебалансировка". Причина использования "Каскада" заключается в том, что результат несбалансированных меток необходим на входе в последующие модели и полезен сам по себе.

Аналогично предположим, что причина, по которой мы тренируем модель предсказывать склонность клиента к покупке, состоит в том, чтобы предложить товар со скидкой. Сам факт того, предлагаем ли мы товар со скидкой и собственно размер скидки, очень часто будет зависеть от того, совершает ли этот клиент сравнительную покупку или нет. С учетом этого нам нужна внутренняя согласованность между двумя моделями (моделью для клиентов, совершающих сравнительные покупки, и моделью для склонности к покупке). В этом случае может потребоваться паттерн "Каскад".

Предварительно натренированные модели

"Каскад" также необходим, когда мы хотим использовать результат предварительно натренированной модели многократно на входе в нашу модель. Например, предположим, что мы строим модель для обнаружения посетителей, которым вход в здание разрешен, чтобы иметь возможность открывать ворота автоматически. Одним из входов в нашу модель может быть номерной знак автомобиля. Вместо того чтобы прямо анализировать фотографию службы безопасности в нашей модели, проще было бы использовать результат модели оптического распознавания символов (optical character recognition, OCR). Очень важно понимать, что системы OCR будут выдавать ошибки, и поэтому мы не должны тренировать модель с использованием идеальной информации о номерных знаках. Вместо этого нам следует тренировать модель на фактическом результате работы системы OCR. И действительно, поскольку разные модели OCR будут вести себя по-разному и выдавать разные ошибки, необходимо будет перетренировать модель, к примеру, если мы изменим поставщика нашей системы OCR.

Распространенным сценарием использования предварительно натренированной модели в качестве первого шага конвейера является использование модели обнаружения объектов, за которой следует модель классификацирования снимков. Например, модель обнаружения объектов может найти на снимке все сумки, промежуточный шаг может обрезать снимок до ограничительных рамок обнаруженных объектов, а последующая модель — определить тип сумки. Мы рекомендуем ис-

пользовать "Каскад", чтобы иметь возможность перетренировывать весь конвейер целиком всякий раз, когда обновляется модель обнаружения объектов (например, новой версией API).

Изменение контекста вместо каскада

Обратите внимание, что в нашем примере мы пытались предсказывать вероятность того, что товар будет возвращен, и поэтому речь шла о кассификационной задаче. Предположим, вместо этого мы хотим предсказывать почасовые объемы продаж. Бóльшую часть времени мы будем обслуживать только розничных покупателей, но от случая к случаю (возможно, четыре или пять раз в год) у нас будет оптовый покупатель.

Здесь имеется в виду условно регрессионная задача предсказания ежедневных объемов продаж, в которой мы имеем посторонний фактор в виде оптовых покупателей. Подход на основе переформулировки регрессионной задачи в задачу классифицирования разных объемов продаж может быть более оптимальным. Хотя указанный подход будет предусматривать тренировку классификационной модели для каждой корзины объема продаж, он позволяет избежать необходимости добиваться правильной классификации розничной торговли относительно оптовой.

Регрессия в редких ситуациях

Паттерн "Каскад" бывает полезным при проведении регрессии, когда некоторые значения гораздо более распространены, чем другие. Например, мы могли бы предсказывать количество осадков по спутниковому снимку. И может оказаться, что на 99% пикселах дождя не будет. В таком случае бывает полезно создавать наложенную (stacked) классификационную модель, за которой следует регрессионная модель:

1. Сначала предсказать возможность дождя.
2. Для пикселов, где модель предсказывает, что дождь маловероятен, предсказать количество осадков равным нулю.
3. Натренировать регрессионную модель предсказывать количество осадков на пикселах, где модель предсказывает, что дождь вероятен.

Очень важно понимать, что классификационная модель не совершенна, и поэтому регрессионную модель приходится тренировать на пикселах, которые классификационная модель предсказывает, что, скорее всего, будет дождь (а не только на пикселах, которые соответствуют дождю в помеченном наборе данных). Дополнительные решения этой задачи см. также в обсуждениях в *разд. "Паттерн 10. Перебалансировка" далее* и *"Паттерн 5. Переформулировка" ранее в этой главе*.

ПАТТЕРН 9. Нейтральный класс

Паттерн "Нейтральный класс" (Neutral Class) охватывает многочисленные ситуации, возникающие во время классифицирования, когда полезно создавать нейтральный класс. Например, вместо тренировки двоичного классификатора, который

выводит вероятность события, лучше тренировать трехклассовый классификатор, который выводит дизъюнктивные вероятности для "Да", "Нет" и "Возможно". Термин "дизъюнктный" здесь означает, что классы не пересекаются. Например, тренировочный образец может принадлежать только одному классу, и поэтому нет никакого наложения между "Да" и "Возможно". "Возможно" в данном случае является нейтральным классом.

Постановка задачи

Представьте себе, что мы пытаемся создать модель, которая дает рекомендации по обезболивающим препаратам. Есть два варианта: ибупрофен и ацетаминофен[25], и в нашем историческом наборе данных оказывается, что ацетаминофен, как правило, назначается пациентам с риском желудочно-кишечных проблем, а ибупрофен преимущественно назначается пациентам с риском повреждения печени. Сверх этого все, как правило, имеет довольно случайный характер; некоторые врачи по умолчанию используют ацетаминофен, а другие — ибупрофен.

Тренировка двоичного классификатора на таком наборе данных приведет к плохой точности, поскольку модели потребуется правильно понимать по существу произвольные случаи.

Решение

Представьте себе другой сценарий. Предположим, что электронный журнал, который фиксирует рекомендации врача, также спрашивает его о том, будут ли приемлемы альтернативные обезболивающие препараты. Если врач назначает ацетаминофен, то приложение спрашивает врача о том, может ли пациент использовать ибупрофен, если у него в аптечке уже он есть.

Основываясь на ответе на второй вопрос, мы имеем нейтральный класс. Рекомендации по-прежнему могут быть записаны как "ацетаминофен", но запись фиксирует, что врач был нейтрален по отношению к этому пациенту. Обратите внимание, что это принципиально требует от нас соответствующего дизайна процедуры сбора данных — мы не можем производить нейтральный класс постфактум. Мы должны правильно спланировать задачу машинного обучения. Правильный дизайн в данном случае начинается в первую очередь с того, как именно мы ставим задачу.

Если у нас есть лишь исторический набор данных, то нам нужно будет задействовать разметку[26]. Мы могли бы попросить разметчиков-людей валидировать изначальный выбор врача и отвечать на вопрос о приемлемости альтернативного обезболивающего.

---

[25] Этот пример используется всего лишь в иллюстративных целях; не воспринимайте его как медицинские рекомендации!

[26] См. **https://oreil.ly/OSZsi**.

Почему это работает

Мы можем разведать механизм, с помощью которого это работает, просимулировав механизм, причастный к синтетическому набору данных. Затем мы покажем, что нечто подобное происходит и в реальном мире с предельными случаями.

Синтетические данные

Давайте создадим синтетический набор данных длиной N, в котором 10% данных представляют пациентов с желтухой в анамнезе. Поскольку они подвержены риску повреждения печени, правильным предписанием для них будет ибупрофен (полный исходный код[27] находится в репозитории на GitHub):

```
jaundice[0:N//10] = True
prescription[0:N//10] = 'ibuprofen'
```

Еще 10% данных будут представлять пациентов с язвенной болезнью желудка в анамнезе; поскольку они подвержены риску повреждения желудка, правильным назначением для них будет ацетаминофен:

```
ulcers[(9*N)//10:] = True
prescription[(9*N)//10:] = 'acetaminophen'
```

Остальным пациентам будет назначен любой из этих препаратов произвольно. Естественно, это случайное назначение приведет к тому, что общая точность модели, натренированной только на двух классах, будет низкой. На самом деле мы можем вычислить верхнюю границу точности. Поскольку 80% тренировочных примеров имеют случайные метки, лучшее, что может сделать модель, — это правильно угадывать половину из них. Таким образом, точность на этом подмножестве тренировочных примеров составит 40%. Остальные 20% тренировочных примеров имеют систематические метки, и идеальная модель это усвоит, поэтому мы ожидаем, что совокупная точность может быть в лучшем случае 60%.

И действительно, натренировав модель с помощью scikit-learn следующим ниже образом, мы получим точность 0,56:

```
ntrain = 8*len(df)//10 # 80% данных для тренировки
lm = linear_model.LogisticRegression()
lm = lm.fit(df.loc[:ntrain-1, ['jaundice', 'ulcers']],
            df[label][:ntrain])
acc = lm.score(df.loc[ntrain:, ['jaundice', 'ulcers']],
            df[label][ntrain:])
```

Если мы создадим три класса и поместим все случайно назначенные предписания в этот класс, то, как и ожидалось, получим идеальную (100%-ную) точность. Цель синтетических данных состояла в иллюстрировании того, что в условиях, когда задействуется случайное назначение, паттерн "Нейтральный класс" поможет нам избежать потери модельной точности из-за произвольно размеченных данных.

---

[27] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/03_problem_representation/neutral.ipynb**.

В реальном мире

В реальных ситуациях все может быть не вполне случайным, как в синтетическом наборе данных, но парадигма произвольного назначения по-прежнему сохраняется. Например, сразу после рождения младенца оценивают по шкале Апгар, т. е. фиксируют баллы от 1 до 10, при этом 10 — это младенец, для которого процесс родов прошел идеально.

Рассмотрим модель, которая натренирована предсказывать, что младенец родится здоровым либо что ему потребуется проведение неотложных медицинских мероприятий (полный исходный код[28] находится в репозитории на GitHub):

```
CREATE OR REPLACE MODEL mlpatterns.neutral_2classes
OPTIONS(model_type='logistic_reg', input_label_cols=['health']) AS

SELECT
  IF(apgar_1min >= 9, 'Healthy', 'NeedsAttention') AS health,
  plurality,
  mother_age,
  gestation_weeks,
  ever_born
FROM `bigquery-public-data.samples.natality`
WHERE apgar_1min <= 10
```

Мы ограничиваем балл по шкале Апгар порогом на уровне 9 и трактуем младенцев, у которых балл по шкале Апгар составляет 9 или 10, как здоровых, а младенцев, у которых балл по шкале Апгар составляет 8 или ниже, как требующих внимания. Точность этой двоичной классификационной модели после тренировки на наборе данных о рождаемости и оценивании на отложенных данных составляет 0,56.

Тем не менее назначение балла по шкале Апгар предусматривает ряд относительно субъективных оценок, и вопрос о назначении младенцу 8 или 9 баллов часто сводится к вопросам предпочтения врача. Такие младенцы не являются абсолютно здоровыми и не нуждаются в серьезном медицинском вмешательстве. Что делать, если мы создадим нейтральный класс, который будет содержать эти предельные баллы? Для этого необходимо создать три класса с баллом по шкале Апгар, равным 10, определяемым как здоровый, баллами от 8 до 9, определяемыми как нейтральный, и более низкими баллами, определяемыми как требующий внимания:

```
CREATE OR REPLACE MODEL mlpatterns.neutral_3classes
OPTIONS(model_type='logistic_reg', input_label_cols=['health']) AS
SELECT
  IF(apgar_1min = 10, 'Healthy',
    IF(apgar_1min >= 8, 'Neutral', 'NeedsAttention')) AS health,
  plurality,
  mother_age,
```

[28] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/03_problem_representation/neutral.ipynb**.

```
  gestation_weeks,
  ever_born
FROM `bigquery-public-data.samples.natality`
WHERE apgar_1min <= 10
```

Эта модель достигает точности 0,79 на отложенном оценочном наборе данных, что намного выше, чем точность 0,56, которая была достигнута с двумя классами.

Компромиссы и альтернативы

Паттерн "Нейтральный класс" следует иметь в виду в самом начале решения задачи машинного обучения. Стоит только собрать правильные данные, и мы сможем избежать многих неприятных проблем в будущем. Вот несколько ситуаций, когда наличие нейтрального класса бывает полезным.

Когда эксперты расходятся во мнениях

Нейтральный класс полезен в разрешении разногласий между экспертами. Предположим, у нас есть разметчики-люди, которым мы показываем историю болезни пациента и спрашиваем, какой лекарственный препарат они бы назначили. В некоторых случаях у нас может быть четкий сигнал для ацетаминофена, в других случаях четкий сигнал для ибупрофена и огромное количество случаев, в отношении которых разметчики не согласны. Нейтральный класс обеспечивает подход к решению таких случаев.

В случае разметки, производимой человеком (в отличие от исторического набора фактических действий врача, где пациент наблюдался только одним врачом), каждый образец размечается несколькими экспертами. Поэтому мы априори знаем, в каких случаях люди расходятся во мнениях. Может показаться, что гораздо проще отбросить такие случаи и просто натренировать двоичный классификатор. В конце концов, не имеет значения, что модель делает на нейтральных случаях. Это решение имеет две проблемы:

1. Ложная уверенность стремится повлиять на принятие модели экспертами-людьми. Модель, которая выводит нейтральное определение, часто более приемлема для экспертов, чем модель, которая ошибочно уверена в тех случаях, когда эксперт выбрал бы альтернативу.
2. Если мы тренируем каскад моделей, то нижестоящие модели будут чрезвычайно чувствительны к нейтральным классам. Если мы продолжим совершенствовать эту модель, то нижестоящие модели могут кардинально меняться от версии к версии.

Еще одной альтернативой является использование согласия между разметчиками-людьми в качестве веса образца во время тренировки. Так, если 5 экспертов с диагнозом согласны, то тренировочный образец получает вес 1, а если эксперты разделились в соотношении 3:2, то вес образца может составить только 0,6. Это позволяет нам натренировать двоичный классификатор, но перегрузит классификатор в сторону "верных" случаев. Недостатком этого подхода является то, что при ра-

венстве вероятностного результата модели 0,5 неясно, связано ли это с тем, что он отражает ситуацию, когда было недостаточно тренировочных данных, либо это ситуация, когда эксперты расходятся во мнениях. Использование нейтрального класса для улавливания областей разногласий позволяет нам устранять двусмысленность этих двух ситуаций.

Удовлетворенность клиента

Потребность в нейтральном классе возникает также в моделях, которые пытаются предсказать удовлетворенность клиентов. Если тренировочные данные состоят из ответов на опросы, в которых клиенты оценивают свое впечатление по шкале от 1 до 10, то было бы полезно разделять оценки на три категории: от 1 до 4 — плохо, от 8 до 10 — хорошо и от 5 до 7 — нейтрально. Если вместо этого мы попытаемся натренировать двоичный классификатор, установив порог равным 6, то модель потратит слишком много усилий, пытаясь правильно понять по существу нейтральные ответы.

В качестве подхода к улучшению векторных вложений

Предположим, мы создаем ценовую модель для авиарейсов и хотим предсказывать возможность покупки клиентом билета на рейс по какой-либо цене. Для этого мы можем обратиться к историческим транзакциям приобретения авиабилетов и брошенных корзин покупок. Однако предположим, что многие транзакции также включают покупки со стороны консолидаторов и турагентов — это люди, которые заключили контракты на тарифы, и поэтому тарифы для них фактически не были установлены динамически. Другими словами, они не платят текущую цену.

Мы могли бы отбросить все нединамические покупки и натренировать модель только на клиентах, которые приняли решение покупать или не покупать, основываясь на отображаемой цене. Однако такая модель будет пропускать всю информацию, хранящуюся в пунктах назначения, которые в разное время интересовали консолидатора или турагента, — это повлияет на характер векторного вложения аэропортов и отелей. Один из подходов к удержанию этой информации, не влияя на решение о ценообразовании, — использовать для этих транзакций нейтральный класс.

Переформулировка с использованием нейтрального класса

Предположим, мы тренируем автоматизированную торговую систему, которая совершает сделки, основываясь на том, куда по ее ожиданиям поползет стоимость ценной бумаги: вверх или вниз. Из-за волатильности фондового рынка и скорости, с которой новая информация отражается в ценах акций, попытка торговать на малых предсказываемых взлетах и падениях, вероятно, с течением времени приведет к высоким торговым издержкам и низкой прибыли.

В таких случаях полезно подумать о том, какова конечная цель. Конечная цель модели ML состоит не в том, чтобы предсказывать направление движения акции: вверх или вниз. Мы не сможем купить все акции, которые, по нашим прогнозам, пойдут вверх, и не сможем продать акции, которых у нас нет.

Более совершенной стратегией может быть покупка колл-опционов[29] для 10 акций, которые, скорее всего, вырастут более чем на 5% в течение следующих 6 месяцев, и покупка пут-опционов для акций, которые, скорее всего, упадут более чем на 5% в течение следующих 6 месяцев.

Решение, таким образом, состоит в том, чтобы создать тренировочный набор данных, состоящий из трех классов:

- акции, которые выросли более чем на 5%, — колл-опционы;
- акции, которые упали более чем на 5%, — пут-опционы;
- остальные акции относятся к нейтральной категории.

Вместо того чтобы тренировать регрессионную модель тому, насколько акции будут расти, теперь мы можем натренировать классификационную модель с этими тремя классами и выбирать наиболее уверенные предсказания из нашей модели.

ПАТТЕРН 10. Перебалансировка

Паттерн "Перебалансировка" (Rebalancing) предоставляет разнообразные подходы для манипулирования наборами данных, которые являются несбалансированными. Под несбалансированностью мы подразумеваем наборы данных, в которых один класс составляет большинство набора данных, оставляя гораздо меньше примеров других классов.

В этом паттерне *не* учитываются сценарии, в которых набору данных не хватает достаточной представленности в какой-либо популяции или реальной среде. Подобные проблемы часто можно решить только путем дополнительного сбора данных. Паттерн "Перебалансировка" в первую очередь касается ответа на вопрос: как строить модели с наборами данных, в которых существует мало примеров какого-либо класса или классов?

Постановка задачи

Модели обучаются лучше всего, когда в них подается одинаковое число примеров по каждому классу меток в наборе данных. Однако многие реально прикладные задачи сбалансированы не так четко. Возьмем, к примеру, случай обнаружения мошенничества, когда вы строите модель для выявления мошеннических операций по кредитным картам. Мошеннические операции встречаются гораздо реже, чем обычные, и в силу этого для тренировки модели имеется меньше данных о случаях мошенничества. То же самое относится и к другим задачам, таким как обнаружение невыплаты по кредиту, выявление дефектных продуктов, предсказание наличия заболевания по медицинским снимкам, фильтрация спама, пометка (флагирование) журналов ошибок в программно-информационном приложении и многое другое.

---

[29] См. учебник по колл- и пут-опционам для начинающих на веб-странице **https://oreil.ly/kDndF**.

Несбалансированные наборы данных применимы ко многим типам моделей, включая двоичную классификацию, многоклассовую классификацию, мультиметочную классификацию и регрессию. В регрессионных случаях несбалансированные наборы данных относятся к данным с выбросными значениями, которые намного выше либо ниже медианы в наборе данных.

Распространенной западней в тренировке моделей с несбалансированными классами меток является безосновательная вера в обманчивые значения точности при оценивании модели. Если мы тренируем модель обнаружения мошенничества и только 5% нашего набора данных содержат мошеннические транзакции, то, скорее всего, наша модель натренируется до 95%-ной точности без каких-либо изменений в наборе данных или опорной архитектуре модели. Хотя это число 95%-ной точности технически верно, есть хороший шанс того, что модель просто угадывает мажоритарный класс (в данном случае — не мошенничество) для каждого примера. В силу этого она ничего не усваивает о том, как отличать миноритарный класс от других примеров в наборе данных.

С целью недопущения излишней опоры на это вводящее в заблуждение значение точности стóит посмотреть на матрицу путаницы (confusion matrix), в которой можно увидеть точность по каждому классу. Матрица путаницы для модели со слабой результативностью, натренированной на несбалансированном наборе данных, часто выглядит примерно так, как показано на рис. 3.18.

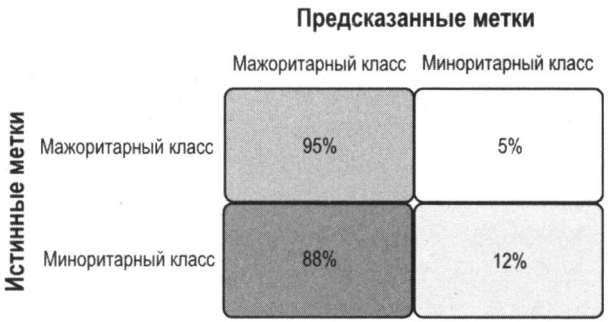


Рис. 3.18. Матрица путаницы для модели, натренированной на несбалансированном наборе данных без корректировки набора данных или модели

В этом примере модель правильно угадывает мажоритарный класс в 95% случаев, но правильно угадывает миноритарный класс только в 12% случаев. Как правило, на диагонали матрицы путаницы для высокорезультативной модели стоят значения, близкие к 100%.

Решение

Прежде всего, поскольку точность на несбалансированных наборах данных может вводить в заблуждение, во время разработки модели важно выбрать надлежащую метрику оценивания. Затем можно задействовать различные технические приемы манипулирования изначально несбалансированными наборами данных как на

уровне набора данных, так и на уровне модели. *Понижающий отбор* изменяет баланс опорного набора данных, а *взвешивание* изменяет то, как модель оперирует некоторыми классами. *Повышающий отбор* дублирует примеры из миноритарного класса и часто предусматривает применение аугментаций[30] для генерирования дополнительных образцов. Мы также взглянем на подходы к *переформулировке* задачи: ее замену регрессионной задачей, анализ значений ошибок модели по каждому примеру либо кластеризацию.

Выбор метрики оценивания

Для несбалансированных наборов данных, таких как в нашем примере обнаружения мошенничества, в качестве метрик лучше всего использовать прецизионность, полноту или *F*-меру, чтобы получить общую картину того, как работает модель. *Прецизионность* (precision) измеряет процент положительных классификаций, которые были правильными из всех сделанных моделью положительных предсказаний. И наоборот, *полнота* (recall) измеряет долю фактических положительных примеров, которые были идентифицированы моделью правильно. Самое большое различие между этими двумя метриками кроется в используемом для их вычисления знаменателе. Для метрики прецизионности знаменателем является суммарное число сделанных моделью предсказаний положительного класса. Для метрики полноты он является числом *фактических* примеров положительных классов, присутствующих в наборе данных.

Идеальная модель имела бы прецизионность и полноту, равные 1,0, но на практике эти две меры часто расходятся друг с другом. *F*-мера (или оценка F1) — это метрика, которая варьируется между 0 до 1 и учитывает как прецизионность, так и полноту. Она рассчитывается следующим образом:

$$2 \cdot (\text{прецизионность} \cdot \text{полнота} / (\text{прецизионность} + \text{полнота})).$$

Давайте вернемся к сценарию обнаружения мошенничества, чтобы увидеть, как каждая из этих метрик работает на практике. Предположим, что в этом примере тестовый набор в общей сложности содержит 1000 примеров, 50 из которых должны быть помечены как мошеннические транзакции. Для этих примеров наша модель правильно предсказывает 930/950 не мошеннических примеров и 15/50 мошеннических примеров. Эти результаты можно наглядно представить на рис. 3.19.

В этом случае прецизионность нашей модели составляет 15/35 (42%), полнота — 15/50 (30%), а *F*-мера — 35%. Эти метрики гораздо лучше справляются со своей работой, улавливая неспособность нашей модели правильно идентифицировать мошеннические транзакции, по сравнению с метрикой точности (accuracy), которая составляет 945/1000 (94,5%). Поэтому в случае моделей, натренированных на несбалансированных наборах данных, предпочтительно использовать метрики, отличные от точности. На самом деле при выполении оптимизации под эти метрики точность может даже снижаться, и это нормально, поскольку прецизионность, пол-

[30] Аугментация (augmentation) — это пропорциональное расширение или увеличение каких-либо характеристик данных по отношению к изначальным данным. — *Прим. перев.*

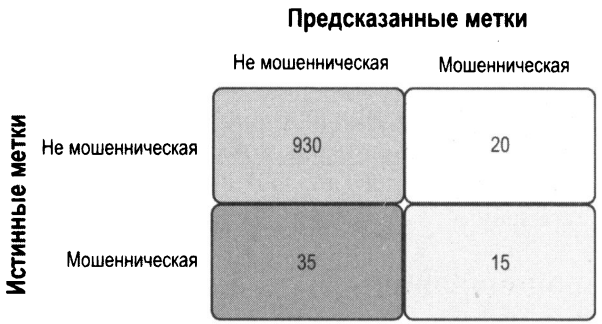


Рис. 3.19. Образцы предсказаний для модели обнаружения мошенничества

нота и F-мера в данном случае являются более качественными индикаторами результативности модели.

Обратите внимание на то, что при оценивании моделей, натренированных на несбалансированных наборах данных, мы не должны использовать выборочные данные во время калькуляции метрик успеха. Это означает, что независимо от того, как мы модифицируем наш набор данных для тренировки в соответствии с техническими решениями, которые мы опишем далее, мы должны оставлять тестовый набор как есть, чтобы он обеспечивал точное представление изначального набора данных. Другими словами, тестовый набор должен иметь примерно такой же баланс классов, как и изначальный набор данных. В приведенном выше примере это будет 5% мошенничества/95% не мошенничества.

Если мы ищем метрику, которая улавливает результативность модели по всем пороговым значениям, то метрика усредненной прецизионности-полноты является более информативной[31] для модельного оценивания, чем площадь под ROC-кривой (area under the ROC curve, AUC). Это обусловлено тем, что усредненная прецизионность-полнота делает больший акцент на том, сколько предсказаний модель сделала правильно из суммарного числа, которое она отнесла к положительному классу. Она придает больший вес положительному классу, что важно для несбалансированных наборов данных. Метрика AUC, с другой стороны, трактует оба класса одинаково и менее чувствительна к улучшениям модели, что не является оптимальным в ситуациях с несбалансированными данными.

Понижающий отбор

Понижающий отбор (downsampling) — это техническое решение для манипулирования несбалансированными наборами данных путем изменения не модели, а лежащего в основе набора данных. С помощью понижающего отбора мы уменьшаем число примеров из мажоритарного класса, используемых во время тренировки модели[32]. В целях ознакомления с тем, как он работает, давайте взглянем на синтети-

[31] См. **https://oreil.ly/5iJX2**.

[32] То есть используем меньше примеров с преобладающим классом. — *Прим. перев.*

ческий набор данных обнаружения мошенничества из Kaggle[33, 34]. Каждый пример в наборе данных содержит разнообразную информацию о транзакции, включая тип транзакции, сумму транзакции и баланс счета как до совершения транзакции, так и после нее. Набор данных содержит 6,3 млн примеров, из которых только 8 тыс. являются мошенническими транзакциями. Это число составляет всего лишь 0,1% всего набора данных.

Хотя крупный набор данных нередко улучшает способность модели выявлять закономерности, он менее полезен, когда данные значительно несбалансированы. Если мы натренируем модель[35] на всем этом наборе данных (6,3 млн строк) без каких-либо модификаций, то, скорее всего, увидим вводящую в заблуждение точность 99,9% как следствие того, что модель каждый раз угадывает не мошеннический класс случайно. Мы можем решить эту проблему, удалив из набора данных большой кусок мажоритарного класса.

Мы возьмем все 8 тыс. мошеннических примеров и отложим их в сторону, чтобы использовать во время тренировки модели. Затем возьмем небольшую случайную выборку не мошеннических транзакций. Далее мы объединим ее с 8 тыс. мошеннических примерами, перетасуем данные и применим этот новый, меньший набор данных для тренировки модели. Вот как мы могли бы реализовать это с помощью библиотеки pandas:

```
data = pd.read_csv('fraud_data.csv')

# Разбить на отдельные кадры данных для мошеннических/не мошеннических транзакций
fraud = data[data['isFraud'] == 1]
not_fraud = data[data['isFraud'] == 0]

# Взять случайную выборку не мошеннических строк
not_fraud_sample = not_fraud.sample(random_state=2, frac=.005)

# Положить их обратно вместе и перетасовать
df = pd.concat([not_fraud_sample,fraud])
df = shuffle(df, random_state=2)
```

После этого наш набор данных будет содержать 25% мошеннических транзакций, являясь гораздо более сбалансированным, чем изначальный набор данных, в котором только 0,1% приходится на миноритарный класс. При понижающем отборе неплохо поэкспериментировать с точным используемым балансом. Здесь мы приме-

---

[33] См. **https://oreil.ly/WqUM**.

[34] Указанный набор данных был создан на основе исследования, предложенного в работе Эдгара Лопес-Рохаса, Ахмада Эльмира и Стефана Аксельссона "PaySim: финансовый мобильный денежный симулятор для обнаружения мошенничества" (Lopez-Rojas E., Elmir A., Axelsson S. PaySim: a financial mobile money simulator for fraud detection // 28th European Modeling and Simulation Symposium, EMSS, Larnaca, Cyprus (2016). — P. 249–255).

[35] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/03_problem_representation/rebalancing.ipynb**.

няли разбивку 25/75, но для достижения приличной точности может потребоваться разбивка, более близкая к 50/50.

Понижающий отбор обычно сочетают с паттерном "Ансамбли", соблюдая следующие шаги:

1. Подвергнуть мажоритарный класс понижающему отбору и использовать все экземпляры миноритарного класса.
2. Натренировать модель и добавить ее в ансамбль.
3. Повторить.

Во время предсказательного вывода следует брать медианное значение на выходе из ансамблевых моделей.

Мы рассмотрели здесь классификационный пример, но понижающий отбор также может применяться и к регрессионным моделям, в которых мы предсказываем числовое значение. В этом случае взятие случайной выборки из мажоритарного класса будет более точным, поскольку мажоритарный "класс" в наших данных включает в себя не одиночную метку, а диапазон значений.

Взвешенные классы

Еще один подход к манипулированию несбалансированными наборами данных заключается в изменении *веса*, который наша модель придает примерам из каждого класса. Обратите внимание, что здесь термин "вес" используется в другом контексте, чем веса (или параметры), усваиваемые моделью во время тренировки, которые вы не можете устанавливать вручную. Взвешивая классы, мы говорим нашей модели относиться к специфическим классам меток во время тренировки с большей важностью. Мы хотим, чтобы наша модель придавала больший вес примерам из миноритарного класса. Весовое значение, которое ваша модель должна придавать тем или иным примерам, зависит только от вас, и это тот параметр, с которым вы можете поэкспериментировать.

В Keras мы можем передать параметр `class_weights` в модель, когда тренируем ее с помощью метода `fit()`. Параметр `class_weights` является словарем `dict` с соотнесением каждого класса с весом, который Keras должен назначать примерам из этого класса. Но как определить точные веса для каждого класса? Весовые значения класса должны относиться к балансу каждого класса в наборе данных. Например, если миноритарный класс составляет только 0,1% набора данных, то разумно заключить, что наша модель должна трактовать примеры из этого класса как имеющие 1000-кратно больше веса, чем из мажоритарного класса. На практике принято делить это весовое значение на 2 по каждому классу, вследствие чего средний вес примера составлет 1,0. Следовательно, имея набор данных, в котором 0,1% значений представляет миноритарный класс, мы могли бы вычислить веса классов с помощью следующего фрагмента исходного кода:

```
num_minority_examples = 1
num_majority_examples = 999
total_examples = num_minority_examples + num_majority_examples
```

```
minority_class_weight = 1/(num_minority_examples/total_examples)/2
majority_class_weight = 1/(num_majority_examples/total_examples)/2

# Передать веса в Keras в словаре dict.
# Ключом является индекс каждого класса.
keras_class_weights = {0: majority_class_weight, 1: minority_class_weight}
```

Затем мы передаем эти веса в модель во время тренировки:

```
model.fit(
    train_data,
    train_labels,
    class_weight=keras_class_weights
)
```

В BigQuery ML мы можем установить `AUTO_CLASS_WEIGHTS = True` в блоке `OPTIONS` при создании модели, чтобы разные классы взвешивались на основе их частоты встречаемости в тренировочных данных.

Хотя бывает полезно следовать эвристике баланса классов для установки весов классов, деловое приложение модели также вполне может предписывать веса классов, которые мы выбираем для назначения. Допустим, у нас есть модель, которая классифицирует снимки дефектных продуктов. Если стоимость доставки дефектного продукта в 10 раз превышает стоимость неправильного классифицирования нормального продукта, то в качестве веса нашего миноритарного класса мы выбрали бы 10.

Смещенность выходного слоя

В сочетании с назначением весов классов также полезно инициализировать выходной слой модели смещенностью, для того чтобы учитывать дисбаланс набора данных. Почему возникает потребность вручную устанавливать начальную смещенность в выходном слое? Когда у нас есть несбалансированные наборы данных, установка выходной смещенности будет помогать модели быстрее сходиться. Это обусловлено тем, что смещенность последнего (предсказывающего) слоя натренированной модели будет выводить в среднем логарифм отношения миноритарных примеров к мажоритарным в наборе данных. После установки смещенности, модель уже начинает с "правильного" значения, без необходимости его обнаруживать с помощью градиентного спуска.

По умолчанию в Keras используется смещенность, равная нулю. Это соответствует смещенности, которую мы хотели бы использовать для идеально сбалансированного набора данных, где log(1/1) = 0. Для вычисления правильной смещенности с учетом баланса набора данных следует использовать:

```
bias = log(num_minority_examples / num_majority_examples)
```

Повышающий отбор

Еще одним распространенным техническим приемом манипулирования несбалансированными наборами данных является *повышающий отбор* (upsampling). В этом

случае мы увеличиваем представительство миноритарного класса, одновременно реплицируя примеры миноритарного класса и генерируя дополнительные синтетические примеры[36]. Он нередко выполняется в сочетании с понижающим отбором мажоритарного класса. Такой подход с сочетанием понижающего и повышающего отборов был предложен в 2002 году и получил название "Техника избыточного отбора синтетического меньшинства" (Synthetic Minority Over-sampling Technique, SMOTE[37]). Метод SMOTE предоставляет алгоритм, который строит эти синтетические примеры, анализируя признаковое пространство примеров миноритарных классов в наборе данных, а затем генерирует аналогичные примеры в этом признаковом пространстве, используя подход, основанный на ближайших соседях. В зависимости от числа похожих точек данных, которые мы решаем рассматривать одновременно (также именуемого числом ближайших соседей), подход на основе SMOTE случайно генерирует новый пример миноритарного класса между этими точками.

Давайте посмотрим на набор данных Pima Indian Diabetes Dataset[38] о заболеваемости сахарным диабетом в племени Пима, чтобы увидеть, как это работает на высоком уровне. 34% этого набора данных содержат примеры пациентов, страдавших сахарным диабетом, поэтому мы будем считать этот класс миноритарным. В табл. 3.3 показано подмножество столбцов для двух примеров миноритарного класса.

Таблица 3.3. *Подмножество признаков для двух тренировочных примеров из миноритарного класса (имеет сахарный диабет) в наборе данных о заболеваемости сахарным диабетом в племени Пима, Индия*

Глюкоза	Артериальное давление	Толщина кожи	Индекс массы тела
148	72	35	33,6
183	64	0	23,3

Новый синтетический пример, основанный на этих двух фактических примерах из набора данных, может выглядеть, как в табл. 3.4, рассчитанный по срединной точке между каждым значением столбца.

Таблица 3.4. *Синтетический пример, созданный на основе двух миноритарных тренировочных примеров с использованием метода SMOTE*

Глюкоза	Артериальное давление	Толщина кожи	Индекс массы тела
165,5	68	17,5	28,4

Технический прием SMOTE в первую очередь касается табличных данных, но аналогичная логика может быть применена и к наборам снимковых данных. Например,

---

[36] То есть используем больше примеров с редкими классами, по мере необходимости задействуя бутстрапирование. — *Прим. перев.*

[37] См. **https://oreil.ly/CFJPz**.

[38] См. **https://oreil.ly/ljqnc**.

если мы строим модель, которая позволит нам различать бенгальских и сиамских кошек, и только 10% нашего набора данных содержат снимки бенгальских кошек, то мы можем сгенерировать дополнительные вариации бенгальских кошек в нашем наборе данных путем аугментации снимка, воспользовавшись классом `ImageDataGenerator` в Keras. Получив значения для нескольких параметров, этот класс будет генерировать несколько вариаций одного и того же изображения, вращая, обрезая, регулируя яркость и выполняя другие модификации.

Компромиссы и альтернативы

Для строительства моделей с изначально несбалансированными наборами данных существует несколько других альтернативных решений, включая переформулировку задачи и манипулирование случаями обнаружения аномалий. Мы также обсудим несколько важных соображений для несбалансированных наборов данных: совокупный размер набора данных, оптимальные архитектуры моделей для разных типов задач и объяснение логики предсказания миноритарного класса.

Переформулировка и каскад

Переформулировка задачи — еще один подход к работе с несбалансированными наборами данных. Сначала мы могли бы подумать о возможности переключения с классификационной задачи на регрессионную или наоборот, задействовав технические приемы, описанные в *разд. "Паттерн 5. Переформулировка" главы 3*, и натренировав каскад моделей. Например, предположим, что у нас есть регрессионная задача, в которой большинство тренировочных данных попадает в отдельно взятый диапазон, с несколькими выбросами. Допустим, что нас интересует предсказание выбросных значений. Тогда мы могли бы преобразовать эту задачу в классификационную, поместив большинство данных в одну корзину, а выбросы — в другую.

Представьте, что мы строим модель для предсказания веса младенца, используя набор натальных данных BigQuery. С помощью библиотеки pandas мы можем создать гистограмму выборки из данных о весе младенцев, чтобы увидеть распределение веса:

```
%%bigquerydf
SELECT
  weight_pounds
FROM
  `bigquery-public-data.samples.natality`
LIMIT 10000

df.plot(kind='hist')
```

На рис. 3.20 показана результирующая гистограмма.

Если мы подсчитаем число младенцев весом 3 фунта (1,36 кг) во всем наборе данных, то их будет примерно 96 тыс. (0,06% данных). Младенцы весом 12 фунтов (5,44 кг) составляют только 0,05% набора данных. В целях получения хорошей

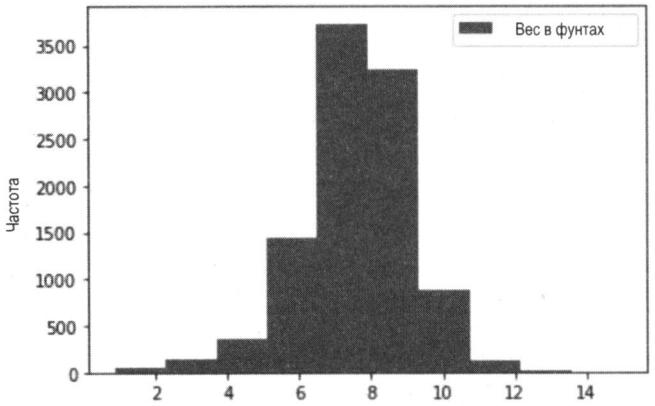


Рис. 3.20. Гистограмма, изображающая распределение веса младенцев для 10 тыс. примеров в наборе натальных данных BigQuery

результативности регрессии по всему диапазону мы можем объединить понижающий отбор с паттернами "Переформулировка" и "Каскад". Сначала мы разобьем данные на три группы: "недостаточный вес", "средний вес" и "избыточный вес". Это можно сделать с помощью следующего запроса:

```
SELECT
  CASE
    WHEN weight_pounds < 5.5 THEN "underweight"
    WHEN weight_pounds > 9.5 THEN "overweight"
  ELSE
  "average"
END
  AS weight,
  COUNT(*) AS num_examples,
  round(count(*) / sum(count(*)) over(), 4) as percent_of_dataset
FROM
  `bigquery-public-data.samples.natality`
GROUP BY
  1
```

Результаты приведены в табл. 3.5.

***Таблица 3.5.** Процент каждой весовой категории, присутствующей в наборе натальных данных*

Вес	num_examples	percent_of_dataset
Средний	123781044	0.8981
Недостаточный	9649724	0.07
Избыточный	4395995	0.0319

Для демонстрационных целей мы возьмем 100 тыс. примеров из каждого класса, чтобы натренировать модель на обновленном сбалансированном наборе данных:

```
SELECT
  is_male,
  gestation_weeks,
  mother_age,
  weight_pounds,
  weight
FROM (
  SELECT
    *,
    ROW_NUMBER() OVER (PARTITION BY weight ORDER BY RAND()) AS row_num
  FROM (
    SELECT
      is_male,
      gestation_weeks,
      mother_age,
      weight_pounds,
      CASE
        WHEN weight_pounds < 5.5 THEN "underweight"
        WHEN weight_pounds > 9.5 THEN "overweight"
      ELSE
      "average"
    END
      AS weight,
    FROM
      `bigquery-public-data.samples.natality`
    LIMIT
      4000000) )
WHERE
  row_num < 100000
```

Мы можем сохранить результаты этого запроса в таблице и, имея более сбалансированный набор данных, теперь можем натренировать классификационную модель для назначения младенцам меток "недостаточный вес", "средний вес" или "избыточный вес":

```
CREATE OR REPLACE MODEL
  `project.dataset.baby_weight_classification` OPTIONS(model_type='logistic_reg',
    input_label_cols=['weight']) AS
SELECT
  is_male,
  weight_pounds,
  mother_age,
  gestation_weeks,
  weight
FROM
  `project.dataset.baby_weight`
```

Еще один подход заключается в использовании паттерна "Каскад" для тренировки трех отдельных регрессионных моделей для каждого класса. Затем мы можем при-

менить наше техническое решение, передавая первоначальной классификационной модели пример и используя результат классифицирования, чтобы принимать решение о том, в какую регрессионную модель отправлять этот пример для численного предсказания.

Обнаружение аномалий

Для несбалансированных наборов данных существуют два подхода к манипулированию регрессионными моделями:

- использовать ошибку модели на предсказании в качестве сигнала;
- выполнять кластеризацию входящих данных и сравнивать расстояние от каждой новой точки данных до существующих кластеров.

В целях более глубокого понимания каждого технического решения предположим, что мы тренируем модель на собранных датчиком данных, чтобы предсказывать температуру. В этом случае нам нужно, чтобы результат работы модели был числовым значением.

В случае первого подхода — использование ошибки в качестве сигнала — после тренировки модели мы бы сравнивали предсказанное моделью значение с фактическим значением для текущей временно́й точки. Если бы имелась существенная разница между предсказанным и фактическим текущим значениями, то мы могли бы отметить входящую точку данных как аномальную. Разумеется, для этого требуется модель, натренированная с хорошей точностью на достаточном количестве исторических данных, чтобы мы могли полагаться на ее качество в отношении будущих предсказаний. Главное предостережение для этого подхода состоит в том, что он требует от нас наличия новых легкодоступных данных, благодаря которым мы получаем возможность сравнивать поступающие данные с модельным предсказанием. Как следствие, он лучше всего подходит для задач с участием потоковой передачи данных или данных временно́го ряда.

Во втором подходе — кластеризация данных — мы начинаем со строительства модели с помощью кластеризационного алгоритма, технического приема моделирования, который организует наши данные в кластеры. Кластеризация — это метод *неконтролируемого обучения*, означающий, что он занимается поиском закономерностей в наборе данных, ничего не зная о метках эмпирических наблюдений. Распространенным алгоритмом кластеризации является алгоритм *k* средних, который можно реализовать с помощью BigQuery ML. Приведенный ниже фрагмент исходного кода демонстрирует процесс тренировки модели, основанной на *k* средних, на наборе натальных данных BigQuery с использованием трех признаков:

```
CREATE OR REPLACE MODEL
  `project-name.dataset-name.baby_weight` OPTIONS(model_type='kmeans',
    num_clusters=4) AS
SELECT
  weight_pounds,
  mother_age,
  gestation_weeks
```

```
FROM
  `bigquery-public-data.samples.natality`
LIMIT 10000
```

Результирующая модель соберет наши данные в четыре группы. После создания модели мы сможем генерировать предсказания на новых данных и смотреть на удаленность каждого предсказания от существующих кластеров. Если удаленность будет большой, мы сможем отмечать точку данных как аномальную. В целях генерирования кластерного предсказания на нашей модели мы можем выполнить следующий ниже запрос, передав ему вымышленный усредненный пример из набора данных:

```
SELECT
  *
FROM
  ML.PREDICT (MODEL `project-name.dataset-name.baby_weight`,
    (
    SELECT
      7.0 as weight_pounds,
      28 as mother_age,
      40 as gestation_weeks
  )
)
```

Результаты запроса в табл. 3.6 показывают расстояние между этой точкой данных и сгенерированными моделью кластерами, именуемыми центроидами.

Таблица 3.6. *Расстояние между примером средневесовой точки данных и каждым кластером, сгенерированным моделью, основанной на k средних*

CENTROID_ID	NEAREST_CENTROIDS_DISTANCE.CENTROID_ID	NEAREST_CENTROIDS_DISTANCE.DISTANCE
4	4	0,29998627812137374
1	1,2370167418282159	
2	1,376651161584178	
3	1,6853517159990536	

Как видно из малого расстояния (0,29), этот пример четко вписывается в центроид 4.

Указанный результат можно сравнить с результатами, которые мы получим, если отправим в модель выброс, т. е. пример с недостаточным весом (табл. 3.7).

Здесь расстояние между этим примером и каждым центроидом довольно велико. Мы могли бы использовать эти значения больших расстояний, чтобы сделать вывод о том, что рассматриваемая точка данных может быть аномальной. Такой подход на основе неконтролируемой кластеризации особенно полезен, если мы заранее не знаем меток данных. После того как мы сгенерировали кластерные предсказания на достаточном количестве примеров, мы бы могли построить модель контролируемого обучения, используя предсказанные кластеры в качестве меток.

Таблица 3.7. Расстояние между примером точки данных с недостаточным весом и каждым кластером, сгенерированным моделью, основанной на k средних

CENTROID_ID	NEAREST_CENTROIDS_DISTANCE.CENTROID_ID	NEAREST_CENTROIDS_DISTANCE.DISTANCE
3	3	3,061985789261998
4	3,3124603501734966	
2	4,330205096751425	
1	4,658614918595627	

Число примеров миноритарного класса

Хотя миноритарный класс в первом примере обнаружения мошенничества составлял всего 0,1% данных, набор данных был крупным настолько, что для работы у нас все же было 8 тыс. мошеннических точек данных. В случае наборов данных с еще меньшим числом примеров миноритарного класса понижающий отбор может сделать результирующий набор данных слишком малым, для того чтобы модель могла на нем учиться. Жесткого и быстрого правила определения числа примеров, которое будет считаться слишком малым, чтобы использовать понижающий отбор, не существует, поскольку всё в значительной степени зависит от нашей задачи и архитектуры модели. Общее эмпирическое правило состоит в том, что если у вас лишь сотни примеров миноритарного класса, то для манипулирования дисбалансом набора данных можно рассмотреть техническое решение, которое отличается от понижающего отбора.

Также стоит отметить, что естественным эффектом удаления подмножества мажоритарного класса является потеря хранящейся в этих примерах некоторой информации. Данная потеря может немного снизить способность модели выявлять мажоритарный класс, но часто выгоды от понижающего отбора все же этот эффект перевешивают.

Сочетание разных технических приемов

Описанные выше технические приемы понижающего отбора и взвешивания классов можно совместить в целях достижения оптимальных результатов. Для этого следует начать с понижающего отбора из данных, пока не будет найден баланс, который хорошо работает в нашем варианте использования. Затем, основываясь на соотношениях меток для перебалансированного набора данных, нужно использовать метод, описанный в *разд. "Взвешенные классы" ранее в этой главе*, чтобы передать в модель новые веса. Сочетание этих подходов может быть особенно полезным, когда мы сталкиваемся с задачей обнаружения аномалий и больше всего интересуемся предсказаниями для миноритарного класса. Например, если мы строим модель обнаружения мошенничества, то мы, вероятно, гораздо больше заинтересованы в транзакциях, которые модель помечает как "мошенническая", чем в тех, которые она помечает как "не мошенническая". В добавление к этому, как упоминается в SMOTE, подход по генерированию синтетических примеров из миноритар-

ного класса нередко сочетается с удалением случайной выборки примеров из мажоритарного класса.

Понижающий отбор также часто сочетается с паттерном "Ансамбли". Используя этот подход, вместо полного удаления случайной выборки из мажоритарного класса мы используем разные его подмножества для тренировки нескольких моделей, а затем совмещаем эти модели. В целях иллюстрации предположим, что у нас есть набор данных со 100 примерами миноритарного класса и 1000 примерами мажоритарного класса. Вместо удаления 900 примеров из мажоритарного класса мы бы разбили мажоритарные примеры случайно на 10 групп по 100 примеров в каждой, чтобы идеально сбалансировать набор данных. Затем мы бы натренировали 10 классификаторов, каждый с теми же 100 примерами из миноритарного класса и 100 разными, случайно отобранными значениями из мажоритарного класса. Проиллюстрированный на рис. 3.11 технический прием бэггинга хорошо вписывается в этот подход.

В дополнение к сочетанию подходов, которые центрированы на данных, мы также можем корректировать порог классификатора, чтобы выполнять оптимизацию под метрики прецизионности или полноты в зависимости от варианта использования. Если бы мы были больше заинтересованы в том, чтобы модель была правильной всякий раз, когда она делает предсказание положительного класса, мы бы оптимизировали предсказательный порог под метрику прецизионности. Это применимо в любой ситуации, когда мы хотим избежать ложных утверждений о данных (ложноположительных результатов). Если же дороже обходится упущение потенциальной положительной классификации, даже если мы можем ошибаться, то мы оптимизируем нашу модель под метрику полноты.

Выбор архитектуры модели

В зависимости от задачи предсказания существуют разные архитектуры, которые следует учитывать при решении задач с помощью паттерна "Перебалансировка". Если мы работаем с табличными данными и строим классификационную модель для обнаружения аномалий, то, как иллюстрирует одна научно-исследовательская работа[39], модели, построенные на деревьях решений, показывают хорошую результативность на этих типах задач. Древесные модели также хорошо работают на задачах, связанных с малыми и несбалансированными наборами данных. XGBoost, scikit-learn и TensorFlow имеют соответствующие методы для реализации моделей, основанных на деревьях решений.

Мы можем создать двоичный классификатор в XGBoost, написав следующий фрагмент исходного кода:

```
# Построить модель
model = xgb.XGBClassifier(
    objective='binary:logistic'
)
```

[39] См. **https://oreil.ly/EnAab**.

```
# Натренировать модель
model.fit(
    train_data,
    train_labels
)
```

Понижающий отбор и веса классов могут использоваться в каждом из этих фреймворков для дальнейшей оптимизации модели с использованием паттерна "Перебалансировка". Например, в целях добавления взвешенных классов в приведенный выше классификатор XGBClassifier мы добавим параметр `scale_pos_weight`, рассчитанный на основе баланса классов в наборе данных.

Если мы занимаемся обнаружением аномалий в данных временно́го ряда, то модели на основе долгой кратковременной памяти (long short-term memory, LSTM) хорошо работают для выявления закономерностей, присутствующих в последовательностях. Кластеризационные модели являются неплохим вариантом для табличных данных с несбалансированными классами. В случае несбалансированных наборов данных с подачей на вход снимков следует использовать архитектуры глубокого обучения с понижающим отбором, взвешенными классами, повышающим отбором или комбинацией этих технических приемов. Однако, что касается текстовых данных, в этом случае генерировать синтетические данные не так просто[40], и лучше всего опираться на понижающий отбор и взвешенные классы.

Независимо от того, с какой модальностью данных мы работаем, полезно поэкспериментировать с разными архитектурами, чтобы увидеть, какая из них работает на несбалансированных данных лучше всего.

Важность объяснимости

Во время строительства моделей для выявления редких случаев в данных, таких как аномалии, особенно важно понимать, каким образом модель делает предсказания. Это помогает получать подтверждения о том, что модель в своих предсказаниях действует на правильных сигналах, а также объяснять поведение модели конечным пользователям. Существует несколько инструментов, которые помогают интерпретировать модели и объяснять предсказания, включая фреймворк с открытым исходным кодом SHAP[41], инструмент What-If[42] и инструментарий Explainable AI в составе инфраструктуры Google Cloud[43].

Объяснения моделей могут принимать различные формы, одна из которых называется *атрибуционными значениями*. Атрибуционные значения говорят нам о том, насколько каждый признак в модели повлиял на модельное предсказание. Положительные атрибуции означают, что тот или иной признак подтолкнул модельное

---

[40] См. **https://oreil.ly/2ai2k**.

[41] См. **https://github.com/slundberg/shap**.

[42] См. **https://oreil.ly/Vf3D**.

[43] См. **https://oreil.ly/lDocn**.

предсказание вверх, а отрицательные атрибуции означают, что этот признак подтолкнул модельное предсказание вниз. Чем выше абсолютное атрибуционное значение, тем большее влияние оно оказывает на модельное предсказание. В снимковых и текстовых моделях атрибуции могут указывать на пикселы или слова, которые больше всего сигнализировали о предсказании. В табличных моделях атрибуции предоставляют числовые значения по каждому признаку, свидетельствуя о его совокупном влиянии на модельное предсказание.

После тренировки модели в TensorFlow на синтетическом наборе данных Kaggle по обнаружению мошенничества и ее развертывания в Explainable AI облачной инфраструктуры Google Cloud давайте взглянем на несколько примеров атрибуций уровня экземпляров. На рис. 3.21 представлены два примера транзакций, которые наша модель правильно определила как мошеннические, вместе с их признаковыми атрибуциями.

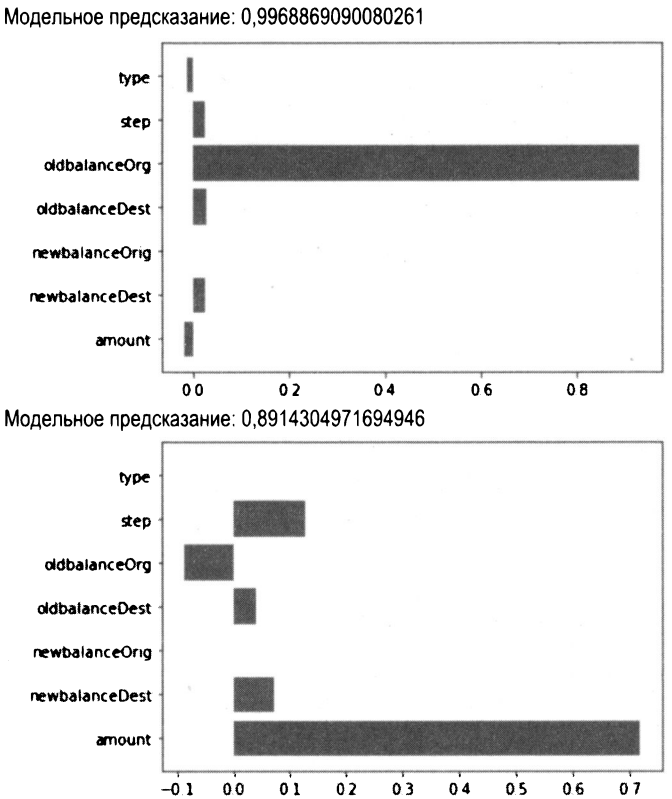


Рис. 3.21. Признаковые атрибуции, полученные из инструмента Explainable AI для двух правильно классифицированных мошеннических транзакций

В первом примере, где модель предсказала 99%-ную вероятность мошенничества, старый остаток на изначальном счете до совершения транзакции был самым большим показателем мошенничества. Во втором примере модель была уверена в своем предсказании мошенничества на 89%, где сумма сделки была идентифицирована

как самый большой сигнал мошенничества. Однако остаток на изначальном счете сделал модель менее уверенной в своем предсказании мошенничества и объясняет, почему предсказательная уверенность чуть ниже на 10 процентных пунктов.

Объяснения важны для любого типа модели машинного обучения, но мы видим их особую полезность для моделей, которые следуют паттерну "Перебалансировка". Когда мы имеем дело с несбалансированными данными, важно заглядывать за пределы метрик точности и ошибки модели в той части, которая касается подтверждения того, что она действует на содержательных сигналах в наших данных.

Резюме

Эта глава была посвящена разным подходам к представлению задачи предсказания через призму архитектуры модели и результата. Рассуждая о том, как вы будете применять свою модель, вы помогаете себе принимать правильное решение относительно типа создаваемой модели и о том, как форматировать результат предсказания. Итак, мы начали с паттерна "Переформулировка" (Reframing), который описывает подходы к замене вашей задачи с регрессионной на классификационную (или наоборот) с целью улучшения качества модели. Это делается за счет переформатирования столбца метки в данных. Далее мы описали паттерн "Мультиметка" (Multilabel), который обращается к случаям, когда подаваемая на вход модели переменная может ассоциироваться с более чем одной меткой. В этом случае следует использовать сигмоидную активационную функцию на выходном слое с двоичной перекрестно-энтропийной потерей.

В то время как паттерны "Переформулировка" и "Мультиметка" сконцентрированы на форматировании результата работы модели, паттерн "Ансамбли" (Ensembles) обращается к архитектуре и предусматривает различные методы сочетания нескольких моделей с целью улучшения результатов машинного обучения из одной модели. В частности, паттерн "Ансамбли" включает в себя бэггинг (модельное усреднение), бустинг (модельное усиление) и стэкинг (модельное наложение) — все эти совершенно разные технические приемы служат для агрегирования многочисленных моделей в одну систему. Паттерн "Каскад" (Cascade) также является подходом на уровне модели и предусматривает подразделение задачи машинного обучения на несколько более мелких задач. В отличие от ансамблевых моделей, паттерн "Каскад" требует, чтобы результаты на выходе из первоначальной модели были входами в нижестоящие модели. Из-за сложности, которую могут создавать каскадные модели, их следует использовать только тогда, когда у вас есть сценарий, в котором первоначальные классификационные метки хаотичны и одинаково важны.

Далее мы рассмотрели паттерн "Нейтральный класс" (Neutral Class), который решает задачу представления на выходном уровне. Указанный паттерн совершенствует двоичный классификатор, добавляя третий "нейтральный" класс. Это полезно в тех случаях, когда вы хотите улавливать произвольные или менее поляризующие классификации, которые не попадают ни в одну из отличимых двоичных категорий. Наконец, паттерн "Перебалансировка" (Rebalancing) предоставляет технические

решения для случаев, когда у вас есть изначально несбалансированный набор данных. Указанный паттерн предлагает использовать понижающий отбор, взвешенные классы или специальные технические приемы переформулировки задачи с целью отыскания решений для наборов данных с несбалансированными классами меток.

Главы 2 и *3* посвящены начальным шагам структурирования задачи машинного обучения, в частности форматированию входных данных, параметрам архитектуры модели и представлению результатов модели на выходе. В следующей главе мы перейдем к очередному шагу рабочего потока процессов машинного обучения — паттернам для тренировки моделей.

ГЛАВА 4

Паттерны для тренировки моделей

Тренировка моделей машинного обучения обычно выполняется итеративно, и этот итеративный процесс неофициально называется *циклом тренировки*. В данной главе мы обсудим внешний вид типичного цикла тренировки и приведем каталог ситуаций, в которых вы, возможно, захотите делать что-то другое.

Типичный цикл тренировки

Модели машинного обучения могут тренироваться с использованием разных типов оптимизации. Деревья решений нередко строятся в поузловом порядке на основе меры прироста информации. В генетических алгоритмах параметры модели представлены в виде генов, и метод оптимизации предусматривает участие технических приемов, которые основываются на теории эволюции. Однако наиболее распространенным подходом к определению параметров моделей машинного обучения является *градиентный спуск*.

Стохастический градиентный спуск

На крупных наборах данных градиентный спуск применяется к мини-пакетам входных данных для тренировки чего угодно, начиная от линейных моделей и бустированных деревьев до глубоких нейронных сетей (deep neural networks, DNN) и опорно-векторных машин (support vector machines, SVM). Он называется *стохастическим градиентным спуском* (stochastic gradient descent, SGD), а расширения стохастического градиентного спуска (такие как Adam и Adagrad) являются де-факто оптимизаторами, используемыми в новейших фреймворках машинного обучения.

Поскольку стохастический градиентный спуск предписывет, чтобы тренировка осуществлялась итеративно на малых пакетах тренировочного набора данных, тренировка модели машинного обучения происходит в цикле. Стохастический градиентный спуск отыскивает минимум, но не является решением в замкнутой форме, и поэтому мы должны определять момент, когда произошло схождение модели. По этой причине следует выполнять мониторинг ошибки (именуемой *потерей*) на тренировочном наборе данных. Переобучение может возникать, если модельная сложность выше, чем это допускается размером и охватом набора данных. К сожалению, узнать о том, что сложность модели является слишком высокой для того или

иного набора данных, невозможно до тех пор, пока модель не будет натренирована на этом наборе данных фактически. Следовательно, оценивание должно выполняться внутри цикла тренировки, и *метрики ошибки* на отложенном срезе тренировочных данных, именуемом *валидационным набором данных*, также должны мониторироваться. Поскольку тренировочный и валидационный наборы данных использовались в цикле тренировки, необходимо откладывать еще одну часть тренировочного набора данных, именуемую *тестовым набором данных*, призванную сообщать фактические метрики ошибки, которые можно было бы ожидать на новых и ранее не встречавшихся данных. Это оценивание проводится в конце.

Цикл тренировки в Keras

Типичный цикл тренировки в Keras выглядит так:

```
model = keras.Model(...)
model.compile(optimizer=keras.optimizers.Adam(),
              loss=keras.losses.categorical_crossentropy(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=64,
                    epochs=3,
                    validation_data=(x_val, y_val))

results = model.evaluate(x_test, y_test, batch_size=128))
model.save(...)
```

Здесь в модели используется оптимизатор Adam для выполнения стохастического градиентного спуска на перекрестной энтропии над тренировочным набором данных, который сообщает окончательную точность, полученную на тестовом наборе данных. Процедура подгонки модели прокручивает в цикле тренировочный набор данных 3 раза (каждый обход тренировочного набора данных называется *эпохой*), при этом модель каждый раз видит пакеты, состоящие из 64 тренировочных примеров. В конце каждой эпохи метрики ошибки вычисляются на проверочном наборе данных, а затем добавляются в историю. В конце цикла подгонки модель оценивается на тестовом наборе данных, сохраняется и потенциально развертывается в качестве сервиса (рис. 4.1).

Вместо использования предварительно построенной функции `fit()` мы могли бы также написать пользовательский цикл тренировки, который прокручивает пакеты

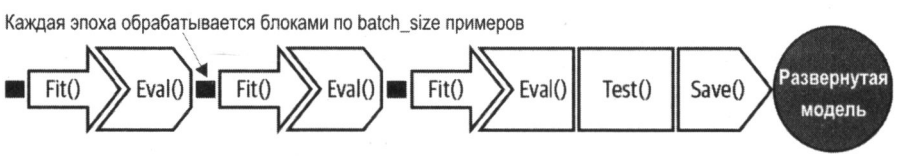


Рис. 4.1. Типичный цикл тренировки состоит из трех эпох. Каждая эпоха обрабатывается блоками по `batch_size` примеров. В конце третьей эпохи модель оценивается на тестовом наборе данных и сохраняется для потенциального развертывания в качестве веб-службы

в явной форме, но нам это не потребуется ни для одного обсуждаемого в данной главе паттерна.

Паттерны для выполнения тренировки

Все описанные в этой главе паттерны так или иначе связаны с модифицированием типичного цикла тренировки. В паттерне "Полезное переобучение" (Useful Overfitting) мы отказываемся от использования валидационного или тестового набора данных, потому что намеренно хотим добиться переобучения на тренировочном наборе данных. В паттерне "Контрольные точки" (Checkpoints) мы периодически сохраняем полное состояние модели, чтобы иметь доступ к частично натренированным моделям. Отмечая контрольные точки, мы обычно также используем виртуальные эпохи, в которых решаем выполнять внутренний цикл функции `fit()` не на полном тренировочном наборе данных, а на фиксированном числе тренировочных примеров. В паттерне "Трансферное обучение" (Transfer Learning) мы берем часть ранее натренированной модели, замораживаем веса и встраиваем эти нетренируемые слои в новую модель, которая решает ту же задачу, но на меньшем наборе данных. В паттерне "Распределительная стратегия" (Distribution Strategy) цикл тренировки выполняется в требуемом масштабе на нескольких воркерах, часто с кэшированием, аппаратным ускорением и параллелизацией. Наконец, в паттерне "Гиперпараметрическая настройка" (Hyperparameter Tuning) сам цикл тренировки вставляется в метод оптимизации, чтобы можно было найти оптимальный набор модельных гиперпараметров.

ПАТТЕРН 11. Полезное переобучение

В паттерне "Полезное переобучение" (Useful Overfitting) мы отказываемся от использования механизмов обобщения, потому что намеренно хотим добиться переобучения модели на тренировочном наборе данных[1]. В ситуациях, когда переобучение полезно, этот паттерн рекомендует проводить машинное обучение без регуляризации, отсева или валидационного набора данных для досрочной остановки.

Постановка задачи

Цель модели машинного обучения — обобщать и делать надежные предсказания на новых, ранее не встречавшихся данных. Если ваша модель *переобучена* на тренировочных данных (например, она продолжает уменьшать тренировочную ошибку за пределами точки, в которой валидационная ошибка начинает увеличиваться), тогда ее способность к обобщению страдает, как и ваши будущие предсказания. Базовые

[1] Переподгонка (overfitting) возникает, когда ML-модель слишком плотно прилегает к тренировочным данным и поэтому не способна давать точные предсказания на ранее не встречавшихся данных. Неплохой аналогией является визит к портному: переподгонка — костюм подогнан слишком плотно, недоподгонка — наоборот. — *Прим. перев.*

учебники по машинному обучению советуют избегать переобучения, используя методы досрочной (ранней) остановки и регуляризации.

Рассмотрим, однако, ситуацию симулирования поведения физических или динамических систем, подобных тем, которые встречаются в климатологии, вычислительной биологии или вычислительных финансах. В таких системах временна́я зависимость наблюдений может быть описана математической функцией или набором дифференциальных уравнений в частных производных (partial differential equation, PDE). Хотя уравнения, управляющие многими из этих систем, могут быть выражены формально, они не имеют решения в замкнутой форме. Вместо этого были разработаны классические численные методы, которые аппроксимируют решения этих систем. К сожалению, для многих реальных приложений эти методы могут быть слишком медленными, что препятствует их использованию на практике.

Рассмотрим ситуацию, показанную на рис. 4.2. Наблюдения, собранные из физической среды, используются на входе (или в качестве первоначальных стартовых условий) в основанную на законах физики модель, которая выполняет итерационные численные вычисления для расчета высокоточного состояния системы. Предположим, что все наблюдения имеют конечное число возможностей (например, температура будет изменяться от 60 до 80 °C с шагом 0,01 °C). Тогда можно будет создать тренировочный набор данных для системы машинного обучения, состоящий из полного входного пространства, и вычислять метки с использованием физической модели.

Рис. 4.2. Одна из ситуаций, когда переобучение допустимо, заключается в том, что все пространство наблюдений в области определения может быть сведено в таблицу и имеется физическая модель, способная вычислять высокоточное решение

Модель ML должна запомнить эту высокоточно рассчитанную и неперекрывающуюся справочную таблицу входных и выходных данных. Выбор в пользу разбивки такого набора данных на тренировочный и оценочный наборы будет контрпро-

дуктивным, поскольку мы будем ожидать, что модель станет запоминать части входного пространства, которые она на самом деле не увидит в тренировочном наборе данных.

Решение

В этом сценарии нет "ранее не встречавшихся" данных, для которых необходимо выполнять обобщение, поскольку все возможные входные данные были сведены в таблицу. Во время разработки модели машинного обучения для обучения такой основанной на физике модели или динамической системы не существует понятия "переобучение". Базовая тренировочная парадигма машинного обучения слегка отличается. Здесь есть некое физическое явление, которое вы пытаетесь запомнить, и оно управляется лежащим в его основе набором дифференциальных уравнений в частных производных или их системой. Машинное обучение обеспечивает управляемый данными подход лишь для аппроксимирования высокоточного решения, и такие понятия, как переобучение, должны оцениваться по-новому.

Например, метод трассировки лучей используется для симулирования спутниковых снимков, которые получаются в результате работы численных моделей предсказания погоды. Он предусматривает расчет степени поглощения солнечного луча предсказанными гидрометеорами (дождем, снегом, градом, ледяными крупинками и т. д.) на каждом атмосферном уровне. Существует конечное число возможных типов гидрометеоров и конечное число высот, которые численная модель предсказывает. Следовательно, модель трассировки лучей должна применять оптические уравнения к крупному, но конечному набору входных данных.

Уравнения излучательного переноса управляют сложной динамической системой распространения электромагнитного излучения в атмосфере, а модели прямого излучательного переноса являются эффективным средством предсказательного вывода о будущем состоянии спутниковых снимков. Однако классические численные методы, которые вычисляют решения этих уравнений, могут потребовать огромных вычислительных усилий и являются слишком медленными для использования на практике.

В этой ситуации на первый план выходит машинное обучение. Его можно использовать для создания модели, которая аппроксимирует решения[2] модели прямого излучательного переноса (рис. 4.3). Эта аппроксимация с привлечением средств технологии ML может быть сделана достаточно близкой к модельному решению, которое изначально было достигнуто с помощью более классических методов. Ее преимущество заключается в том, что предсказательный вывод с применением аппроксимации (которая должна лишь вычислять замкнутую формулу) занимает лишь часть времени, необходимого для выполнения трассировки лучей (которое потребовало бы численных методов). В то же время тренировочный набор данных слишком велик (несколько терабайт) и слишком громоздок для использования в качестве справочной таблицы в производстве.

---

[2] См. **https://oreil.ly/IkYKm**.

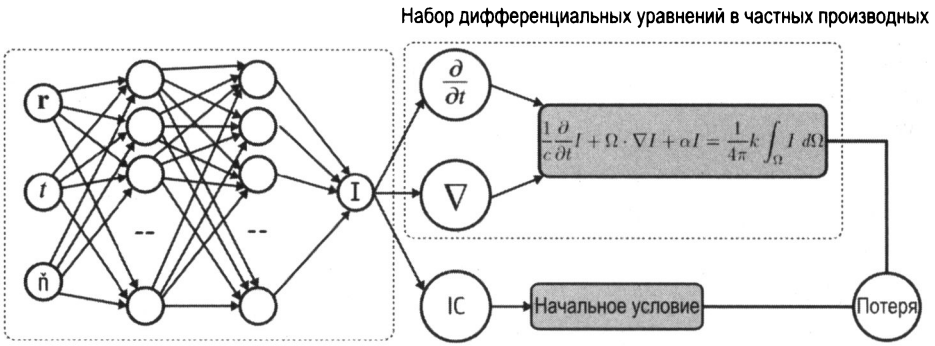


Рис. 4.3. Архитектура использования нейронной сети для моделирования решения уравнения в частных производных для $I(r, t, n)$

Существует важное различие между тренировкой ML-модели для аппроксимирования решения такой динамической системы и тренировкой ML-модели для предсказания веса младенца на данных о рождаемости, собранных в течение многих лет. А именно динамическая система представляет собой набор уравнений, управляемых законами электромагнитного излучения — нет ни ненаблюдаемой переменной, ни шума и ни статистической изменчивости. Для конкретного набора входных данных существует лишь один высокоточно рассчитываемый результат. В тренировочном наборе данных нет никакого наложения между разными примерами. По этой причине мы можем отбросить проблему в отношении обобщения. Мы хотим, чтобы наша ML-модель вписывалась в тренировочные данные как можно лучше, став "переобученной".

Это противоречит типичному подходу к тренировке модели ML, при котором важны соображения смещенности, дисперсии и ошибки обобщения. Традиционно понимаемый процесс тренировки предусматривает наличие возможности, при которой модель будет запоминать тренировочные данные "слишком хорошо", и если в результате тренировки своей модели ваша функция тренировочной потери равна нулю, то это, скорее, является тревожным сигналом, чем поводом для торжества. Переобучение на тренировочном наборе данных в таком ключе приводит к тому, что модель дает ошибочные предсказания на новых, ранее не встречавшихся точках данных. Но мы заведомо знаем, что ранее не встречавшихся данных не будет, поэтому модель аппроксимирует решение набора дифференциальных уравнений в частных производных (PDE) над полным входным спектром. Если ваша нейронная сеть способна усваивать набор параметров, где функция потери равна нулю, то этот набор параметров определяет фактическое решение рассматриваемого дифференциального уравнения.

Почему это работает

Если все возможные входные данные можно свести в таблицу, то, как показано пунктирной кривой на рис. 4.4, переобученная модель все равно будет делать те же предсказания, что и "истинная" модель, при условии, что она будет натренирована на всех возможных входных точках. И потому переобучение не вызывает озабо-

ченности. Мы должны позаботиться о том, чтобы предсказательные выводы делались на округленных значения входных данных, при этом округление определяется разрешающей способностью, с которой входное пространство было разделено сеткой.

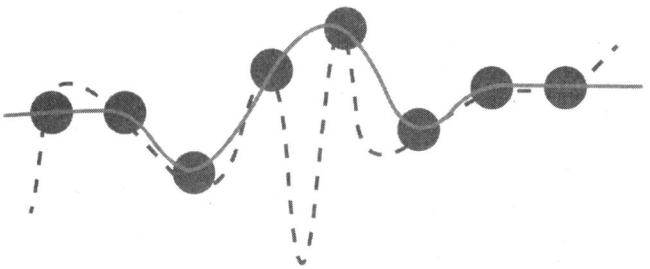


Рис. 4.4. Переобучение не вызывает озабоченности, если модель натренирована на всех возможных входных точках, потому что предсказания одинаковы для обеих кривых

Можно ли найти модельную функцию, которая будет сколь угодно близкой к истинным меткам? Объяснение того, почему это работает, вытекает из теоремы равномерной аппроксимации, относящейся к глубокому обучению, которая утверждает, что любая функция (и ее производные) может быть аппроксимирована нейронной сетью по меньшей мере с одним скрытым слоем и любой "сплющивающей" активационной функцией, такой как сигмоида. Из нее следует, что независимо от данной нам функции, до тех пор, пока она ведет себя относительно хорошо, существует нейронная сеть только с одним скрытым слоем, которая аппроксимирует эту функцию так близко, как мы хотим[3].

Основанные на глубоком обучении подходы к решению дифференциальных уравнений или сложных динамических систем призваны представлять функцию, определенную в неявной форме дифференциальным уравнением или системой уравнений, с использованием нейронной сети.

Переобучение полезно при соблюдении двух следующих условий:

- нет шума, поэтому метки точны для всех случаев;
- в вашем распоряжении имеется полный набор данных (у вас есть все возможные примеры), в этом случае переобучение становится интерполированием набора данных.

Компромиссы и альтернативы

Мы ввели понятие переобучения как полезного метода, когда набор входных данных исчерпывающий и точная метка для каждого набора входов может быть вычислена. Если полное входное пространство можно свести в таблицу, то пере-

[3] Разумеется, возможно, дело не в том, что мы можем обучать сеть с помощью градиентного спуска только потому, что такая нейронная сеть существует (вот почему выручает изменение архитектуры модели путем добавления слоев — это делает ландшафт потери более поддающимся стохастическому градиентному спуску).

обучение не является проблемой, потому что нет ранее не встречавшихся данных. Однако паттерн "Полезное переобучение" полезен и за пределами этого узкого варианта использования. Во многих реальных ситуациях, даже если приходится смягчать одно или несколько условий, постулат о пользе переобучения остается в силе.

Интерполяция и теория хаоса

Модель машинного обучения функционирует как приближение (аппроксимация) справочной таблицы входных данных к выходным. Если справочная таблица мала, ее следует использовать именно как справочную таблицу! Нет необходимости аппроксимировать ее моделью машинного обучения. Аппроксимация ML полезна в ситуациях, когда справочная таблица слишком велика для ее эффективного использования. Именно в случае громоздкости справочной таблицы предпочтительнее трактовать ее как тренировочный набор данных для ML-модели, которая аппроксимирует эту справочную таблицу.

Обратите внимание: мы исходили из того, что наблюдения будут иметь конечное число возможностей. Например, мы постулировали, что температура будет измеряться в приращениях 0,01 °C и находиться между 60 и 80 °C. Это справедливо, если наблюдения фиксируются цифровыми приборами. Если это не так, то модель ML необходима для интерполирования между записями из справочной таблицы.

Модели машинного обучения интерполируют, взвешивая ранее не встречавшиеся значения по расстоянию этих ранее не встречавшихся значений от тренировочных примеров. Такая интерполяция работает только в том случае, если лежащая в ее основе система не является хаотической. В хаотических системах, даже если система детерминирована, малые различия в начальных условиях могут привести к кардинально разным исходам.

Тем не менее на практике каждое отдельное хаотическое явление имеет конкретный порог разрешающей способности[4], за пределами которого существует возможность, что модели будут прогнозировать его в течение коротких промежутков времени. Следовательно, если справочная таблица достаточно детализирована и понятны пределы разрешающей способности, можно получить полезные аппроксимации.

Методы Монте-Карло

В реальности сведение в таблицу всех вероятных входных значений может оказаться практически неосуществимым, и вы, возможно, воспользуетесь подходом, основанным на методах Монте-Карло[5], для выборки из входного пространства для создания набора входных значений, в особенности там, где не все возможные комбинации входных значений физически возможны.

---

[4] См. **https://oreil.ly/F-drU**.

[5] См. **https://oreil.ly/pTgS9**.

В таких случаях переобучение технически возможно (на рис. 4.5 незаполненные кружки аппроксимированы неверными оценками, показанными кружками с крестиками).

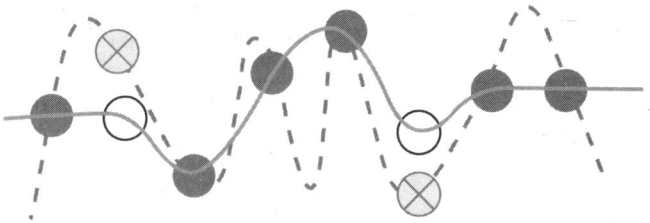


Рис. 4.5. Если данные входного пространства не сведены в таблицу и вместо этого из него отбираются пробы, то вам нужно позаботиться об ограничении модельной сложности

Однако даже здесь вы увидите, что ML-модель будет интерполировать между известными ответами. Вычисление всегда является детерминированным, и только входные точки подвергаются случайному отбору. Следовательно, эти известные ответы не содержат шума, а поскольку ненаблюдаемых переменных нет, ошибки в неотобранных точках будут строго ограничены модельной сложностью. Здесь опасность переобучения исходит не от подгонки к шуму, а от сложности модели. Переобучение не так уж и страшно, когда размер набора данных больше, чем число свободных параметров. Следовательно, использование комбинации моделей с низкой сложностью и мягкой регуляцией обеспечивает практический прием, позволяющий избегать неприемлемого переобучения в случае отбора проб из входного пространства методом Монте-Карло.

Дискретизации под управлением данных

Хотя получение решения в замкнутой форме является возможным для некоторых наборов дифференциальных уравнений в частных производных (PDE), определение решений с использованием численных методов более распространено. Численные методы PDE уже являются обширной областью научных исследований, и этой теме посвящено много книг[6], курсов[7] и журналов[8]. Одним из распространенных подходов является использование конечно-разностных методов, подобных методам Рунге — Кутты (Runge — Kutta), для решения обыкновенных дифференциальных уравнений. Обычно это делается путем дискретизирования дифференциального оператора набора дифференциальных уравнений в частных производных и поиска решения дискретной задачи на пространственно-временной сетке начальной области. Однако когда размерность задачи становится большой, этот основанный на сетке подход резко отказывает из-за проклятия размерности, поскольку расстояние между ячейками сетки должно быть достаточно малым[9], чтобы можно было

[6] См. **https://oreil.ly/RJWVQ**.

[7] См. **https://oreil.ly/wcl_n**.

[8] См. **https://msp.org/apde**.

[9] См. **https://oreil.ly/TxHD-**.

захватывать наименьший размер признака в решении. Поэтому для достижения 10-кратного увеличения разрешающей способности снимка требуется 10 000-кратное увеличение вычислительной мощности, поскольку сетка должна быть масштабирована в четырех измерениях с учетом пространства и времени.

Однако существует возможность использовать не методы Монте-Карло, а машинное обучение, отбирая образцы точек для создания управляемых данными дискретизаций наборов дифференциальных уравнений в частных производных. В статье "Автоматическое усвоение дискретизаций, управляемых данными, для PDE" (Learning data-driven discretizations for PDEs)[10] Бар-Синай (Bar-Sinai) и соавт. демонстрируют эффективность этого подхода. Авторы используют сетку низкой разрешающей способности, состоящую из фиксированных точек, для аппроксимирования решения посредством кусочно-полиномиальной интерполяции с применением стандартных конечно-разностных методов, а также решения, полученного из нейронной сети. Решение, полученное из нейронной сети, значительно превосходит численную симуляцию в минимизировании абсолютной ошибки, в некоторых местах достигая улучшения порядка в 10^2 раз. В то время как увеличение разрешающей способности с применением конечно-разностных методов требует существенно большей вычислительной мощности, нейронная сеть способна поддерживать высокую производительность лишь с минимальными дополнительными затратами. Такие методы, как глубокий метод Галеркина (deep Galerkin method), могут использовать глубокое обучение, чтобы обеспечивать бессеточную аппроксимацию решения данного дифференциального уравнения в частных производных. При подобном подходе решение дифференциального уравнения сводится к задаче цепной оптимизации (*см. разд. "Паттерн 8. Каскад" главы 3*).

Глубокий метод Галеркина

Глубокий метод Галеркина[11] — это алгоритм глубокого обучения для решения дифференциальных уравнений в частных производных. Указанный алгоритм аналогичен по духу методам Галеркина, используемым в области численного анализа, в котором решение аппроксимируется с помощью нейронной сети, а не линейной комбинации базисных функций.

Неограниченные области

И в методах Монте-Карло, и в управляемых данными методах дискретизации исходят из того, что отбор проб из всего входного пространства, даже если он будет несовершенным, возможен. Вот почему ML-модель трактовалась как интерполяция между известными точками.

Обобщение и проблему переобучения трудно игнорировать, когда мы не способны отбирать пробы точек из полной области определения функции, например в случае

[10] См. **https://oreil.ly/djDkK**.

[11] См. **https://oreil.ly/rQy4d**.

функций с неограниченными областями или проекциями вдоль оси времени в будущее. В этих условиях важно учитывать переобучение, недоподгонку и ошибку обобщения. На самом деле, хотя технические приемы, подобные глубокому методу Галеркина, удовлетворительно работают на участках, откуда отбираются хорошие пробы, усваиваемая таким образом функция обобщает не всегда с приемлемым результатом на участках вне области, которые не подверглись отбору проб в фазе тренировки. А это может создавать проблемы в использовании ML для решения набора дифференциальных уравнений в частных производных, определенных на неограниченных областях, поскольку невозможно захватывать представительную выборку для тренировки.

Дистиллирование знаний нейронной сети

Еще одна ситуация, когда переобучение оправдано, заключается в дистиллировании или перенеесе (трансфере) знаний из большой модели машинного обучения в меньшую. Дистилляция (перегонка) знаний полезна, когда емкость обучения у большой модели задействуется не в полной мере. В таком случае вычислительная сложность большой модели может не потребоваться. Однако при этом тренировать малые модели сложнее. Хотя меньшая модель обладает достаточной емкостью для представления знаний, она может не обладать достаточной емкостью для эффективного обучения знаний.

Решение состоит в том, чтобы тренировать меньшую модель на крупном объеме сгенерированных данных, которые помечены большей моделью. Меньшая модель усваивает не фактические метки на реальных данных, а мягкий результат на выходе из большей модели. Это задача проще и может быть усвоена меньшей моделью. Как и при аппроксимировании числовой функции ML-моделью, цель состоит в том, чтобы меньшая модель была добросовестным представителем предсказаний большей модели машинного обучения. На этом втором тренировочном шаге можно использовать паттерн "Полезное переобучение" (Useful Overfitting).

Переобучение на пакете данных

На практике тренировка нейронных сетей требует большого числа экспериментов, и специалист-практик должен выбирать из большого числа вариантов, от размера и архитектуры сети до скорости обучения, инициализации весов или других гиперпараметров.

Переобучение на малом пакете — хорошая проверка на вменяемость[12] как модели, так и конвейера ввода данных. Если модель компилируется и исходный код выполняется без ошибок, это не означает, что вы вычислили то, что у вас есть, или что целевая установка тренировки сконфигурирована правильно. Сложная модель должна быть способной к переобучению на достаточно малом пакете данных в предположении, что все настроено правильно. Поэтому, если вы не можете переобучить любую модель на малом пакете, то стоит перепроверить код модели, кон-

[12] См. **https://oreil.ly/AcLtu**.

вейер ввода данных и функцию потери на наличие каких-либо ошибок или простых дефектов. Переобучение на пакете данных — полезный технический прием во время тренировки и отладки нейронных сетей.

Переобучение выходит за рамки просто пакета. С более целостной точки зрения, переобучение подчиняется общему совету, который обычно дается в отношении глубокого обучения и регуляризации. Оптимально подогнанная модель — это большая модель, которая была надлежаще регуляризована[13]. Иными словами, если ваша глубокая нейронная сеть не способна достичь переобучения к тренировочному набору данных, то вам следует использовать более крупную. А после того как у вас появится большая модель, переобученная на тренировочном наборе, вы сможете применить регуляризацию для повышения валидационной точности, даже если тренировочная точность может снизиться.

Вы можете протестировать свой код модели Keras, используя `tf.data.Dataset`, который вы написали для своего конвейера ввода данных. Например, если ваш конвейер ввода тренировочных данных называется `trainds`, то мы будем использовать `batch()` для извлечения одного пакета данных. Полный исходный код для данного примера[14] можно найти в репозитории, прилагаемом к этой книге:

```
BATCH_SIZE = 256
single_batch = trainds.batch(BATCH_SIZE).take(1)
```

Затем, во время тренировки модели, вместо того чтобы вызывать полный набор данных `trainds` внутри метода `fit()`, мы используем созданный нами один-единственный пакет:

```
model.fit(single_batch.repeat(),
          validation_data=evalds,
          ...)
```

Обратите внимание, что мы применяем метод `repeat()`, чтобы не исчерпывать данные во время тренировки на этом единственном пакете. За счет этого обеспечивается возможность многократно брать по одному пакету во время тренировки. Все остальное (валидационный набор данных, модельный код, сгенерированные признаки и т. д.) остается прежним.

Вместо извлечения произвольной выборки из тренировочного набора данных мы рекомендуем выполнять переобучение на малом наборе данных, каждый пример которого был старательно проверен на наличие правильных меток. Дизайн нейросетевой архитектуры должен быть выстроен таким образом, чтобы она могла высокоточно усваивать этот пакет данных и получать нулевую потерю. Затем следует взять ту же самую сеть и натренировать ее на полном тренировочном наборе данных.

---

[13] См. **https://oreil.ly/A7DFC**.

[14] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/04_hacking_training_loop/distribution_strategies.ipynb**.

ПАТТЕРН 12. Контрольные точки

В контрольных точках мы периодически сохраняем полное состояние модели, чтобы иметь частично натренированные модели, которые могут служить окончательной моделью (в случае досрочной остановки) либо выступать в качестве отправной точки для продолжения тренировки (в случае тонкой настройки и аварийного отказа машины). Паттерн "Контрольные точки" (Checkpoints) обеспечивает такую функциональность.

Постановка задачи

Чем сложнее модель (например, чем больше слоев и узлов в нейронной сети), тем больше набор данных, необходимый для эффективного выполнения ее тренировки. Это обусловлено тем, что более сложные модели, как правило, имеют более настраиваемые параметры. По мере роста размеров модели время, необходимое для подгонки одного пакета примеров, тоже увеличивается. По мере увеличения размера данных (и при условии, что размеры пакетов фиксированы) число пакетов тоже увеличивается. Следовательно, с точки зрения вычислительной сложности этот двойной удар означает, что тренировка будет занимать много времени.

На момент написания этой книги тренировка модели перевода с английского языка на немецкий на новейшем модуле тензорного процессора (tensor processing unit, TPU) с использованием относительно малого набора данных занимает около 2 часов[15]. На реальных наборах данных, вроде тех, которые используются для тренировки умных устройств, тренировка может длиться несколько дней.

Когда у нас тренировка длится так долго, шансы на аварийный отказ машины неприятно высоки. В таком случае при возникновении проблемы нужно иметь возможность запустить работу не с самого начала, а продолжить ее с промежуточной точки.

Решение

В конце каждой эпохи мы можем сохранять состояние модели. И если цикл тренировки по какой-либо причине прерывается, мы можем возвращаться в сохраненное состояние модели и перезапускаться. Однако при этом мы должны обеспечивать сохранение не только самой модели, но и ее промежуточного состояния. Что это означает?

После завершения тренировки мы сохраняем или экспортируем модель, чтобы иметь возможность ее развертывать для предсказательного вывода. Экспортированная модель содержит не всё свое состояние, а только информацию, необходимую для создания функции предсказания. Например, для дерева решений это будут окончательные правила по каждому промежуточному узлу и предсказанное значе-

[15] См. **https://oreil.ly/vDRve**.

ние для каждого листового узла. Для линейной модели это будут окончательные значения весов и смещенностей. Для полносвязной нейронной сети нам также нужно добавить активационные функции и веса скрытых соединений.

Какие данные о состоянии модели нам нужны при восстановлении из контрольной точки, которые экспортированная модель не содержит? Экспортированная модель не содержит эпоху и номер пакета, которые модель в данный момент обрабатывает, что, очевидно, важно для возобновления тренировки. Но цикл тренировки модели может содержать больше информации. Для того чтобы эффективно выполнять градиентный спуск, оптимизатор может изменять скорость обучения по расписанию. Состояние этой скорости обучения отсутствует в экспортируемой модели. Кроме того, в модели может иметься стохастическое поведение, например отсев. Оно тоже не фиксируется в экспортированном состоянии модели. Модели, наподобие рекуррентных нейронных сетей, включают в себя историю предыдущих входных значений. В общем случае полное состояние модели во много раз больше размера экспортируемой модели.

Сохранение полного состояния модели с целью обеспечения возможности возобновления тренировки с заданной точки называется *фиксацией состояния в контрольной точке* (checkpointing), а сохраненные модельные файлы называются *контрольными точками*. Как часто мы должны фиксировать состояния в контрольных точках? Состояние модели изменяется после каждого пакета из-за градиентного спуска. Поэтому, если мы не хотим потерять проделанную работу, то должны фиксировать состояние после каждого пакета. Однако контрольные точки огромны, и связанный с ними ввод-вывод добавит значительные накладные расходы. Вместо этого модельные фреймворки обычно предоставляют возможность фиксации состояния модели в контрольной точке в конце каждой эпохи. Это разумный компромисс между отсутствием какой-либо фиксации состояния вообще и фиксацией состояния после каждого пакета.

В целях фиксирования состояния модели в контрольной точке в рамках Keras методу `fit()` следует предоставить функцию обратного вызова:

```
checkpoint_path = '{}/checkpoints/taxi'.format(OUTDIR)
cp_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                                                 save_weights_only=False,
                                                 verbose=1)
history = model.fit(x_train, y_train,
                    batch_size=64,
                    epochs=3,
                    validation_data=(x_val, y_val),
                    verbose=2,
                    callbacks=[cp_callback])
```

С добавлением фиксации состояния в контрольной точке цикл тренировки становится таким, каким он показан на рис. 4.6.

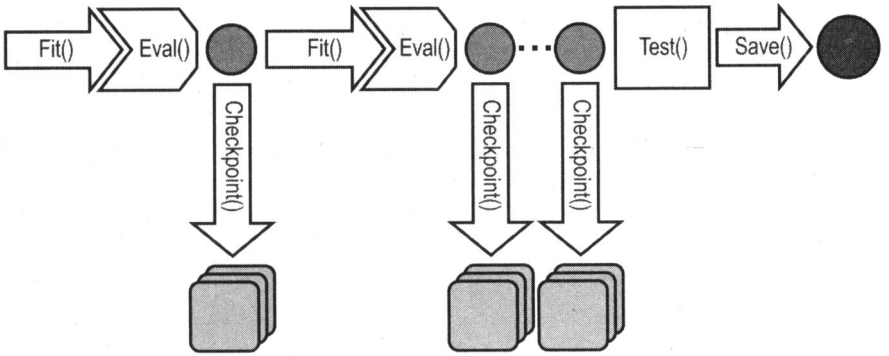


Рис. 4.6. При фиксировании состояния в контрольной точке сохраняется полное состояние модели в конце каждой эпохи

Контрольные точки в PyTorch

На момент написания этой книги библиотека PyTorch напрямую не поддерживает контрольные точки. Однако она поддерживает экстернализацию состояния большинства объектов. В целях реализации контрольных точек в PyTorch следует запросить, чтобы эпоха, состояние модели, состояние оптимизатора и любая другая информация, необходимая для возобновления тренировки, были сериализованы вместе с моделью:

```
torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': loss,
            ...
            }, PATH)
```

Во время загрузки из контрольной точки вам нужно создать необходимые классы, а затем загрузить их из контрольной точки:

```
model = ...
optimizer = ...
checkpoint = torch.load(PATH)
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
epoch = checkpoint['epoch']
loss = checkpoint['loss']
```

Этот уровень является более низким, чем в TensorFlow, но он обеспечивает гибкость хранения нескольких моделей в контрольной точке и выбора того, какие части состояния модели загружать или не загружать.

Почему это работает

TensorFlow и Keras автоматически возобновляют тренировку с контрольной точки, если контрольные точки находятся в выходном пути. Следовательно, для того чтобы начать тренировку с нуля, вы должны начать с нового выходного каталога (или удалить предыдущие контрольные точки из выходного каталога). Это работает, потому что фреймворки машинного обучения учитывают наличие файлов контрольных точек.

Несмотря на то что контрольные точки предназначены в первую очередь для поддержания отказоустойчивости, наличие частично натренированных моделей открывает ряд других вариантов использования. Это обусловлено тем, что частично натренированные модели обычно более способны к обобщению, чем модели, созданные в более поздних итерациях. Понять, почему это происходит, можно из игровой площадки Tensorflow[16] (рис. 4.7).

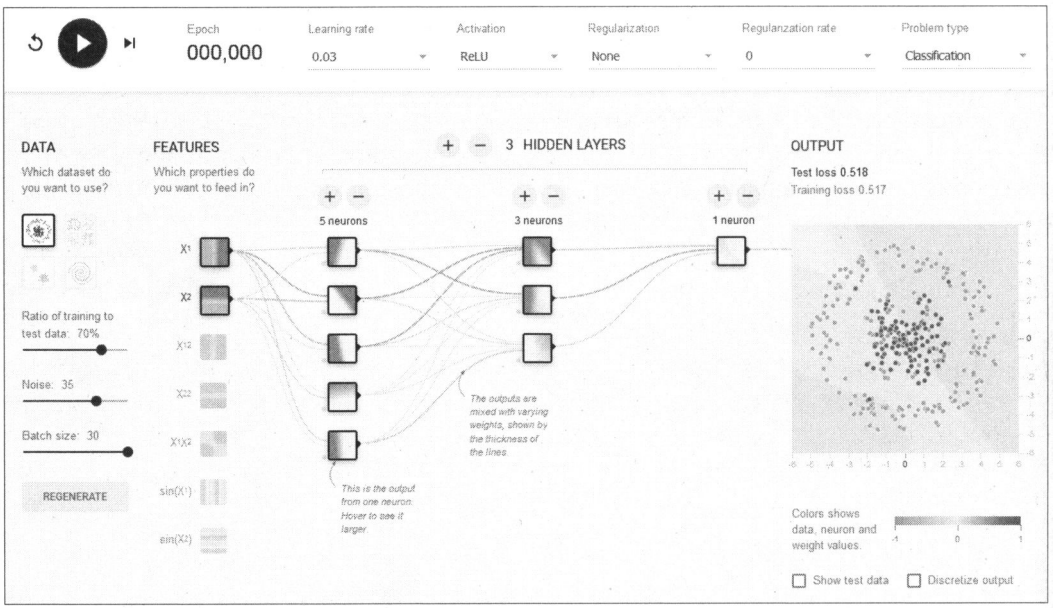


Рис. 4.7. Стартовая точка спиральной классификационной задачи. Вы можете перейти к этой настройке, открыв соответствующую ссылку[17] в веб-браузере

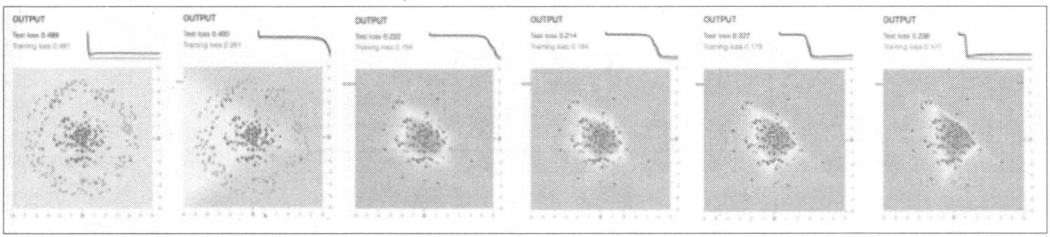
В окне игровой площадки мы пытаемся построить классификатор для различения синих и оранжевых точек (если вы читаете этот текст в печатной книге, перейдите по ссылке в веб-браузере). Двумя входными признаками являются x_1 и x_2, т. е. координаты точек. Основываясь на этих признаках, модель должна выдать вероятность того, что точка является синей. Модель начинает работу со случайных весов, и фон точек показывает модельное предсказание для каждой координатной точки.

[16] См. **https://oreil.ly/sRjkN**.

[17] См. **https://oreil.ly/ISg9X**.

Поскольку веса являются случайными, вероятность имеет тенденцию зависать вблизи центрального значения для всех пикселов.

Запустив тренировку нажатием кнопки-стрелки в левом верхнем углу изображения, мы увидим, как модель медленно начинает учиться с каждой последующей эпохой (рис. 4.8).



а *б* *в* *г* *д* *е*

Рис. 4.8. Шаги, демонстрирующие, что́ именно модель усваивает в ходе тренировки. Графики вверху — это тренировочная потеря и валидационная ошибка, в то время как изображения показывают, как модель на этом этапе предсказала бы цвет точки в каждой координате сетки

Первый намек на усвоение отображен на рис. 4.8, *б*, и видно, что модель усвоила высокоуровневое представление данных, на рис. 4.8, *в*. С этого момента модель корректирует границы, чтобы получить все больше и больше синих точек в центральном участке, оставляя при этом оранжевые точки за пределами. Это выручает, но только до определенного момента. К тому времени, когда мы доберемся до рис. 4.8, *д*, корректировка весов начинает отражать случайные возмущения в тренировочных данных, и они контрпродуктивны на валидационном наборе данных.

Следовательно, мы можем разбить тренировку на три фазы. В первой фазе, между этапами (*а*) и (*в*), модель усваивает высокоуровневую организацию данных. Во второй фазе, между этапами и (*в*) и (*д*), модель усваивает детали. К тому времени, когда мы переходим к третьей фазе, этапу (*е*), модель уже переобучена. Частично натренированная модель с конца первой фазы или со второй фазы имеет некоторые преимущества именно потому, что она усвоила высокоуровневую организацию, но при этом не увязла в деталях.

Компромиссы и альтернативы

Помимо обеспечения отказоустойчивости сохранение промежуточных контрольных точек также позволяет нам реализаовать возможности досрочной остановки и тонкой настройки.

Досрочная остановка

В общем случае чем дольше вы тренируете модель, тем меньше потеря на тренировочном наборе данных. Однако в какой-то момент ошибка на валидационном наборе данных может перестать уменьшаться. Если вы начинаете достигать переподгонки к тренировочному набору данных, то валидационная ошибка может даже увеличиваться (рис. 4.9).

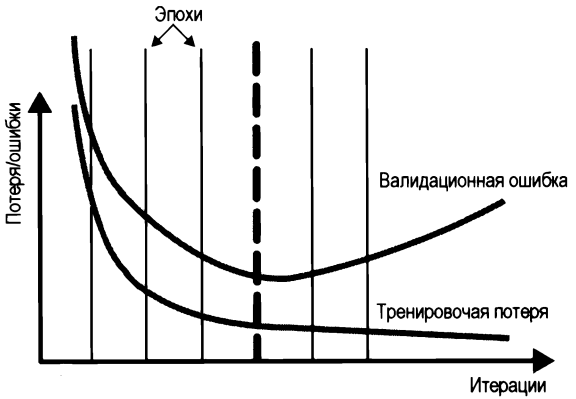


Рис. 4.9. Как правило, тренировочная потеря с течением времени снижается. Но как только начинается переобучение, валидационная ошибка на отложенном наборе данных начинает расти

В таких случаях полезно посмотреть на валидационную ошибку в конце каждой эпохи и остановить тренировочный процесс, когда валидационная ошибка больше, чем в предыдущей эпохе. На рис. 4.9 это произойдет в конце четвертой эпохи, отмеченной толстой штриховой линией. Это *досрочная (ранняя) остановка.*

Если бы мы фиксировали состояние в конце каждого пакета, то могли бы уловить истинный минимум, который мог бы оказаться чуть-чуть ранее или позже границы эпохи. См. обсуждение темы виртуальных эпох в этом разделе, в котором приводится более частый подход к фиксированию состояния в контрольной точке.

Более частое фиксирование состояния полезно, если досрочная остановка не слишком чувствительна к малым возмущениям в валидационной ошибке. Вместо этого мы можем применить досрочную остановку только после того, как валидационная ошибка не улучшится через *N* контрольных точек или более.

Подбор контрольной точки. Хотя досрочная остановка может быть реализована путем остановки тренировки, как только валидационная ошибка начинает увеличиваться, мы рекомендуем тренировать дольше и выбирать оптимальный прогон в качестве шага постобработки. Причина, по которой мы предлагаем хорошо тренировать в третьей фазе (объяснение трех фаз цикла тренировки *см. в разд. "Почему это работает" ранее в этой главе*), заключается в том, что нередко валидационная ошибка увеличивается на некоторое время, а затем снова начинает падать. Обычно это обусловлено тем, что тренировка сначала сосредоточивается на более распространенных сценариях (фаза 1), а затем начинает сходиться к более редким ситуациям (фаза 2). Поскольку отбор редких ситуаций между тренировочным и валидационным наборами данных может осуществляться несовершенным образом, следует ожидать нерегулярных увеличений валидационной ошибки во время тренировочного прогона в фазе 2. Кроме того, существуют характерные для больших моделей ситуации, в которых можно ожидать появления глубокого двойного спуска[18], и поэтому на всякий случай необходимо тренировать немного дольше.

[18] См. **https://oreil.ly/Kya8h**.

В нашем примере вместо экспорта модели в конце цикла тренировки мы загрузим четвертую контрольную точку и оттуда экспортируем нашу окончательную модель. Этот шаг называется *подбором контрольной точки* (checkpoint selection), и в TensorFlow он может осуществляться с помощью его класса `BestExporter`[19].

Регуляризация. Вместо досрочной остановки или подбора контрольной точки бывает полезно попробовать добавить в модель регуляризацию L2, вследствие которой валидационная ошибка не будет увеличиваться и модель никогда не попадет в фазу 3. Взамен и тренировочная потеря, и валидационная ошибка должны достичь плато (рис. 4.10). Мы называем такой цикл тренировки (где метрики тренировки и валидации достигают плато) *хорошо отработанным* (well-behaved) циклом тренировки.

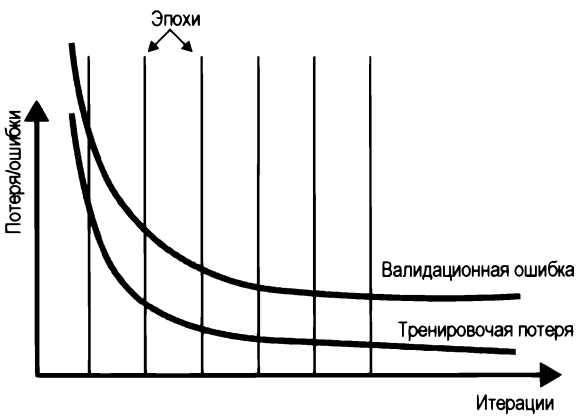


Рис. 4.10. В идеальной ситуации валидационная ошибка не увеличивается. Вместо этого и тренировочная потеря, и валидационная ошибка достигают плато

Если досрочная остановка не выполняется и для определения схождения используется только тренировочная потеря, мы можем избежать необходимости откладывать в сторону отдельный тестовый набор данных. Даже если мы не делаем досрочной остановки, полезно выводить на экран ход тренировки модели, в особенности если тренировка модели занимает много времени. Хотя результативность и ход тренировки модели обычно мониторятся на валидационном наборе данных во время цикла тренировки, это делается только для целей визуализации. Поскольку нам не нужно предпринимать никаких действий, основываясь на отображаемых метриках, мы можем выполнять визуализацию на тестовом наборе данных.

Причина, по которой регуляризация — лучшее решение, нежели досрочная остановка, заключается в том, что регуляризация позволяет использовать весь набор данных целиком, изменяя веса модели, в то время как досрочная остановка требует от 10 до 20% набора данных только для того, чтобы решить, когда прекратить тренировку. Другие методы ограничения переподгонки (такие как отсев и использование моделей с меньшей сложностью) тоже являются хорошей альтернативой дос-

[19] См. **https://oreil.ly/UpN1a**.

рочной остановке. В дополнение к этому, как показывает одна исследовательская работа[20], двойной спуск происходит в самых разных задачах машинного обучения, и, следовательно, лучше тренировать дольше, чем рисковать субоптимальным техническим решением путем ее остановки в досрочном порядке.

Два среза данных. Разве советы в отношении регуляризации не противоречат более ранним советам в отношении досрочной остановки или отбора контрольной точки? Не совсем.

Мы рекомендуем разбивать данные на две части: тренировочный набор данных и оценочный набор данных. Оценочный набор данных играет роль тестового набора данных во время экспериментирования (где нет валидационного набора данных) и роль валидационного набора данных в производстве (где нет тестового набора данных).

Чем объемнее ваш тренировочный набор данных, тем более сложную модель вы можете использовать и тем более точную модель вы можете получить. Использование регуляризации вместо досрочной остановки или отбора контрольной точки позволяет использовать более крупный тренировочный набор данных. В фазе экспериментирования (когда вы разведываете разные модельные архитектуры, технические приемы тренировки и гиперпараметры) рекомендуется отключать досрочную остановку и тренировать с более крупными моделями (*см. также разд. "Паттерн 11. Полезное переобучение"*). Это делается в обеспечение того, чтобы модель обладала достаточной емкостью для обучения предсказательных паттернов. Во время этого процесса следует мониторить схождение ошибки на тренировочном срезе данных. В конце экспериментирования вы можете использовать оценочный набор данных для диагностики качества работы вашей модели на данных, с которыми она не сталкивалась во время тренировки.

Во время тренировки модели для развертывания в реальной среде подготовьтесь к тому, чтобы иметь возможность проводить непрерывное оценивание и перетренировку модели. Сосредоточившись на досрочной остановке или подборе контрольной точки, проводите мониторинг метрики ошибки на оценочном наборе данных. Выбор между досрочной остановкой и подбором контрольной точки зависит от необходимости контролировать затраты (в этом случае выбор падает на досрочную остановку) либо от желания отдать приоритет точности модели (тогда лучшим решением будет подбор контрольной точки).

Тонкая настройка

В хорошо отработанном цикле тренировки градиентный спуск позволит вам быстро добраться до окрестности оптимальной ошибки на основе большинства ваших данных, а затем медленно продвигаться к самой низкой ошибке, оптимизируя на угловых случаях.

Теперь представьте, что вам нужно периодически перетренировывать модель на свежих данных. Как правило, вы хотите подчеркивать именно свежие данные, а не

[20] См. **https://oreil.ly/FJ_iy**.

угловые случаи из прошлого месяца. Нередко предпочтительнее возобновить тренировку не с последней контрольной точки, а с контрольной точки, отмеченной толстой штриховой линией на рис. 4.11, что соответствует началу фазы 2 в нашем обсуждении фаз тренировки модели, описанных ранее в *разд. "Почему это работает"*. Это помогает обеспечивать наличие общего метода, который позднее можно тонко настроить в течение нескольких эпох только на свежих данных.

Когда вы возобновите процесс с контрольной точки, отмеченной толстой пунктирной вертикальной линией, вы окажетесь в четвертой эпохе, и поэтому скорость обучения будет довольно низкой. Следовательно, свежие данные не будут менять модель кардинально. Однако модель будет вести себя оптимально (в контексте более крупной модели) на свежих данных, потому что вы отрегулировали ее на этом меньшем наборе данных. Это называется *тонкой настройкой*. Тонкая настройка также обсуждается в *разд. "Паттерн 13. Трансферное обучение" далее в этой главе*.

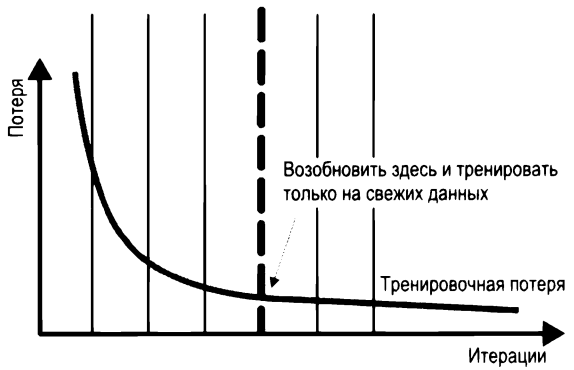


Рис. 4.11. Возобновить работу с контрольной точки до того, как тренировочная потеря начнет достигать плато. Для последующих итераций тренировать только на свежих данных

Тонкая настройка работает только при условии, что вы не меняете архитектуру модели.

Не обязательно всегда начинать с более ранней контрольной точки. В некоторых случаях окончательная контрольная точка (которая будет развернута как сервис) может использоваться в качестве теплого старта для другой итерации тренировки модели. Тем не менее более ранняя контрольная точка, как правило, обеспечивает более качественное обобщение.

Переопределение эпохи

Учебники по машинному обучению нередко содержат вот такой исходный код:

```
model.fit(X_train, y_train,
          batch_size=100,
          epochs=15)
```

Приведенный выше исходный код основывается на допущении о том, что у вас есть набор данных, который вписывается в память, и, следовательно, ваша модель может прокручивать 15 эпох без риска аварийного отказа машины. Оба эти допущения необоснованы — наборы данных ML варьируются, достигая терабайтных объемов, и когда тренировка может длиться часами, высока вероятность аварийного отказа машины.

В целях придания приведенному выше исходному коду большей отказоустойчивости следует подавать набор данных TensorFlow[21] (а не просто массив NumPy), потому что набор данных TensorFlow превышает пределы оперативной памяти. За счет этого обеспечивается возможность итерации и ленивой загрузки. Теперь исходный код выглядит следующим образом:

```
cp_callback = tf.keras.callbacks.ModelCheckpoint(...)
history = model.fit(trainds,
                    validation_data=evalds,
                    epochs=15,
                    batch_size=128,
                    callbacks=[cp_callback])
```

Однако идея использовать эпохи на крупных наборах данных неудачна. Эпохи легко понятны, но приводят к плохим эффектам в реальных ML-моделях. Для того чтобы понять причину этого, представьте, что у вас есть тренировочный набор данных с миллионом примеров. Может возникнуть соблазн просто прокрутить этот набор данных 15 раз (например), установив число эпох равным 15. И здесь вскрывается несколько проблем.

- Число эпох — целое, но разница во времени тренировки между обработкой набора данных 14,3 раза и 15 раз может составлять часы. Если модель сошлась после просмотра 14,3 млн примеров, вы вполне можете остановить процесс и не тратить вычислительные ресурсы, необходимые для обработки еще 0,7 млн примеров.
- Вы фиксируете состояние в контрольной точке один раз за эпоху, и ожидание миллиона примеров между контрольными точками может быть слишком долгим. В целях обеспечения отказоустойчивости у вас, возможно, возникнет потребность фиксировать состояние в контрольной точке чаще.
- Наборы данных растут с течением времени. Если вы добавляете еще 100 тыс. примеров и тренируете модель, получая более высокую ошибку, то возникает вопрос: это потому что вам нужно сделать досрочную остановку или же потому что новые данные каким-то образом повреждены? Вы не можете назвать причину точно, т. к. предыдущая тренировка была на 15 млн примеров, а новая — на 16,5 млн.
- В распределенной тренировке с архитектурой, основанной на сервере параметров (*см. разд. "Паттерн 14. Распределительная стратегия" далее в этой главе*)

[21] См. **https://oreil.ly/EKJ4V**.

с параллелизмом данных и надлежащей перетасовкой, концепция эпохи больше не ясна. Из-за потенциально отстающих воркеров вы можете поручить системе выполнять тренировку только на некотором числе мини-пакетов.

Шаги в расчете на эпоху. Вместо того чтобы тренировать модель в течение 15 эпох, мы могли бы решить тренировать в течение 143 тыс. шагов, где размер пакета `batch_size` равен 100:

```
NUM_STEPS = 143000
BATCH_SIZE = 100
NUM_CHECKPOINTS = 15
cp_callback = tf.keras.callbacks.ModelCheckpoint(...)
history = model.fit(trainds,
                    validation_data=evalds,
                    epochs=NUM_CHECKPOINTS,
                    steps_per_epoch=NUM_STEPS // NUM_CHECKPOINTS,
                    batch_size=BATCH_SIZE,
                    callbacks=[cp_callback])
```

Каждый шаг предусматривает обновление веса, основываясь на одном мини-пакете данных, и это позволяет нам останавливаться на 14,3 эпох. Этот подход обеспечивает гораздо бóльшую гранулярность, но мы должны задавать "эпоху" как 1/15-ю долю от суммарного числа шагов:

```
steps_per_epoch=NUM_STEPS // NUM_CHECKPOINTS
```

Это делается для того, чтобы получать верное число контрольных точек. Такое решение будет работать при условии, что мы обеспечим бесконечный повтор набора данных `trainds`:

```
trainds = trainds.repeat()
```

Необходимость метода `repeat()` обусловлена тем, что мы больше не устанавливаем `num_epochs`, поэтому число эпох по умолчанию равно единице. Без метода `repeat()` модель выйдет из цикла, как только тренировочные образцы будут исчерпаны после однократного чтения набора данных.

Перетренировка с бóльшим количеством данных. Что произойдет, когда мы получим еще 100 тыс. примеров? Нет проблем! Мы добавим их в склад данных, но не будем обновлять исходный код. Наш исходный код все равно захочет обрабатывать 143 тыс. шагов, и он примется обрабатывать этот объем данных, за исключением того, что 10% видимых им примеров являются более новыми. Если модель сходится, то все отлично. Если нет, то проблема кроется в этих новых точках данных, потому что мы не тренируем дольше, чем раньше. Сохраняя число шагов постоянным, мы смогли предотвратить влияние новых данных на тренировку на большем объеме данных.

После того как мы прошли 143 тыс. шагов, перезапускаем тренировку и выполняем ее немного дольше (скажем, 10 тыс. шагов), и пока модель продолжает сходиться, мы тренируем ее дольше. Затем мы обновляем число `143000` в приведенном выше

фрагменте исходного кода (на самом деле это будет параметр исходного кода), чтобы отразить новое число шагов.

Все это прекрасно работает до тех пор, пока вы не захотите выполнить гиперпараметрическую настройку. В этом случае вам захочется изменить размер пакета. К сожалению, если вы поменяете размер пакета на 50, то обнаружите, что тренируете в течение половины времени, потому что мы тренируем в течение 143 тыс. шагов, и каждый шаг лишь наполовину длиннее, чем раньше. Очевидно, что это никуда не годится.

Виртуальные эпохи. Решение заключается в том, чтобы держать неизменным суммарное число показываемых модели тренировочных примеров (а не число шагов; рис. 4.12):

```
NUM_TRAINING_EXAMPLES = 1000 * 1000
STOP_POINT = 14.3
TOTAL_TRAINING_EXAMPLES = int(STOP_POINT * NUM_TRAINING_EXAMPLES)
BATCH_SIZE = 100
NUM_CHECKPOINTS = 15
steps_per_epoch = (TOTAL_TRAINING_EXAMPLES //
                   (BATCH_SIZE*NUM_CHECKPOINTS))
cp_callback = tf.keras.callbacks.ModelCheckpoint(...)
history = model.fit(trainds,
                    validation_data=evalds,
                    epochs=NUM_CHECKPOINTS,
                    steps_per_epoch=steps_per_epoch,
                    batch_size=BATCH_SIZE,
                    callbacks=[cp_callback])
```

Рис. 4.12. Определение виртуальной эпохи с точки зрения желательного числа шагов между контрольными точками

Получив больше данных, сначала следует натренировать модель со старыми настройками, затем увеличить число примеров, чтобы отразить новые данные, и, наконец, изменить `STOP_POINT`, чтобы отразить количество проходов через данные, обеспечивающих схождение.

Теперь этот алгоритм безопасен даже для гиперпараметрической настройки (обсуждаемой далее в этой главе) и сохраняет все преимущества поддержания числа шагов постоянным.

ПАТТЕРН 13. Трансферное обучение

В паттерне "Трансферное обучение" (Transfer Learning) мы берем часть ранее натренированной модели, замораживаем веса и встраиваем эти нетренируемые слои в новую модель, которая решает аналогичную задачу, но на меньшем наборе данных.

Постановка задачи

Тренировка реальных прикладных ML-моделей на неструктурированных данных требует чрезвычайно больших наборов данных, которые не всегда легкодоступны. Возьмем случай модели, выявляющей перелом кости на рентгеновском снимке руки. В целях достижения высокой точности вам понадобятся сотни тысяч снимков, если не больше. Прежде чем ваша модель усвоит внешний вид сломанной кости, она должна сначала научиться понимать пикселы, края и контуры, которые являются частью снимков в вашем наборе данных. То же самое верно и для моделей, натренированных на текстовых данных. Допустим, мы строим модель, которая берет описания симптомов пациента и предсказывает возможные состояния здоровья, связанные с этими симптомами. В дополнение к усвоению того, какие слова отличают простуду от пневмонии, модель также должна усвоить базовую семантику языка и то, как последовательность слов создает смысл. Например, модель должна была бы не только научиться обнаруживать присутствие слова "температура", но и то, что последовательность "температуры нет" несет совсем другое значение, чем "высокая температура".

Для того чтобы понять, какой объем данных требуется для тренировки высокоточных моделей, мы можем обратиться к ImageNet[22] — базе данных из более чем 14 млн помеченных снимков. ImageNet часто используется в качестве эталона для оценивания фреймворков машинного обучения на разнообразном аппаратном обеспечении. Например, в комплекте инструментов для эталонного тестирования MLPerf[23] базу данных ImageNet используют для сравнения времени, потребовавшегося различным фреймворкам ML, работающим на разном аппаратном обеспечении, для достижения точности классификации 75,9%. В тренировочных результатах v0.7 MLPerf модель TensorFlow, работающая на Google TPU v3, требует около 30 секунд для того, чтобы достичь этой целевой точности[24]. В более продолжительных случаях тренировки модели могут достигать еще большей точности на ImageNet. Однако это во многом связано с размером ImageNet. Как правило, организации со специализированными задачами предсказания даже близко не имеют такого же объема данных.

---

[22] См. **https://oreil.ly/t6583**.

[23] См. **https://oreil.ly/hDPiJ**.

[24] MLPerf v0.7 Training Closed ResNet. Взято на веб-сайте **www.mlperf.org** 23 сентября 2020 года, запись 0.7-67. Название и логотип MLPerf являются зарегистрированными торговыми марками. Дополнительную информацию см. на **www.mlperf.org**.

Кроме того, поскольку примеры, подобные описанным выше снимковым и текстовым примерам, предусматривают привлечение специализированных предметных областей данных, отсутствует возможность использовать общецелевую модель, которая обеспечивала бы успешное выявление переломов костей или диагностику заболеваний. Модель, натренированная на ImageNet, возможно, будет способна помечать рентгеновский снимок как рентгеновский или медицинский, но вряд ли сможет помечать его как "сломанная бедренная кость". Поскольку такие модели нередко тренируются на широком спектре высокоуровневых категорий меток, мы не ожидаем, что они будут понимать состояния, присутствующие на снимках, специфичных для нашего набора данных. Значит, требуется техническое решение, которое позволит нам строить конкретную прикладную модель, используя только те данные, которые у нас есть, и с теми метками, которые нам небезразличны.

Решение

Паттерн "Трансферное обучение" (Transfer Learning) позволит нам взять модель, которая была натренирована на том же типе данных для аналогичной задачи, и применить ее к специализированной задаче, используя наши конкретное прикладные данные. Под "одним и тем же типом данных" мы подразумеваем одну и ту же модельность данных — снимки, текст и т. д. Помимо такой широкой категории, как снимки, также идеально использовать модель, которая была предварительно натренирована на тех же типах снимков. Например, использовать модель, которая была предварительно натренирована на фотографиях, если вы собираетесь использовать ее для классифицирования фотоизображений, и модель, которая была предварительно натренирована на снимках дистанционного зондирования, если вы собираетесь применять ее для классифицирования спутниковых снимков. Под *похожей задачей* мы подразумеваем решаемую в настоящий момент задачу. Например, в целях выполнения трансфера ранее усвоенных знаний для классифицирования снимков лучше начать с модели, которая была натренирована классифицировать снимки, а не обнаруживать объекты.

Продолжая этот пример, предположим, что мы строим двоичный классификатор, чтобы на рентгеновском снимке определять наличие сломанной кости. У нас есть только по 200 снимков каждого класса: сломана и не сломана. Этого количества не хватит, чтобы натренировать качественную модель с нуля, но достаточно для трансферного обучения. Для решения этой задачи с помощью трансферного обучения нам понадобится найти модель, которая уже была натренирована на крупном наборе данных выполнять классифицирование снимков. Затем мы удалим из этой модели последний слой, заморозим вес этой модели и продолжим тренировку, используя наши 400 рентгеновских снимков. В идеале мы получим модель, натренированную на наборе данных со снимками, похожими на наши рентгеновские снимки, например снимки, сделанные в лаборатории или в другой контролируемой обстановке. Однако мы все равно сможем эффективно задействовать трансферное обучение, даже если наборы данных различаются при условии, что задача предсказания одна и та же. В данном случае мы выполняем классифицирование снимков.

Трансферное обучение можно использовать для многих задач предсказания помимо классифицирования снимков при условии, что существует предварительно натренированная модель, решающая задачу, аналогичную вашей, которую вы хотите выполнять на своем наборе данных. Например, трансферное обучение также нередко применяется в обнаружении объектов на снимках, передаче стиля снимка, генерировании изображений, классифицировании текста, машинном переводе и т. д.

Трансферное обучение ранее усвоенных знаний работает, потому что позволяет нам использовать модели, которые уже были натренированы на чрезвычайно больших размеченных наборах данных. Мы можем применять трансферное обучение благодаря многолетним научным исследованиям и усилиям, которые другие приложили к созданию этих наборов данных за нас, что и продвинуло вперед современное состояние дел в трансферном обучении. Одним из примеров такого набора данных является проект ImageNet, начатый в 2006 году специалистом в области информатики Фей-Фей Ли (Fei-Fei Li) и опубликованный в 2009 году. Проект ImageNet[25] сыграл важную роль в развитии трансферного обучения и проложил путь для других больших наборов данных, таких как COCO[26] и Open Images[27].

В основе трансферного обучения лежит идея о том, что вы можете использовать веса и слои модели, натренированной в той же предметной области, что и предсказания в вашей задаче. В большинстве ML-моделей последний слой содержит классификационную метку или результат, специфичный для вашей задачи предсказания. При трансферном обучении мы удаляем этот слой, замораживаем натренированные модельные веса и заменяем последний слой результатом для нашей специализированной задачи предсказания перед тем, как продолжить тренировку. Принцип его работы мы видим на рис. 4.13.

Как правило, в *качестве узкого* (bottleneck) *слоя* выбирается предпоследний слой модели (слой перед выходным слоем модели). Далее мы дадим пояснение в отношении узкого слоя, а также объясним разные способы реализации трансферного обучения в TensorFlow.

Узкий слой

По отношению ко всей модели узкий слой представляет собой входную переменную (обычно снимок или текстовый документ) в пространстве наименьшей размерности. Говоря точнее, когда мы подаем данные в модель, первые слои видят эти данные почти в их изначальной форме. В целях ознакомления с тем, как это работает, давайте продолжим рассмотрение примера медицинской визуализации, но на этот раз мы построим модель[28] с набором данных колоректальной гистологии[29], чтобы классифицировать гистологические снимки в одну из восьми категорий.

---

[25] Дэнг С. и др. ImageNet: крупномасштабная иерархическая база данных снимков (Jia Deng. ImageNet: A large-scale hierarchical image database // IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). — 2009. — P. 248–255).

[26] См. **https://oreil.ly/mXt77**.

[27] См. **https://oreil.ly/QN9KU**.

[28] См. **https://oreil.ly/QfOU_**.

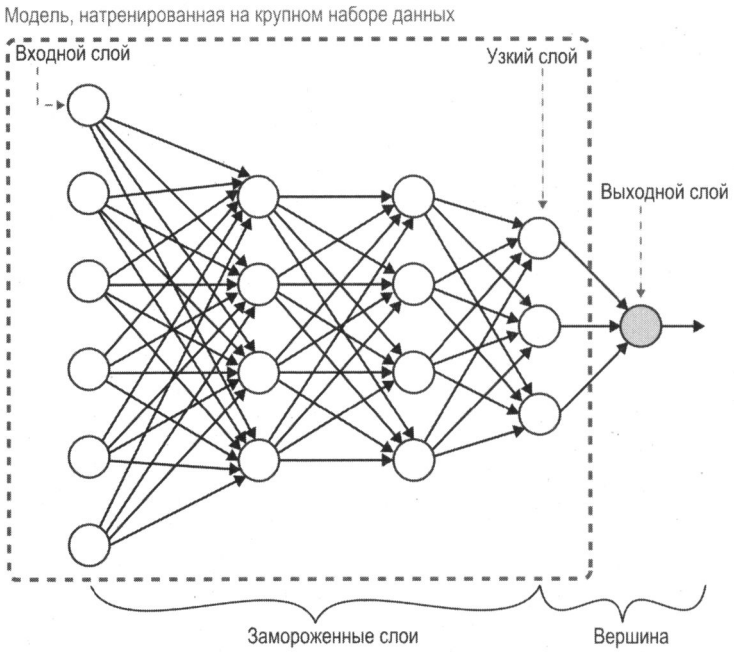


Рис. 4.13. Трансферное обучение предусматривает тренировку модели на крупном наборе данных. "Вершина" модели (как правило, только выходной слой) удаляется, а остальные слои замораживаются. Последний слой оставшейся модели называется узким слоем (или бутылочным горлышком)

В целях проведения разведывательного анализа модели, которую мы собираемся использовать для трансферного обучения, загрузим модельную архитектуру VGG, предварительно натренированную на наборе данных ImageNet:

```
vgg_model_withtop = tf.keras.applications.VGG19(
    include_top=True,
    weights='imagenet',
)
```

Обратите внимание, что мы установили `include_top=True`, а значит, загружаем полную модель VGG, включая выходной слой. В случае ImageNet модель классифицирует снимки на 1000 разных классов, поэтому выходной слой представляет собой массив из 1000 элементов. Посмотрим на результаты работы `model.summary()`, чтобы понять, какой слой будет использоваться в качестве узкого. Для краткости мы опустили здесь несколько средних слоев:

```
Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
```

[29] См. **https://oreil.ly/r4HHq**.

```
block1_conv1 (Conv2D)        (None, 224, 224, 64)        1792
...more layers here...
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)         2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)           0
_________________________________________________________________
flatten (Flatten)            (None, 25088)               0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)                102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)                16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)                4097000
=================================================================
Total params: 143,667,240
Trainable params: 143,667,240
Non-trainable params: 0
```

Как видите, модель VGG принимает снимки в качестве массива размером 224×224×3 пиксела. Этот 128-элементный массив затем пропускается через поочередные слои (каждый из которых может изменять размерность массива) до тех пор, пока он не будет сплющен в одномерный 25 088-элементный массив в слое, именуемом `flatten`. Наконец, он подается в выходной слой, который возвращает массив из 1000 элементов (для каждого класса в ImageNet). В этом примере мы выберем слой `block5_pool` в качестве узкого слоя, когда будем адаптировать эту модель для тренировки на наших медицинских гистологических снимках. Узкий слой производит массив размером 7×7×512, т. е. низкоразмерное представление входного снимка. В нем удерживается довольно много информации из входного снимка, чтобы его можно было классифицировать. Применяя эту модель к нашей задаче классифицирования медицинских снимков, мы надеемся, что дистилляция информации из модели VGG будет достаточной для успешного выполнения классифицирования на нашем наборе данных.

Набор гистологических данных поставляется со снимками в форме (150, 150, 3)-размерных массивов. Представление 150×150×3 является самой высокой размерностью. В целях использования модели VGG с нашими снимковыми данными мы можем загрузить ее следующим образом:

```
vgg_model = tf.keras.applications.VGG19(
    include_top=False,
    weights='imagenet',
    input_shape=((150,150,3))
)

vgg_model.trainable = False
```

Установив `include_top=False`, мы указываем на то, что последний слой VGG, который мы хотим загрузить, является узким слоем. Переданная нами входная форма

input_shape соответствует входной форме наших гистологических изображений. Краткое описание последних нескольких слоев этой обновленной модели VGG выглядит следующим образом:

```
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 20,024,384
Trainable params: 0
Non-trainable params: 20,024,384
_________________________________________________________________
```

Последний слой теперь является нашим узким слоем. Вы, возможно, обратили внимание на то, что размер block5_pool равен (4, 4, 512), тогда как раньше он был (7, 7, 512). Это обусловлено тем, что мы создали экземпляр VGG с параметром input_shape для учета размера изображений в нашем наборе данных. Стоит также отметить, что установка include_top=False жестко запрограммирована на использование block5_pool в качестве узкого слоя, но если вы хотите настроить модель под свои нужды, то можете загрузить полную модель и удалить любые дополнительные слои, которые не хотите использовать.

Прежде чем эта модель будет готова к тренировке, нам понадобится добавить несколько слоев поверх, специфичных для наших данных и классификационной задачи. Также важно отметить, что поскольку мы установили trainable=False, в текущей модели есть 0 тренируемых параметров.

В качестве общего эмпирического правила следует принять, что узким слоем обычно является последний, уплощенный слой наименьшей размерности перед операцией уплощения.

Узкие слои концептуально похожи на векторные вложения, поскольку и те и другие представляют признаки в редуцированной размерности. Например, в автокодировочной модели с архитектурой "кодировщик — декодировщик" узким слоем и вправду является векторное вложение. В этом случае узкий слой служит средним слоем модели, соотнося изначальные входные данные с более низкоразмерным представлением, которое декодировщик (вторая половина сети) использует для соотнесения результата обратно с изначальным, более высокоразмерным представлением. Для ознакомления с диаграммой узкого слоя в автокодировщике см. рис. 2.13.

Слой векторного вложения — это, по сути, справочная таблица весов, соотносящая конкретный признак с некоторой размерностью в векторном пространстве. Главное различие заключается в том, что веса в слое векторного вложения могут быть натренированы, тогда как веса всех слоев, ведущих к узкому слою и включающие узкий слой, заморожены. Другими словами, вся сеть, вплоть до узкого слоя и включая его, не поддается тренировке, и веса в слоях после узкого слоя являются единственными поддающимися тренировке слоями в модели.

Стоит также отметить, что в паттерне "Трансферное обучение" (Transfer Learning) могут использоваться предварительно натренированные векторные вложения. При построении модели, которая включает слой векторного вложения, вы можете либо использовать поиск по существующему (предварительно натренированному) векторному вложению, либо натренировать собственный слой векторного вложения с нуля.

Итак, трансферное обучение представляет собой техническое решение, которое вы можете использовать для выполнения аналогичной задачи на меньшем наборе данных. Трансферное обучение всегда использует узкий слой с нетренируемыми, замороженными весами. Векторные вложения — это разновидность представления данных. В конечном счете все сводится к цели. Если цель заключается в том, чтобы натренировать похожую модель, то вы будете использовать трансферное обучение. Если же цель состоит в том, чтобы представить входной снимок более сжато, вы будете использовать векторное вложение. Исходный код может быть точно таким же.

Реализация трансферного обучения

Трансферное обучение реализуется в Keras с помощью одного из нижеследующих методов.

- Загрузить предварительно натренированную модель самостоятельно, удалить слои после узкого слоя и добавить новый окончательный слой с вашими собственными данными и метками.
- Использовать модуль предварительно натренированных моделей из хаба TensorFlow Hub[30] в качестве основы для вашей задачи трансферного обучения.

Давайте начнем с того, как загружать и использовать предварительно натренированную модель самостоятельно. Для этого будем отталкиваться от примера представленной нами ранее модели VGG. Обратите внимание, что VGG — это архитектура модели, в то время как ImageNet — это данные, на которых она была натренирована. Вместе они составляют предварительно натренированную модель, которую мы будем использовать для трансферного обучения. Здесь трансферное обучение используется для классифицирования изображений колоректальной гистологии. В то время как изначальный набор данных ImageNet содержит 1000 меток, наша результирующая модель вернет только 8 заданных нами возможных классов, в отличие от тысяч меток, присутствующих в ImageNet.

Загрузка предварительно натренированной модели и ее использование для получения классификаций на изначальных метках, на которых модель была натренирована, не являются трансферным обучением. Трансферное обучение идет дальше еще на один шаг вперед, заменяя окончательные слои модели вашей собственной задачей предсказания.

Загруженная нами модель VGG будет нашей базовой моделью. Нам понадобится добавить несколько слоев, чтобы сгладить результат на выходе из узкого слоя и передать этот сглаженный результат в 8-элементный softmax-массив:

---

[30] См. **https://tfhub.dev/**.

```
global_avg_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_avg = global_avg_layer(feature_batch)

prediction_layer = tf.keras.layers.Dense(8, activation='softmax')
prediction_batch = prediction_layer(feature_batch_avg)
```

Наконец, мы можем использовать API Sequential для создания новой, основанной на трансферном обучении модели в качестве стека слоев:

```
histology_model = keras.Sequential([
    vgg_model,
    global_avg_layer,
    prediction_layer
])
```

Давайте обратимся к результату фукции `model.summary()` на нашей модели с использованием трансферного обучения:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Model)                (None, 4, 4, 512)         20024384
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 8)                 4104
=================================================================
Total params: 20,028,488
Trainable params: 4,104
Non-trainable params: 20,024,384
_________________________________________________________________
```

Важным моментом здесь является то, что единственные тренируемые параметры находятся после узкого слоя. В этом примере узким слоем являются признаковые векторы из модели VGG. После компиляции этой модели мы можем ее натренировать, используя наш набор гистологических снимков.

Предварительно натренированные векторные вложения

Хотя мы можем загрузить предварительно натренированную модель самостоятельно, мы также можем реализовать трансферное обучение, используя многочисленные предварительно натренированные модели, имеющиеся в TF Hub — библиотеке предварительно натренированных моделей (именуемых модулями). Указанные модули охватывают массу предметных областей данных и вариантов использования, включая классификацию, обнаружение объектов, машинный перевод и многое другое. В TensorFlow эти модули можно загрузить как слой, а затем добавить собственный классифицирующий слой сверху.

В целях ознакомления с тем, как работает TF Hub, давайте построим модель, которая классифицирует отзывы о кинофильмах как позитивные либо негативные. Сначала мы загрузим предварительно натренированную модель, основанную на

векторных вложениях и натренированную на крупном корпусе новостных статей. Эта модель может быть инстанцирована как `hub.KerasLayer`:

```
hub_layer = hub.KerasLayer(
    "https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1",
    input_shape=[], dtype=tf.string, trainable=True)
```

Поверх нее можно уложить дополнительные слои и построить классификатор:

```
model = keras.Sequential([
    hub_layer,
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

Теперь мы можем натренировать эту модель, передав ей на вход наш собственный текстовый набор данных. Результирующее предсказание будет представлять собой одноэлементный массив, указывающий на то, каким по мнению модели данный текст является: позитивным или негативным.

Почему это работает

Для того чтобы понять, почему трансферное обучение работает, давайте сначала рассмотрим аналогию. Когда ребенок учится говорить, ему демонстрируют многочисленные примеры и поправляют, если он что-то называет неправильно. Например, в первый раз, когда он учится распознавать кошку, он будет видеть, как его мама или папа показывает на кошку и произносит слово "кошка", и это повторение укрепляет связи в его мозге. Аналогично его поправляют, когда он говорит "кошка", имея в виду животное, которое не является кошкой. Когда ребенок учится распознавать собаку, ему не нужно начинать с нуля. Он может использовать процесс распознавания, подобный тому, который он использовал для кошки, но применять его к немного другой задаче. Благодаря этому ребенок закладывает основу для обучения в области других знаний. В дополнение к усвоению новых предметов он также учится узнавать новые вещи. Применение этих методов обучения к другим предметным областям примерно соответствует тому, как работает трансферное обучение.

Как это происходит в нейронных сетях? В типичной сверточной нейронной сети (CNN-сети) обучение является иерархическим. Первые слои учатся распознавать присутствующие на снимке края и контуры. В примере с кошкой это означает, что модель может выявлять на снимке участки, в которых край тела кошки встречается с фоном. Следующие слои в модели начинают понимать группы краев — возможно, что там будут два края, которые сходятся к верхнему левому углу снимка. Последние слои CNN могут собирать эти группы краев воедино, развивая понимание разных признаков снимка. В примере с кошкой модель может выявить два треугольных контура в верхней части снимка и два овальных контура под ними. С точки зрения человека, что эти треугольные контуры являются ушами, а овальные — глазами.

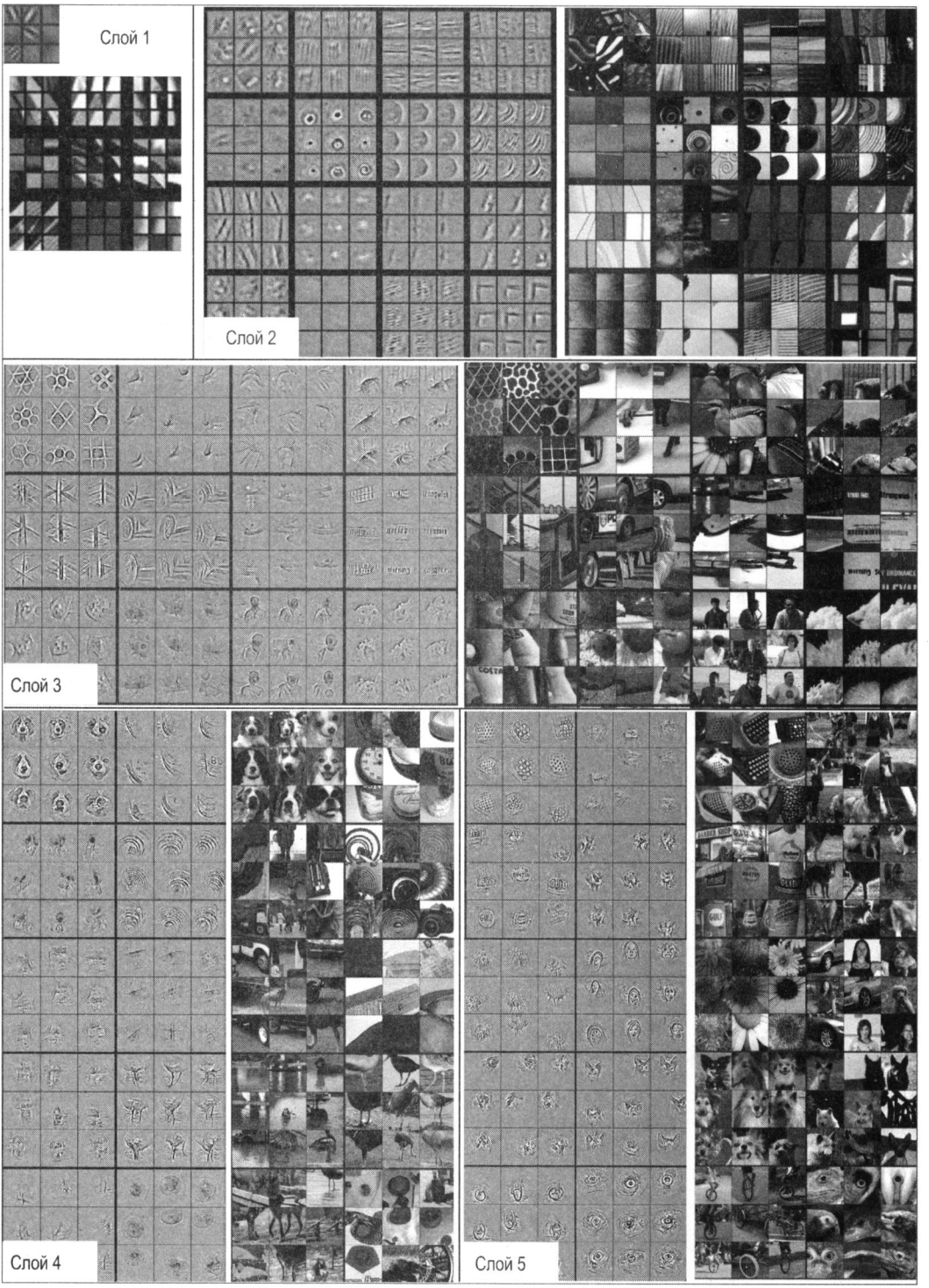


Рис. 4.14. Исследование Цайлера и Фергюса (2013) по деконструированию CNN-сети помогает нам визуализировать то, как CNN-сеть видит снимки в каждом слое сети

Мы можем визуализировать этот процесс на рис. 4.14, основываясь на исследованиях Цайлера (Zeiler) и Фергюса (Fergus)[31] по деконструированию CNN-сети, чтобы понять, какие признаки были активированы в каждом слое модели. Для каждого слоя в пятислойной CNN-сети деконструирование показывает признаковую карту снимка для данного слоя рядом с фактическим снимком. Это позволяет нам увидеть, каким образом восприятие снимка моделью продвигается вперед по мере того, как он движется по сети. Слои 1 и 2 распознают только края, слой 3 начинает распознавать объекты, а слои 4 и 5 могут распознавать фокусные точки на всем снимке.

Тем не менее всегда следует помнить о том, что для нашей модели все это просто группы пиксельных значений. Она не знает, что треугольные и овальные контуры являются ушами и глазами — она знает лишь то, как ассоциировать конкретные группы признаков с метками, на которых она была натренирована. При таком подходе процесс обучения моделью групп признаков, которые составляют кошку, не сильно отличается от обучения групп признаков, которые являются частью других объектов, таких как стол, гора или даже знаменитость. Для модели все это просто разные комбинации пиксельных значений, краев и контуров.

Компромиссы и альтернативы

Мы пока не обсуждали методы модифицирования весов изначальной модели во время реализации трансферного обучения. Здесь мы проштудируем два предназначенных для этого подхода: извлечение признаков и тонкую настройку. Мы также обсудим вопрос, почему трансферное обучение в первую очередь ориентировано на снимковые и текстовые модели, и рассмотрим взаимосвязь между векторными вложениями текстовых предложений и трансферным обучением.

Тонкая настройка против извлечения признаков

Извлечение признаков является подходом в трансферном обучении, при котором вы замораживаете веса всех слоев перед узким слоем и тренируете следующие слои на собственных данных и метках. Еще один вариант — применять тонкую настройку весов слоев предварительно натренированной модели. При выполнении тонкой настройки вы можете обновлять веса каждого слоя в предварительно натренированной модели либо только нескольких слоев непосредственно перед узким слоем. Тренировка модели, основанной на трансферном обучении с помощью тонкой настройки, обычно занимает больше времени, чем извлечение признаков. В приведенном выше примере классифицирования текста мы устанавливали `trainable=True` во время инициализации слоя TF Hub. Это и есть пример тонкой настройки.

Во время тонкой настройки веса исходных слоев модели обычно оставляют замороженными, т. к. эти слои были натренированы распознавать базовые признаки, которые часто встречаются во многих типах снимков. Например, в целях тонкой настройки модели MobileNet мы устанавливаем `trainable=False` только для под-

[31] См. **https://oreil.ly/VzRV_**.

множества слоев в модели, не делая каждый слой не поддающимся тренировке. Для тонкой настройки после 100-го слоя мы могли бы выполнить:

```
base_model = tf.keras.applications.MobileNetV2(input_shape=(160,160,3),
                                                include_top=False,
                                                weights='imagenet')
for layer in base_model.layers[:100]:
  layer.trainable = False
```

Один из рекомендуемых подходов к определению числа слоев для замораживания называется *прогрессивной тонкой настройкой*[32], и он предусматривает итеративное размораживание слоев после каждого тренировочного прогона с целью отыскания идеального для тонкой настройки числа слоев. Подход работает лучше всего и наиболее эффективно, если вы поддерживаете низкую скорость обучения (обычно 0,001) и относительно малое число итераций тренировки. В целях реализации прогрессивной тонкой настройки следует начать с размораживания только последнего слоя переданной модели (слоя, ближайшего к выходу) и рассчитывать потерю модели после тренировки. Затем, один за другим, разморозить еще несколько слоев, пока не будет достигнут входной слой либо пока потеря не начнет достигать плато. Эту процедуру следует использовать для получения числа слоев, подлежащих тонкой настройке.

Как определить, следует ли выполнять тонкую настройку или же замораживать все слои своей предварительно натренированной модели? В типичной ситуации, когда у вас малый набор данных, предварительно натренированную модель лучше всего использовать в качестве извлекателя признаков, чем выполнять ее тонкую настройку. Если вы выполняете перетренировку весов модели, которая, скорее всего, была натренирована на тысячах или миллионах примеров, то тонкая настройка может привести к тому, что обновленная модель будет переподогнана к вашему малому набору данных и потеряет более общую информацию, усвоенную из этих миллионов примеров. Хотя все зависит от ваших данных и вашей задачи предсказания, когда мы здесь говорим "малый набор данных", мы имеем в виду наборы данных с сотнями или несколькими тысячами тренировочных примеров.

Еще один фактор, который следует учитывать во время принятия решения о необходимости проведения тонкой настройки, заключается в степени подобия вашей задачи предсказания задаче изначальной предварительно натренированной модели, которую вы используете. Когда задача предсказания аналогична или служит продолжением предыдущей тренировки, как это было в нашей модели анализа настроений с использованием отзывов о кинофильмах, тонкая настройка способна привести к более точным результатам. Когда задача иная или наборы данных существенно отличаются, лучше всего заморозить все слои предварительно натренированной модели вместо их тонкой настройки. В табл. 4.1 кратко резюмируются ключевые моменты[33].

---

[32] См. **https://oreil.ly/fAv1S**.

[33] Дополнительные сведения см. в статье "Сверточные нейронные сети для визуального распознавания" (Convolutional Neural Networks for Visual Recognition, **https://oreil.ly/w109T**).

Таблица 4.1. *Критерии, помогающие выбирать между извлечением признаков и тонкой настройкой*

Критерий	Извлечение признаков	Тонкая настройка
Насколько большим является набор данных?	Малый	Большой
Является ли ваша задача предсказания такой же, как и у предварительно натренированной модели?	Разные задачи	Та же самая задача или аналогичная задача с одинаковым распределением меток по классам
Каков бюджет времени тренировки и вычислительных затрат?	Низкий	Высокий

В нашем текстовом примере предварительно натренированная модель была натренирована на корпусе новостных текстовых сообщений, но нашим примером использования был анализ настроений. Поскольку эти задачи различаются, мы должны использовать изначальную модель, скорее, в качестве извлекателя признаков, чем выполнять ее тонкую настройку. Примером других задач предсказания в области изображений может служить использование нашей модели MobileNet, натренированной на ImageNet, в качестве основы для выполнения трансферного обучения на наборе медицинских снимков. Хотя обе задачи предусматривают классифицирование изображений, природа снимков в каждом наборе данных очень индивидуальна.

Фокус внимания на снимковых и текстовых моделях

Возможно, вы заметили, что все примеры в этом разделе демонстрируют обработку снимковых и текстовых данных. Это обусловлено тем, что трансферное обучение в первую очередь предназначено для случаев, когда вы можете применить решение аналогичной задачи к задаче из той же предметной области данных. Однако модели, натренированные табличными данными, охватывают потенциально бесконечное число возможных задач предсказания и типов данных. Вы могли бы натренировать модель на табличных данных предсказывать цену билетов, которую следует назначать для вашего мероприятия, вероятность невозврата кем-то кредита, выручку вашей компании в следующем квартале, продолжительность поездки в такси и т. д. Конкретные данные для этих задач также невероятно разнообразны: задача о билетах зависит от информации о художниках и местах проведения выставки, задача с кредитами — от персональных доходов, а продолжительность поездки в такси — от структуры городского дорожного движения. По этим причинам трансфер ранее усвоенных знаний из одной табличной модели в другую характеризуется рядом трудностей.

Хотя трансферное обучение еще не так распространено на табличных данных, как для снимковых и текстовых предметных областей, архитектура TabNet[34] представляет собой новое научно-исследовательское направление в этой области. Большин-

[34] См. **https://oreil.ly/HI5XI**.

ство табличных моделей требуют значительных усилий по генерированию признаков по сравнению со снимковыми и текстовыми моделями. В TabNet применяется технический прием, в котором для обучения представлений табличных признаков сначала используется обучение без учителя, а затем эти усвоенные представления тонко настраиваются для предсказаний. При таком подходе в TabNet автоматизируется генерирование признаков для табличных моделей.

Векторные вложения слов и предложений

В нашем обсуждении темы векторных вложений текста мы до сих пор ссылались главным образом на векторное вложение *слов*. Еще одним типом векторного вложения текста является векторное вложение *предложений*. Там, где векторные вложения слов представляют отдельные слова в векторном пространстве, векторные вложения предложений представляют целые предложения. Следовательно, векторные вложения слов являются контекстно-агностическими. Давайте посмотрим, как это происходит со следующим предложением:

I've left you fresh baked cookies on the left side of the kitchen counter.
(Свежеиспеченное печенье найдешь на кухне на левой стороне стола.)

Обратите внимание, что слово left встречается в этом предложении дважды: сначала как глагол, а затем как прилагательное. Если бы нам нужно было сгенерировать векторные вложения слов для этого предложения, мы бы получили отдельный массив для каждого слова. В векторном вложении слов массив для обоих экземпляров слова left будет одинаковым. Однако, используя векторные вложения на уровне предложений, мы бы получили один-единственный вектор, который представлял бы все предложение. Существует несколько подходов к генерированию векторных вложений предложений — от усреднения векторных вложений слов предложения до тренировки модели обучения с учителем на крупном корпусе текста для генерирования векторных вложений.

Как это связано с трансферным обучением? Последний метод — тренировка модели обучения с учителем для создания векторных вложений на уровне предложений — на самом деле является формой трансферного обучения. Этот подход используется универсальным кодировщиком предложений Universal Sentence Encoder[35] компании Google (доступен в TF Hub) и BERT[36]. Эти методы отличаются от векторного вложения слов тем, что они выходят за рамки простого предоставления весовой справочной таблицы для отдельных слов. Вместо этого они были построены путем тренировки модели на крупном наборе разнообразного текста с целью понимания смысла, передаваемого последовательностями слов. При таком подходе они по своему дизайну предназначены для переноса на разные естественно-языковые задачи и, следовательно, могут использоваться для строительства моделей, в которых реализуется трансферное обучение.

---

[35] См. **https://oreil.ly/Y0Ry9**.

[36] См. **https://oreil.ly/l_gQf**.

ПАТТЕРН 14. Распределительная стратегия

В паттерне "Распределительная стратегия" (Distribution Strategy) цикл тренировки выполняется в требуемом масштабе на нескольких воркерах, нередко с кэшированием, аппаратным ускорением и параллелизацией.

Постановка задачи

В наши дни крупные нейронные сети обычно имеют миллионы параметров и тренируются на массивных объемах данных. На самом деле, как было показано, увеличение масштаба глубокого обучения по отношению к числу тренировочных примеров, числу параметров модели или и тому и другому вместе резко улучшает результативность модели. Однако по мере увеличения размера моделей и данных требования к вычислительным мощностям и памяти пропорционально возрастают, что делает время, необходимое для тренировки этих моделей, одной из самых больших проблем глубокого обучения.

Графические процессоры (GPU) обеспечивают значительный вычислительный скачок и приближают время тренировки глубоких нейронных сетей умеренного размера к пределам достигаемости. Однако в случае очень больших моделей, тренируемых на массивных объемах данных, отдельных графических процессоров недостаточно, чтобы сократить время тренировки. Например, на момент написания этой книги тренировка ResNet-50 на эталонном наборе данных ImageNet в течение 90 эпох на одном графическом процессоре NVIDIA M40 требует 1018 операций одинарной прецизионности и занимает 14 дней. Поскольку ИИ все больше и больше используется для решения задач в сложных областях, а библиотеки с открытым исходным кодом, такие как Tensorflow и PyTorch, делают создание основанных на глубоком обучении моделей доступнее, крупные нейронные сети, сравнимые с ResNet-50, стали нормой.

И в этом проблема. Если тренировка нейронной сети занимает две недели, то вам придется подождать, прежде чем вы сможете повторить процесс, реализовав в нем новые идеи, или поэкспериментировать с доработкой настроек. Более того, в случае некоторых сложных задач, таких как медицинская визуализация, автономное вождение или естественно-языковой перевод, не всегда есть возможность подразделить задачу на меньшие компоненты или работать только с подмножеством данных. Только с полным набором данных вы можете оценить, работает ли что-то или нет.

Время тренировки переводится в деньги в буквальном смысле. В мире бессерверного машинного обучения вместо покупки собственного дорогостоящего графического процессора можно отправлять тренировочные задания в облачный сервис, где с клиента взимается плата за время тренировки. Стоимость тренировки модели, будь то плата за графический процессор или плата за облачный сервис, быстро нарастает.

Существует ли подход, который позволял бы ускорять тренировку этих крупных нейронных сетей?

Решение

Одним из подходов к ускорению тренировки является использование распределительных стратегий в цикле тренировки. Существуют разные технические приемы распределения, но общая идея состоит в том, чтобы разделять усилия по тренировке модели между многочисленными машинами. Это можно сделать двумя способами: *посредством параллелизма данных* и *посредством параллелизма модели*. При параллелизме данных разделяются вычисления, которые распределяются между разными машинами, и разные воркеры выполняют тренировку на разных подмножествах тренировочных данных. При параллелизме модели разделяется модель, и разные воркеры выполняют вычисления для разных частей модели. В этом разделе книги мы сосредоточимся на параллелизме данных и покажем реализации в TensorFlow с использованием библиотеки `tf.distribute.Strategy`. Мы обсудим параллелизм модели в *разд. "Компромиссы и альтернативы" далее в этой главе*.

Для реализации параллелизма данных должен существовать метод, позволяющий разным воркерам вычислять градиенты и делиться этой информацией, чтобы делать обновления параметров модели. За счет него обеспечиваются согласованность всех вокреров и работа каждого шага градиента на тренировку модели. В широком смысле параллелизм данных может осуществляться как синхронно, так и асинхронно.

Синхронная тренировка

В синхронной тренировке воркеры осуществляют тренировку на разных срезах входных данных параллельно, и значения градиента агрегируются в конце каждого тренировочного шага. Указанная работа выполняется с помощью алгоритма AllReduce. Из этого следует, что каждый воркер, обычно графический процессор, имеет копию модели прямо на устройстве, и для одного шага стохастического градиентного спуска (SGD) мини-пакет данных разделяется между каждым отдельным воркером. Каждое устройство выполняет прямой проход со своей частью мини-пакета и вычисляет градиенты для каждого модельного параметра. Эти локально вычисленные градиенты затем собираются с каждого устройства и агрегируются (например, усредняются), производя единое обновление градиента для каждого параметра[37]. Центральный сервер хранит самую последнюю копию модельных параметров и выполняет шаг градиента в соответствии с градиентами, полученными от нескольких воркеров. Как только параметры модели будут обновлены в соответствии с этим агрегированным шагом градиента, новая модель отправляется обратно воркерам вместе с еще одной частью следующего мини-пакета, и процесс повторяется. На рис. 4.15 показана типичная архитектура AllReduce для синхронного распределения данных.

Как и в случае любой стратегии обеспечения параллелизма, такая архитектура требует дополнительных накладных расходов на управление временем и обменом

[37] Алгоритм AllReduce сводит целевые массивы во всех процессах-воркерах к одному массиву и возвращает результирующий массив всем воркерам. — *Прим. перев.*

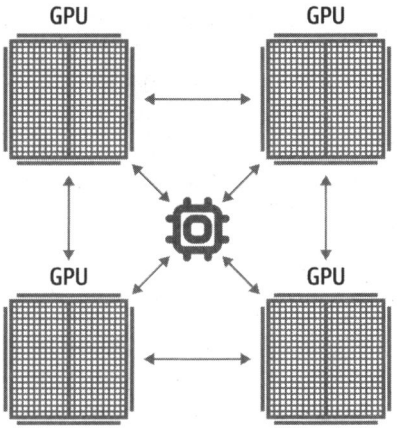


Рис. 4.15. Во время синхронной тренировки каждый воркер содержит копию модели и вычисляет градиенты, используя срез мини-пакета тренировочных данных

информацией между воркерами. Крупные модели могут стать причиной узких мест в операциях ввода-вывода при передаче данных от центрального процессора к графическому процессору во время тренировки, а медленные сети могут обусловливать задержки.

В TensorFlow зеркальная стратегия `tf.distribute.MirroredStrategy` поддерживает синхронную распределенную тренировку на многочисленных графических процессорах на одной и той же машине. Каждый параметр модели зеркально отражается во всех воркерах и хранится как единая концептуальная переменная, именуемая `MirroredVariable`. Во время шага алгоритма AllReduce все градиентные тензоры становятся доступными на каждом устройстве. Это помогает значительно снизить накладные расходы на синхронизацию. Помимо описанной выше реализации, существуют и другие реализации алгоритма AllReduce, во многих из которых используется библиотека примитивов NVIDIA NCCL[38].

В целях реализации этой зеркальной стратегии в Keras сначала следует создать экземпляр зеркальной распределительной стратегии, а затем переместить создание и компиляцию модели внутрь области видимости этого экземпляра. В следующем фрагменте исходного кода показано, как использовать `MirroredStrategy` во время тренировки трехслойной нейронной сети:

```
mirrored_strategy = tf.distribute.MirroredStrategy()
with mirrored_strategy.scope():
    model = tf.keras.Sequential([tf.keras.layers.Dense(32, input_shape=(5,)),
                                 tf.keras.layers.Dense(16, activation='relu'),
                                 tf.keras.layers.Dense(1)])
    model.compile(loss='mse', optimizer='sgd')
```

При построении модели внутри этой области модельные параметры создаются не как обычные переменные, а как зеркально отраженные. Когда дело доходит до под-

[38] См. **https://oreil.ly/HX4NE**.

гонки модели на наборе данных, все выполняется точно так же, как и раньше. Исходный код модели остается прежним! Для того чтобы активировать распределенную тренировку, нужно всего лишь обернуть модельный код в область распределительной стратегии, и на этом все. Стратегия `MirroredStrategy` занимается реплицированием модельных параметров на доступных графических процессорах, агрегированием градиентов и многим другим. В целях тренировки и оценивания модели мы, как обычно, просто вызываем метод `fit()` или `evaluate()`:

```
model.fit(train_dataset, epochs=2)
model.evaluate(train_dataset)
```

Во время тренировки каждый пакет входных данных делится поровну между несколькими воркерами. Например, если вы используете два графических процессора, то пакет размером 10 будет разделен между двумя графическими процессорами, каждый из которых на каждом шаге будет получать 5 тренировочных примеров. В рамках Keras также существуют и другие стратегии синхронного распределения, такие как `CentralStorageStrategy` и `MultiWorkerMirroredStrategy`. Стратегия `MultiWorkerMirroredStrategy` обеспечивает возможность выполнять распределение не только на графических процессорах на одной машине, но и на многочисленных машинах. В стратегии `CentralStorageStrategy` модельные переменные не отражаются зеркально; вместо этого они помещаются на центральный процессор (CPU), а операции реплицируются на все локальные графические процессоры. Таким образом, обновления переменных происходят только в одном месте.

При выборе между разными распределительными стратегиями наилучший вариант зависит от топологии вашего компьютера и от того, насколько быстро центральные и графические процессоры могут обмениваться между собой информацией. В табл. 4.2 кратко резюмировано то, как разные описанные здесь стратегии выглядят с точки зрения этих критериев.

***Таблица 4.2.** Выбор между распределительными стратегиями, зависящими от топологии компьютера и скорости обмена информацией между центральными и графическими процессорами*

	Более быстрое соединение CPU — GPU	**Более быстрое соединение GPU — GPU**
Одна машина с многочисленными GPU	`CentralStorageStrategy`	`MirroredStrategy`
Много машин с многочисленными GPU	`MultiWorkerMirroredStrategy`	`MultiWorkerMirroredStrategy`

Параллелизм распределенных данных в PyTorch

В исходном коде PyTorch всегда используется контейнер параллелизма распределенных данных `DistributedDataParallel` независимо от того, сколько у вас графических процессоров — один или много, и какое количество машин требуется для работы модели — одна или несколько. В отличие от TensorFlow, распределительная стратегия здесь определяется тем, как и где вы запускаете процессы, как проводите отбор и загрузку данных и т. д.

Сначала мы инициализируем процесс и ждем запуска других процессов, а затем организуем обмен данными с помощью инструкции:

```
torch.distributed.init_process_group(backend="nccl")
```

Потом указываем номер устройства, получив ранг из командной строки. Ранг, равный 0, означает главный процесс, а 1, 2, 3, ... — воркеров:

```
device = torch.device("cuda:{}".format(local_rank))
```

Модель создается, как обычно, в каждом процессе, но отправляется на это устройство. Распределенная версия модели, которая будет обрабатывать свой шард[39] пакета, создается с помощью контейнера `DistributedDataParallel`:

```
model = model.to(device)
ddp_model = DistributedDataParallel(model, device_ids=[local_rank],
                                    output_device=local_rank)
```

Сами данные шардируются с помощью `DistributedSampler`, и каждый пакет данных также отправляется на устройство:

```
sampler = DistributedSampler(dataset=trainds)
loader = DataLoader(dataset=trainds, batch_size=batch_size,
                    sampler=sampler, num_workers=4)
...
for data in train_loader:
    features, labels = data[0].to(device), data[1].to(device)
```

Когда запускается тренер PyTorch, ему сообщается суммарное число узлов и его собственный ранг:

```
python -m torch.distributed.launch --nproc_per_node=4 \
       --nnodes=16 --node_rank=3 --master_addr="192.168.0.1" \
       --master_port=1234 my_pytorch.py
```

Если число узлов равно 1, то мы имеем эквивалент стратегии `MirroredStrategy` TensorFlow, а если число узлов больше 1, то мы имеем эквивалент стратегии `MultiWorkerMirroredStrategy` TensorFlow. Если число процессов в расчете на узел и число узлов равны 1, то мы имеем стратегию `OneDeviceStrategy`. Оптимизированный обмен информацией для всех этих случаев, при условии что он поддерживается бэкендом (в данном случае, NCCL), передается в группу `init_process_group`.

Асинхронная тренировка

В асинхронной тренировке[40] воркеры выполняют тренировку на разных срезах входных данных независимо, а модельные веса и параметры обновляются асинхронно, как правило, посредством архитектуры на основе сервера параметров. Из

[39] Шардирование (sharding) — это горизонтальное разделение данных в базе данных или поисковой системе с целью распределения нагрузки. В рамках указанной операции каждый подраздел называется шардом (осколком). — *Прим. перев.*

[40] См. **https://oreil.ly/Wkk5B**.

этого следует, что ни один воркер не ждет обновлений модели от других воркеров. В архитектуре на основе сервера параметров существует один-единственный сервер параметров, который управляет текущими значениями модельных весов (рис. 4.16).

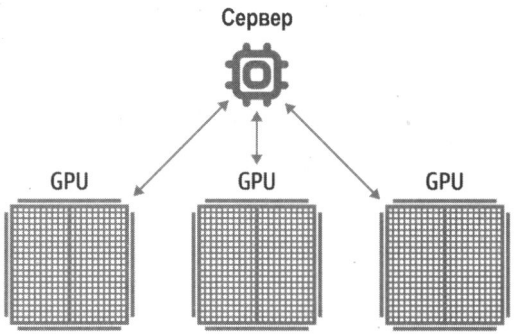


Рис. 4.16. В асинхронной тренировке каждый воркер выполняет шаг градиентного спуска с частью мини-пакета. Ни один воркер не ждет обновления модели от других воркеров

Как и в синхронной тренировке, мини-пакет данных распределяется между каждым отдельным воркером для каждого шага стохастического градиентного спуска. Каждое устройство выполняет прямой проход со своей порцией мини-пакета и вычисляет градиенты для каждого модельного параметра. Эти градиенты отправляются на сервер параметров, который выполняет обновление параметров, а затем возвращает новые модельные параметры воркеру с еще одной частью следующего мини-пакета.

Ключевое различие между синхронной и асинхронной тренировкой состоит в том, что сервер параметров не выполняет всю работу алгоритма AllReduce. Вместо этого он периодически вычисляет новые модельные параметры, основываясь на тех изменениях градиента, которые он получил с момента последнего вычисления. В типичной ситуации асинхронное распределение обеспечивает более высокую пропускную способность, чем синхронная тренировка, потому что медленный воркер не блокирует продвижение тренировочных шагов. Если один воркер аварийно отказывает, то тренировка продолжается, как и планировалось, с другими воркерами до тех пор, пока вышедший из строя не перезагрузится. Как следствие, во время тренировки некоторые части мини-пакета могут быть потеряны, что слишком затрудняет точное отслеживание числа обработанных эпох данных. Это еще одна причина, по которой во время тренировки крупных распределенных заданий вместо эпох мы обычно указываем виртуальные эпохи; обсуждение темы виртуальных эпох *см. в разд. "Паттерн 12. Контрольные точки" ранее в этой главе.*

Кроме того, поскольку между обновлениями весов нет синхронизации, существует возможность, что один воркер обновляет модельные веса, основываясь на устаревшем состоянии модели. Однако на практике это не проблема. Как правило, крупные нейронные сети тренируются в течение многочисленных эпох, и эти малые расхождения, в конце концов, становятся незначительными.

В Keras стратегия `ParameterServerStrategy` реализует асинхронную, основанную на сервере параметров тренировку на многочисленных машинах. При использовании этого распределения некоторые машины обозначаются как воркеры, а другие — как серверы параметров. Серверы параметров содержат каждую модельную переменную, а вычисления выполняются на воркерах, как правило, графических процессорах[41].

Реализация аналогична реализации других распределительных стратегий в Keras. Например, в вашем исходном коде вы просто замените вызов `MirroredStrategy()` на `ParameterServerStrategy()`.

Заслуживает упоминания еще одна поддерживаемая в Keras распределительная стратегия — `OneDeviceStrategy`. Она помещает любые переменные, созданные в ее области видимости, на указанное устройство. Данная стратегия особенно полезна в качестве подхода к тестированию вашего исходного кода перед тем, как вы решите переключиться на другие стратегии, которые занимаются фактическим распределением на многочисленные устройства/машины.

Синхронная и асинхронная тренировки имеют свои преимущества, и выбор между ними часто сводится к аппаратным и сетевым ограничениям.

Синхронная тренировка особенно уязвима для медленных устройств или плохого сетевого соединения, поскольку тренировка будет останавливаться в ожидании обновлений от всех воркеров. Это означает, что синхронное распределение предпочтительнее, когда все устройства находятся на одном хосте и есть быстрые устройства (например, TPU или GPU) с сильными связями. С другой стороны, асинхронное распределение предпочтительнее, если есть много маломощных или ненадежных воркеров. Если один воркер аварийно отказывает или зависает при возврате обновления градиента, то он не будет останавливать цикл тренировки. Единственное ограничение накладывается операциями ввода-вывода.

Почему это работает

Крупные, сложные нейронные сети требуют огромных объемов тренировочных данных для своей эффективной работы. Схемы распределенной тренировки резко увеличивают пропускную способность обрабатываемых этими моделями данных и могут действительно сокращать время тренировки с недель до часов. Совместное использование ресурсов между заданиями воркеров и сервера параметров приводит к резкому увеличению пропускной способности обрабатываемых данных. На рис. 4.17 сравнивается пропускная способность обрабатываемых тренировочных данных, в данном случае изображений, в разных условиях распределения[42]. Наибо-

---

[41] В рамках указанной архитектуры существуют два типа сущностей: воркеры и серверы. Параметры модели хранятся распределенно на сервере (серверах), тогда как тренировочные данные распределяются между воркерами. Сервер предоставляет воркерам интерфейс хранилища в формате "ключ — значение" для доступа к параметрам модели. Серверу параметров вменяется в обязанность поддержание текущего значения параметров. — *Прим. перев.*

[42] Кампос В. и др. Распределенные стратегии тренировки алгоритма глубокого обучения для компьютерного зрения на распределенном GPU-кластере (Victor Campos. Distributed training strategies for a computer vision deep learning algorithm on a distributed GPU cluster // International Conference on Computational Science, ICCS 2017. — 2017. — June 12–14).

лее примечательно, что пропускная способность возрастает с увеличением числа узлов-воркеров, и хотя серверы параметров выполняют задачи, не связанные с вычислениями, выполняемыми на воркерах графического процессора, разделение рабочей нагрузки между бóльшим числом машин является наиболее выгодной стратегией.

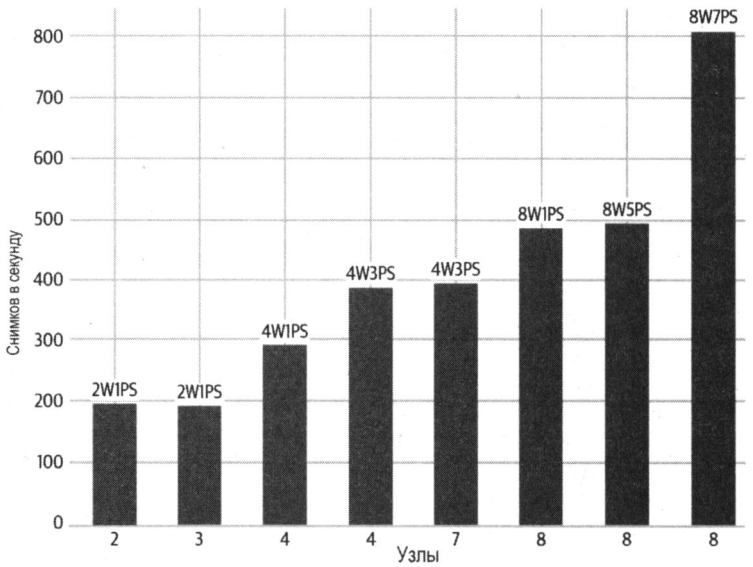


Рис. 4.17. Сравнение пропускной способности между разными условиями распределения. Здесь 2W1PS обозначает двух воркеров и один сервер параметров

Вдобавок параллелизация данных уменьшает время, необходимое для схождения при выполнении тренировки. В аналогичном исследовании было показано, что увеличение числа воркеров ведет к минимальной потере гораздо быстрее[43]. На рис. 4.18 сравнивается время до минимума для разных распределительных стратегий. По

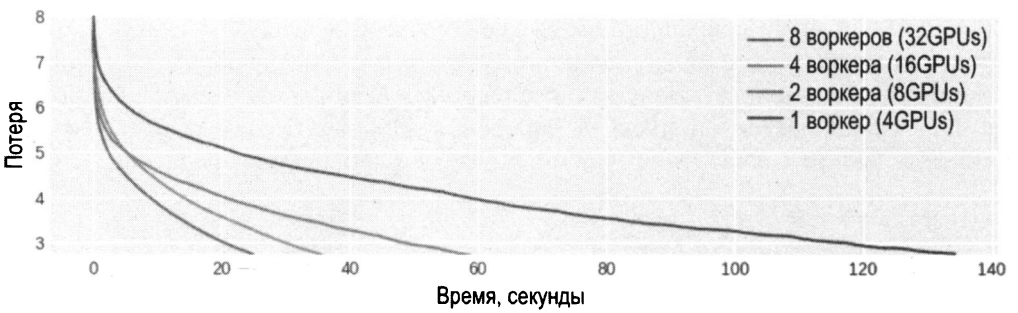


Рис. 4.18. По мере увеличения числа графических процессоров время схождения при выполнении тренировки уменьшается

[43] См. там же.

мере увеличения числа воркеров время до минимума тренировочной потери резко уменьшается, демонстрируя почти пятикратное ускорение с 8 воркерами по сравнению с 1.

Компромиссы и альтернативы

Помимо параллелизма данных, необходимо учитывать и иные аспекты распределения, такие как параллелизм модели, другие ускорители тренировки (например, TPU) и другие соображения (например, пределы, связанные с операциями ввода-вывода, и размер пакета).

Параллелизм модели

В некоторых случаях нейронная сеть настолько велика, что не может поместиться в памяти одного устройства; например, нейронный машинный перевод компании Google (Google's Neural Machine Translation)[44] имеет миллиарды параметров. Для того чтобы натренировать столь большие модели, их нужно разделить между многочисленными устройствами[45] (рис. 4.19). Это называется *параллелизмом модели*. Разделяя части сети и связанные с ними вычисления между многочисленными ядрами, рабочую нагрузку на вычисления и память распределяют по многочисленным устройствам. Каждое устройство оперирует одним и тем же мини-пакетом данных во время тренировки, но выполняет вычисления, связанные только с его отдельными компонентами модели.

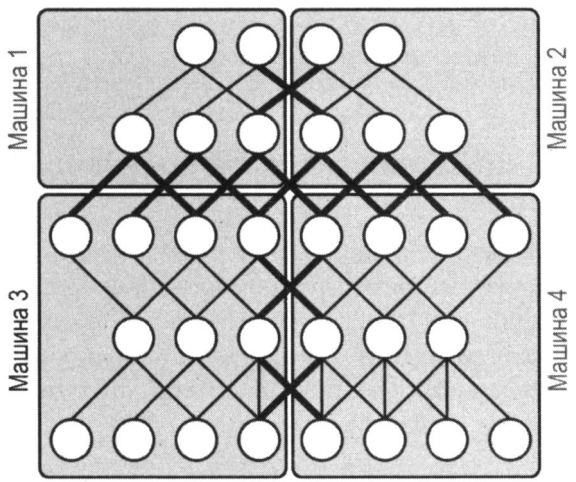


Рис. 4.19. Параллелизм модели распределяет модель по многочисленным устройствам

[44] См. **https://oreil.ly/xL4Cu**.

[45] Дин Дж. и др. Крупномасштабные распределенные глубокие сети (Jeffrey Dean. Large Scale Distributed Deep Networks // NIPS Proceedings — 2012).

Параллелизм модели или параллелизм данных?

Априори ни одна из схем не лучше другой. У каждой есть свои преимущества. Как правило, архитектура модели определяет, что лучше использовать: параллелизм данных или параллелизм модели.

В частности, параллелизм модели повышает эффективность, когда объем вычислений в расчете на активность нейрона является большим, например в широких моделях с большим числом полносвязных слоев. Это обусловлено тем, что именно значение нейрона передается между разными компонентами модели. Вне парадигмы тренировки параллелизм модели обеспечивает дополнительную выгоду от обработки запросов очень крупными моделями там, где требуется низкая задержка. Распределение вычислений крупной модели между многочисленными устройствами может значительно сократить суммарное время вычислений при генерировании онлайновых предсказаний.

В то же время параллелизм данных более эффективен, когда объем вычислений в расчете на вес является большим, например, когда задействованы сверточные слои. Это обусловлено тем, что именно веса моделей (и их градиентные обновления) передаются между разными воркерами.

В зависимости от масштаба вашей модели и задачи может потребоваться использование и того и другого. Mesh TensorFlow[46] — это библиотека[47], оптимизированная под распределенное глубокое обучение, которая сочетает синхронный параллелизм данных с параллелизмом модели. Она реализована в качестве слоя над TensorFlow и позволяет легко распределять тензоры по разным размерностям. Распределение по пакетному слою является синонимом параллелизма данных, в то время как распределение по любой другой размерности, например размерности, представляющей размер скрытого слоя, обеспечивает параллелизм модели.

Микросхемы ASIC для более высокой производительности при меньших затратах

Еще один подход к ускорению процесса тренировки состоит в ускорении опорного аппаратного обеспечения, например, с помощью конкретно-прикладных интегральных схем (application-specific integrated circuit, ASIC). В машинном обучении это относится к аппаратным компонентам, разработанным специально для оптимизирования производительности при больших матричных вычислениях, лежащих в основе цикла тренировки. Тензорные процессоры (TPU) в Google Cloud представляют собой микросхемы ASIC, которые можно использовать как для тренировки моделей, так и для генерирования предсказаний. Подобным образом, Microsoft Azure предлагает программируемую пользователем вентильную матрицу Azure FPGA (field-programmable gate array), которая также является прикладным чипом

[46] См. **https://oreil.ly/svS4q**.

[47] См. **https://github.com/tensorflow/mesh**.

машинного обучения, как и ASIC, за исключением того, что его можно при необходимости переконфигурировать. Эти чипы способны значительно сокращать комбинированную метрику времени до точности (time-to-accuracy) при тренировке крупных, сложных нейросетевых моделей. Модель, тренировка которой занимает две недели на графических процессорах, может сходиться за часы на тензорных процессорах.

Есть и другие преимущества использования прикладных чипов машинного обучения. Например, в то время как ускорители (графические процессоры, программируемые пользователем вентильные матрицы, тензорные процессоры и т. д.) стали быстрее, ввод-вывод превратился в весьма узкое место в тренировке ML. Многие процессы тренировки тратят циклы на ожидание чтения и перемещения данных в ускоритель и ожидание обновления градиента для выполнения алгоритма AllReduce. Модули TPU имеют высокоскоростное межсоединение, поэтому, как правило, не нужно беспокоиться о накладных расходах на связь внутри модуля (модуль состоит из тысяч TPU). Кроме того, имеется много дискового пространства, а это означает, что можно извлекать данные упреждающе и обращаться к процессору менее часто. Поэтому следует обрабатывать более крупные пакеты, чтобы в полной мере воспользоваться преимуществами чипов с высокоскростной памятью и высокоскоростным межсоединением, таких как TPU.

С точки зрения распределенной тренировки, стратегия `TPUStrategy` позволяет выполнять распределенные тренировочные задания на тензорных процессорах. Стратегия `TPUStrategy` по сути аналогична стратегии `MirroredStrategy`, хотя тензорные процессоры имеют собственную реализацию алгоритма AllReduce.

Использование стратегии `TPUStrategy` подобно использованию других распределительных стратегий в TensorFlow. Одно из отличий заключается в том, что сначала вы должны настроить регулятор `TPUClusterResolver`, который указывает на местоположение TPU. В настоящее время TPU доступны для бесплатного использования в Google Colab, и там вам не нужно указывать какие-либо аргументы для адреса `tpu_address`:

```
cluster_resolver =
tf.distribute.cluster_resolver.TPUClusterResolver(tpu=tpu_address)
tf.config.experimental_connect_to_cluster(cluster_resolver)
tf.tpu.experimental.initialize_tpu_system(cluster_resolver)
tpu_strategy = tf.distribute.experimental.TPUStrategy(cluster_resolver)
```

Выбор размера пакета

Еще одним важным фактором, который следует учитывать, является размер пакета. В частности, при синхронном параллелизме данных, когда модель особенно велика, предпочтительнее уменьшать суммарное число итераций тренировки, поскольку каждый тренировочный шаг требует совместного использования обновленной модели разными воркерами, что приводит к замедлению времени передачи. Таким образом, важно максимально увеличивать размер мини-пакета, чтобы иметь возможность достигать той же самой производительности за меньшее число шагов.

Однако, как было отмечено[48], очень крупные размеры пакетов отрицательно влияют на скорость схождения стохастического градиентного спуска, а также на качество окончательного решения[49]. На рис. 4.20 показано, что одно только увеличение размера пакета в конечном счете приводит к увеличению валидационной ошибки top-1. Фактически авторы исследования утверждают, что для поддержания низкой валидационной ошибки при одновременном сокращении времени распределенной тренировки необходима линейная нормализация скорости обучения как функции от крупного размера пакета.

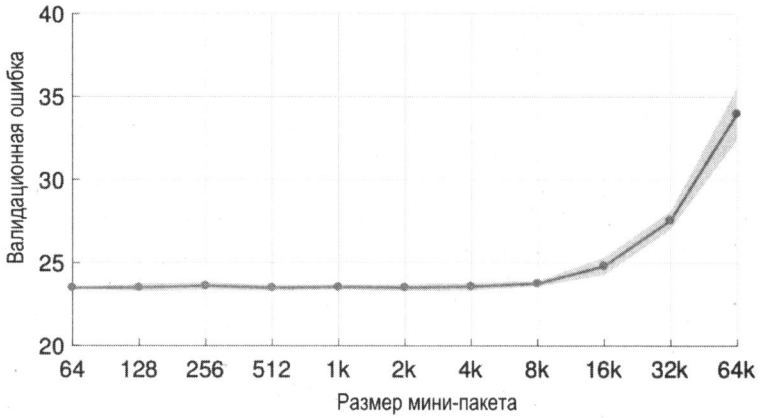


Рис. 4.20. Как было показано, крупные размеры пакетов отрицательно влияют на качество окончательной натренированной модели

Таким образом, задание размера мини-пакета в контексте распределенной тренировки само по себе является сложной оптимизационной задачей, т. к. влияет и на статистическую точность (обобщение), и на аппаратную эффективность использования модели. В смежной исследовательской работе[50], посвященной этой оптимизации, приведен технический прием послойной адаптивной крупнопакетной оптимизации под названием LAMB, который смог сократить время тренировки BERT с 3 дней до 76 минут.

Минимизация ожидания ввода-вывода

Графические и тензорные процессоры могут обрабатывать данные гораздо быстрее, чем центральные процессоры, и при использовании распределительных стратегий с многочисленными ускорителями конвейеры ввода-вывода едва поспевают за ними, создавая узкое место для выполнения более эффективной тренировки. В частности, перед завершением тренировочного шага данные для следующего шага

[48] См. **https://oreil.ly/FOtIX**.

[49] Гоял П. и др. Точный стохастический градиентный спуск с крупным мини-пакетом: тренировка ImageNet за 1 час (Priya Goyal et al. Accurate, large minibatch SGD: training ImageNet in 1 hour. — 2017, arXiv: 1706.02677v2 [cs.CV]).

[50] См. **https://oreil.ly/yeALI**.

недоступны для обработки. Это показано на рис. 4.21. Центральный процессор манипулирует входным конвейером: считывает данные из хранилища, предобрабатывает и отправляет в ускоритель для вычисления. По мере того как распределительные стратегии ускоряют тренировку, всё острее ощущается необходимость иметь эффективные конвейеры ввода данных для полного задействования имеющихся вычислительных мощностей.

Рис. 4.21. При распределенной тренировке на нескольких GPU/TPU необходимо иметь эффективные конвейеры ввода данных

Это достигается несколькими путями, включая использование оптимизированных форматов файлов, таких как TFRecords, и строительство конвейеров данных с применением API `tf.data` в TensorFlow. API `tf.data` позволяет обрабатывать крупные объемы данных и имеет встроенные преобразования, полезные для создания гибких и эффективных конвейеров. Например, `tf.data.prefetch` накладывает предобработку и модельное исполнение тренировочного шага друг на друга, вследствие чего, пока модель исполняет тренировочный шаг N, конвейер ввода данных читает и готовит данные для тренировочного шага $N + 1$ (рис. 4.22).

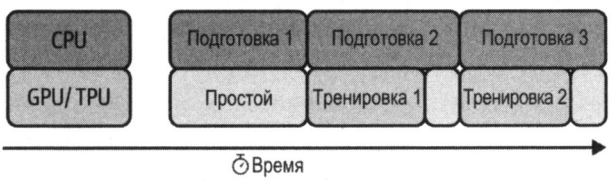


Рис. 4.22. Предварительная доставка накладывает предобработку и исполнение модели друг на друга, и благодаря этому пока модель исполняет один тренировочный шаг, конвейер ввода данных читает и готовит данные для следующего

ПАТТЕРН 15. Гиперпараметрическая настройка

В паттерне "Гиперпараметрическая настройка" (Hyperparameter Tuning) сам цикл тренировки вставляется в метод оптимизации для поиска оптимального набора гиперпараметров модели.

Постановка задачи

В машинном обучении тренировка модели предусматривает поиск оптимального набора точек ветвления (в случае деревьев решений), весов (в случае нейронных сетей) или опорных векторов (в случае опорно-векторных машин). Мы называем их

параметрами *модели*. Однако для того чтобы провести тренировку модели и найти оптимальные параметры модели, нам нередко приходится жестко кодировать различные величины. Например, мы можем решить, что максимальная глубина дерева будет равна 5 (в случае деревьев решений) или активационной функцией будет ReLU (для нейронных сетей), либо выбрать набор задействуемых нами ядер (в SVM-машинах). Эти параметры называются *гиперпараметрами*.

Параметры модели относятся к весам и смещенностям, усвоенным вашей моделью. Вы не имеете прямого контроля над параметрами модели, поскольку они в значительной степени зависят от ваших тренировочных данных, архитектуры модели и многих других факторов. Другими словами, задать параметры вручную невозможно. Веса вашей модели инициализируются случайными значениями, а затем оптимизируются вашей моделью по мере прохождения итераций тренировки. Гиперпараметры относятся к любым параметрам, которые вы, как строитель модели, можете контролировать. Они включают такие значения, как скорость обучения, число эпох, число слоев в вашей модели и многое другое.

Ручная настройка

Поскольку значения для разных гиперпараметров можно отбирать вручную, вашим первым инстинктивным подходом при поиске оптимальной комбинации значений гиперпараметров будет метод проб и ошибок. Этот вариант может сработать для моделей, которые тренируются за секунды или минуты, но быстро начнет дорожать на более крупных моделях, которые требуют значительного времени тренировки и инфраструктуры. Представьте, что вы тренируете модель классифицирования снимков, тренировка которой занимает несколько часов на графических процессорах. Вы определяетесь с несколькими значениями гиперпараметров, чтобы их попробовать, а потом ждете результатов первого тренировочного прогона. Основываясь на этих результатах, вы дорабатываете настройку гиперпараметров, снова тренируете модель, сравниваете результаты с первым прогоном, а затем определяетесь с лучшими значениями гиперпараметров, глядя на тренировочный прогон с лучшими метриками.

С таким подходом есть несколько проблем. Во-первых, вы потратили почти день и много вычислительных часов на эту задачу. Во-вторых, невозможно никак узнать, что вы пришли к оптимальной комбинации значений гиперпараметров. Вы испытали только две разные комбинации, и поскольку вы меняли несколько значений одновременно, вы не знаете, какой параметр оказал наибольшее влияние на результативность. Даже при дополнительных попытках использование этого подхода быстро израсходует ваше время и вычислительные ресурсы и, возможно, не даст самых оптимальных значений гиперпараметров.

Мы используем термин *"испытание"* для обозначения одного тренировочного прогона с набором значений гиперпараметров.

Поиск по сетке и комбинаторный взрыв

Более структурированная версия описанного выше метода проб и ошибок называется *поиском по сетке* (grid search). Во время реализации гиперпараметрической настройки с помощью поиска по сетке мы выбираем список возможных значений, подлежащих испытанию для каждого гиперпараметра, который мы хотим оптимизировать. Например, предположим, что в модели `RandomForestRegressor()` библиотеки scikit-learn мы хотим испытать следующую комбинацию значений для модельных гиперпараметров `max_depth` и `n_estimators`:

```
grid_values = {
  'max_depth': [5, 10, 100],
  'n_estimators': [100, 150, 200]
}
```

Применяя поиск по сетке, мы испытываем каждую комбинацию указанных значений, а затем используем комбинацию, которая дает наилучшую метрику оценивания на нашей модели. Давайте посмотрим, как это работает на модели, построенной на основе случайного леса и натренированной на наборе данных о недвижимости в районе Бостона, США (Boston Housing), который поставляется с предустановленной программой scikit-learn. Модель будет предсказывать цену дома на основе ряда факторов. Мы можем выполнить поиск по сетке, создав экземпляр класса `GridSearchCV` и натренировав модель, передав ей значения, которые мы определили ранее:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import load_boston

X, y = load_boston(return_X_y=True)
housing_model = RandomForestRegressor()

grid_search_housing = GridSearchCV(
    housing_model, param_grid=grid_vals, scoring='max_error')
grid_search_housing.fit(X, y)
```

Обратите внимание, что здесь внутренний параметр `scoring`[51] представляет метрику, подлежащую оптимизации. В случае этой регрессионной модели мы хотим использовать комбинацию гиперпараметров, которая приводит к наименьшей для нашей модели ошибке. В целях получения наилучшей комбинации значений из поиска по сетке мы можем выполнить инструкцию `grid_search_housing.best_params_`. Она возвращает следующее:

```
{'max_depth': 100, 'n_estimators': 150}
```

Мы хотели бы сравнить этот результат с ошибкой, которую мы бы получили, тренируя модель, основанную на регрессоре случайного леса, без гиперпараметриче-

[51] В библиотеке scikit-learn параметр `scoring` используется внутренней стратегией оценивания с участием специального оценщика, т. е. объекта, который выполняет оптимизацию под указанную в параметре метрику. — *Прим. перев.*

ской настройки, используя значения, принятые в scikit-learn для этих параметров по умолчанию. Этот подход с использованием поиска по сетке хорошо работает на малом примере, который мы определили выше, но в более сложных моделях нам наверняка потребуется оптимизировать более двух гиперпараметров, каждый из которых имеет широкий диапазон возможных значений. В конечном счете поиск по сетке приведет к комбинаторному взрыву — по мере внесения дополнительных гиперпараметров и значений в нашу сетку вариантов число возможных комбинаций, которые нам нужно испытать, и время, необходимое для всех их, значительно увеличиваются.

Еще одна проблема с таким подходом заключается в том, что при выборе разных комбинаций никакая логика не срабатывает. Поиск по сетке — это, по сути, техническое решение методом грубой силы, в котором испытываются все возможные комбинации значений. Предположим, что после некоторого значения `max_depth` ошибка нашей модели увеличивается. Алгоритм поиска по сетке не учится на предыдущих испытаниях, поэтому он не будет знать, что после определенного порога следует прекратить испытывать значения `max_depth`. Он просто будет испытывать каждое предоставляемое вами значение, независимо от результатов.

Библиотека scikit-learn поддерживает альтернативу поиску по сетке под названием *рандомизированного поиска*, `RandomizedSearchCV`, в котором реализован случайный поиск. Вместо того чтобы испытывать каждую возможную комбинацию гиперпараметров из набора, вы определяете, сколько раз вы хотели бы произвольно выбирать значения для каждого гиперпараметра. Для реализации случайного поиска в scikit-learn мы создаем экземпляр класса `RandomizedSearchCV` и передаем ему словарь, аналогичный приведенному выше `grid_values`, указав диапазоны вместо конкретных значений. Случайный поиск работает быстрее, чем поиск по сетке, т. к. он не испытывает каждую комбинацию в вашем наборе возможных значений, но при этом вполне может получиться так, что среди случайно отобранного оптимального набора гиперпараметров не окажется.

Для робастной гиперпараметрической настройки нам требуется решение, которое масштабируется и учится на предыдущих испытаниях, отыскивая оптимальную комбинацию значений гиперпараметров.

Решение

В библиотеке keras-tuner реализована байесова оптимизация, чтобы выполнять гиперпараметрический поиск непосредственно в Keras. Для того чтобы использовать keras-tuner, определим модель внутри функции, которая принимает гиперпараметрический аргумент, в данном случае `hp`. Затем мы можем использовать `hp` в функции везде, где хотим включить гиперпараметр, указывая имя гиперпараметра, тип данных, диапазон значений, которые мы хотели бы найти, и величину его прироста всякий раз, когда мы испытываем новый.

Вместо того чтобы жестко кодировать значение гиперпараметра во время определения слоя в модели Keras, мы определяем его с помощью гиперпараметрической переменной. Мы хотим настроить число нейронов в первом скрытом слое нашей нейронной сети:

```
keras.layers.Dense(hp.Int('first_hidden', 32, 256, step=32), activation='relu')
```

Здесь `first_hidden` — имя, которое мы дали этому гиперпараметру; `32` — минимальное значение, которое мы для него определили; `256` — максимальное значение; `32` — величина, на которую мы должны увеличивать это значение в пределах определенного нами диапазона. Если бы мы строили классификационную модель MNIST, то полная функция, которую мы передали бы в keras-tuner, могла бы выглядеть следующим образом:

```
def build_model(hp):
    model = keras.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),
        keras.layers.Dense(
            hp.Int('first_hidden', 32, 256, step=32), activation='relu'),
        keras.layers.Dense(
            hp.Int('second_hidden', 32, 256, step=32), activation='relu'),
        keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(
        optimizer=tf.keras.optimizers.Adam(
            hp.Float('learning_rate', .005, .01, sampling='log')),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

    return model
```

Библиотека keras-tuner поддерживает много разных алгоритмов оптимизации. Здесь мы создадим экземпляр тюнера с байесовой оптимизацией и оптимизируем его под метрику валидационной точности:

```
import kerastuner as kt

tuner = kt.BayesianOptimization(
    build_model,
    objective='val_accuracy',
    max_trials=10
)
```

Исходный код для выполнения настроечного задания похож на тренировку модели с помощью функции `fit()`. В процессе работы мы сможем увидеть значения трех гиперпараметров, отобранных для каждого испытания. Когда работа будет завершена, мы сможем увидеть гиперпараметрическую комбинацию, которая привела к наилучшему испытанию. На рис. 4.23 приведен пример результата одного пробного прогона с использованием keras-tuner.

В дополнение к приведенным здесь примерам, библиотека keras-tuner предоставляет дополнительную функциональность, которую мы еще не рассматривали. Ее можно использовать для экспериментирования с разным количеством слоев модели, определив параметр `hp.Int()` внутри цикла. Кроме того, вместо диапазона для гиперпараметра можно предоставить фиксированный набор значений.

```
Hyperparameters:
|-first_hidden: 35
|-learning_rate: 0.005798007789002127
|-second_hidden: 160
Epoch 1/10
1688/1688 [==============================] - 5s 3ms/step - loss: 1.5554 - accuracy: 0.7540 - val_loss: 0.4973 - val_accuracy: 0.8753
Epoch 2/10
1688/1688 [==============================] - 5s 3ms/step - loss: 0.4308 - accuracy: 0.8874 - val_loss: 0.3429 - val_accuracy: 0.9042
Epoch 3/10
1688/1688 [==============================] - 5s 3ms/step - loss: 0.3867 - accuracy: 0.9051 - val_loss: 0.2888 - val_accuracy: 0.9343
Epoch 4/10
1688/1688 [==============================] - 5s 3ms/step - loss: 0.3864 - accuracy: 0.9070 - val_loss: 0.2665 - val_accuracy: 0.9333
Epoch 5/10
1688/1688 [==============================] - 5s 3ms/step - loss: 0.4957 - accuracy: 0.8849 - val_loss: 0.3942 - val_accuracy: 0.9165
Epoch 6/10
1688/1688 [==============================] - 5s 3ms/step - loss: 0.4518 - accuracy: 0.8968 - val_loss: 0.3776 - val_accuracy: 0.9260
Epoch 7/10
1688/1688 [==============================] - 5s 3ms/step - loss: 0.4181 - accuracy: 0.9065 - val_loss: 0.3471 - val_accuracy: 0.9287
Epoch 8/10
1688/1688 [==============================] - 5s 3ms/step - loss: 0.4361 - accuracy: 0.9017 - val_loss: 0.3558 - val_accuracy: 0.9222
Epoch 9/10
1688/1688 [==============================] - 5s 3ms/step - loss: 0.4278 - accuracy: 0.9047 - val_loss: 0.3847 - val_accuracy: 0.9132
Epoch 10/10
1688/1688 [==============================] - 5s 3ms/step - loss: 0.4383 - accuracy: 0.9004 - val_loss: 0.4232 - val_accuracy: 0.9243

Trial complete

Trial summary
|-Trial ID: 9b9b7bb5dbeb5e2dff1cae569202b6a1
|-Score: 0.934333324432373
|-Best step: 0
```

Рис. 4.23. Результат одного пробного прогона гиперпараметрической настройки с помощью тюнера keras-tuner. В верхней части показаны отобранные тюнером гиперпараметры, а в разделе резюме — результирующая метрика оптимизации

В более сложных моделях параметр `hp.Choice()` можно использовать для экспериментирования с разными типами слоев, такими как `BasicLSTMCell` и `BasicRNNCell`. Библиотека keras-tuner работает в любой среде, где вы можете тренировать модель Keras.

Почему это работает

Хотя поиск по сетке и случайный поиск в гиперпараметрической настройке работают эффективнее метода проб и ошибок, они быстро становятся дорогостоящими в случае моделей, требующих значительного времени тренировки или имеющих большое пространство гиперпараметрического поиска.

Поскольку и сами модели машинного обучения, и процесс гиперпараметрического поиска являются задачами оптимизации, из этого следует, что мы могли бы использовать подход, который учится отыскивать оптимальную комбинацию гиперпараметров в заданном диапазоне возможных значений точно так же, как наши модели учатся на тренировочных данных.

Гиперпараметрическую настройку можно рассматривать как внешний цикл оптимизации (рис. 4.24), в котором внутренний цикл состоит из типичной тренировки модели. Несмотря на то что мы изображаем нейронные сети как модель, параметры которой оптимизируются, это техническое решение применимо и к другим типам моделей машинного обучения. Кроме того, хотя более распространенным вариантом использования является выбор одной наилучшей модели из всех потенциальных гиперпараметров, в некоторых случаях может использоваться гиперпараметрический каркас, генерирующий семейства моделей, которые могут действовать как ансамбль (см. обсуждение паттерна "Ансамбли" в *главе 3*).

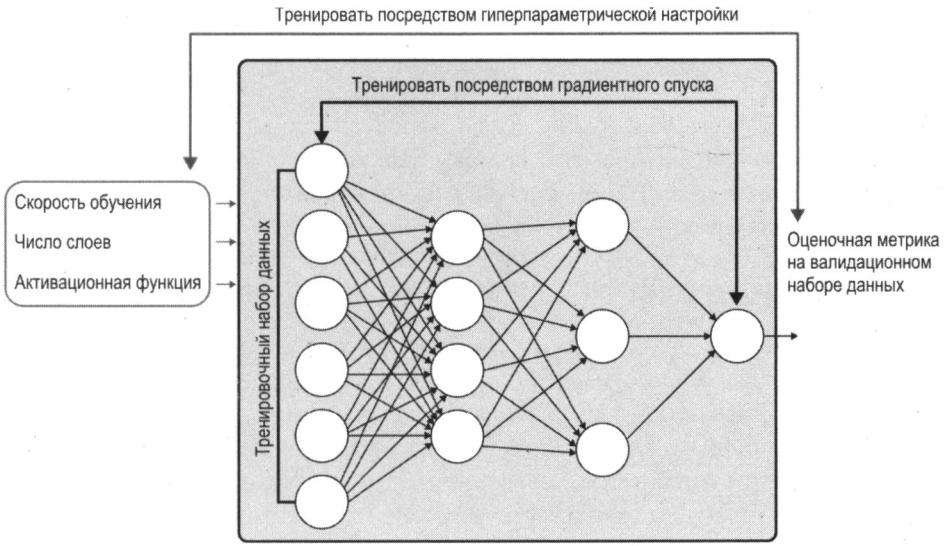


Рис. 4.24. Гиперпараметрическая настройка может рассматриваться как внешний цикл оптимизации

Нелинейная оптимизация

Подлежащие настройке гиперпараметры делятся на две группы: гиперпараметры, которые связаны с модельной *архитектурой*, и гиперпараметры, которые связаны с *тренировкой* модели. Гиперпараметры архитектуры модели, такие как число слоев в модели или количество нейронов в слое, управляют математической функцией, лежащей в основе модели машинного обучения. Параметры, связанные с тренировкой модели, такие как число эпох, скорость обучения и размер пакета, управляют циклом тренировки и часто имеют отношение к тому, как работает оптимизатор, основанный на градиентном спуске. Принимая во внимание оба этих типа параметров, можно сделать вывод, что совокупная модельная функция по отношению к этим гиперпараметрам в общем случае недифференцируема.

Внутренний цикл тренировки является дифференцируемым, и поиск оптимальных параметров может осуществляться посредством стохастического градиентного спуска. Один шаг модели, натренированной с помощью стохастического градиента, может длиться всего несколько миллисекунд. С другой стороны, одно испытание в задаче гиперпараметрической настройки предусматривает тренировку полной модели на тренировочном наборе данных и может занимать несколько часов. Более того, задача оптимизации гиперпараметров должна решаться с помощью нелинейных методов оптимизации, применимых к недифференцируемым задачам.

Как только мы принимаем решение о том, что будем использовать методы нелинейной оптимизации, наш выбор метрики становится шире. Эта метрика будет оцениваться на валидационном наборе данных и не обязательно должна совпадать с тренировочной потерей. В случае классификационной модели метрикой оптимизации может быть точность, и поэтому вам нужно найти комбинацию гиперпара-

метров, которая приводит к максимальной точности модели, даже если потерей будет двоичная перекрестная энтропия. В регрессионной модели может потребоваться оптимизация медианной абсолютной ошибки, даже если потерей будет квадрат ошибки. В этом случае вам нужно будет найти гиперпараметры, которые дают наименьшую среднеквадратическую ошибку. Эта метрика может быть даже выбрана на основе бизнес-целей. Например, мы можем максимизировать ожидаемую выручку либо минимизировать потери из-за мошенничества.

Байесова оптимизация

Байесова оптимизация — это технический прием для оптимизирования функций черного ящика, изначально разработанный в 1970-х годах Джонасом Мокусом (Jonas Mockus)[52]. Указанный метод использовался во многих областях и впервые был применен к гиперпараметрической настройке в 2012 году. Здесь мы сосредоточимся на байесовой оптимизации, связанной с гиперпараметрической настройкой. В данном контексте модель машинного обучения является *функцией черного ящика*, поскольку модели производят серию результатов на выходе из подаваемых нами на вход переменных, не требуя от нас знания внутренних деталей самой модели. Процесс тренировки модели называется *вызовом целевой функции.*

Цель байесовой оптимизации состоит в том, чтобы тренировать модель напрямую как можно меньше раз, поскольку это дорого. Следует помнить, что всякий раз, когда мы испытываем новую комбинацию гиперпараметров на нашей модели, нам нужно пробегать по всему циклу тренировки в нашей модели. Это может показаться тривиальным с малой моделью, такой как модель scikit-learn, которую мы тренировали ранее, но для многих производственных моделей процесс тренировки требует значительной инфраструктуры и времени.

Вместо того чтобы тренировать модель всякий раз, когда мы испытываем новую комбинацию гиперпараметров, байесова оптимизация определяет новую функцию, которая эмулирует нашу модель, но которую гораздо дешевле выполнять. Такая функция называется *суррогатной*: на вход в данную функцию поступают значения гиперпараметра, а на выходе из нее получается ваша метрика оптимизации. Суррогатная функция вызывается гораздо чаще, чем целевая функция для отыскания оптимальной комбинации гиперпараметров перед выполнением тренировочного прогона на вашей модели. При таком подходе на выбор гиперпараметров для каждого испытания тратится больше вычислительного времени, чем в поиске по сетке. Однако, поскольку байесов подход использования суррогатной функции обходится значительно дешевле, чем запуск нашей целевой функции, всякий раз, когда мы испытываем разные гиперпараметры, он предпочтительнее. Распространенные подходы к генерированию суррогатной функции включают гауссов процесс[53] или деревья Парзена[54].

---

[52] См. **https://oreil.ly/Ak24H**.

[53] См. **https://oreil.ly/-Srjj**.

[54] См. **https://oreil.ly/UqxDd**.

До сих пор мы касались разных частей байесовой оптимизации, но как они работают вместе? Сперва мы должны выбрать гиперпараметры, которые хотим оптимизировать, и определить диапазон значений для каждого гиперпараметра. Эта часть процесса является ручной и определяет пространство, в котором наш алгоритм будет искать оптимальные значения. Нам также нужно определить целевую функцию, т. е. код, вызывающий процесс тренировки модели. С этого места байесова оптимизация формирует суррогатную функцию для симулирования процесса тренировки модели и использует эту функцию для определения наилучшей комбинации гиперпараметров для выполнения на нашей модели. Только когда этот суррогатная функция приходит по ее мнению к хорошей комбинации гиперпараметров, мы делаем полный тренировочный (пробный) прогон на нашей модели. Результаты этого процесса затем передаются обратно в суррогатную функцию, и процесс повторяется указанное нами число испытаний.

Компромиссы и альтернативы

Генетические алгоритмы являются альтернативой байесовым методам гиперпараметрической настройки, но они, как правило, требуют гораздо большего числа тренировочных прогонов моделей, чем байесовы методы. Мы покажем, как использовать сервис для оптимизации гиперпараметрической настройки на моделях, построенных с использованием различных фреймворков ML.

Полноуправляемая гиперпараметрическая настройка

Подход, принятый в библиотеке keras-tuner, может не масштабироваться до больших задач машинного обучения, потому что мы хотели бы, чтобы испытания проходили параллельно, а вероятность ошибки машины и других отказов возрастает по мере того, как время тренировки модели растягивается в часы. Следовательно, полноуправляемый, отказоустойчивый подход, обеспечивающий оптимизацию функции черного ящика, полезен для гиперпараметрической настройки. Пример сервиса с байесовой оптимизацей представлен сервисом гиперпараметрической настройки[55], поставляемым платформой Google Cloud AI. Этот сервис основан на Vizier[56], инструменте оптимизации черного ящика, используемом внутри компании Google.

Лежащие в основе указанного облачного сервиса концепции работают аналогично тюнеру keras-tuner: вы указываете имя, тип, диапазон и масштаб каждого гиперпараметра, и эти значения отражаются в тренировочном коде модели. Мы покажем вам, как запускать гиперпараметрическую настройку на платформе AI Platform с помощью модели PyTorch, натренированной на наборе натальных данных BigQuery, чтобы предсказывать вес младенца при рождении.

Первый шаг — создание файла config.yaml, задающего гиперпараметры, которые вы хотите оптимизировать в задании, а также некоторые другие метаданные в ва-

[55] См. **https://oreil.ly/MO8FZ**.

[56] См. **https://oreil.ly/tScQa**.

шем задании. Одно из преимуществ использования облачного сервиса заключается в том, что вы можете масштабировать свое настроечное задание, выполняя его на графических или тензорных процессорах и распределяя его по нескольким серверам параметров. В этом конфигурационном файле вы также указываете суммарное количество гиперпараметрических испытаний, которые вы хотите выполнить, и число из этих испытаний, которые вы хотите выполнять параллельно. Чем больше процессов будет работать параллельно, тем быстрее будет выполняться ваша работа. Однако преимущество параллельного выполнения меньшего числа испытаний заключается в том, что служба сможет учиться на результатах каждого завершенного испытания и оптимизировать следующее.

Для нашей модели пример конфигурационного файла, использующего графические процессоры, может выглядеть следующим образом. В этом примере мы настроим три гиперпараметра — скорость обучения, значение моментума (импульса) оптимизатора[57] и число нейронов в скрытом слое нашей модели. Мы также укажем нашу метрику оптимизации. В данном примере наша цель — минимизировать потерю модели на валидационном наборе:

```
trainingInput:
  scaleTier: BASIC_GPU
  parameterServerType: large_model
  workerCount: 9
  parameterServerCount: 3
  hyperparameters:
    goal: MINIMIZE
    maxTrials: 10
    maxParallelTrials: 5
    hyperparameterMetricTag: val_error
    enableTrialEarlyStopping: TRUE
    params:
    - parameterName: lr
      type: DOUBLE
      minValue: 0.0001
      maxValue: 0.1
      scaleType: UNIT_LINEAR_SCALE
    - parameterName: momentum
      type: DOUBLE
      minValue: 0.0
      maxValue: 1.0
      scaleType: UNIT_LINEAR_SCALE
    - parameterName: hidden-layer-size
      type: INTEGER
      minValue: 8
      maxValue: 32
      scaleType: UNIT_LINEAR_SCALE
```

[57] См. **https://oreil.ly/8mHPQ**.

Вместо того чтобы для определения этих значений использовать конфигурационный файл, вы также можете сделать это с помощью API Python платформы AI Platform.

Для этого нам понадобится добавить в исходный код обработчик аргументов, который будет задавать аргументы, определенные нами выше в файле, а затем ссылаться на эти гиперпараметры там, где они будут появляться во всем модельном коде.

Далее мы построим модель, используя API nn.Sequential библиотеки PyTorch с оптимизатором, основанным на стохастическом градиентном спуске. Поскольку наша модель предсказывает вес младенца в виде вещественного числа, эта модель будет регрессионной. Мы задаем каждый гиперпараметр с помощью переменной `args`, которая содержит переменные, определенные в обработчике аргументов:

```
import torch.nn as nn

model = nn.Sequential(nn.Linear(num_features, args.hidden_layer_size),
                      nn.ReLU(),
                      nn.Linear(args.hidden_layer_size, 1))

optimizer = torch.optim.SGD(model.parameters(), lr=args.lr,
                            momentum=args.momentum)
```

В конце нашего кода тренировки модели мы создадим экземпляр `HyperTune()` и передадим ему метрику, которую пытаемся оптимизировать. Он будет сообщать результирующее значение метрики оптимизации после каждого тренировочного прогона. Важно, чтобы любая выбранная нами метрика оптимизации вычислялась не на тренировочном наборе данных, а на тестовом или валидационном наборе данных:

```
import hypertune

hpt = hypertune.HyperTune()

val_mse = 0
num_batches = 0

criterion = nn.MSELoss()

with torch.no_grad():
    for i, (data, label) in enumerate(validation_dataloader):
        num_batches += 1
        y_pred = model(data)
        mse = criterion(y_pred, label.view(-1,1))
        val_mse += mse.item()
    avg_val_mse = (val_mse / num_batches)

hpt.report_hyperparameter_tuning_metric(
    hyperparameter_metric_tag='val_mse',
```

```
    metric_value=avg_val_mse,
    global_step=epochs
)
```

После того как мы отправили тренировочное задание на платформу AI Platform, мы можем отслеживать журналы в облачной консоли. По завершении каждого испытания вы сможете увидеть значения, выбранные для каждого гиперпараметра, и результирующее значение метрики оптимизации (рис. 4.25).

HyperTune trials

	Trial ID	avg_val_mse ↑	Training step	lr	momentum	hidden-layer-size
○ ✓	6	1.58062	10	0.04151	0.37651	31
○ ✓	7	1.58216	10	0.00821	0.97651	19
○ ✓	1	1.58262	10	0.00547	0.90981	30
○ ✓	10	1.58374	10	0.07828	0.69754	24
○ ✓	4	1.58463	10	0.00905	0.4407	20
○ ✓	2	1.59563	10	0.07565	0.0407	14
○ ✓	3	1.60248	10	0.04235	0.6407	26
○ ✓	9	1.60607	10	0.05561	0.62768	19
○ ✓	8	1.61204	10	0.06797	0.85666	22
○ ✓	5	1.80907	10	0.0484	0.29004	27

Рис. 4.25. Пример сводки HyperTune в консоли платформы AI Platform. Она относится к модели PyTorch, оптимизирующей три модельных параметра с целью минимизирования корня из среднеквадратической ошибки (RMSE) на валидационном наборе данных

По умолчанию сервис AI Platform Training будет использовать для вашего настроечного задания байесову оптимизацию, но вместо нее вы также по желанию можете указать алгоритмы поиска по сетке или случайного поиска. Указанный сервис оптимизирует ваш гиперпараметрический поиск по всем тренировочным заданиям. Если мы запустим еще одно тренировочное задание, аналогичное вышеприведенному, но с некоторыми доработками в настройках гиперпараметров и пространстве поиска, то оно будет использовать результаты последнего задания с целью эффективного выбора значений для следующего набора испытаний.

Мы привели здесь пример PyTorch, но вы можете использовать AI Platform Training для гиперпараметрической настройки в любом фреймворке машинного обучения, упаковав свой тренировочный код и предоставив файл setup.py, инсталлирующий любые библиотечные зависимости.

Генетические алгоритмы

Мы рассмотрели различные алгоритмы оптимизации гиперпараметров: ручного поиска, поиска по сетке, случайного поиска и байесовой оптимизации. Еще одной, менее распространенной, альтернативной является *генетический алгоритм*, который примерно основан на эволюционной теории естественного отбора Чарльза Дарвина. Указанная теория, также именуемая "выживанием наиболее приспособ-

ленных", утверждает, что наиболее эффективные ("наиболее приспособленные") члены популяции будут выживать и передавать свои гены будущим поколениям, тогда как менее приспособленные члены не будут. Генетические алгоритмы применялись к разным типам оптимизационных задач, включая гиперпараметрическую настройку.

Применительно к гиперпараметрическому поиску генетический подход начинает с того, что сначала определяет *функцию приспособленности*. Она измеряет качество конкретного испытания и обычно может определяться метрикой оптимизации в вашей модели (точностью, ошибкой и т. д.). Задав функцию приспособленности, вы случайно отбираете несколько комбинаций гиперпараметров из пространства поиска и выполняете пробную версию для каждой этой комбинации. Затем вы берете гиперпараметры из испытаний, которые показали наилучшие результаты, и используете эти значения для определения нового пространства поиска. Это пространство поиска становится новой "популяцией", и вы используете ее для генерирования новых комбинаций значений в следующем наборе испытаний. Вы продолжаете этот процесс, сокращая число выполняемых вами испытаний до тех пор, пока не получите результат, удовлетворяющий вашим требованиям.

Поскольку генетические алгоритмы используют для улучшения результаты предыдущих испытаний, они "умнее", чем ручной поиск, случайный поиск и поиск по сетке. Однако при большом пространстве гиперпараметрического поиска сложность генетических алгоритмов возрастает. Вместо того чтобы для тренировки модели использовать суррогатную функцию в качестве прокси, как в байесовой оптимизации, генетические алгоритмы требуют тренировки модели для каждой возможной комбинации значений гиперпараметров. Вдобавок на момент написания этой книги генетические алгоритмы были менее распространены, и существует меньше фреймворков ML, которые поддерживают их из коробки в части гиперпараметрической настройки.

Резюме

Эта глава посвящена паттернам, которые модифицируют типичный для машинного обучения цикл тренировки, основанный на стохастическом градиентном спуске (SGD). Мы начали с рассмотрения паттерна "Полезное переобучение" (Useful Overfitting), который охватывал ситуации, когда переобучение бывает полезным. Например, при использовании таких управляемых данными методов, как машинное обучение, для аппроксимирования решений сложных динамических систем или набора дифференциальных уравнений в частных производных (PDE), где может быть покрыто все входное пространство, конечной целью является переобучение на тренировочном наборе. Переобучение также полезно в качестве технического приема при разработке и отладке архитектуры. Далее мы рассмотрели паттерн "Контрольные точки" (Checkpoints) и приемы использования контрольных точек во время тренировки моделей. В этом паттерне мы периодически сохраняем полное состояние модели во время тренировки. Указанные контрольные точки могут рассматриваться в качестве окончательной модели, как в случае досрочной остановки,

либо в качестве стартовых точек в случае сбоев тренировки или в случае тонкой настройки.

Паттерн "Трансферное обучение" (Transfer Learning) охватывает многократное использование частей ранее натренированной модели. Трансферное обучение представляет собой полезный подход к эффективному использованию слоев усвоенных признаков из предварительно натренированной модели, когда ваш собственный набор данных ограничен. Указанный паттерн также можно применять для тонкой настройки предварительно натренированной модели, которая была натренирована на крупном обобщенном наборе данных, под более специализированный набор данных. Затем мы обсудили паттерн "Распределительная стратегия" (Distribution Strategy). Тренировка крупных, сложных нейронных сетей может занимать значительное количество времени. Распределительные стратегии предлагают разнообразные способы, с помощью которых цикл тренировки можно модифицировать для выполнения в требуемом масштабе на нескольких воркерах, используя параллелизацию и аппаратные ускорители.

Наконец, паттерн "Гиперпараметрическая настройка" (Hyperparameter Tuning) касался обсуждения вопроса о том, как сам цикл тренировки, основанный на стохастическом градиентном спуске, может оптимизироваться по отношению к модельным гиперпараметрам. Мы познакомились с несколькими полезными библиотеками, которые можно использовать для реализации гиперпараметрической настройки в отношении моделей, созданных в Keras и PyTorch.

В следующей главе рассматриваются паттерны, связанные с обеспечением отказоустойчивости (при большом числе запросов, трафике с пиковой нагрузкой или управлении изменениями) при внедрении моделей в производство.

ГЛАВА 5

Паттерны для отказоустойчивой обработки

Предназначение модели машинного обучения состоит в том, чтобы использовать ее для предсказательного вывода на данных, которые она не встречала во время тренировки. Поэтому после тренировки модель обычно разворачивают в промышленной среде и используют для генерирования предсказаний в ответ на поступающие запросы. Ожидается, что программно-информационное обеспечение, развернутое в промышленных средах, будет отказоустойчивым и незначительно нуждаться во вмешательстве человека для поддержания его работы. Паттерны этой главы решают задачи, связанные с обеспечением отказоустойчивости в разных обстоятельствах, применительно к моделям ML.

Паттерн "Функция обслуживания без поддержки состояния" (Stateless Serving Function) позволяет инфраструктуре масштабироваться и манипулировать тысячами или даже миллионами предсказательных запросов в секунду. Паттерн "Пакетное обслуживание" (Batch Serving) дает возможность модели обрабатывать случайные или периодические запросы на миллионы или миллиарды предсказаний асинхронно. Эти паттерны полезны помимо обеспечения отказоустойчивости, поскольку они уменьшают зависимость между создателями и пользователями моделей машинного обучения.

Паттерн "Непрерывное оценивание модели" (Continued Model Evaluation) решает распространенную задачу обнаружения ситуаций, когда развернутая модель больше не соответствует целевому назначению. Паттерн "Двухфазные предсказания" (Two-Phase Predictions) предоставляет подход к решению задачи обеспечения техничности и производительности моделей, когда их приходится развертывать на распределенных устройствах. Паттерн "Предсказания по ключу" (Keyed Predictions) необходим для масштабируемой реализации нескольких паттернов из этой главы.

ПАТТЕРН 16. Функция обслуживания без поддержки состояния

Паттерн "Функция обслуживания без поддержки состояния" (Stateless Serving Function) позволяет системе ML обрабатывать синхронно от тысяч до миллионов предсказательных запросов в секунду. Система ML разработана на основе функции без сохранения состояния, которая фиксирует архитектуру и веса обученной модели.

Функции без поддержки состояния

Функция без поддержки состояния — это функция, выходные значения которой определяются исключительно ее входными значениями. Приведенная ниже функция, например, не имеет состояния:

```
def stateless_fn(x):
    return 3*x + 15
```

Функция без поддержки состояния еще может рассматриваться как неизменяемый объект, в которой веса и смещенности хранятся как константы:

```
class Stateless:
    def __init__(self):
        self.weight = 3
        self.bias = 15
    def __call__(self, x):
        return self.weight*x + self.bias
```

Функция, которая поддерживает счетчик количества, когда она была вызвана, и которая возвращает другое значение в зависимости от четности этого счетчика, является примером функции с поддержкой состояния:

```
class State:
    def __init__(self):
        self.counter = 0
    def __call__(self, x):
        self.counter += 1
        if self.counter % 2 == 0:
            return 3*x + 15
        else:
            return 3*x - 15
```

Вызов `stateless_fn(3)` или `Stateless()(3)` всегда возвращает 24, тогда как

```
a = State()
```

а потом вызов

```
a(3)
```

возвращает значение, которое варьируется между –6 и 24. Счетчик в этом случае является внутренним состоянием функции, а значение на выходе из функции зависит как от значения на входе (`x`) в нее, так и от состояния (`counter`). Состояние обычно поддерживается с помощью переменных класса (как в нашем примере) либо с помощью глобальных переменных.

Поскольку компоненты без поддержки состояния не имеют никакого состояния, они могут использоваться совместно многочисленными клиентами. Серверы обычно создают пул экземпляров компонентов без поддержки состояния и используют их для обработки клиентских запросов по мере их поступления. Компоненты с поддержкой состояния должны помнить состояние каждого клиента. Жизненный цикл компонентов без поддержки состояния должен управляться сервером. Например, они должны инициализироваться при первом запросе и уничтожаться, когда клиент

отключается или истекает время ожидания. Вследствие этих факторов компоненты без поддержки состояния обладают высокой масштабируемостью, тогда как компоненты с поддержкой состояния обходятся дорого и ими трудно управлять. При разработке корпоративных приложений архитекторы стараются свести к минимуму число компонентов с поддержкой состояния. Веб-приложения, например, часто строятся для работы на основе API REST, и они предусматривают передачу состояния от клиента к серверу при каждом вызове.

В машинном обучении во время тренировки фиксируется большое количество состояний. Так, число эпох и скорость обучения являются частью состояния модели и должны запоминаться, потому что, как правило, скорость обучения уменьшается с каждой последующей эпохой. Говоря, что модель должна быть экспортирована как функция без поддержки состояния, мы хотим, чтобы создатели каркаса модели отслеживали эти переменные состояния и не включали их в экспортируемый файл.

В результате применения функции без поддержки состояния серверный код упрощается и делается более масштабируемым, но может усложниться клиентский код. Например, некоторые функции модели по своей сути поддерживают состояние. Модель исправления ошибок орфографии, которая берет слово и возвращает исправленную форму, должна поддерживать состояние, потому что она должна знать предыдущие несколько слов, чтобы исправлять слово наподобие "there" на "their" в зависимости от контекста. Модели, которые оперируют последовательностями, поддерживают историю, используя специальные структуры, такие как рекуррентные нейронные сети. В таких случаях для экспорта модели в качестве функции без поддержки состояния требуется замена входной переменной с одиночного слова, например, на предложение. Из этого следует, что клиенты модели исправления ошибок орфографии должны будут управлять состоянием (собирать последовательность слов и разбивать ее на предложения) и отправлять ее вместе с каждым запросом. Результирующая сложность на стороне клиента наиболее заметна, когда клиент проверки орфографии должен вернуться назад и изменить предыдущее слово из-за контекста, который добавляется позже.

Постановка задачи

Возьмем модель классифицирования текста, которая в качестве тренировочных данных использует отзывы о кинофильмах из интернет-базы данных кинофильмов (Internet Movie Database, IMDb). Для начального слоя модели мы будем использовать предварительно натренированное векторное вложение, которое соотносит текст с 20-мерными векторами вложения (полный исходный код см. в блокноте serving_function.ipynb[1] в репозитории на GitHub этой книги):

```
model = tf.keras.Sequential()
embedding = (
    "https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim-with-oov/1")
```

[1] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/05_resilience/serving_function.ipynb**.

```
hub_layer = hub.KerasLayer(embedding, input_shape=[],
                           dtype=tf.string, trainable=True, name='full_text')
model.add(hub_layer)
model.add(tf.keras.layers.Dense(16, activation='relu', name='h1_dense'))
model.add(tf.keras.layers.Dense(1, name='positive_review_logits'))
```

Слой векторного вложения берется из TensorFlow Hub и помечается как тренируемый, вследствие чего мы имеем возможность выполнять тонкую настройку (*см. разд. "Паттерн 13. Трансферное обучение" главы 4*) на словаре, находящемся в отзывах базы данных IMDb. Последующее слои — это слои простой нейронной сети с одним скрытым слоем и выходным логитным слоем. Затем эту модель можно натренировать на наборе данных с отзывами о кинофильмах, чтобы научиться предсказывать позитивность или негативность отзыва.

После того как модель будет натренирована, мы сможем использовать ее для предсказательного вывода о степени позитивности отзыва:

```
review1 = 'The film is based on a prize-winning novel.'
review2 = 'The film is fast moving and has several great action scenes.'
review3 = 'The film was very boring. I walked out half-way.'
logits = model.predict(x=tf.constant([review1, review2, review3]))
```

В результате получается двумерный массив, например такой:

```
[[ 0.6965847]
 [ 1.61773 ]
 [-0.7543597]]
```

Выполнение операций предсказательного вывода путем вызова `model.predict()` на объекте в памяти (или тренируемом объекте, загруженном в память) несет в себе несколько проблем, как описано в предыдущем фрагменте кода.

- Мы должны загружать всю модель Keras целиком в память. Слой векторного вложения текста, который был задан как тренируемый, может быть довольно большим, потому что должен хранить вложения для полного словаря английских слов. Модели глубокого обучения со многими слоями также могут быть весьма большими.
- Приведенная выше архитектура накладывает ограничения на задержку, которая может быть достигнута за счет того, что вызовы метода `predict()` должны отправляться друг за другом.
- Несмотря на то что предпочтительным языком программирования исследователя данных является Python, предсказательный вывод, скорее всего, будет вызываться программами, написанными разработчиками, которые предпочитают другие языки, или на мобильных платформах, таких как Android или iOS.
- Данные на входе модели и выходе из нее, являясь наиболее эффективными для тренировки, могут быть неудобными для пользователя. В нашем примере на выходе из модели были логиты[2], потому что результаты в этой форме предпочти-

[2] См. **https://oreil.ly/qCWdH**.

тельнее для градиентного спуска. Вот почему второе число в выходном массиве больше 1[3]. Клиенты же, как правило, хотят, чтобы из данных вычислялась сигмоида, вследствие чего выходной интервал находился бы между 0 и 1 и мог бы интерпретироваться в более удобном для пользователя формате как вероятность. Скорее всего, мы захотим проводить эту постобработку на сервере с целью поддержания максимальной простоты клиентского кода. Аналогично модель может быть натренирована с использованием сжатых двоичных записей, в то время как во время использования у нас вполне может возникнуть потребность манипулировать самоописательными входными форматами, такими как JSON.

Решение

Решение состоит из следующих ниже шагов.

1. Экспортировать модель в формат, который фиксирует математическое ядро модели и является независимым от языка программирования.
2. В производственной системе формула, состоящая из "прямых" расчетов модели, восстанавливается как функция без поддержки состояния.
3. Функция без поддержки состояния развертывается на базе фреймворка, который обеспечивает REST-сервисы.

Экспорт модели

Первый шаг решения состоит в экспортировании модели в формат (TensorFlow использует формат SavedModel, но ONNX является еще одним вариантом), который фиксирует математическое ядро модели. Все состояние модели (скорость обучения, отсев, короткое замыкание и т. д.) сохранять не нужно — только математическую формулу, необходимую для вычисления величины на выходе из величины на входе. Как правило, натренированные весовые значения являются константами в математической формуле.

В Keras это достигается так:

```
model.save('export/mymodel')
```

Для обеспечения платформенно-нейтрального эффективного механизма восстановления формат SavedModel опирается на протокольные буферы. Другими словами, метод `model.save()` записывает модель в виде протокольного буфера (с расширением .pb) и экстернализирует натренированные веса, словари и т. д. в другие файлы в стандартной каталожной структуре:

```
export/.../variables/variables.data-00000-of-00001
export/.../assets/tokens.txt
export/.../saved_model.pb
```

---

[3] Логит (logit), также логарифм перевесов или разногласий (log odds) — это функция, которая соотносит вероятность принадлежности классу с интервалом]–∞; +∞[вместо промежутка [0; 1], т. е. по сути является ненормализованным логарифмом вероятности. — *Прим. перев.*

Предсказательный вывод на языке Python

В продуктивной системе модельная формула восстанавливается из протокольного буфера и других ассоциированных файлов в виде функции без поддержки состояния, которая отвечает конкретной модельной сигнатуре с именами входных и выходной переменных и типами данных.

В TensorFlow имеется инструмент saved_model_cli для проверки экспортированных файлов и определения сигнатуры функции без поддержки состояния, которую мы можем использовать в вызове сервиса модели:

```
saved_model_cli show --dir ${export_path} \
    --tag_set serve --signature_def serving_default
```

Этот код выдаст:

```
The given SavedModel SignatureDef contains the following input(s):
  inputs['full_text_input'] tensor_info:
      dtype: DT_STRING
      shape: (-1)
      name: serving_default_full_text_input:0
The given SavedModel SignatureDef contains the following output(s):
  outputs['positive_review_logits'] tensor_info:
      dtype: DT_FLOAT
      shape: (-1, 1)
      name: StatefulPartitionedCall_2:0
Method name is: tensorflow/serving/predict
```

Сигнатура указывает на то, что метод предсказания принимает одноэлементный массив (именуемый `full_text_input`) на входе, представляющий собой строковую переменную, и на выходе выдает одно число с плавающей точкой, имя которого `positive_review_logits`. Эти имена происходят от имен, которые мы назначили слоям Keras:

```
hub_layer = hub.KerasLayer(..., name='full_text')
...
model.add(tf.keras.layers.Dense(1, name='positive_review_logits'))
```

Вот как мы можем получить функцию обслуживания и использовать ее для предсказательного вывода:

```
serving_fn = tf.keras.models.load_model(export_path). \
              signatures['serving_default']
outputs = serving_fn(full_text_input=
                     tf.constant([review1, review2, review3]))
logit = outputs['positive_review_logits']
```

Обратите внимание, как мы используем имена входных и выходных переменных из функции обслуживания в коде.

Создание сервиса

Приведенный выше исходный код можно поместить в веб-приложение или бессерверную платформу, такую как Google App Engine, Heroku, AWS Lambda, Azure Functions, Google Cloud Functions, Cloud Run и т. д. Общим для всех этих платформ является то, что они позволяют разработчику задавать функцию, которая должна быть исполнена. Платформы берут на себя автоматическое масштабирование инфраструктуры, чтобы обеспечить возможность манипулирования большим числом предсказательных запросов в секунду с низкой задержкой.

Например, мы можем вызвать функцию обслуживания из инструмента Cloud Functions следующим образом:

```
serving_fn = None
def handler(request):
    global serving_fn
    if serving_fn is None:
        serving_fn = (tf.keras.models.load_model(export_path)
                                .signatures['serving_default'])
    request_json = request.get_json(silent=True)

    if request_json and 'review' in request_json:
        review = request_json['review']
        outputs = serving_fn(full_text_input=tf.constant([review]))
        return outputs['positive_review_logits']
```

Обратите внимание: нам следует проследить за тем, чтобы функция обслуживания была определена как глобальная переменная (или класс-одиночка), чтобы она не перезагружалась в ответ на каждый запрос. На практике функция обслуживания будет перезагружаться из пути экспорта (в облачном хранилище Google Cloud Storage) только в случае холодных пусков.

Почему это работает

Подход на основе экспортирования модели в функцию без поддержки состояния и развертывания функции без поддержки состояния в среде веб-приложений работает, потому что фреймворки веб-приложений предлагают автоматическое масштабирование, могут быть полноуправляемыми и являются независимыми от языка. Они также знакомы коллективам разработчиков программного обеспечения и развития бизнеса, которые, возможно, не имеют опыта работы с машинным обучением. Указанный подход также обладает преимуществами для гибкой методологии разработки — инженер ML или исследователь данных может самостоятельно изменять модель. Разработчику приложения нужно лишь изменить конечную точку, к которой он обращается.

Автомасштабируемость

Масштабирование сервисов до миллионов запросов в секунду является хорошо проработанной инженерной задачей. Вместо того чтобы заниматься строительст-

вом уникальных для машинного обучения сервисов, мы можем опереться на десятилетия инженерной работы, которая ушла на строительство отказоустойчивых веб-приложений и веб-сервисов. Облачные провайдеры умеют эффективно масштабировать конечные сервисы с минимальным временем прогрева.

Нам даже не нужно самим писать систему обработки запросов моделью. Большинство современных корпоративных систем машинного обучения имеют подсистему модельного обслуживания запросов. Например, TensorFlow предоставляет инструмент TensorFlow Serving, а PyTorch — TorchServe. Если мы будем использовать эти подсистемы обслуживания, то сможем просто предоставлять экспортированный файл, а программное обеспечение позаботится о создании конечного сервиса.

Полная управляемость

Облачные платформы абстрагируются от управления и инсталляции таких компонентов, как TensorFlow. Например, развернуть функцию обслуживания запросов в качестве REST API в Google Cloud так же просто, как выполнить приведенную ниже консольную программу, предоставляющую месторасположение данных в формате сохраненной модели SavedModel:

```
gcloud ai-platform versions create ${MODEL_VERSION} \
       --model ${MODEL_NAME} --origin ${MODEL_LOCATION} \
       --runtime-version $TFVERSION
```

В Amazon SageMaker развертывание сохраненной модели TensorFlow в формате SavedModel является аналогично простым и достигается с помощью:

```
model = Model(model_data=MODEL_LOCATION, role='SomeRole')
predictor = model.deploy(initial_instance_count=1,
                         instance_type='ml.c5.xlarge')
```

Имея REST-сервис, мы можем отправить предсказательный запрос в формате JSON:

```
{"instances":
  [
      {"reviews": "The film is based on a prize-winning novel."},
      {"reviews": "The film is fast moving and has several great action scenes."},
      {"reviews": "The film was very boring. I walked out half-way."}
  ]
}
```

Мы возвращаем предсказанные значения, которые также обернуты в структуру JSON:

```
{"predictions": [{"positive_review_logits": [0.6965846419334412]},
                 {"positive_review_logits": [1.6177300214767456]},
                 {"positive_review_logits": [-0.754359781742096]}]}
```

Давая клиентам возможность отправлять JSON-запросы с многочисленными экземплярами в запросе (процедура создания многосоставных запросов называется *упаковыванием*), мы позволяем клиентам выменивать более высокую пропускную способность, связанную с меньшим числом сетевых вызовов, на увеличение па-

раллелизации, если они отправляют больше запросов с меньшим числом экземпляров в расчете на запрос.

Помимо упаковывания существуют и другие меры повышения производительности или снижения затрат. Например, использование машины с более мощными графическими процессорами обычно помогает повышать производительность моделей глубокого обучения. Выбор машины с многочисленными ускорителями и/или потоками исполнения позволяет увеличить количество запросов в секунду. Использование автомасштабирующего кластера машин снижает затраты при пиковых рабочих нагрузках. Такая доработка нередко выполняется коллективом по интеграции ML в производство (ML/DevOps); некоторые из них специфичны для ML, другие — нет.

Языковая нейтральность

Каждый современный язык программирования умеет говорить на REST, а паттерн предоставляется для автогенерирования необходимых HTTP-заглушек. Так, клиенты Python могут вызывать REST API следующим образом. Обратите внимание, что в приведенном ниже исходном коде нет ничего фрейворк-специфичного. Поскольку облачная служба абстрагируется от конкретики модели ML, предоставлять в Keras и TensorFlow какие-либо ссылки не нужно:

```
credentials = GoogleCredentials.get_application_default()
api = discovery.build("ml", "v1", credentials = credentials,
            discoveryServiceUrl =
"https://storage.googleapis.com/cloudml/discovery/ml_v1_discovery.json")
request_data = {"instances":
 [
  {"reviews": "The film is based on a prize-winning novel."},
  {"reviews": "The film is fast moving and has several great action scenes."},
  {"reviews": "The film was very boring. I walked out half-way."}
 ]
}

parent = "projects/{}/models/imdb".format("PROJECT", "v1")
response = api.projects().predict(body = request_data,
                                  name = parent).execute()
```

Эквивалент приведенного выше фрагмента исходного кода можно написать на многих языках (мы применяем Python, предполагая, что вы хотя бы немного с ним знакомы). На момент написания этой книги разработчики могут получать доступ к API из Java, PHP, .NET, JavaScript, Objective-C, Dart, Ruby, Node.js и Go.

Мощная экосистема

Поскольку фреймворки веб-приложений столь широко используются, существует широкий спектр инструментов для измерения, мониторинга и управления веб-приложениями. Если мы развернем ML-модель на базе фреймворка веб-приложений, то модель можно отслеживать и регулировать с помощью инструментов, знакомых инженерам по обеспечению надежности программных продуктов, IT-ад-

министраторам и сотрудникам коллективов по интеграции программно-информационных продуктов в производство (DevOps). Они не обязаны ничего знать о машинном обучении.

Аналогично ваши коллеги по развитию бизнеса знают, как измерять и монетизировать веб-приложения с помощью API-шлюзов. Они могут переносить эти знания из прежней сферы деятельности и применять их для измерения и монетизации моделей машинного обучения.

Компромиссы и альтернативы

Как гласит шутка Дэвида Уилера (David Wheeler)[4], решение любой задачи в информатике заключается во внесении дополнительного уровня опосредованности. Введение спецификации экспортированной функции без поддержки состояния обеспечивает этот дополнительный уровень опосредованности. Паттерн "Функция обслуживания без поддержки состояния" позволяет изменять сигнатуру модельного обслуживания, чтобы обеспечивать дополнительную функциональность, такую как дополнительная пред- и постобработка, помимо того, что делает модель ML. По сути, этот паттерн можно использовать для предоставления модели многочисленных сервисов. Он также помогает в создании онлайнового предсказания с низкой задержкой для моделей, тренируемых на системах, таких как склады данных, которые обычно связаны с продолжительными запросами.

Прикладная функция обслуживания

Выходной слой нашей модели классифицирования текста представляет собой плотный слой, результат работы которого находится в интервале]–∞; ∞[:

```
model.add(tf.keras.layers.Dense(1, name='positive_review_logits'))
```

Функция потери это учитывает:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(
                      from_logits=True),
              metrics=['accuracy'])
```

Когда мы используем модель для предсказания, она, разумеется, возвращает то, на что была натренирована предсказывать, и выдает логиты. Однако клиенты ожидают вероятность, что отзыв будет положительным. Для решения этой задачи на выходе из модели нам нужно возвращать сигмоидный результат.

Это можно сделать, написав и экспортировав пользовательскую функцию обслуживания. Далее приведена прикладная функция обслуживания в Keras, которая добавляет вероятность и возвращает словарь, содержащий как логиты, так и вероятности для каждого поданного на вход отзыва:

---

[4] См. **https://oreil.ly/uskud**.

```
@tf.function(input_signature=[tf.TensorSpec([None],
                              dtype=tf.string)])
def add_prob(reviews):
    logits = model(reviews, training=False) # вызов модели
    probs = tf.sigmoid(logits)
    return {
        'positive_review_logits' : logits,
        'positive_review_probability' : probs
    }
```

Затем мы можем экспортировать вышеприведенную функцию в качестве модельного обслуживания, принятого по умолчанию:

```
model.save(export_path,
           signatures={'serving_default': add_prob})
```

Определение метода `add_prob()` сохраняется в пути экспорта `export_path()` и вызывается в ответ на запрос клиента.

Сигнатура модельного обслуживания запросов у экспортированной модели отражает новое имя входной переменной (обратите внимание на имя входного параметра `add_prob`), а также ключи выходного словаря и типы данных:

```
The given SavedModel SignatureDef contains the following input(s):
  inputs['reviews'] tensor_info:
      dtype: DT_STRING
      shape: (-1)
      name: serving_default_reviews:0
The given SavedModel SignatureDef contains the following output(s):
  outputs['positive_review_logits'] tensor_info:
      dtype: DT_FLOAT
      shape: (-1, 1)
      name: StatefulPartitionedCall_2:0
  outputs['positive_review_probability'] tensor_info:
      dtype: DT_FLOAT
      shape: (-1, 1)
      name: StatefulPartitionedCall_2:1
Method name is: tensorflow/serving/predict
```

Когда эта модель развертывается и используется для предсказательного вывода, выходной JSON содержит как логиты, так и вероятность:

```
{'predictions': [
    {'positive_review_probability': [0.6674301028251648],
     'positive_review_logits': [0.6965846419334412]},
    {'positive_review_probability': [0.8344818353652954],
     'positive_review_logits': [1.6177300214767456]},
    {'positive_review_probability': [0.31987208127975464],
     'positive_review_logits': [-0.754359781742096]}
]}
```

Обратите внимание, что `add_prob` — это функция, которую мы пишем. В этом случае мы выполнили небольшую постобработку выходных данных. Тем не менее мы могли бы сделать практически что угодно (без поддержки состояния) внутри этой функции.

Несколько сигнатур

Довольно часто модели поддерживают несколько целевых установок или клиентов с разными потребностями. Хотя выдаваемый на выходе словарь может позволить разным клиентам извлекать все, что они захотят, в некоторых случаях это решение бывает неидеальным. Например, функция, которую нам пришлось вызвать, чтобы получить вероятность из логитов, свелась просто к `tf.sigmoid()`. Это ненакладно, и нет никаких проблем с ее вычислением даже для клиентов, которые от нее откажутся. С другой стороны, если бы функция была дорогой, то ее вычисление клиентами, которым ее значение не нужно, может добавить значительные накладные расходы.

Если малое число клиентов требует очень дорогостоящей операции, то полезно предоставлять несколько сигнатур модельного обслуживания и просить клиента сообщать фреймворку модельного обслуживания, какую из них вызывать. Это делается при экспорте модели путем указания имени, отличного от `serving_default`. Например, мы могли бы написать две сигнатуры, используя вот такой фрагмент кода:

```
model.save(export_path, signatures={
        'serving_default': func1,
        'expensive_result': func2,
})
```

Тогда входной JSON-запрос будет содержать имя сигнатуры для выбора желательной модельной конечной точки обслуживания запросов:

```
{
  "signature_name": "expensive_result",
   {"instances": ...}
}
```

Онлайновое предсказание

Поскольку экспортированная функция обслуживания в конечном счете является просто форматом файла, ее можно использовать для обеспечения возможностей онлайнового предсказания, когда исходный фреймворк машинного обучения не поддерживает онлайновые предсказания изначально.

Например, мы можем создать модель, формулирующую вывод о потребности младенца в особом внимании, натренировав логистическую регрессионную модель на наборе натальных данных:

```
CREATE OR REPLACE MODEL
 mlpatterns.neutral_3classes OPTIONS(model_type='logistic_reg',
  input_label_cols=['health']) AS
```

```
SELECT
IF
 (apgar_1min = 10,
  'Healthy',
 IF
  (apgar_1min >= 8,
   'Neutral',
   'NeedsAttention')) AS health,
 plurality,
 mother_age,
 gestation_weeks,
 ever_born
FROM
 `bigquery-public-data.samples.natality`
WHERE
 apgar_1min <= 10
```

Натренировав модель, мы сможем выполнять предсказание с использованием SQL:

```
SELECT * FROM ML.PREDICT(MODEL mlpatterns.neutral_3classes,
    (SELECT
     2  AS plurality,
     32 AS mother_age,
     41 AS gestation_weeks,
     1  AS ever_born
    )
)
```

Однако BigQuery предназначен в первую очередь для распределенной обработки данных. Хотя он отлично подходил для тренировки ML-модели на гигабайтах данных, решение использовать такую систему для выполнения предсказательного вывода на одной строке данных является далеко не лучшим — задержки могут достигать секунды или чуть больше. Функциональность `ML.PREDICT`, скорее, больше подходит для пакетного обслуживания.

Для того чтобы проводить онлайновые предсказания, мы можем попросить службу BigQuery экспортировать модель как сохраненную модель TensorFlow (SavedModel):

```
bq extract -m --destination_format=ML_TF_SAVED_MODEL \
    mlpatterns.neutral_3classes gs://${BUCKET}/export/baby_health
```

Теперь мы можем развернуть модель, сохраненную в формате SavedModel, на платформе модельного обслуживания, таком как Cloud AI Platform, который поддерживает формат SavedModel, получив преимущества модельного обслуживания запросов с низкой задержкой и автомасштабированием. Полный исходный код приведен в блокноте[5] в репозитории на GitHub.

---

[5] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/05_resilience/serving_function.ipynb**.

Даже если бы такой возможности экспортировать модель в формате сохраненной модели не существовало, мы могли бы извлечь веса, написать математическую модель для выполнения линейной модели, контейнеризировать ее и развернуть образ контейнера на платформе модельного обслуживания.

Предсказательная библиотека

Вместо того чтобы развертывать функцию обслуживания запросов как микросервис, которая может вызываться через REST API, есть возможность реализовать код предсказания как библиотечную функцию. Библиотечная функция будет загружать экспортированную модель при первом вызове, вызывать `model.predict()` с предоставленными входными данными и возвращать результат. В этом случае разработчики приложений, которым необходимо выполнять предсказания с помощью библиотеки, смогут включать эту библиотеку в свои приложения.

Библиотечная функция является более подходящей альтернативой микросервису, если модель невозможно вызывать по сети ввиду физических причин (отсутствует сетевое соединение) либо из-за ограничений по производительности. Подход на основе библиотечной функции также возлагает вычислительную нагрузку на клиента, и это бывает предпочтительнее с бюджетной точки зрения. Использование библиотечного подхода с TensorFlow.js позволяет избежать сетевых проблем, когда желательно, чтобы модель работала в браузере.

Главный недостаток библиотечного подхода в том, что техническое сопровождение и обновление модели затруднены — весь клиентский код, который используется в модели, должен будет обновляться, чтобы задействовать новую версию библиотеки. Чем чаще модель обновляется, тем более привлекательным становится подход на основе микросервисов. Вторичным недостатком является то, что библиотечный подход ограничен языками программирования, для которых библиотеки написаны, тогда как подход на основе REST API открывает модель для приложений, написанных практически на любом современном языке программирования.

Разработчик библиотеки должен позаботиться о том, чтобы использовать пул потоков и параллелизацию для поддержания необходимой пропускной способности. Однако при таком подходе обычно существует предел достигаемой масштабируемости.

ПАТТЕРН 17. Пакетное обслуживание

Паттерн "Пакетное обслуживание" (Batch Serving) использует программно-информационную инфраструктуру, широко применяемую для распределенной обработки данных, с целью выполнения предсказательного вывода на большом числе экземпляров одномоментно.

Постановка задачи

Обычно предсказания выполняются по одному и по требованию. Факт мошеннической транзакции по кредитной карте определяется в момент обработки платежа.

Потребность в интенсивной терапии для младенца определяется при осмотре младенца сразу после рождения. Поэтому при развертывании модели с помощью фреймворка в виде сервиса для обработки запросов она настраивается на обработку одного экземпляра или, самое большее, нескольких тысяч экземпляров, вкладываемых в один запрос.

Архитектура фреймворка модельного обслуживания разработана таким образом, чтобы обрабатывать отдельный запрос синхронно и как можно быстрее, как описано в *разд. "Паттерн 16. Функция обслуживания без поддержки состояния"*. Инфраструктура модельного обслуживания обычно строится как микросервис, который выгружает тяжелые вычисления (такие как у глубоких сверточных нейронных сетей) на высокопроизводительное оборудование, такое как тензорные (TPU) или графические процессоры (GPU), и минимизирует неэффективность, связанную с многочисленными слоями программного обеспечения.

Однако существуют обстоятельства, при которых предсказания должны выполняться асинхронно над крупными объемами данных. Например, операция определения необходимости переупорядочивать артикул может выполняться не всякий раз, когда артикул пробивается на кассе, а ежечасно. Музыкальные сервисы могут создавать персонализированные ежедневные плейлисты для каждого пользователя из своей клиенткой базы и передавать их этим пользователям. Персонализированный плейлист не создается по требованию в ответ на каждое взаимодействие пользователя с музыкальным софтом. По этой причине ML-модель должна делать предсказания не для одного экземпляра за раз, а для миллионов экземпляров одновременно.

Попытка вызывать сервис, предназначенный для обработки одного запроса за раз, и отправлять ему миллионы артикулов или миллиарды пользователей приведет к переполнению ML-модели.

Решение

В паттерне "Пакетное обслуживание" используется инфраструктура распределенной обработки данных (MapReduce, Apache Spark, BigQuery, Apache Beam и т. д.) для выполнения предсказательного вывода ML на большом числе экземпляров в асинхронном режиме.

В ходе обсуждения паттерна "Функция обслуживания без поддержки состояния" мы натренировали модель классифицирования текста выводить результат о позитивности либо негативности отзыва. Допустим, мы хотим применить эту модель к каждой жалобе, которая когда-либо поступала в Бюро финансовой защиты потребителей Соединенных Штатов (Consumer Finance Protection Bureau, CFPB).

Мы можем загрузить модель Keras в BigQuery следующим образом (полный исходный код доступен в блокноте[6] на GitHub):

---

[6] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/05_resilience/batch_serving.ipynb**.

```
CREATE OR REPLACE MODEL mlpatterns.imdb_sentiment
OPTIONS(model_type='tensorflow', model_path='gs://.../*')
```

В обычных условиях тренируют модель, используя данные из BigQuery, здесь же мы просто загружаем внешне натренированную модель. Однако теперь есть возможность использовать BigQuery для выполнения предсказаний ML. Например, SQL-запрос

```
SELECT * FROM ML.PREDICT(MODEL mlpatterns.imdb_sentiment,
  (SELECT 'This was very well done.' AS reviews)
)
```

возвращает вероятность позитивного отзыва (`positive_review_probability`), равную 0,82.

Использовать систему распределенной обработки данных, такую как BigQuery, для выполнения разовых предсказаний не очень эффективно. Однако что делать, если мы хотим применить модель машинного обучения к каждой жалобе в базе данных CFPB?[7] Мы можем адаптировать приведенный выше запрос, просто назначив столбцу `consumer_complaint_narrative` во внутреннем `SELECT` псевдоним `reviews` (отзывы) для оценивания:

```
SELECT * FROM ML.PREDICT(MODEL mlpatterns.imdb_sentiment,
  (SELECT consumer_complaint_narrative AS reviews
   FROM `bigquery-public-data`.cfpb_complaints.complaint_database
   WHERE consumer_complaint_narrative IS NOT NULL
   )
)
```

В базе данных содержится более 1,5 млн жалоб, но они обрабатываются примерно за 30 секунд, что доказывает преимущества использования системы распределенной обработки данных.

Почему это работает

Паттерн "Функция обслуживания без поддержки состояния" настроен на модельное обслуживание с низкой задержкой для поддержания тысяч одновременных запросов. Такой способ для нерегулярной или периодической обработки миллионов позиций может оказаться довольно дорогостоящим. Если эти запросы не чувствительны к задержке, гораздо экономичнее использовать архитектуру распределенной обработки данных для вызова моделей машинного обучения на миллионах позиций. Причина в том, что вызов ML-модели на миллионах позиций представляет

[7] Хотите знать, как выглядит "позитивная" жалоба? Тогда вот: я получаю телефонные звонки XXXX утром и вечером. Я сказал звонившим прекратить мне названивать, но они все равно продолжают, даже в воскресенье утром. В воскресенье утром мне дважды подряд звонили из XXXXXXX. В субботу мне позвонили девять раз. Мне также звонят около девяти раз каждый будний день. Единственный намек на то, что жалобщик недоволен, кроется в том, что он попросил звонивших прекратить названивать. В противном случае остальные заявления вполне могут свидетельствовать о том, что кто-то хвастается своей популярностью!

собой ошеломляюще параллельную задачу[8] — можно брать миллион позиций, разбивать их на 1000 групп по 1000 позиций в каждой, отправлять каждую группу позиций в машину, а затем объединять результаты. Результат модели машинного обучения на позиции номер 2000 полностью независим от результата модели машинного обучения на позиции номер 3000, и поэтому работу можно разделить и властвовать над ней.

Возьмем, к примеру, запрос на поиск пяти наиболее позитивных жалоб:

```
WITH all_complaints AS (
SELECT * FROM ML.PREDICT(MODEL mlpatterns.imdb_sentiment,
  (SELECT consumer_complaint_narrative AS reviews
   FROM `bigquery-public-data`.cfpb_complaints.complaint_database
   WHERE consumer_complaint_narrative IS NOT NULL
   )
)
)
SELECT * FROM all_complaints
ORDER BY positive_review_probability DESC LIMIT 5
```

Глядя на детали исполнения в веб-консоли BigQuery, мы видим, что весь запрос занял 35 секунд (см. рамку с пометкой #1 на рис. 5.1).

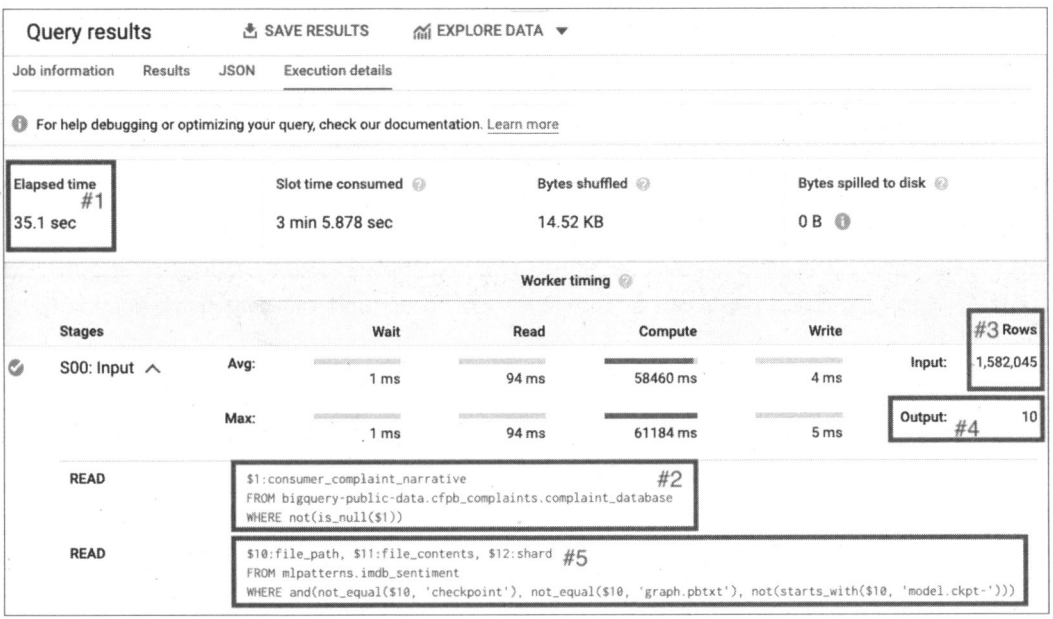


Рис. 5.1. Первые два шага запроса на отыскание пяти наиболее позитивных жалоб в наборе данных о жалобах потребителей в Бюро финансовой защиты потребителей

[8] В параллельных вычислениях ошеломляюще параллельная (embarrassingly parallel) задача, или идеально параллельная задача, — это задача, в которой требуется мало (или вообще никаких) усилий для разделения задачи на ряд параллельных задач. — *Прим. перев.*

Первый шаг (см. рамку #2 на рис. 5.1) читает столбец `consumer_complaint_narrative` из публичного набора данных BigQuery там, где описание жалобы не равно `NULL`. Из числа строк, выделенных в рамке #3, мы узнаем, что это предусматривает чтение 1 582 045 значений. Результат этого шага записывается в 10 шард (см. рамку #4 на рис. 5.1).

Второй шаг читает данные из этого шарда (обратите внимание на `$12:shard` в запросе), но также получает `file_path` и `file_contents` модели машинного обучения `imdb_sentiment` и применяет модель к данным в каждом шарде. Принцип работы MapReduce состоит в том, что каждый шард обрабатывается воркером, поэтому тот факт, что существует 10 шард, указывает на то, что второй шаг выполняется 10 воркерами. Изначальные 1,5 млн строк данных хранились бы разбросанными по многим файлам, и поэтому первый шаг, скорее всего, был бы обработан таким же числом воркеров, что и число файлов, составляющих этот набор данных.

Остальные шаги показаны на рис. 5.2.

```
SORT              $20 DESC
                  LIMIT 5

COMPUTE           $20 := STRUCT_FIELD_OP(1, $30)
                  $21 := STRUCT_FIELD_OP(0, $30)

BUFFERING_COMPUTE $30 := TENSORFLOW_PREDICT_SIGNATURE_ID(MAKE_STRUCT(STRUCT_FIELD_OP(0, $61), STRUCT_FIELD_OP(1, $61),
                  STRUCT_FIELD_OP(2, $61), STRUCT_FIELD_OP(-1, $61)), $60, NULL, ...)

JOIN              CROSS EACH WITH EACH

COMPUTE           $40 := TENSORFLOW_LOAD_TYPE_MODEL_ID(MAKE_STRUCT($52, $51, $50), NULL, '', '', 'mlpatterns.imdb_sentiment',
                  ARRAY<...>)

AGGREGATE         $50 := ARRAY_AGG($12)
                  $51 := ARRAY_AGG($11)
                  $52 := ARRAY_AGG($10)

WRITE             $80, $81, $82
                  TO __stage00_output
```

Рис. 5.2. Третий и последующие шаги запроса позволяют отыскать пять наиболее позитивных жалоб

Третий шаг сортирует набор данных в убывающем порядке и берет пять вариантов. Это делается на каждом воркере, поэтому каждый из 10 воркеров находит 5 самых положительных жалоб в своем шарде. Остальные шаги извлекают и форматируют оставшиеся порции данных и записывают их в выход.

На завершающем шаге (не показан) выбираются 50 жалоб, сортируются и выбираются 5, которые и формируют фактический результат. Возможность разделить работу именно в таком ключе между многими воркерами позволяет BigQuery выполнять операцию целиком на 1,5 млн жалоб за 35 секунд.

Компромиссы и альтернативы

Паттерн "Пакетное обслуживание" зависит от возможности разделения задачи между несколькими воркерами. Поэтому он не ограничивается складами данных или даже языком SQL. Справится любой фреймворк MapReduce. Однако склады дан-

ных SQL, как правило, являются самым простым подходом и нередко выбираются по умолчанию, в особенности когда данные структурированы по своей природе.

Несмотря на то что пакетное обслуживание используется тогда, когда задержка приемлема, существует возможность встраивать предварительно вычисленные результаты и периодическое обновление, чтобы использовать их в сценариях, в которых пространство возможных входных переменных предсказания лимитировано.

Пакетные и потоковые конвейеры

Такие инструменты, как Apache Spark или Apache Beam, полезны, когда входные переменные нуждаются в предобработке перед подачей их в модель, если результаты на выходе из модели машинного обучения требуют постобработки или если предобработка либо постобработка трудно выразима на SQL. Если на вход в модель поступают фото, аудио или видео, то SQL не годится и нужен инструмент, который может обрабатывать неструктурированные данные. Эти инструменты также могут использовать для предобработки снимков преимущества ускоренного аппаратного обеспечения, такого как TPU и GPU.

Еще одна причина применять инструмент, подобный Apache Beam, состоит в необходимости поддержания клиентским кодом внутреннего состояния. Распространенная причина, по которой клиент должен поддерживать состояние, заключается в том, что одной из переменных, поступающих на вход в ML-модель, является среднее значение во временном окне. В этом случае клиентский код должен вычислять скользящее среднее входящего потока данных и подавать скользящее среднее в модель ML.

Представьте себе, что мы строим систему модерации комментариев и хотим отклонять отзывы пользователей, которые пишут комментарии о конкретном человеке более двух раз в день. Например, мы оставляем первые два сообщения, в которых комментатор пишет что-то о президенте Обаме, но блокируем все попытки этого комментатора упомянуть президента Обаму в течение оставшегося дня. Это пример постобработки, которой требуется поддерживать состояние, потому что нам нужен счетчик упоминаний той или иной знаменитости для каждого участника дискуссии. Кроме того, этот счетчик должен обрабатывать циклически чередующийся 24-часовой период времени.

Это можно сделать с помощью инструмента распределенной обработки данных, который способен поддерживать состояние. В этой ситуации на первый план выходит фреймворк Apache Beam. Вызов ML-модели для идентифицирования упоминаний знаменитости и ее привязывания к каноническому графу знаний (чтобы упоминание "Обама" и упоминание "президент Обама" были привязаны к **en.wikipedia.org/wiki/Barack_Obama**) из Apache Beam можно выполнить с помощью следующего фрагмента кода (полный исходный код см. в блокноте[9] на GitHub):

---

[9] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/05_resilience/nlp_api.ipynb**.

```
| beam.Map(lambda x : nlp.Document(x, type='PLAIN_TEXT'))
| nlp.AnnotateText(features)
| beam.Map(parse_nlp_result)
```

где `parse_nlp_result` выполняет разбор JSON-запроса, проходящего через трансформанту `AnnotateText`, которая, в свою очередь, за кулисами вызывает API естественно-языковой обработки (NLP).

Кэшированные результаты пакетного обслуживания

Мы обсудили тему пакетного обслуживания как подход к вызову модели на миллионах элементов в ситуации, когда модель стандартно эксплуатируется как служба в онлайновом режиме с использованием паттерна "Функция обслуживания без поддержки состояния". Разумеется, пакетное обслуживание может работать, даже если модель не поддерживает онлайновое обслуживание. Важно, чтобы фреймворк машинного обучения, который выполняет предсказательный вывод, был способен использовать преимущества параллельной обработки.

Например, рекомендательные механизмы должны заполнять разреженную матрицу, состоящую из каждой пары "пользователь — позиция". Типичное предприятие может иметь 10 млн постоянных клиентов и 10 тыс. позиций в каталоге продуктов. Для того чтобы дать рекомендацию пользователю, баллы рекомендаций должны быть расчитаны по каждой позиции из 10 тыс., ранжированы и верхние 5 представлены пользователю. Это невозможно сделать в режиме, близком к реальному времени, вне функции обслуживания. Тем не менее требование, близкое к реальному времени, означает, что простое использование пакетного обслуживания тоже не будет работать.

В таких случаях пакетное обслуживание следует использовать для предварительного вычисления рекомендаций для всех 10 млн пользователей:

```
SELECT
  *
FROM
  ML.RECOMMEND(MODEL mlpatterns.recommendation_model)
```

Указанное вычисление следует сохранить в реляционной базе данных, такой как MySQL, Datastore или Cloud Spanner (это могут сделать предварительно построенные передаточные службы и заготовки потоковой службы Dataflow[10]). При посещении веб-сайта любым пользователем рекомендации для этого пользователя извлекаются из базы данных и доставляются немедленно и с очень низкой задержкой.

В фоновом режиме рекомендации периодически обновляются. Например, мы могли бы перетренировывать рекомендательную модель ежечасно, основываясь на последних действиях на веб-сайте. Тогда мы сможем выполнять предсказательный вывод только для тех пользователей, которые посетили веб-сайт за последний час:

---

[10] См. **https://github.com/GoogleCloudPlatform/DataflowTemplates/blob/master/src/main/java/com/google/cloud/teleport/templates/BigQueryToDatastore.java**.

```
SELECT
  *
FROM
  ML.RECOMMEND(MODEL mlpatterns.recommendation_model,
    (
    SELECT DISTINCT
      visitorId
    FROM
      mlpatterns.analytics_session_data
    WHERE
      visitTime > TIME_DIFF(CURRENT_TIME(), 1 HOUR)
    ))
```

и обновлять соответствующие строки в реляционной базе данных, используемой для обработки запросов к модели.

Лямбда-архитектура

Производственная ML-система, которая поддерживает и онлайновое, и пакетное обслуживание, называется *лямбда-архитектурой*[11] — такая ML-система позволяет практикам машинного обучения находить компромисс между задержкой (посредством паттерна "Функция обслуживания без поддержки состояния") и пропускной способностью (посредством паттерна "Пакетное обслуживание").

Платформа AWS Lambda[12], несмотря на свое название, не имеет лямбда-архитектуры. Это бессерверная платформа для масштабирования функций без поддержки состояния, аналогичная инструментам Google Cloud Functions или Azure Functions.

Как правило, лямбда-архитектура поддерживается наличием отдельных систем для онлайнового и пакетного обслуживания. Например, в Google Cloud инфраструктура онлайнового обслуживания обеспечивается компонентом Cloud AI Platform Prediction, а инфраструктура пакетного обслуживания — службой BigQuery и потоковой аналитической службой Cloud Dataflow (компонент Cloud AI Platform Predictions обеспечивает удобный интерфейс, благодаря которому пользователям не нужно явным образом использовать Dataflow). Существует возможность импортирования модели TensorFlow в BigQuery для пакетного обслуживания. Также можно брать натренированную модель BigQuery ML и экспортировать ее в специфичном для TensorFlow формате сохраненной модели SavedModel для онлайнового обслуживания. Эта двусторонняя совместимость позволяет пользователям Google Cloud достигать любого компромисса между задержкой и пропускной способностью.

---

[11] См. **https://oreil.ly/jLZ46**.

[12] См. **https://oreil.ly/RqPan**.

ПАТТЕРН 18. Непрерывное оценивание модели

Паттерн "Непрерывное оценивание модели" (Continued Model Evaluation) решает распространенную проблему, связанную с необходимостью обнаружения и принятия мер, когда развернутая модель больше не соответствует целевому назначению.

Постановка задачи

Итак, вы натренировали свою модель. Вы собрали сырые данные, очистили их, сгенерировали признаки, создали слои векторного вложения, настроили гиперпараметры и все такое. Вы способны достичь точности 96% на отложенном тестовом наборе. Удивительно! Вы даже прошли через кропотливый процесс развертывания своей модели, перенеся ее из блокнота Jupyter в продуктивную среду в виде модели машинного обучения, и обрабатываете запросы на модельное предсказание через REST API. Поздравляю, вы сделали это. Всё, конец — делу венец!

Всё так, но не совсем. Развертывание не конец жизненного цикла модели машинного обучения. Откуда вы знаете, что ваша модель в "дикой" природе работает так, как и ожидалось? Что делать, если во входящих данных есть неожиданные изменения? Или если модель больше не дает точных либо полезных предсказаний? Как эти изменения будут обнаруживаться?

Мир динамичен, но разработка модели машинного обучения обычно создает статическую модель из исторических данных. А это означает, что, начав работать, модель может начать деградировать, а ее предсказания могут становиться все более ненадежными. Две главные причины временно́й деградации моделей — это дрейф концепции и дрейф данных.

Дрейф концепции возникает всякий раз, когда связь между входными данными и целью модели меняется. Это часто происходит из-за того, что опорные допущения вашей модели изменились, например в моделях, натренированных распознавать враждебное или конкурентное поведение, таких как обнаружение мошенничества, спам-фильтры, торговля на фондовом рынке, торги онлайновой рекламой или кибербезопасность. В этих сценариях предсказательная модель направлена на выявление регулярностей, характерных для желательной (или нежелательной) активности, тогда как противник учится приспосабливаться и, возможно, модифицирует свое поведение по мере изменения обстоятельств. Подумайте, например, о модели, разработанной для обнаружения мошенничества с кредитными картами. Характер использования людьми кредитных карт со временем изменился, и, следовательно, варианты мошенничества с кредитными картами тоже изменились. Например, когда была введена технология "chip and PIN"[13], мошеннические транзакции стали перемещаться в онлайн. По мере того как мошенническое поведение адаптировалось, результативность модели, которая была разработана до появления этой технологии, внезапно начала страдать, и модельные предсказания стали менее точными.

---

[13] Банковские карты с чипом и ПИН-кодом. — *Прим. ред.*

Еще одной причиной деградирования модельной результативности с течением времени является *дрейф данных*. Мы представили проблему дрейфа данных в *разд. "Распространенные проблемы машинного обучения" главы 1*. Дрейф данных относится к любому изменению, которое произошло в данных, подаваемых в вашу модель для генерирования предсказания, по сравнению с данными, которые использовались для тренировки. Дрейф данных может происходить по ряду причин: схема входных данных изменяется в источнике (например, поля добавляются или удаляются на вышестоящем уровне), признаковое распределение с течением времени меняется (например, в больницу поступает больше молодых людей, потому что поблизости открылся горнолыжный курорт) или смысл данных меняется, даже если структура/схема не изменилась (например, условие, определяющее избыточный вес пациента, может с течением времени измениться). Обновления программного обеспечения могут приводить к появлению новых дефектов или изменению делового сценария и созданию новой метки продукта, ранее отсутствовавшей в тренировочных данных. Конвейеры ETL для строительства, тренировки и предсказания с помощью ML-моделей могут быть хрупкими и непрозрачными, и любое из этих изменений окажет резкое влияние на результативность вашей модели.

Развертывание модели — непрерывный процесс, и для решения проблемы дрейфа концепции или дрейфа данных необходимо обновлять тренировочный набор данных и перетренировывать модель свежими данными с целью улучшения предсказаний. Но как узнать, когда необходима перетренировка? И как часто следует перетренировывать? Предобработка данных и тренировка модели бывают дорогостоящими как по времени, так и по деньгам, и каждый шаг цикла разработки модели вносит дополнительные накладные расходы на разработку, мониторинг и техническое сопровождение.

Решение

Самый простой подход к выявлению ухудшения модели состоит в постоянном мониторинге модельной результативности предсказания во временно́й динамике и ее оценивании с помощью тех же метрик оценивания, которые вы использовали во время разработки. Посредством такого непрерывного оценивания и мониторинга модели мы определяем, что модель или любые изменения, которые мы внесли в модель, работают так, как они должны работать.

Концепция

Подобного рода непрерывное оценивание требует доступа к сырым данным предсказательных запросов и сгенерированным моделью предсказаниям, а также к эмпирическому наблюдению — всё в одном месте. Инструментальная платформа Google Cloud AI Platform предоставляет возможность конфигурировать развернутую версию модели таким образом, чтобы обеспечивать регулярное взятие выборок из входных и выходных данных онлайнового предсказания и сохранение в таблице BigQuery. В целях поддержания результативности службы при большом числе запросов в секунду можно настраивать объем отбираемых данных под свои нужды,

задавая процент от числа входных запросов. Для измерения метрик результативности необходимо соотносить эту сохраненную выборку предсказаний с эмпирическим наблюдением.

В большинстве ситуаций, прежде чем станут доступны метки эмпирических наблюдений, может пройти немало времени. Например, в случае модели оттока клиентуры бывает, что до наступления следующего цикла подписки не ясно, какие клиенты прекратили свое обслуживание. Или, в модели финансового предсказания случается, что истинная выручка остается неизвестной до тех пор, пока не будет опубликован отчет о закрытии и корпоративных заработках текущего квартала. В любом из этих случаев оценивание нельзя проводить до тех пор, пока не будут получены данные эмпирических наблюдений.

Для того чтобы познакомиться с непрерывным оцениванием, развернем модель классифицирования текста, натренированную на наборе данных HackerNews, на платформе Google Cloud AI Platform. Полный исходный код этого примера можно найти в блокноте непрерывного оценивания[14] в репозитории, прилагаемом к этой книге.

Развертывание модели

Входным материалом для нашего тренировочного набора данных является заголовок статьи, а связанная с ним метка — это источник новостей, из которого возникла статья: nytimes, techcrunch либо github. По мере эволюционирования новостных трендов со временем слова, связанные с заголовком "New York Times", будут меняться. Аналогично выпуск новых технологических продуктов повлияет на слова, которые можно найти в TechCrunch. Непрерывное оценивание позволяет нам осуществлять мониторинг модельных предсказаний, следя за тем, как эти тренды влияют на результативность модели, и при необходимости запускать процесс перетренировки.

Предположим, что модель экспортируется с помощью нашей конкретной прикладной входной функции обслуживания (*см. разд. "Паттерн 16. Функция обслуживания без поддержки состояния" ранее в этой главе*):

```
@tf.function(input_signature=[tf.TensorSpec([None], dtype=tf.string)])
def source_name(text):
    labels = tf.constant(['github', 'nytimes', 'techcrunch'],dtype=tf.string)
    probs = txtcls_model(text, training=False)
    indices = tf.argmax(probs, axis=1)
    pred_source = tf.gather(params=labels, indices=indices)
    pred_confidence = tf.reduce_max(probs, axis=1)
    return {'source': pred_source,
            'confidence': pred_confidence}
```

После развертывания этой модели, во время онлайнового предсказания, модель будет возвращать предсказанный источник новостей в виде строкового значения и

[14] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/05_resilience/continuous_eval.ipynb**.

числовой оценки этой предсказательной метки, связанной со степенью уверенности, которая была у модели. Например, мы можем создать онлайновое предсказание, записав входной пример в формате JSON в файл input.json для отправки на выполнение предсказания:

```
%%writefile input.json
{"text":
"YouTube introduces Video Chapters to make it easier to navigate longer videos"}
```

Этот запрос вернет следующий результат предсказания:

```
CONFIDENCE  SOURCE
0.918685    techcrunch
```

Сохранение предсказаний

После того как модель будет развернута, мы можем настроить задание на сохранение выборки предсказательных запросов — причина сохранения не всех запросов, а только выборки, обусловлена стремлением избежать ненужного замедления работы системы модельного обслуживания. Мы можем сделать это в разделе непрерывного оценивания инструментальной платформы Google Cloud AI Platform (Continuous Evaluation section of the Google Cloud AI Platform, CAIP), указав `LabelKey` (столбец, являющийся результатом работы модели, который в нашем случае будет источником, поскольку мы предсказываем источник статьи), `ScoreKey` в результатах предсказания (числовое значение, которое в нашем случае является уверенностью, confidence) и таблицу в BigQuery, где хранится часть онлайновых предсказательных запросов. В нашем примере исходного кода таблица называется `txtcls_eval.swivel`. После всех настроек всякий раз, когда будут делаться онлайновые предсказания, платформа CAIP будет передавать имя модели, версию модели, временну́ю метку предсказательного запроса, сырые входные данные предсказания и результат работы модели в указанную таблицу BigQuery (табл. 5.1).

Таблица 5.1. *Часть запросов на онлайновое предсказание и сырых результатов предсказания, сохраненная в таблице в BigQuery*

Строка	model	model_version	time	raw_data	raw_prediction	groundtruth
1	txtcls	swivel	2020-06-10 01:40:32 UTC	{"instances": [{"text":"Astronauts Dock With Space Station After Historic SpaceX Launch"}]}	{"predictions": [{"source":"github", "confidence": 0.999427556991577 1}]}	null
2	txtcls	swivel	2020-06-10 01:37:46 UTC	{"instances": [{"text":"Senate Confirms First Black Air Force Chief"}]}	{"predictions": [{"source":"nytimes", "confidence":0.9989787 340164185}]}	null
3	txtcls	swivel	2020-06-09 21:21:47 UTC	{"instances": [{"text":"A native Mac app wrapper for WhatsApp Web"}]}	{"predictions": [{"source":"github", "confidence": 0.745254397392273}]}	null

Улавливание эмпирического наблюдения

Кроме того, необходимо уловить эмпирическое наблюдение по каждому экземпляру, отправляемому в модель для предсказания. Это делается с помощью ряда подходов в зависимости от варианта использования и доступности данных. Один из подходов — это привлечение человека к разметке: все экземпляры, отправляемые в модель для предсказания, или, возможно, только те из них, для которых модель имеет маргинальную уверенность, отправляются человеку для аннотирования. Большинство облачных поставщиков предлагают некоторую форму такой службы разметки, которая обеспечивает возможность выполнять разметку экземпляров в требуемом ключе.

Метки эмпирических наблюдений также можно получить из того, как пользователи взаимодействуют с моделью и ее предсказаниями. Побуждая пользователей совершать те или иные действия, можно наладить неявную обратную связь для модельного предсказания или создавать метку эмпирических наблюдений. Например, когда пользователь выбирает один из предложенных альтернативных маршрутов в Google Maps, выбранный маршрут служит неявным эмпирическим наблюдением. Когда пользователь оценивает рекомендованный кинофильм, можно утверждать, что модель, построенная для предсказаниях оценок пользователей с целью получения рекомендаций, действительно работоспособна. Если модель позволяет пользователю изменять предсказание, например, как в медицинской обстановке, когда врач может изменить предложенный моделью диагноз, то это дает четкий сигнал об эмпирическом наблюдении.

Важно учитывать то, как цикл обратной связи модельных предсказаний и улавливание эмпирического наблюдения могут повлиять на тренировочные данные в будущем. Например, предположим, что вы построили модель, чтобы предсказывать ситуации, когда корзина покупок будет заброшена. Вы даже можете проверять состояние корзины через регулярные промежутки времени, чтобы создавать метки эмпирических наблюдений для оценивания модели. Однако если ваша модель говорит о том, что пользователь от своей корзины покупок откажется, а вы предложите ему бесплатную доставку или некую скидку, чтобы повлиять на его поведение, то вы никогда не узнаете, было ли изначальное модельное предсказание верным. Иными словами, вы нарушили допущение принципа оценивания модели и должны будете определить метки эмпирических наблюдений каким-то другим путем. Эта задача оценивания конкретного исхода при другом сценарии называется контрфактическим (counterfactual) рассуждением[15] и часто возникает в таких сферах использования, как обнаружение мошенничества, медицина и реклама, в которых существует вероятность того, что модельные предсказания будут приводить к некоторому вмешательству, способному затмить усвоение актуального эмпирического наблюдения из этого примера.

Оценивание результативности модели

Первоначально столбец `groundtruth` таблицы `txtcls_eval.swivel` в BigQuery пуст. Мы можем предоставить метки эмпирических наблюдений, как только они станут доступны, обновив значение напрямую с помощью команды SQL. Разумеется, пре-

[15] То есть в предположении о том, что было бы в некоторой гипотетической ситуации. — *Прим. перев.*

жде чем мы запустим задание по оцениванию, мы должны обеспечить наличие эмпирического наблюдения. Обратите внимание, что эмпирическое наблюдение придерживается той же структуры JSON, что и результат предсказания на выходе из модели:

```
UPDATE
  txtcls_eval.swivel
SET
  groundtruth = '{"predictions": [{"source": "techcrunch"}]}'
WHERE
  raw_data = '{"instances":
[{"text": "YouTube introduces Video Chapters to help navigate longer videos"}]}' '
```

Для обновления большего числа строк мы могли бы использовать инструкцию `MERGE` вместо инструкции `UPDATE`. После добавления эмпирического наблюдения в таблицу есть возможность легко проверить текстовые входные данные и модельное предсказание и сравнить с эмпирическим наблюдением (табл. 5.2):

```
SELECT
  model,
  model_version,
  time,
  REGEXP_EXTRACT(raw_data, r'.*"text": "(.*)"') AS text,
  REGEXP_EXTRACT(raw_prediction, r'.*"source": "(.*?)"') AS prediction,
  REGEXP_EXTRACT(raw_prediction, r'.*"confidence": (0.\d{2}).*') AS confidence,
  REGEXP_EXTRACT(groundtruth, r'.*"source": "(.*?)"') AS groundtruth,
FROM
  txtcls_eval.swivel
```

Таблица 5.2. *Таблица BigQuery после появления эмпирического наблюдения, его добавления в изначальную таблицу и оценивания результативности модели*

Строка	model	model_version	time	text	prediction	confidence	groundtruth
1	txtcls	swivel	2020-06-10 01:38:13 UTC	A native Mac app wrapper for WhatsApp Web	github	0.77	github
2	txtcls	swivel	2020-06-10 01:37:46 UTC	Senate Confirms First Black Air Force Chief	nytimes	0.99	nytimes
3	txtcls	swivel	2020-06-10 01:40:32 UTC	Astronauts Dock With Space Station After Historic SpaceX Launch	github	0.99	nytimes
4	txtcls	swivel	2020-06-09 21:21:44 UTC	YouTube introduces Video Chapters to make it easier to navigate longer videos	techcrunch	0.77	techcrunch

Имея доступ к этой информации в BigQuery, мы можем загрузить оценочную таблицу в кадр данных `df_evals` и непосредственно вычислить метрики оценивания для

этой версии модели. Поскольку у нас многоклассовая классификация, мы можем вычислить точность, полноту и балл оценки F1 для каждого класса. Мы также можем создать матрицу путаницы, помогающую анализировать ситуации, когда модельные предсказания могут пострадать в рамках некоторых категориальных меток. На рис. 5.3 показана матрица путаницы, сравнивающая предсказания этой модели с эмпирическим наблюдением.

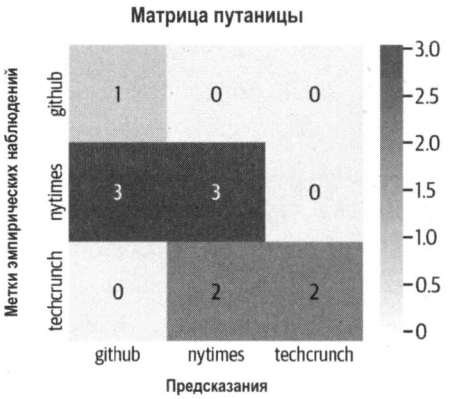


Рис. 5.3. Матрица путаницы показывает все пары меток эмпирических наблюдений и предсказаний, поэтому вы можете провести разведывательный анализ результативности вашей модели в разных классах

Непрерывное оценивание

Вдобавок мы должны обеспечить возможность улавливания выходной таблицей версии модели и временной метки предсказательных запросов, чтобы использовать одну и ту же таблицу для непрерывного оценивания двух разных версий модели в целях сравнения метрик между моделями. Например, если мы развертываем новую версию модели, под названием `swivel_v2`, которая тренируется на более свежих данных или имеет другие гиперпараметры, то мы можем сравнить их результативность, нарезав кадр данных оценивания в соответствии с версией модели:

```
df_v1 = df_evals[df_evals.version == "swivel"]
df_v2 = df_evals[df_evals.version == "swivel_v2"]
```

Аналогично мы можем создать срезы оценивания во времени, сосредоточиваясь только на модельных предсказаниях в течение последнего месяца или последней недели:

```
today = pd.Timestamp.now(tz='UTC')
one_month_ago = today - pd.DateOffset(months=1)
one_week_ago = today - pd.DateOffset(weeks=1)

df_prev_month = df_evals[df_evals.time >= one_month_ago]
df_prev_week = df_evals[df_evals.time >= one_week_ago]
```

В целях проведения вышеуказанных оцениваний в непрерывном режиме можно запланировать работу блокнота (или контейнерезированной формы) по расписа-

нию. Мы можем настроить его так, чтобы он запускал перетренировку модели, если метрика оценивания падает ниже некоторого порога.

Почему это работает

Во время разработки моделей машинного обучения присутствует неявное допущение о том, что тренировочные, валидационные и тестовые данные поступают из одного и того же распределения (рис. 5.4). Когда мы развертываем модели в производстве, то из указанного допущения вытекает, что будущие данные окажутся похожими на прошлые. Однако, как только модель будет запущена в производство "в дикой природе", это статическое предположение о данных может оказаться недействительным. На самом деле многие промышленные ML-системы сталкиваются с быстро меняющимися нестационарными данными, а модели со временем приобретают черты застарелости, что негативно сказывается на качестве предсказаний.

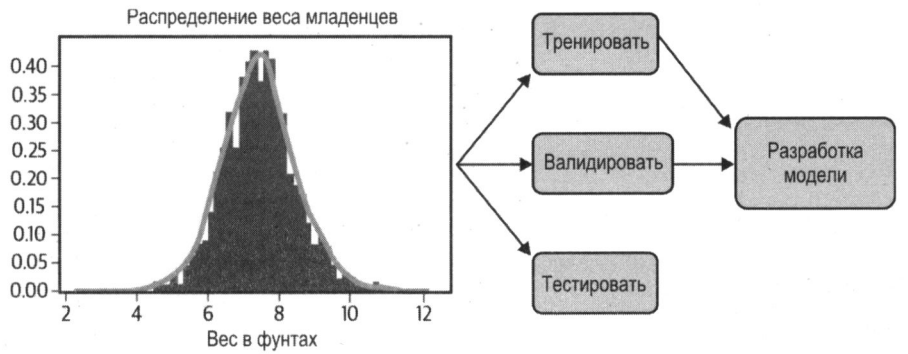


Рис. 5.4. При разработке ML-модели тренировочные, валидационные и тестовые данные поступают из одного и того же распределения данных. Однако после развертывания модели это распределение может измениться, что серьезно повлияет на результативность модели

Непрерывное оценивание модели обеспечивает каркас для оценивания результативности развернутой модели исключительно на новых данных. Это позволяет нам как можно раньше обнаруживать застарелость модели. Указанная информация помогает определять частоту перетренировки модели или время, когда следует полностью заменять ее новой версией.

Захватывая входные и выходные данные предсказания и сравнивая их с эмпирическим наблюдением, можно количественно отслеживать результативность модели или измерять качество работы разных версий модели с помощью A/B-тестирования в текущей среде, независимо от качества работы версий в прошлом.

Компромиссы и альтернативы

Цель непрерывного оценивания состоит в том, чтобы обеспечивать средства для мониторинга результативности моделей и поддержания моделей в производственной среде в актуальном состоянии. При таком подходе непрерывное оценивание обеспечивает триггер, сигнализирующий о ситуациях, когда следует выполнять

перетренировку модели. И здесь важно учитывать допустимые пределы для результативности модели, создаваемые ими компромиссы и роль плановой перетренировки. Существуют также технические приемы и инструменты, такие как TFX, которые помогают упреждающе обнаруживать дрейф данных и дрейф концепции, осуществляя непосредственный мониторинг распределения во входных данных.

Триггеры для перетренировки

Результативность модели обычно со временем деградирует. Непрерывное оценивание позволяет высокоточно структурированно измерять степень ее устаревания и обеспечивает триггер для перетренировки модели. Значит ли это, что вы должны перетренировывать свою модель, как только ее результативность начинает падать? Все зависит от обстоятельств. Ответ на этот вопрос тесно связан с бизнес-кейсом использования и должен обсуждаться наряду с метриками оценивания и оцениванием модели. В зависимости от сложности модели и конвейеров ETL стоимость перетренировки бывает дорогостоящей. Следует учитывать компромисс между тем, какая величина ухудшения результативности является приемлемой, и ее стоимостью.

Бессерверные триггеры

Cloud Functions, AWS Lambda и Azure Functions предоставляют бессерверные подходы к автоматизированию перетренировки посредством триггеров. Тип триггера определяет способ и время исполнения вашей функции. Эти триггеры могут быть сообщениями, публикуемыми в очереди сообщений, уведомлением об изменениях корзины облачного хранилища, указывающим на добавление нового файла, изменениями данных в базе данных или даже HTTPS-запросом. После того как событие запущено, исходный код функции исполняется.

В контексте перетренировки триггер облачного события будет представлен значительным изменением или падением точности модели. Функция, или действие, будет заключаться в вызове тренировочного конвейера для перетренировки модели и развертывания новой версии. В *разд. "Паттерн 25. Конвейер рабочего потока" главы 6* описан принцип выполнения этого процесса. Конвейеры рабочих потоков содержат и организуют сквозной рабочий поток процессов машинного обучения от сбора данных и проверки до разработки, тренировки и развертывания моделей. После развертывания новой версии модели ее можно сравнивать с текущей версией, чтобы определять необходимость ее замены.

Сам порог может быть установлен как абсолютное значение; например, перетренировка модели происходит, когда точность модели падает ниже 95%. Как вариант, порогом может быть темп изменения результативности, например, когда результативность начинает испытывать нисходящую траекторию. Каким бы ни был подход, философия выбора порога аналогична философии фиксации состояния модели в контрольной точке во время тренировки. При более высоком, более чувствительном пороге промышленные модели остаются актуальными, правда, при этом высоки затраты на частую перетренировку, а также технические накладные расходы на

сопровождение и переключение между разными версиями моделей. При более низком пороге затраты на тренировку снижаются, но модели становятся более застарелыми. На рис. 5.5 отражен компромисс между порогом результативности и тем, как он влияет на число заданий по перетренировке моделей.

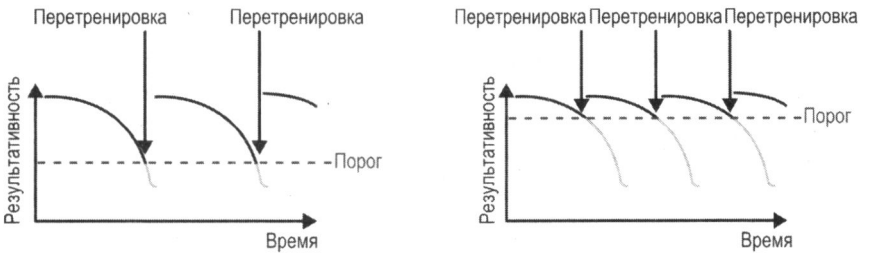


Рис. 5.5. Установка более высокого порога для результативности модели обеспечивает более качественную промышленную модель, но потребует более частых перетренировок, что бывает дорогостоящим

Если конвейер перетренировки моделей автоматически запускается таким порогом, то важно также отслеживать и валидировать триггеры. Незнание момента, когда ваша модель была перетренирована, неизбежно приводит к проблемам. Даже если процесс автоматизирован, вам всегда следует контролировать перетренировку модели, чтобы лучше понимать и отлаживать производственную модель.

Плановая перетренировка

Непрерывное оценивание дает ключевой сигнал для понимания времени, когда необходимо перетренировывать модель. Этот процесс перетренировки нередко осуществляется путем тонкой настройки предыдущей модели с использованием любых вновь собранных тренировочных данных. Там, где непрерывное оценивание можно выполнять каждый день, плановая перетренировка может происходить только каждую неделю или каждый месяц (рис. 5.6).

После тренировки новой версии модели ее результативность сравнивается с текущей версией модели. Обновленная модель развертывается в качестве замены только в том случае, если она по результативности превосходит предыдущую модель по отношению к тестовому набору текущих данных.

Рис. 5.6. Непрерывный мониторинг обеспечивает оценивание модели каждый день по мере сбора новых данных. Периодическая перетренировка и сравнение моделей используются для оценивания в дискретные моменты времени

И как часто следует планировать перетренировку по расписанию? Расписание перетренировки будут зависеть от бизнес-кейса использования, распространенности новых данных и стоимости (временно́й и денежной) исполнения конвейера перетренировки. Иногда временной горизонт модели естественным образом определяет момент, когда следует планировать работу по перетренировке. Например, если цель модели состоит в предсказывании корпоративных заработков в следующем квартале, поскольку вы будете получать новые метки эмпирических наблюдений только раз в квартал, то нет смысла тренировать чаще. Однако если объем и встречаемость новых данных велики, то было бы выгодно выполнять перетренировку чаще. Самой экстремальной версией перетренировки является онлайновое машинное обучение[16]. Некоторые приложения машинного обучения, такие как размещение рекламы или рекомендации в новостной ленте, требуют принятия решений в режиме реального времени и могут постоянно улучшать результативность путем перетренировки и обновления параметрических весов с каждым новым тренировочным примером.

В общем случае оптимальный временной интервал вы, как специалист-практик, будете определять с помощью опыта и экспериментов. Если вы пытаетесь смоделировать задачу с быстро изменяющимися условиями, такую как враждебное или конкурентное поведение, то имеет смысл устанавливать более свободный план перетренировки. Если задача достаточно статична, например предсказание веса младенца при рождении, то менее частых перетренировок будет достаточно.

В любом случае полезно иметь автоматизированный конвейер, который может исполнять полный процесс перетренировки с помощью одного-единственного вызова API. Такие инструменты, как службы Cloud Composer/Apache Airflow и AI Platform Pipelines, полезны для создания, планирования запусков и мониторинга рабочих процессов ML — от предобработки сырых данных и тренировки до гиперпараметрической настройки и развертывания. Мы обсудим это в *разд. "Паттерн 25. Конвейер рабочего потока" главы 6.*

Валидация данных с помощью TFX

Распределение данных с течением времени может изменяться (рис. 5.7). Например, рассмотрим вес младенцев при рождении в наборе натальных данных. По мере того как с течением времени медицина и социальные стандарты меняются, связь между модельными признаками, такими как возраст матери или срок беременности, меняется по отношению к модельной метке, весу младенца. Этот дрейф данных отрицательно влияет на способность модели обобщать на новых данных. Одним словом, модель приобрела черты застарелости, и ее нужно перетренировать на свежих данных.

В то время как непрерывное оценивание обеспечивает постфактумный подход к мониторингу развернутой модели, также полезно выполнять мониторинг новых данных, получаемых во время модельного обслуживания, и упреждающе выявлять изменения в распределениях данных.

---

[16] См. **https://oreil.ly/Mj-DA**.

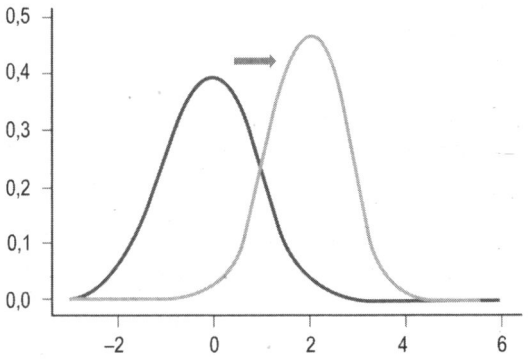


Рис. 5.7. Распределение данных с течением времени может меняться. Дрейф данных относится к любому изменению, которое произошло в данных, подаваемых в модель для генерирования предсказания, по сравнению с данными, использованными для тренировки

Компонент валидации данных TFX Data Validation является полезным инструментом для достижения этой цели. TFX[17] — это сквозная платформа с открытым исходным кодом для развертывания моделей машинного обучения разработки компании Google. Компонент Data Validation можно применять для сравнения примеров данных, используемых в тренировке, с данными, собранными во время обработки запросов. Проверки валидности обнаруживают аномалии в данных, асимметрию между тренировкой и обслуживанием или дрейф данных. TensorFlow Data Validation создает визуализации данных с помощью Facets[18], инструмента визуализации с открытым исходным кодом для машинного обучения. Facets Overview предоставляет высокоуровневый взгляд на распределение значений по различным признакам и может раскрывать несколько обычных и необычных проблем, таких как неожиданные значения признаков, пропущенные значения признаков и асимметрия между тренировкой и обслуживанием.

Оценивание интервала перетренировки

Полезная и относительно дешевая тактика, позволяющая понять характер влияния дрейфа данных и дрейфа концепции на модель, заключается в тренировке модели на устаревших данных и оценивании результативности этой модели на более свежих данных (рис. 5.8). Таким образом можно сымитировать непрерывный процесс оценивания модели в офлановой среде, т. е. надо собрать данные полугодовой или годовой давности и пройти обычный рабочий маршрут разработки модели, генерируя признаки, оптимизируя гиперпараметры и захватывая соответствующие метрики оценивания, а затем сравнить эти метрики оценивания с модельными предсказаниями для более свежих данных, собранных всего за месяц до этого. Насколько хуже устаревшая модель работает с текущими данными? Ответ на тот вопрос позволяет оценить значение темпа падения результативности модели с течением времени и частоты перетренировки, которая может потребоваться.

---

[17] См. **https://oreil.ly/RP2e9**.

[18] См. **https://oreil.ly/NE-SQ**.

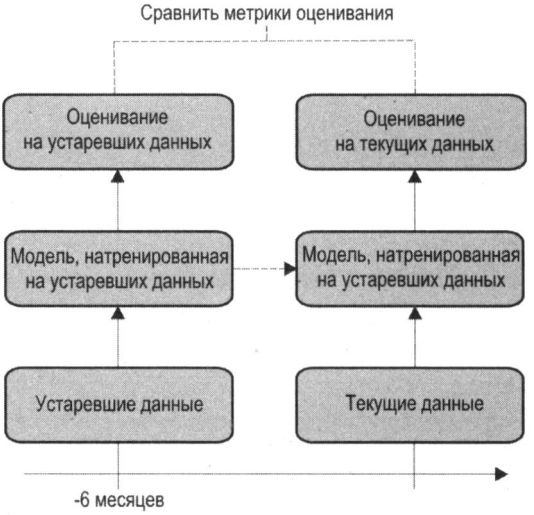


Рис. 5.8. Тренировка модели на устаревших данных и оценивание на текущих данных имитируют непрерывный процесс оценивания модели в офлайновой среде

ПАТТЕРН 19. Двухфазные предсказания

Паттерн "Двухфазные предсказания" (Two-Phase Predictions) обеспечивает подход к решению задачи поддержания результативности крупных сложных моделей в ситуациях, когда есть потребность в их развертывании на распределенных устройствах, путем разделения вариантов использования на две фазы, при этом на периферии выполняется только более простая фаза.

Постановка задачи

При развертывании моделей машинного обучения мы не всегда можем полагаться на конечных пользователей в том, что у них будут надежные интернет-соединения. В таких ситуациях модели развертываются на периферии, т. е. скачиваются на устройство пользователя и не требуют подключения к Интернету для генерирования предсказаний. Если учитывать технические пределы устройств, следует отметить, что модели, развернутые на периферии, должны быть меньше, чем модели, развернутые в облаке, и поэтому требуют балансировки компромиссов между модельной сложностью и размером, частотой обновления, точностью и низкой задержкой.

Существуют разнообразные сценарии, по которым можно развертывать свои модели на периферийном устройстве. Один из примеров — устройство контроля физической подготовки, в котором модель дает рекомендации пользователям на основе их активности, отслеживаемой с помощью акселерометра и гироскопа. Вполне вероятно, что пользователь будет заниматься физкультурой дистанционно на открытой площадке без возможности интернет-соединения. В этих случаях мы хотим, чтобы приложение все равно работало. Еще одним примером является экологическое приложение, в котором используются температура и другие метеоданные для

предсказаний в отношении будущих трендов. В обоих примерах, даже если у нас есть интернет-соединение, непрерывное генерирование предсказаний из модели, развернутой в облаке, может оказаться медленным и дорогостоящим.

В целях конвертирования натренированной модели в формат, работающий на периферийных устройствах, модели часто проходят процесс, именуемый *квантизацией*, при котором усвоенные модельные веса представляются меньшим числом байтов. В TensorFlow, например, используется формат TensorFlow Lite для конвертирования[19] сохраненных моделей в меньший формат, оптимизированный под обработку запросов на периферии. Модели, предназначенные для периферийных устройств, в дополнение к квантизации могут также первоначально иметь меньший размер, чтобы вписываться в строгие технологические пределы по памяти и процессору.

Квантизация и другие методы, используемые в TF Lite, значительно уменьшают размер и задержку предсказания результирующих ML-моделей, но вместе с этим может снижать и точность модели. Кроме того, поскольку мы не можем неизменно опираться на периферийные устройства, имеющие возможность подключения, также представляет проблему и своевременное развертывание новых версий моделей на этих устройствах.

Как компромиссы достигаются на практике, можно увидеть, взглянув на варианты тренировки периферийных моделей в службе Cloud AutoML Vision[20] на рис. 5.9.

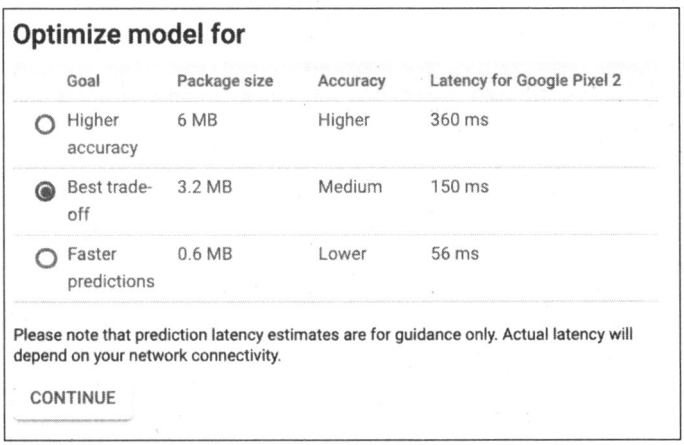


Рис. 5.9. Поиск компромиссов между модельной точностью, размером и задержкой для моделей, развернутых на периферии в службе Cloud AutoML Vision

В целях учета этих компромиссов нам нужно техническое решение, которое уравновешивает уменьшенный размер и задержку периферийных моделей с добавленной усложненностью и точностью облачных моделей.

---

[19] См. **https://oreil.ly/UaMq7**.

[20] См. **https://oreil.ly/MWsQH**.

Решение

С помощью паттерна "Двухфазные предсказания" мы подразделяем нашу задачу на две части. Мы начинаем с меньшей, более дешевой модели, которую можно развернуть на периферийном устройстве. Поскольку эта модель обычно имеет более простую задачу, она способна выполнять эту задачу на периферийном устройстве с относительно высокой точностью. За ней следует вторая, более сложная модель, развернутая в облаке и запускаемая только при необходимости. Разумеется, указанный паттерн требует, чтобы у вас была задача, которую можно подразделить на две части с разными уровнями сложности. Одним из примеров такой задачи являются умные устройства, такие как Google Home[21], которые активируются сигналом пробуждения и затем могут отвечать на вопросы и откликаться на команды, связанные с настройкой будильника, чтением новостей и взаимодействием со встроенными устройствами, такими как лампы и термостаты. Google Home, например, активируется словами "OK Google" или "Hey Google". После того как устройство распознает слово пробуждения, пользователи могут задавать более сложные вопросы, такие как: "Сможешь назначить встречу с Сарой на 10 часов утра?"

Эту задачу можно разбить на две отличимые части: первоначальную модель, которая прослушивает слово пробуждения, и более сложную модель, которая способна понимать и отвечать на любой другой запрос пользователя. Обе модели будут выполнять распознавание звука. Первая модель, однако, должна будет выполнять только двоичное классифицирование: соответствует ли звук, который она только что услышала, слову пробуждения или нет? Хотя эта модель по сложности проще, она должна работать постоянно, что будет обходиться дорого, если ее развернуть в облаке. Вторая модель потребует распознавания звука и понимания естественного языка для разбора запроса пользователя. Эта модель должна работать только тогда, когда пользователь задает вопрос, но делает больший акцент на высокой точности. Паттерн "Двухфазные предсказания" может решить эту задачу путем развертывания модели слова пробуждения на периферийном устройстве, а более сложной модели — в облаке.

В дополнение к этому варианту использования умного устройства паттерн "Двухфазные предсказания" может задействоваться в целом ряде других ситуаций. Допустим, вы являетесь сотрудником завода, где в определенный момент времени работает много разных машин. Когда машина перестает работать правильно, она обычно издает шум, который может быть связан с неисправностью. Разные шумы соответствуют каждой отдельной машине, и каждая машина может сломаться поразному. В идеале вы можете построить модель, которая будет помечать проблемные шумы и выявлять смысл, который они означают. С помощью паттерна "Двухфазные предсказания" можно было бы построить первую, офлайновую, модель для обнаружения аномальных звуков. Затем задействовать вторую, облачную, модель, чтобы выявлять, не является ли обычный звук признаком какого-то неисправного состояния.

---

[21] См. **https://oreil.ly/3ROKg**.

Паттерн "Двухфазные предсказания" можно также использовать для сценария на основе снимков. Предположим, у вас развернуты камеры в дикой природе, чтобы идентифицировать и отслеживать исчезающие виды животных. Одна модель у вас может быть на устройстве, и в ее обязанность входит обнаружение исчезающего животного на последнем снимке. Если факт обнаружения подтверждается, этот снимок можно отправить в облачную модель, которая определяет конкретный тип животного на фото.

В целях иллюстрации паттерна "Двухфазные предсказания" давайте воспользуемся общецелевым набором данных распознавания звука из Kaggle[22]. Указанный набор данных содержит около 9 тыс. звуковых семплов знакомых звуков с 41 категорией меток, включая "виолончель", "стук", "телефон", "труба" и многие другие. Первой фазой нашего технического решения будет модель, которая предсказывает, исходит ли данный звук от музыкального инструмента. Затем для звуков, которые первая модель предсказывает как звуки музыкальных инструментов, мы будем получать предсказание из модели, развернутой в облаке, чтобы предсказывать конкретный инструмент из общего числа 18 возможных вариантов. На рис. 5.10 показан двухфазный процесс для данного примера.

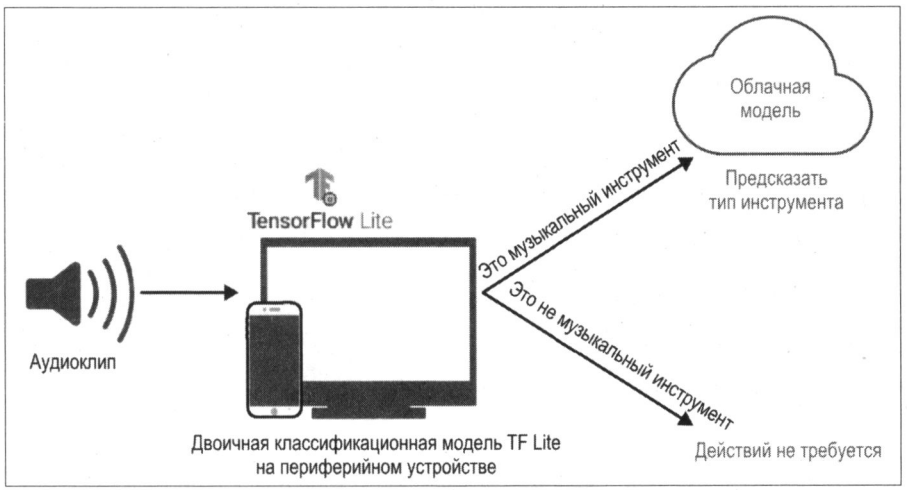


Рис. 5.10. Использование паттерна "Двухфазные предсказания" для идентифицирования звуков музыкальных инструментов

Для построения каждой из этих моделей мы выполним конвертирование звуковых данных в спектрограммы, которые являются визуальными представлениями звука. Это позволит нам использовать архитектуры распространенных снимковых моделей наряду с паттерном "Трансферное обучение" для решения этой задачи. На рис. 5.11 приведена спектрограмма аудиофрагмента звучания саксофона из нашего набора данных.

[22] См. **https://oreil.ly/189Pr**.

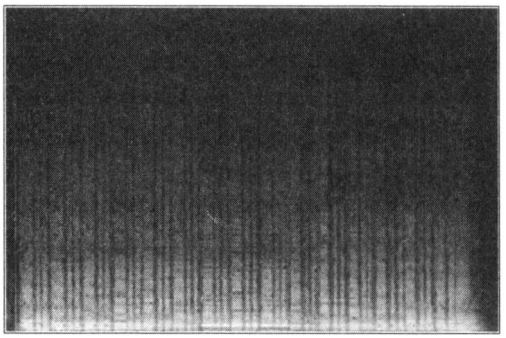


Рис. 5.11. Снимок (спектрограмма) аудиофрагмента звучания саксофона из нашего тренировочного набора данных. Исходный код для конвертирования WAV-файлов в спектрограммы можно найти в репозитории[23] на GitHub

Фаза 1: построение офлайновой модели

Первая модель в нашем техническом решении на основе паттерна "Двухфазные предсказания" должна быть достаточно малой, чтобы ее можно было загружать на мобильное устройство для быстрого предсказательного вывода, не полагаясь на интернет-соединение. Отталкиваясь от приведенного выше примера с музыкальным инструментом, мы приведем пример первой фазы предсказания, построив двоичную классификационную модель, оптимизированную под предсказательный вывод прямо на периферийном устройстве.

Исходный набор звуковых данных содержит 41 метку для разных типов аудиоклипов. Наша первая модель будет иметь только две метки: "инструмент" или "не инструмент". Мы построим нашу модель, используя архитектуру модели MobileNetV2[24], натренированную на наборе данных ImageNet. MobileNetV2 доступна непосредственно в Keras и представляет собой архитектуру, оптимизированную под обработку запросов на периферийном устройстве. В случае нашей модели мы заморозим веса MobileNetV2 и загрузим ее *без* верхней части, вследствие чего сможем добавить наш собственный выходной слой двоичного классифицирования:

```
mobilenet = tf.keras.applications.MobileNetV2(
    input_shape=((128,128,3)),
    include_top=False,
    weights='imagenet'
)
mobilenet.trainable = False
```

Если мы организуем наши изображения спектрограмм в каталоги с соответствующим именем метки, то сможем использовать класс `ImageDataGenerator` фреймворка Keras для создания тренировочного и валидационного наборов данных:

---

[23] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/05_resilience/audio_to_spectro.ipynb**.

[24] См. **https://oreil.ly/zvbzR**.

```
train_data_gen = image_generator.flow_from_directory(
    directory=data_dir,
    batch_size=32,
    shuffle=True,
    target_size=(128,128),
    classes = ['not_instrument','instrument'],
    class_mode='binary')
```

Когда тренировочный и валидационный наборы данных будут готовы, мы сможем натренировать модель так, как это делается обычно. Типичным подходом к экспортированию натренированных моделей для обработки запросов является использование TensorFlow-метода `model.save()`. Вместе с тем следует помнить, что эта модель будет обслуживать запросы на периферийном устройстве, и следовательно, она должна быть как можно меньше. Для создания модели, которая вписывается в эти требования, мы будем использовать библиотеку TensorFlow Lite[25], оптимизированную под разработку и обработку запросов непосредственно на мобильных и встроенных устройствах, которые могут не иметь надежного интернет-соединения. В TF Lite есть несколько встроенных утилит для квантизации моделей как во время тренировки, так и после нее.

В целях подготовки натренированной модели для периферийной обработки запросов мы используем TF Lite для ее экспортирования в оптимизированный формат:

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()
open('converted_model.tflite', 'wb').write(tflite_model)
```

Этот подход к квантизированию модели *после* ее тренировки является самым быстрым. Используя настройки оптимизации, принятые в TF Lite по умолчанию, указанный подход будет сводить веса модели к их 8-битному представлению. Он также будет квантизировать входные данные во время предсказательного вывода, когда мы делаем предсказания на нашей модели. Выполнив приведенный выше исходный код, результирующая экспортированная модель TF Lite будет иметь размер в одну четвертую размера, который был бы, если бы мы экспортировали ее без квантизации.

В целях еще большего оптимизирования модели под офлайновый предсказательный вывод вы также можете квантизировать веса модели во время тренировки либо квантизировать все математические операции модели в дополнение к весам. На момент написания этой книги оптимизируемая квантизацией тренировка моделей TensorFlow 2 числится в дорожной карте[26].

Для генерирования предсказания на модели TF Lite мы используем интерпретатор TF Lite Interpreter, который оптимизирован под низкую задержку. Скорее всего, вы захотите загрузить свою модель на устройство Android или iOS и генерировать

[25] См. **https://oreil.ly/dyx93**.

[26] См. **https://oreil.ly/RuONn**.

предсказания непосредственно из кода приложения. Для обеих платформ имеются свои API, но здесь мы покажем исходный код генерирования предсказаний на языке Python, чтобы можно было запускать его из того же блокнота, в котором вы создали свою модель. Сначала мы создаем экземпляр интерпретатора TF Lite и получаем подробную информацию о входном и выходном форматах, которые он ожидает:

```
interpreter = tf.lite.Interpreter(model_path="converted_model.tflite")
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

Для двоичной классификационной модели MobileNetV2, которую мы натренировали выше, переменная `input_details` с деталями выходных данных выглядит следующим образом:

```
[{'dtype': numpy.float32,
  'index': 0,
  'name': 'mobilenetv2_1.00_128_input',
  'quantization': (0.0, 0),
  'quantization_parameters': {'quantized_dimension': 0,
  'scales': array([], dtype=float32),
  'zero_points': array([], dtype=int32)},
  'shape': array([  1, 128, 128,   3], dtype=int32),
  'shape_signature': array([  1, 128, 128,   3], dtype=int32),
  'sparsity_parameters': {}}]
```

Затем мы передадим первый снимок из нашего валидационного пакета в загруженную модель TF Lite для предсказания, вызовем интерпретатор и получим результат:

```
input_data = np.array([image_batch[21]], dtype=np.float32)
interpreter.set_tensor(input_details[0]['index'], input_data)

interpreter.invoke()
output_data = interpreter.get_tensor(output_details[0]['index'])
print(output_data)
```

На выходе мы получим сигмоидный массив с одним-единственным значением в промежутке [0; 1], указывающим на то, является ли поступивший на вход звук звуком музыкального инструмента.

В зависимости от того, насколько дорого обходится вызов облачной модели, можете также изменить метрику, под которую вы оптимизируете модель, когда тренируете ее для периферийного устройства. Например, если вы больше всего стремитесь избегать ложных утверждений (ложноположительных результатов), можете оптимизировать не под полноту, а под прецизионность.

Теперь, когда модель работает на устройстве, мы можем получать быстрые предсказания без необходимости полагаться на интернет-соединение. Если модель уверена в том, что данный звук не является звуком музыкального инструмента, мы

можем на этом остановиться. Если же модель предсказывает "инструмент", то самое время перейти к отправке аудиоклипа в более сложную облачную модель.

Какие модели подходят для периферии?

Как определить, что модель подходит для работы на периферии? На этот счет есть несколько соображений, связанных с размером модели, ее сложностью и имеющимся аппаратным обеспечением. Как правило, малые, менее сложные модели оптимизированы под работу на периферийном устройстве лучше. Это обусловлено тем, что периферийные модели ограничены хранилищем, которое доступно на устройстве. Нередко, когда модели масштабируются с помощью квантизации или других технических приемов, это делается в ущерб точности. В силу этого модели с более простой задачей предсказания и архитектурой подходят для периферийных устройств лучше всего. Под "проще" мы подразумеваем такие компромиссы, как предпочтение двоичной классификации перед многоклассовой или выбор менее сложной архитектуры модели (например, модели, основанной на дереве решений, или линейно-регрессионной модели), когда это возможно.

Когда вам приходится разворачивать модели на периферии, при этом все-таки придерживаясь некоторых пределов по размеру модели и ее сложности, стоит взглянуть на периферийное аппаратное обеспечение, специально разработанное с учетом предсказательного вывода ML. Например, плата Coral Edge TPU[27] обеспечивает прикладную микросхему ASIC, оптимизированную под высокопроизводительный офлайновый предсказательный вывод ML на моделях TensorFlow Lite. Подобным же образом, компания NVIDIA предлагает набор инструментов Jetson Nano[28] для оптимизированного под периферию маломощного предсказательного вывода ML. Аппаратная поддержка предсказательного вывода ML быстро эволюционирует по мере того, как встроенное периферийное ML становится все более распространенным.

Фаза 2: построение облачной модели

Поскольку наша облачная модель не нуждается в оптимизации под предсказательный вывод без сетевого соединения, мы можем следовать более традиционному подходу к тренировке, экспортированию и развертыванию этой модели. В зависимости от вашего варианта использования на основе паттерна "Двухфазные предсказания" эта вторая модель может принимать множество разных форм. В примере с Google Home фаза 2 могла бы включать в себя несколько моделей: одну, которая на входе конвертирует речь говорящего пользователя в текст, и вторую, которая выполняет естественно-языковую обработку, чтобы понять текст и маршрутизировать запрос пользователя. Если пользователь просит что-то более сложное, можно предусмотреть и третью модель, которая могла бы предоставлять рекомендацию на основе предпочтений пользователя или прошлой активности.

---

[27] См. **https://oreil.ly/N2NOs**.

[28] См. **https://oreil.ly/GUOQc**.

В нашем примере с музыкальным инструментом второй фазой решения будет мультиклассовая модель, которая классифицирует звуки в одну из 18 возможных категорий музыкальных инструментов. Поскольку эта модель не нуждается в развертывании на периферийном устройстве, в качестве отправной точки можно использовать более крупную модельную архитектуру, такую как VGG, а затем следовать паттерну "Трансферное обучение", описанному в *главе 4*.

Мы загрузим VGG, натренированную на наборе данных ImageNet, зададим размер наших спектрограмм в параметре `input_shape` и заморозим модельные веса перед добавлением нашего собственного классификационного выходного слоя с функцией мягкого максимума:

```
vgg_model = tf.keras.applications.VGG19(
    include_top=False,
    weights='imagenet',
    input_shape=((128,128,3))
)
vgg_model.trainable = False
```

На выходе получим 18-элементный массив softmax-вероятностей:

```
prediction_layer = tf.keras.layers.Dense(18, activation='softmax')
```

Мы ограничим наш набор данных только аудиоклипами музыкальных инструментов, а затем преобразуем метки инструментов в 18-элементные векторы, кодированные с одним активным состоянием. Для подачи снимков в модель и ее тренировки мы можем использовать такой же подход, как в вышеприведенном генераторе `image_generator`. Вместо того чтобы экспортировать нашу модель как модель TF Lite, мы можем применить `model.save()`, чтобы экспортировать ее для обработки запросов.

В целях демонстрации развертывания модели фазы 2 в облаке мы будем использовать службу Cloud AI Platform Prediction[29]. Нужно закачать сохраненные активы модели в корзину хранилища Cloud Storage, а затем развернуть модель, указав фреймворк и направив службу AI Platform Prediction в нашу корзину хранения.

Для второй фазы паттерна "Двухфазные предсказания" можно использовать любой другой облачный инструмент развертывания моделей. Помимо службы AI Platform Prediction облачного инструментария Google Cloud, инфраструктуры AWS SageMaker[30] и Azure Machine Learning[31] предлагают службы для развертывания моделей.

Когда мы экспортируем модель в формате SavedModel фреймворка TensorFlow, мы можем передавать URL-адрес корзины хранилища непосредственно в метод сохранения модели `save()`:

```
model.save('gs://your_storage_bucket/path')
```

---

[29] См. **https://oreil.ly/P5Cn9**.

[30] См. **https://oreil.ly/zIHey**.

[31] См. **https://oreil.ly/dCxHE**.

Приведенная выше инструкция экспортирует модель в формате TF SavedModel и закачивает ее в корзину хранилища Cloud Storage.

На платформе AI Platform ресурс модели содержит разные версии вашей модели. Каждая модель может иметь сотни версий. Сначала мы создадим ресурс модели с помощью интерфейса командной строки gcloud инфраструктуры Google Cloud:

```
gcloud ai-platform models create instrument_classification
```

Существует несколько подходов к развертыванию модели. Мы воспользуемся интерфейсом gcloud и направим платформу AI Platform в подкаталог хранилища, содержащий наши сохраненные модели:

```
gcloud ai-platform versions create v1 \
  --model instrument_classification \
  --origin 'gs://your_storage_bucket/path/model_timestamp' \
  --runtime-version=2.1 \
  --framework='tensorflow' \
  --python-version=3.7
```

Теперь мы можем отправлять предсказательные запросы в нашу модель через API службы AI Platform Prediction, который поддерживает онлайновое и пакетное предсказание. Онлайновое предсказание позволяет получать предсказания почти в режиме реального времени сразу на нескольких примерах. Если у нас сотни или тысячи примеров, которые мы хотим отправить для предсказания, мы можем создать пакетное предсказательное задание, которое будет работать асинхронно в фоновом режиме и выводить результаты предсказания в файл по завершении.

Для урегулирования ситуаций, когда вызывающее нашу модель периферийное устройство не всегда соединено с Интернетом, мы могли бы хранить аудиоклипы для предсказания музыкального инструмента на периферийном устройстве до тех пор, пока оно находится в офлайновом режиме. Когда устройство будет восстанавливать связь, мы будем отправлять эти клипы в облачную модель для предсказания.

Компромиссы и альтернативы

Хотя паттерн "Двухфазные предсказания" хорошо работает во многих случаях, следует учитывать ситуации, когда у ваших конечных пользователей интернет-соединение очень слабое, и поэтому шансы на вызов модели, размещенной в облаке, весьма невелики. В этом разделе мы обсудим две исключительно офлайновые альтернативы — сценарий, в котором клиент должен делать много предсказательных запросов за один раз, и рекомендации в отношении того, как выполнять непрерывное оценивание для офлайновых моделей.

Автономная однофазная модель

Иногда у конечных пользователей вашей модели вообще может отсутствовать интернет-соединение. Даже если устройства этих пользователей не смогут надежно получать доступ к облачной модели, все равно важно предоставлять им какой-то способ доступа к вашему приложению. В этом случае вместо того, чтобы полагать-

ся на двухфазный предсказательный процесс, вы можете сделать свою первую модель довольно-таки устойчивой, чтобы она была самодостаточной.

Для этого мы можем создать уменьшенную версию нашей сложной модели и предоставить пользователям возможность скачивать эту более простую, меньшую модель для использования в офлайновом режиме. Эти офлайновые модели могут быть не такими точными, как их более крупные онлайновые аналоги, но такое техническое решение будет лучше, чем отсутствие офлайновой поддержки вообще. Для построения более сложных моделей, предназначенных для офлайнового предсказательного вывода, лучше всего использовать инструмент, который позволяет квантизировать модельные веса и другие математические операции как во время тренировки, так и после нее. Этот технический прием называется *тренировкой с учетом квантизации* (quantization aware training)[32].

Одним из примеров приложения, которое предоставляет более простую офлайновую модель, является Google Переводчик[33]. Это надежная служба онлайнового перевода с сотен языков. Однако во многих сценариях вы будете нуждаться в использовании службы перевода без интернет-доступа. В таких ситуациях Google Переводчик позволяет скачивать офлайновые переводы на более чем 50 разных языков. Эти офлайновые модели невелики, около 40–50 Мбайт, и близки по точности к более сложным онлайновым версиям. На рис. 5.12 показано сравнение качества периферийной и онлайновой моделей перевода.

Рис. 5.12. Сравнение периферийной фразовой и (более новой) нейронной моделей машинного перевода и онлайнового нейронного машинного перевода (источник: The Keyword[34])

Еще одним примером автономной однофазной модели является Google Bolo[35] — речевое приложение для изучения языка, предназначенное для детей. Приложение работает полностью в офлайновом режиме и было разработано с намерением помочь населению там, где надежный доступ в Интернет не всегда имеется.

---

[32] См. **https://oreil.ly/ABd8r**.

[33] См. **https://oreil.ly/uEWAM**.

[34] См. **https://oreil.ly/S_woM**.

[35] См. **https://oreil.ly/zTy79**.

Офлайновая поддержка для специфических вариантов использования

Еще одно техническое решение, которое позволяет вашему приложению работать для пользователей с минимальным интернет-соединением, состоит в том, чтобы делать доступными в офлайновом режиме только некоторые части вашего приложения. Сюда может входить предоставление нескольких общих признаков в офлайновом режиме или кэширование результатов предсказания ML-модели для последующего использования в офлайновом режиме. С этой альтернативой мы попрежнему задействуем две фазы предсказания, но ограничиваемся вариантами использования, охватываемыми нашей офлайновой моделью. При таком подходе приложение работает в офлайновом режиме, но обеспечивает полную функциональность, когда восстанавливает соединение.

Например, Google Maps позволяет заранее скачивать карты и маршруты движения. В целях экономии места на мобильном устройстве за счет ограничения числа хранимых направлений движения в офлайновом режиме могут быть доступны только направления движения на автомобиле (не ходьба или езда на велосипеде). Еще одним примером может быть фитнес-приложение, которое считает ваши шаги и дает рекомендации для будущей активности. Скажем, самое распространенное использование этого приложения — это проверка количества пройденных шагов за текущий день. В целях поддержания этого варианта использования в офлайновом режиме мы могли бы синхронизировать данные фитнес-трекера с устройством пользователя по Bluetooth, там самым обеспечивая возможность проверять состояние физической формы за текущий день в офлайновом режиме. Для оптимизации результативности такого приложения можно было бы сохранять историю физической формы и делать рекомендации доступными только в онлайновом режиме.

Мы могли бы продолжать наращивать функциональность приложения, сохраняя запросы пользователя, пока его устройство находится в офлайновом режиме, и отправляя их в облачную модель, когда оно восстановит соединение, чтобы обеспечивать более подробные результаты. Вдобавок, мы могли бы даже предоставлять базовую рекомендательную модель, функционирующую в офлайновом режиме, с намерением дополнять его улучшенными результатами, когда приложение в состоянии отправлять запросы пользователя в облачную модель. При таком техническом решении пользователь по-прежнему получает некоторую функциональность, даже если интернет-соединение отсутствует. Когда он вернется в сеть, то сможет воспользоваться полнофункциональным приложением и надежной ML-моделью.

Манипулирование многочисленными предсказаниями почти в режиме реального времени

Бывает, что надежное интернет-соединение все-таки имеется у конечных пользователей вашей ML-модели, но им может потребоваться делать сотни или даже тысячи предсказательных запросов к вашей модели одномоментно. Если у вас есть лишь облачная модель и каждое предсказание требует API-вызова внешней службы, то

получение предсказательных ответов на тысячах примеров одномоментно будет занимать слишком много времени.

Давайте предположим, что у нас есть встроенные устройства, и они развернуты в различных местах по всему дому пользователя. Эти устройства собирают данные о температуре, давлении и качестве воздуха. Наша модель развернута в облаке для обнаружения аномалий в этих датчиковых данных. Поскольку датчики собирают новые данные непрерывно, было бы неэффективно и дорого отправлять каждую входящую точку данных в облачную модель. Вместо этого мы можем развернуть модель непосредственно на датчиках, чтобы выявлять в поступающих данных возможных аномальных кандидатов. Тогда мы сможем отправлять в облачную модель только потенциальные аномалии для консолидированной валидации, принимая во внимание показания датчиков со всех мест. Этот подход является вариацией описанного ранее паттерна "Двухфазные предсказания", главное отличие которой состоит в том, что и офлайновая, и облачная модели выполняют одну и ту же задачу предсказания, но с разными входными переменными. В этом случае модели будут регулировать число предсказательных запросов, отправляемых в облачную модель за один раз.

Непрерывное оценивание для офлайновых моделей

Как обеспечить актуальность и независимость от дрейфа данных наших моделей на периферийных устройствах? Существует несколько вариантов выполнения непрерывного оценивания на моделях, не имеющих сетевого соединения. Прежде всего, мы могли бы сохранять подмножество предсказаний, полученных на периферийном устройстве. Тогда мы могли бы периодически оценивать результативность нашей модели на этих примерах и определять потребность модели в перетренировке. В случае нашей двухфазной модели важно проводить это оценивание регулярно, т. к. вполне вероятно, что многие вызовы нашей модели на периферийном устройстве не будут переходить на облачную модель второй фазы. Еще один вариант — создавать реплику нашей модели на периферийном устройстве для выполнения в онлайновом режиме только для целей непрерывного оценивания. Это техническое решение более предпочтительно, если офлайновая и облачная модели выполняют похожие задачи предсказания, как в упомянутом ранее случае естественно-языкового перевода.

ПАТТЕРН 20. Предсказания по ключу

В обычных условиях тренировка модели осуществляется на том же наборе входных признаков, который будет предоставляться ей в режиме реального времени при ее развертывании. Во многих ситуациях, однако, бывает также выгодно, чтобы модель пропускала через себя поставляемый клиентом ключ. Такой подход реализуется паттерном "Предсказания по ключу" (Keyed Predictions), и ключ является необходимым условием для масштабируемой реализации нескольких рассмотренных в этой главе паттернов.

Постановка задачи

Если ваша модель развернута как сервис и на входе принимает всего одну переменную, то совершенно ясно, что выходная переменная соответствует входной переменной. Но что делать, если ваша модель на входе принимает файл с миллионом переменных и отправляет на выход файл с миллионом предсказаний?

Вы, возможно, подумаете, что первый выходной экземпляр, очевидно, соответствует первому входному экземпляру, второй выходной экземпляр — второму входному экземпляру и т. д. Однако при соотношении 1:1 каждый узел сервера должен будет обрабатывать последовательно полный набор входных данных. Было бы гораздо выгоднее, если бы вы использовали систему распределенной обработки данных и распределяли экземпляры по нескольким машинам, собирали все полученные выходные данные и отправляли их обратно. Проблема с этим подходом заключается в том, что выходные данные будут перемешаны. Требование, чтобы выходные данные были одинаково упорядочены, создает трудности для масштабируемости, а предоставление выходных данных в неупорядоченном виде предполагает, что клиенты знают, какая переменная на входе соответствует своему результату на входе.

Эта же проблема возникает, если ваша система обработки запросов принимает массив экземпляров, как описано в паттерне "Функция обслуживания без поддержки состояния". Проблема в том, что обработка большого числа экземпляров локально приведет к "горячим" точкам. Серверные узлы, получающие лишь несколько запросов, смогут поспевать, но любой серверный узел, получающий особо большой массив, начнет отставать. Эти "горячие" точки обяжут вас делать серверные машины гораздо мощнее, чем они должны быть. Поэтому многие обработки запросов накладывают ограничение на число экземпляров, которые могут отправляться в одном запросе. Если такого ограничения нет либо модель настолько затратна в вычислительном отношении, что запросы с меньшим числом экземпляров, чем это ограничение, могут приводить к избыточной нагрузке на сервер, то вы столкнетесь с проблемой "горячих" точек. Следовательно, любое решение задачи пакетного обслуживания также решит проблему "горячих" точек в онлайновой обработке.

Решение

Решение заключается в использовании сквозных ключей. Попросите клиента предоставить ключ, связанный с каждой входной переменной. Например, предположим, что ваша модель тренируется с использованием трех переменных (A, B, C) на входе (рис. 5.13, слева), чтобы на выходе получать результат D. Тогда надо лишь дать вашим клиентам возможность поставлять в модель (K, A, B, C), где K — это ключ с уникальным идентификатором. Ключ может быть чем-то простым, вроде нумерации входных экземпляров 1, 2, 3, ... Ваша модель тогда будет возвращать (K, D), и клиент сможет выяснять, какой экземпляр на выходе соответствует своему экземпляру на входе.

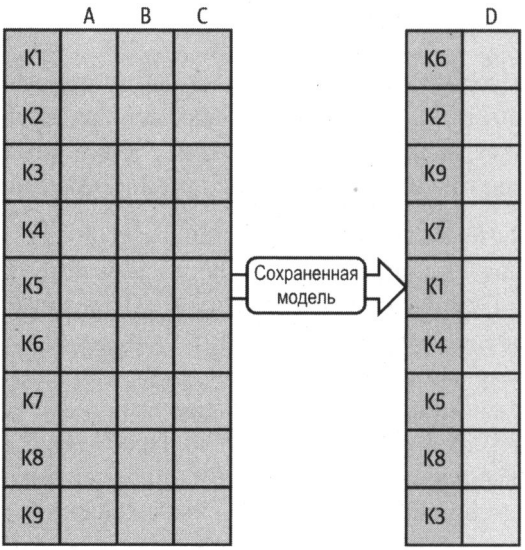


Рис. 5.13. Клиент предоставляет на входе уникальный ключ для каждого экземпляра. Система обработки запросов присоединяет указанные ключи к соответствующему предсказанию. Это позволяет клиенту получать правильное предсказание для каждой переменной на входе, даже если результаты на выходе не упорядочены

Как проносить сквозные ключи в Keras

Для того чтобы ваша модель Keras проносила сквозные ключи, во время экспортирования модели следует предоставлять сигнатуру модельного обслуживания.

Например, ниже приведен фрагмент исходного кода, вызывающий модель, которая в противном случае принимала бы четыре входные переменные (`is_male`, `mother_age`, `plurality` и `gestation_weeks`), и дающий ей возможность брать ключ, который она будет проносить через всю обработку вместе с изначальным результатом работы модели (`babyweight`):

```
# Функция обслуживания, которая проносит сквозные ключи
@tf.function(input_signature=[{
    'is_male': tf.TensorSpec([None,], dtype=tf.string, name='is_male'),
    'mother_age': tf.TensorSpec([None,], dtype=tf.float32, name='mother_age'),
    'plurality': tf.TensorSpec([None,], dtype=tf.string, name='plurality'),
    'gestation_weeks': tf.TensorSpec([None,], dtype=tf.float32,
                                     name='gestation_weeks'),
    'key': tf.TensorSpec([None,], dtype=tf.string, name='key')
}])
def keyed_prediction(inputs):
    feats = inputs.copy()
    key = feats.pop('key') # взять ключ из входной переменной
    output = model(feats)  # вызвать модель
    return {'key': key, 'babyweight': output}
```

Затем эта модель сохраняется, как описано в паттерне "Функции обслуживания без поддержки состояния":

```
model.save(EXPORT_PATH,
           signatures={'serving_default': keyed_prediction})
```

Добавление возможности предсказания по ключу в существующую модель

Обратите внимание, что приведенный выше исходный код работает даже в том случае, если изначальная модель не была сохранена с функцией обслуживания. Нужно лишь загрузить модель, используя метод `tf.saved_model.load()`, прикрепить функцию обслуживания и применить приведенный выше фрагмент исходного кода (рис. 5.14).

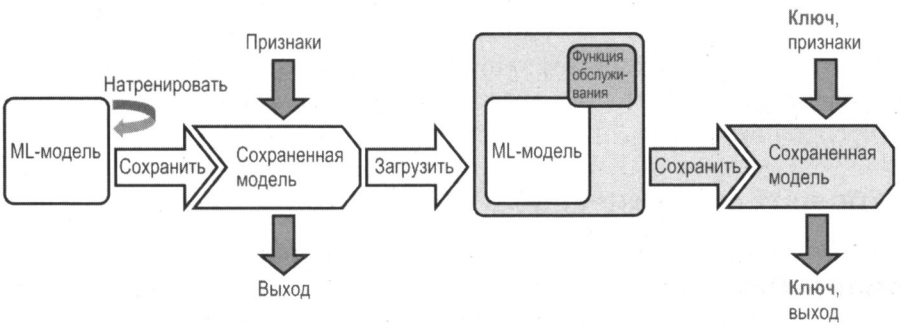


Рис. 5.14. Загрузить модель, сохраненную в формате SavedModel, прикрепить не заданную по умолчанию функцию обслуживания и сохранить модель

При этом предпочтительно предоставить функцию обслуживания, которая воспроизводит старое поведение без ключа:

```
# Функция обслуживания, которая не требует ключа
@tf.function(input_signature=[{
    'is_male': tf.TensorSpec([None,], dtype=tf.string, name='is_male'),
    'mother_age': tf.TensorSpec([None,], dtype=tf.float32, name='mother_age'),
    'plurality': tf.TensorSpec([None,], dtype=tf.string, name='plurality'),
    'gestation_weeks': tf.TensorSpec([None,], dtype=tf.float32,
name='gestation_weeks')
}])
def nokey_prediction(inputs):
    output = model(inputs) # вызвать модель
    return {'babyweight': output}
```

Приведенное выше поведение следует использовать по умолчанию и добавлять `keyed_prediction` в качестве новой функции обслуживания:

```
model.save(EXPORT_PATH,
           signatures={'serving_default': nokey_prediction,
                       'keyed_prediction': keyed_prediction
})
```

Компромиссы и альтернативы

Почему сервер не может просто назначать ключи данным, которые он получает на входе? При онлайновом предсказании серверы могут назначать уникальные идентификаторы запросов, которые не содержат никакой семантической информации. В случае пакетного предсказания проблема заключается в том, что переменные на входе должны быть ассоциированы с результатами на выходе, поэтому сервера, который назначает уникальный идентификатор, недостаточно, т. к. указанный ID не может быть присоединен к переменным на входе. Сервер должен назначать ключи данным, которые он получает на входе до того, как вызывать модель, использовать ключи для упорядочения результатов на выходе, а затем удалять ключи перед отправкой результатов. Проблема заключается в том, что упорядочение вычислений в распределенной обработке данных обходится очень дорого.

Вдобавок, есть несколько других ситуаций, когда поставляемые клиентом ключи полезны: это асинхронное обслуживание и оценивание. Учитывая обе ситуации, предпочтительно, чтобы ключ был уникальным для варианта использования и поддавался идентификации. Поэтому обращение к клиентам с просьбой предоставить ключ упрощает решение.

Асинхронное обслуживание

В наши дни многие промышленные ML-модели являются нейронными сетями, а нейронные сети предусматривают наличие матричных умножений. Матричное умножение на таких аппаратных средствах, как графические процессоры (GPU) и тензорные процессоры (TPU), выполняется эффективнее, если вы можете обеспечивать, чтобы матрицы находились в определенных диапазонах размеров и/или были кратны определенному числу. Поэтому полезно накапливать запросы (разумеется, вплоть до максимальной задержки) и обрабатывать входящие запросы по частям. Поскольку блоки будут состоять из чередующихся запросов от многочисленных клиентов, ключ в этом случае тоже должен обладать каким-то идентификатором клиента.

Непрерывное оценивание

Если вы выполняете непрерывное оценивание, то полезно регистрировать метаданные о предсказательных запросах, чтобы осуществлять мониторинг случаев, когда результативность падает по всем направлениям или только в определенных ситуациях. Такая нарезка случаев значительно облегчается, если ключ идентифицирует ситуацию, о которой идет речь. Например, предположим, что нам нужно применить паттерн "Призма объективности" (Fairness Lens; *см. главу 7*) с целью обеспечения объективной результативности модели для разных клиентских сегментов (например, по возрасту клиента и/или расе клиента). Модель не будет использовать клиентский сегмент на входе, но нам нужно оценивать результативность модели, нарезанной по клиентскому сегменту. В таких случаях наличие встроенного в ключ клиентского сегмента (например, ключ может быть 35-Темнокожий-Мужчина-34324323) облегчает нарезку.

Альтернативное решение состоит в том, чтобы модель игнорировала нераспознанные входные переменные и отправляла обратно не только результаты предсказания, но и все входные переменные, включая нераспознанные.

Это позволяет клиенту соотносить переменные на входе с результатами на выходе, но стоит дороже с точки зрения пропускной способности и вычислений на стороне клиента.

Поскольку высокопроизводительные серверы будут поддерживать многочисленных клиентов, подкрепляться кластером и группировать запросы в пакеты с целью получения преимуществ производительности, то лучше заранее принимать в расчет следующее — просить клиентов предоставлять ключи с каждым предсказанием и указывать ключи, которые не станут причиной коллизии с другими клиентами.

Резюме

В этой главе мы рассмотрели технические приемы операционализации моделей машинного обучения с целью обеспечения их отказоустойчивости и возможности масштабирования, чтобы справляться с производственной нагрузкой. Каждый рассмотренный нами паттерн обеспечения отказоустойчивости связан с шагами модельного развертывания и обслуживания в типичном рабочем потоке процессов ML.

Мы начали эту главу с рассмотрения подходов к инкапсулированию вашей натренированной ML-модели в качестве функции без поддержки состояния с использованием паттерна "Функция обслуживания без поддержки состояния" (Stateless Serving Function). Функция обслуживания устраняет сцепленность среды тренировки со средой развертывания модели, определяя функцию, которая выполняет предсказательный вывод на экспортированной версии модели, и развертывается в виде REST-сервиса. Не все промышленные модели требуют немедленных результатов предсказания, т. к. существуют ситуации, когда необходимо отправить крупный пакет данных в модель для предсказания, но нет надобности получать результаты сразу. Мы ознакомились с тем, как паттерн "Пакетное обслуживание" (Batch Serving) решает эту задачу, используя инфраструктуру распределенной обработки данных, предназначенную для асинхронного выполнения многочисленных запросов на модельное предсказание в качестве фонового задания, при котором результат пишется в указанное место.

Затем на основе паттерна "Непрерывное оценивание модели" (Continued Model Evaluation) мы рассмотрели подход к верификации развернутой модели в той части, которая касается того, что она по-прежнему показывает хорошую результативность на новых данных. Этот паттерн решает проблемы дрейфа данных и дрейфа концепции, регулярно оценивая вашу модель и используя эти результаты для определения необходимости в перетренировке. В паттерне "Двухфазные предсказания" (Two-Phase Predictions) мы нашли техническое решение для специфических вариантов использования, когда существует потребность в развертывании модели на перифе-

рии. Когда можно подразделить задачу на две логические части, этот паттерн сначала создает более простую модель, которую можно развертнуть на периферийном устройстве. Такая периферийная модель соединена с более сложной моделью, размещенной в облаке. Наконец, в паттерне "Предсказания по ключу" (Keyed Predictions) мы обсудили вопрос, почему при исполнении предсказательных запросов полезно предоставлять уникальный ключ для каждого примера. За счет этого ваш клиент ассоциирует каждый результат предсказания на выходе с правильным примером на входе.

В следующей главе мы рассмотрим паттерны обеспечения воспроизводимости. Указанные технические приемы решают проблемы, связанные с врожденной случайностью, присутствующей во многих аспектах машинного обучения, и фокусируются на обеспечении надежных и согласованных результатов при каждом выполнении рабочего потока машинного обучения.

ГЛАВА 6

Паттерны обеспечения воспроизводимости

Лучшие практические приемы разработки программного обеспечения, такие как модульное тестирование, базируются на том, что при выполнении нами фрагмента кода он будет выдавать детерминированный результат:

```
def sigmoid(x):
    return 1.0 / (1 + np.exp(-x))

class TestSigmoid(unittest.TestCase):
    def test_zero(self):
        self.assertAlmostEqual(sigmoid(0), 0.5)

    def test_neginf(self):
        self.assertAlmostEqual(sigmoid(float("-inf")), 0)

    def test_inf(self):
        self.assertAlmostEqual(sigmoid(float("inf")), 1)
```

В машинном обучении такого рода воспроизводимость затруднена. Во время тренировки ML-модели инициализируются случайными значениями, а затем корректируются на основе тренировочных данных. Простой алгоритм *k* средних, который реализован в библиотеке scikit-learn, требует задания переменной `random_state` (случайное состояние) для того, чтобы алгоритм всякий раз возвращал одни и те же результаты:

```
def cluster_kmeans(X):
    from sklearn import cluster
    k_means = cluster.KMeans(n_clusters=10, random_state=10)
    labels = k_means.fit(X).labels_[::]
    return labels
```

Помимо случайного начального числа существует ряд других артефактов, которые нужно урегулировать в целях обеспечения воспроизводимости во время тренировки. Вдобавок машинное обучение состоит из разных этапов, таких как тренировка, развертывание и перетренировка. Нередко бывает важно, чтобы некоторые компоненты оставались воспроизводимыми и на этих этапах.

В данной главе мы рассмотрим паттерны, которые затрагивают разные аспекты обеспечения воспроизводимости. Паттерн "Преобразователь" (Transform) захваты-

вает зависимости процесса подготовки данных из конвейера тренировки модели, чтобы воспроизводить их во время модельного обслуживания запросов. Паттерн "Повторяемая разбивка" (Repeatable Splitting) фиксирует подход, на основе которого данные разбиваются между тренировочным, валидационным и тестовым наборами данных для того, чтобы используемый в тренировке тренировочный пример никогда не использовался для оценивания или тестирования даже по мере роста набора данных. Паттерн "Мостовая схема" (Bridged Schema) обращается к задаче обеспечения воспроизводимости, когда тренировочный набор данных является гибридом, соответствуя разным схемам данных. Паттерн "Конвейер рабочего потока" (Workflow Pipeline) охватывает все этапы рабочего потока машинного обучения, чтобы при повторной тренировке модели части конвейера могли использоваться многоразово. Паттерн "Хранилище признаков" (Feature Store) обращается к вопросу обеспечения воспроизводимости и неоднократного использования признаков в разных заданиях по машинному обучению. Паттерн "Оконный предсказательный вывод" (Windowed Inference) обеспечивает, чтобы признаки, рассчитанные динамическим, зависящим от времени путем, могли повторяться правильно между тренировкой и обслуживанием. Паттерн "Управление версиями" (Versioning) связан с контролем версий данных и моделей и является необходимым условием для манипулирования многими описанными в этой главе паттернами.

ПАТТЕРН 21. Преобразователь

Паттерн "Преобразователь" (Transform) значительно облегчает перенесение ML-модели в промышленную среду, держа входные данные, признаки и преобразования тщательным образом отделенными друг от друга.

Постановка задачи

Проблема заключается в том, что переменные, поступающие *на вход* в ML-модель, не являются *признаками*, которые ML-модель использует в своих вычислениях. Например, в модели классифицирования текста на вход подаются сырые текстовые документы, а признаками являются числовые представления этого текста в виде векторных вложений. Когда мы тренируем ML-модель, то делаем это при помощи признаков, которые извлекаются из сырых данных. Возьмем приведенную ниже модель, которая натренирована предсказывать продолжительность велосипедных прогулок по Лондону, используя BigQuery ML:

```
CREATE OR REPLACE MODEL ch09eu.bicycle_model
OPTIONS(input_label_cols=['duration'],
        model_type='linear_reg')
AS
SELECT
 duration
, start_station_name
, CAST(EXTRACT(dayofweek from start_date) AS STRING)
```

```
 as dayofweek
, CAST(EXTRACT(hour from start_date) AS STRING)
 as hourofday
FROM
 `bigquery-public-data.london_bicycles.cycle_hire`
```

Указанная модель имеет три признака (`start_station_name`, `dayofweek` и `hourofday`), вычисляемых из двух входных переменных — `start_station_name` и `start_date` (рис. 6.1).

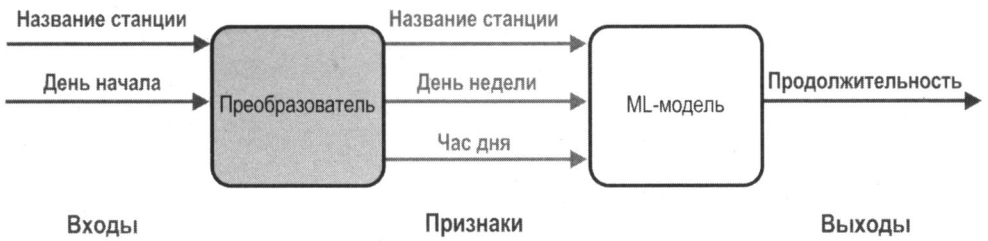


Рис. 6.1. Модель имеет три признака, вычисляемых из двух входных переменных

Приведенный выше SQL-код смешивает входные переменные и признаки и не отслеживает выполненные преобразования. Это вернется бумерангом, когда мы попытаемся с помощью данной модели делать предсказания. Поскольку модель была натренирована на трех признаках, сигнатура предсказания должна выглядеть именно так, как показано ниже:

```
SELECT * FROM ML.PREDICT(MODEL ch09eu.bicycle_model,(
   'Kings Cross' AS start_station_name
 , '3' as dayofweek
 , '18' as hourofday
))
```

Обратите внимание, что во время предсказательного вывода нам приходится знать о том, на каких признаках модель была натренирована, как они должны интерпретироваться, а также детали примененных преобразований. Нам следует знать о том, что "3" нам нужно отправлять для `dayofweek`. Так, а эта тройка... она означает вторник или среду? Все зависит от того, какая библиотека использовалась в модели, или от того, что мы считаем началом недели!

Асимметрия между тренировкой и обслуживанием, вызванная различиями в любом из этих факторов между средами тренировки модели и обслуживания запросов, — одна из ключевых причин, по которым продукционализация ML-моделей является столь сложной.

Решение

Решение состоит в том, чтобы в явной форме захватывать преобразования, применяемые для конвертирования входных модельных данных в признаки. В BigQuery ML это делается с помощью спецификатора инструкции `TRANSFORM`, который обеспечи-

вает, чтобы эти преобразования автоматически применялись во время выполнения инструкции ML.PREDICT.

С учетом поддержки инструкции TRANSFORM приведенную выше модель нужно переписать следующим образом:

```
CREATE OR REPLACE MODEL ch09eu.bicycle_model
OPTIONS(input_label_cols=['duration'],
        model_type='linear_reg')
TRANSFORM(
 SELECT * EXCEPT(start_date)
 , CAST(EXTRACT(dayofweek from start_date) AS STRING)
 as dayofweek -- feature1
 , CAST(EXTRACT(hour from start_date) AS STRING)
 as hourofday -- feature2
)
AS
SELECT
 duration, start_station_name, start_date -- inputs
FROM
 `bigquery-public-data.london_bicycles.cycle_hire`
```

Обратите внимание, как мы четко отделили входные переменные (в инструкции SELECT) от признаков (в инструкции TRANSFORM). Теперь предсказывать стало гораздо проще. Мы можем отправлять в модель просто название станции и метку времени (входные данные):

```
SELECT * FROM ML.PREDICT(MODEL ch09eu.bicycle_model,(
   'Kings Cross' AS start_station_name
  , CURRENT_TIMESTAMP() as start_date
))
```

Далее модель позаботится о проведении соответствующих преобразований, создав необходимые признаки. Она делает это путем захвата как преобразовательной логики, так и артефактов (таких как константы нормализации, коэффициенты векторного вложения, справочные таблицы и т. д.), выполняя преобразования.

Коль скоро в инструкции SELECT мы осмотрительно используем только сырые входные данные и помещаем всю последующую обработку входных данных в инструкцию TRANSFORM, BigQuery ML будет применять эти преобразования автоматически во время предсказания.

Компромиссы и альтернативы

Описанное выше решение работает, потому что BigQuery ML отслеживает преобразовательную логику и артефакты за нас, сохраняет их в модельном графе и автоматически применяет преобразования во время предсказания.

Если мы используем фреймворк, в который поддержка паттерна "Преобразователь" не встроена, то мы должны спланировать нашу архитектуру модели таким образом, чтобы выполняемые во время тренировки преобразования легко воспроизводились во время обработки запросов. Это можно сделать, сохранив преобразования в графе

модели или создав хранилище преобразованных признаков (*см. разд. "Паттерн 26. Хранилище признаков"*).

Преобразования в TensorFlow и Keras

Допустим, что мы тренируем модель ML для оценивания стоимости поездки (`fare`) в нью-йоркском такси и имеем шесть входных переменных (широту места посадки (`pickup_latitude`), долготу места посадки (`pickup_longitude`), широту места высадки (`dropoff_latitude`), долготу места высадки (`drop off_longitude`), количество пассажиров (`passanger_count`) и время посадки (`pickup_datetime`)). TensorFlow поддерживает концепцию признаковых столбцов, которые сохраняются в модельном графе. Однако API разработан исходя из того, что сырые входные данные будут такими же, что и признаки.

Предположим, что мы хотим пронормализовать широты и долготы (подробности *см. в разд. "Простые представления данных" главы 2*), создать преобразованный признак, а именно евклидово расстояние (`euclidean`), и извлечь час дня[1] из метки времени (`hourofday`). Нам придется старательно проработать дизайн модельного графа (рис. 6.2), твердо помня о концепции паттерна "Преобразователь". Просматривая приведенный ниже фрагмент исходного кода, обратите внимание на то, как мы четко расположили три отдельных слоя в модели Keras — слой входов `Input`, слой преобразования `Transform` и слой плотных признаков `DenseFeatures`.

Во-первых, каждая переменная на входе в модель Keras делается слоем `Input` (полный исходный код находится на GitHub[2]):

```
inputs = {
        colname : tf.keras.layers.Input(
                    name=colname, shape=(), dtype='float32')
           for colname in ['pickup_longitude', 'pickup_latitude',
                           'dropoff_longitude', 'dropoff_latitude']
}
```

На рис. 6.2 эти прямоугольники помечены как `dropoff_latitude` (широта высадки), `dropoff_longitude` (долгота высадки) и т. д.

Во-вторых, поддерживается словарь преобразованных признаков и каждое преобразование делается либо слоем `Preprocessing`, либо слоем `Lambda`. Здесь мы нормализуем входные переменные с помощью слоев `Lambda`:

```
transformed = {}
for lon_col in ['pickup_longitude', 'dropoff_longitude']:
            transformed[lon_col] = tf.keras.layers.Lambda(
                lambda x: (x+78)/8.0,
                name='scale_{}'.format(lon_col)
            )(inputs[lon_col])
```

---

[1] Для пояснения: `hourofday(datetime(2015-12-14 18:54)) == 18`.

[2] См. **https://github.com/GoogleCloudPlatform/training-data-analyst/blob/master/quests/serverlessml/06_feateng_keras/solution/taxifare_fc.ipynb**.

```
for lat_col in ['pickup_latitude', 'dropoff_latitude']:
            transformed[lat_col] = tf.keras.layers.Lambda(
                lambda x: (x-37)/8.0,
                name='scale_{}'.format(lat_col)
            )(inputs[lat_col])
```

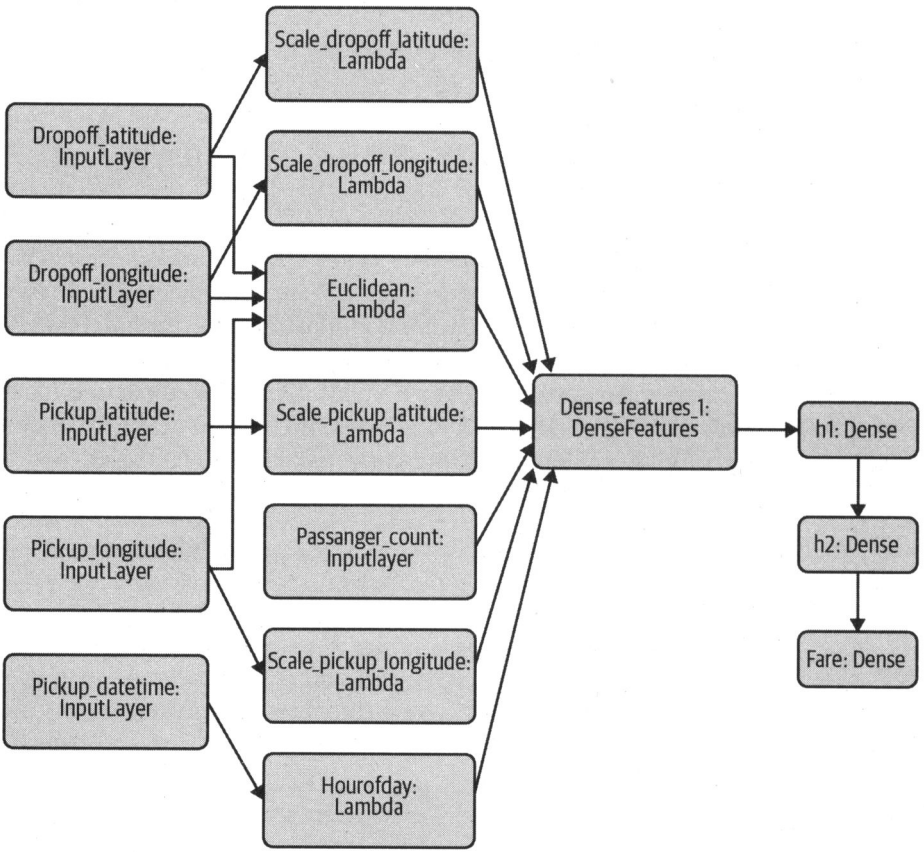


Рис. 6.2. Модельный граф Keras для задачи оценивания стоимости проезда в такси

На рис. 6.2 эти прямоугольники помечены как `scale_dropoff_latitude` (нормализованная широта высадки), `scale_drop off_longitude` (нормализованная долгота высадки) и т. д.

У нас также будет один слой `Lambda` для евклидова расстояния (`euclidean`), который вычисляется из четырех слоев `Input` (рис. 6.2):

```
def euclidean(params):
    lon1, lat1, lon2, lat2 = params
    londiff = lon2 - lon1
    latdiff = lat2 - lat1
    return tf.sqrt(londiff*londiff + latdiff*latdiff)
```

```
transformed['euclidean'] = tf.keras.layers.Lambda(euclidean, name='euclidean')([
            inputs['pickup_longitude'],
            inputs['pickup_latitude'],
            inputs['dropoff_longitude'],
            inputs['dropoff_latitude']
        ])
```

Аналогично столбец для создания часа дня из метки времени (`hourofday`) является слоем `Lambda`:

```
transformed['hourofday'] = tf.keras.layers.Lambda(
            lambda x: tf.strings.to_number(tf.strings.substr(x, 11, 2),
                                           out_type=tf.dtypes.int32),
            name='hourofday'
         )(inputs['pickup_datetime'])
```

В-третьих, все эти преобразованные слои будут объединены в слой `DenseFeatures`:

```
dnn_inputs = tf.keras.layers.DenseFeatures(feature_columns.values())(transformed)
```

Поскольку конструктор для слоя `DenseFeatures` требует набора признаковых столбцов, нам придется указать, каким образом брать все преобразованные значения и преобразовывать их в данные для подачи на вход нейронной сети. Мы могли бы использовать их без модификаций, закодировать их с одним активным состоянием либо решить сгруппировать числа в корзины. Для простоты будем их использовать как есть:

```
feature_columns = {
        colname: tf.feature_column.numeric_column(colname)
           for colname in ['pickup_longitude', 'pickup_latitude',
                           'dropoff_longitude', 'dropoff_latitude']
}
feature_columns['euclidean'] = \
               tf.feature_column.numeric_column('euclidean')
```

Имея входной слой `DenseFeatures`, мы можем построить остальную часть модели Keras, как обычно:

```
h1 = tf.keras.layers.Dense(32, activation='relu', name='h1')(dnn_inputs)
h2 = tf.keras.layers.Dense(8, activation='relu', name='h2')(h1)
output = tf.keras.layers.Dense(1, name='fare')(h2)
model = tf.keras.models.Model(inputs, output)
model.compile(optimizer='adam', loss='mse', metrics=['mse'])
```

Полный пример[3] находится в репозитории на GitHub.

Обратите внимание на то, как мы организовали слои модели Keras: первым слоем был `Inputs`, второй слой — `Transform` и третий слой — `DenseFeatures`, который их объединил. После этой последовательности слоев начинается обычная архитектура

[3] См. **https://github.com/GoogleCloudPlatform/training-data-analyst/blob/master/quests/serverlessml/06_feateng_keras/solution/taxifare_fc.ipynb**.

модели. Поскольку слой `Transform` является частью модельного графа, обычные технические решения на основе паттернов "Функция обслуживания без поддержки состояния" и "Пакетное обслуживание" (*см. главу 5*) будут работать как есть.

Эффективные преобразования с помощью библиотеки tf.transform

Один из недостатков приведенного выше подхода заключается в том, что преобразования будут выполняться во время каждой итерации тренировки. Это не так уж и важно, если мы лишь нормализуем на известные константы. А если наши преобразования окажутся более дорогими с точки зрения вычислений? Или если мы хотим нормализовать, используя среднее значение и дисперсию, и в этом случае нам нужно прокручивать все данные, чтобы вычислять эти переменные? Что делать тогда?

Полезно различать преобразования *уровня экземпляра*, которые могут быть непосредственно частью модели (где единственный недостаток — это их применение на каждой итерации тренировки), и преобразования *уровня набора данных*, где нам нужен полный проход по данным для вычисления совокупной статистической величины или словаря категориальной переменной. Такие преобразования уровня набора данных не могут быть частью модели и должны применяться в качестве масштабируемого шага предобработки, который производит преобразование, захватывающее логику и артефакты (среднее значение, дисперсию, словарь и т. д.), подлежащие прикреплению к модели. Для преобразования уровня набора данных следует использовать библотеку tf.transform.

Библиотека tf.transform (входящая в состав TensorFlow Extended[4]) обеспечивает эффективный подход к выполнению преобразований в ходе предобрабатывающего прохода по данным и сохранению результирующих признаков и артефактов преобразования, чтобы эти преобразования могли применяться системой TensorFlow Serving во время предсказания.

Первый шаг — определение функции преобразования. Например, для того чтобы пронормализовать все входные данные, сведя их среднее к нулю, а дисперсию — к единице, и сгруппировать их в корзины, мы создадим следующую функцию предобработки (полный исходный код[5] см. в репозитории на GitHub):

```
def preprocessing_fn(inputs):
    outputs = {}
    for key in ...:
        outputs[key + '_z'] = tft.scale_to_z_score(inputs[key])
        outputs[key + '_bkt'] = tft.bucketize(inputs[key], 5)
    return outputs
```

Перед тренировкой сырые данные читаются и преобразуются с помощью предварительной функции в Apache Beam:

---

[4] См. **https://oreil.ly/OznI3**.

[5] См. **https://github.com/tensorflow/tfx/blob/master/tfx/examples/chicago_taxi_pipeline/taxi_utils_native_keras.py**.

```
transformed_dataset, transform_fn = (raw_dataset |
    beam_impl.AnalyzeAndTransformDataset(preprocessing_fn))
transformed_data, transformed_metadata = transformed_dataset
```

Преобразованные данные затем записываются в формат, подходящий для чтения тренировочным конвейером:

```
transformed_data | tfrecordio.WriteToTFRecord(
    PATH_TO_TFT_ARTIFACTS,
    coder=example_proto_coder.ExampleProtoCoder(
        transformed_metadata.schema))
```

Конвейер Beam также сохраняет подлежащую выполнению функцию предобработки вместе с любыми артефактами, необходимыми этой функции, в артефакт в формате графа TensorFlow. В приведенном выше случае, например, этот артефакт будет включать среднее значение и дисперсию для нормализации чисел, а также границы корзин для группирования чисел в корзины. Функция тренировки читает преобразованные данные, и, следовательно, не приходится повторять преобразования внутри цикла тренировки.

Функция обслуживания должна загружать эти артефакты и создавать слой `Transform`:

```
tf_transform_output = tft.TFTransformOutput(PATH_TO_TFT_ARTIFACTS)
tf_transform_layer = tf_transform_output.transform_features_layer()
```

Затем функция обслуживания может применять слой `Transform` к разобранным входным признакам и вызывать модель с преобразованными данными для вычисления результата:

```
@tf.function
def serve_tf_examples_fn(serialized_tf_examples):
    feature_spec = tf_transform_output.raw_feature_spec()
    feature_spec.pop(_LABEL_KEY)
    parsed_features = tf.io.parse_example(serialized_tf_examples, feature_spec)
    transformed_features = tf_transform_layer(parsed_features)
    return model(transformed_features)
```

При таком подходе мы обеспечиваем вставку преобразований в граф модели для обработки запросов. В то же время, поскольку тренировка модели происходит на преобразованных данных, циклу тренировки не приходится выполнять эти преобразования в течение каждой эпохи.

Преобразования текста и снимков

В текстовых моделях принято предварительно обрабатывать входной текст (например, удалять знаки препинания, стоп-слова, заглавные буквы, выделять основы слов и т. д.) перед тем, как передавать очищенный текст в модель в качестве признака. Другие распространенные приемы инженерии признаков на текстовых данных включают лексемизацию (токенизацию) и сопоставление с регулярными выражениями. Важно, чтобы те же самые шаги очистки или извлечения выполнялись во время предсказательного вывода.

Необходимость захвата преобразований важна даже в том случае, если нет явного генерирования признаков, как при использовании глубокого обучения со снимками. Снимковые модели обычно имеют входной слой, который принимает снимки конкретного размера. Входные снимковые данные другого размера перед подачей в модель должны быть обрезаны, дополнены или передискретизированы до этого фиксированного размера. Другие распространенные преобразования в снимковых моделях включают манипуляции с цветом (гамма-коррекцию, преобразование оттенков серого и т. д.) и коррекцию ориентации. Важно, чтобы такие преобразования были идентичными для тренировочного набора данных и предсказательного вывода. Паттерн "Преобразователь" позволяет обеспечивать эту воспроизводимость.

В снимковых моделях есть некоторые преобразования (например, аугментация данных путем случайной обрезки и масштабирования), которые применяются только во время тренировки. Эти трансформации не нужно захватывать во время предсказательного вывода. Такие преобразования не будут частью паттерна "Преобразователь".

Альтернативные подходы

Альтернативным подходом к решению проблемы асимметрии между тренировкой и обслуживанием является использование паттерна "Хранилище признаков" (Feature Store). Хранилище признаков содержит скоординированный вычислительный механизм и хранилище данных в виде преобразованных признаков. Вычислительный механизм поддерживает доступ с низкой задержкой для предсказательного вывода и пакетного создания преобразованных признаков, в то время как хранилище данных обеспечивает быстрый доступ к преобразованным признакам для тренировки модели. Хранилище признаков не обязывает к вписыванию операций преобразования в граф модели. И в этом его преимущество. Например, коль скоро хранилище признаков поддерживает Java, операции предобработки можно выполнять на Java, в то время как саму модель можно написать с использованием библиотеки PyTorch языка Python. Недостатком хранилища признаков является то, что оно делает модель зависимой от хранилища признаков и значительно усложняет инфраструктуру обработки запросов.

Еще один подход к отделению языка программирования и фреймворка, предназначенных для преобразования признаков, от языка, применяемого для написания модели, состоит в выполнении предобработки в контейнерах и использовании этих контейнеров в рамках тренировки и обслуживания. Этот подход обсуждается в *разд. "Паттерн 25. Конвейер рабочего потока" далее в этой главе* и внедрен на практике в инструментарии Kubeflow Serving.

ПАТТЕРН 22. Повторяемая разбивка

В целях обеспечения повторимости и воспроизводимости выборок данных необходимо выполнять разбивку имеющихся данных на тренировочный, валидационный и тестовый наборы данных, используя хорошо распределенный столбец и детерми-

нированную хеш-функцию. Паттерн "Повторяемая разбивка" (Repeatable Splitting) предлагает легковесную, повторимую разбивку данных.

Постановка задачи

Многие учебные пособия по машинному обучению предлагают разбивать данные случайно на тренировочный, валидационный и тестовый наборы данных с использованием исходного кода, подобного приведенному ниже:

```
df = pd.DataFrame(...)
rnd = np.random.rand(len(df))
train = df[ rnd < 0.8 ]
valid = df[ rnd >= 0.8 & rnd < 0.9 ]
test  = df[ rnd >= 0.9 ]
```

К сожалению, во многих практических ситуациях этот подход не работает. Причина в том, что строки данных редко бывают независимыми. Например, если мы тренируем модель предсказывать задержки рейсов, то задержки прибытия рейсов в один и тот же день будут сильно коррелировать друг с другом. Это приводит к утечке информации между тренировочным и тестовым наборами данных, когда некоторые рейсы в какой-либо конкретный день находятся в тренировочном наборе данных, а другие рейсы в тот же день находятся в тестовом наборе. Эта утечка из-за коррелированных строк — распространенная проблема, и мы должны ее избегать при выполнении машинного обучения.

Кроме того, функция `rand` упорядочивает данные при каждом ее выполнении поразному, поэтому, если мы выполним программу снова, то получим разные 80% строк. Это может привнести хаос, в случае если мы экспериментируем с разными моделями машинного обучения с целью выбора лучшей из них — нам необходимо сравнивать результативность модели на одном и том же тестовом наборе данных. Для решения этой проблемы нам нужно заранее устанавливать случайное начальное число либо сохранять данные после их разбивки. Идея жесткого кодирования пропорций разбивки данных нехороша тем, что при выполнении таких технических приемов, как метод складного ножа, бутстрапирование, перекрестная валидация и гиперпараметрическая настройка, нам нужно будет изменять этот срез данных и делать это так, чтобы иметь возможность проводить индивидуальные испытания.

Для машинного обучения нам нужно иметь легковесную, повторимую разбивку данных, которая работает независимо от языка программирования или случайных начальных чисел. Мы также хотим обеспечить попадание коррелированных строк в один и тот же срез данных. Например, мы не хотим, чтобы рейсы 2 января 2019 года включались в тестовый набор данных, если рейсы в тот день находятся в тренировочном наборе данных.

Решение

Сначала мы выявляем столбец, который улавливает корреляционную связь между строками. В наборе данных о задержке рейсов это столбец даты (`date`). Затем мы

берем последние несколько цифр хеш-функции на этом столбце, чтобы разбить данные. В задаче о задержке авиарейсов мы можем применить алгоритм хеширования Farm Fingerprint на столбце `date`, чтобы разбить имеющиеся данные на тренировочный, валидационный и тестовый наборы данных.

Дополнительные сведения об алгоритме цифрового отпечатка Farm Fingerprint, поддержке им других каркасов и языков, а также о взаимосвязи между хешированием и криптографией *см. в разд. "Паттерн 1. Хешированный признак" главы 2.* В частности, существуют обертки алгоритма Farm Hash[6] с открытым исходным кодом на нескольких языках (включая Python[7]), и поэтому указанный паттерн может применяться, даже если данные не находятся на складе данных, поддерживающем повторимый хеш прямо из коробки.

Вот как можно разбить набор данных на хеше столбца `date`:

```
SELECT
  airline,
  departure_airport,
  departure_schedule,
  arrival_airport,
  arrival_delay
FROM
  `bigquery-samples`.airline_ontime_data.flights
WHERE
  ABS(MOD(FARM_FINGERPRINT(date), 10)) < 8 -- 80% для ТРЕНИРОВКИ
```

В целях разбивки по столбцу `date` мы вычисляем его хеш с помощью функции `FARM_FINGERPRINT`, а затем используем функцию деления по модулю для отыскания произвольного 80%-ного подмножества строк. Теперь этот срез повторяем — поскольку функция `FARM_FINGERPRINT` возвращает одно и то же значение всякий раз, когда она вызывается на конкретной дате, мы можем быть уверены в том, что всякий раз будем получать одни и те же 80% данных. Как следствие, все рейсы на любую заданную дату будут принадлежать одному и тому же срезу данных — тренировочному, валидационному или тестовому. Этот результат повторим независимо от случайного начального числа.

Если мы хотим разбить данные по аэропорту прибытия `arrival_airport` (чтобы 80% аэропортов находились в тренировочном наборе данных, возможно, потому, что мы пытаемся что-то предсказать об удобствах аэропорта), то будем вычислять хеш не по дате, а по аэропорту прибытия.

Также не представляет труда получить валидационные данные: в приведенном выше запросе надо лишь поменять `< 8` на `=8`, а для тестовых данных заменить это на `=9`. При таком подходе мы получаем 10% образцов в валидационный набор и 10% в тестовый.

---

[6] См. **https://github.com/google/farmhash**.

[7] См. **https://oreil.ly/526Dc**.

Какие соображения следует учитывать при выборе столбца, по которому следует проводить разбивку? Столбец даты `date` должен обладать несколькими характеристиками, чтобы иметь возможность его использовать в качестве столбца разбивки на части.

- Строки с одной и той же датой имеют тенденцию коррелировать. Это ключевая причина, обязывающая нас обеспечивать нахождение всех строк с одной и той же датой в одном и том же срезе.
- Столбец даты `date` не является входной переменной для модели, даже если дата используется в качестве критерия для разбивки. Извлеченные из даты признаки, такие как день недели или час дня, могут подаваться на вход, но мы не можем использовать фактическое входное значение в качестве поля, по которому будет проводиться разбивка, потому что натренированная модель не увидит 20% возможных входных значений для столбца `date`, если 80% данных используется нами для тренировки.
- В столбце `date` должно быть достаточно значений. Поскольку мы вычисляем хеш и находим значение по модулю 10, нам нужно иметь по крайней мере 10 уникальных хеш-значений. Чем больше у нас уникальных значений, тем лучше. Для надежности эмпирическое правило нацеливает на 3–5-кратное превышение знаменателя операции деления по модулю, поэтому в данном случае мы хотим 40 уникальных дат или около того.
- Метка должна быть хорошо распределена между датами. Если окажется, что все задержки произошли 1 января, и не было никаких задержек до конца года, то это не сработает, т. к. разбитые на срезы наборы данных будут искажены. Для надежности следует посмотреть на модельный граф и обеспечить, чтобы все три среза имели одинаковое распределение меток. Для дополнительной надежности следует посмотреть на граф и обеспечить, чтобы распределение метки по отношению к задержке отправления и другим входным переменным было одинаковым во всех трех наборах данных.

Проверка на одинаковость распределения меток в трех наборах данных может быть автоматизирована с помощью теста Коломогорова — Смирнова: надо лишь построить график кумулятивных функций распределения меток в трех наборах данных и найти максимальное расстояние между каждой парой. Чем меньше максимальное расстояние, тем лучше срез.

Компромиссы и альтернативы

Давайте рассмотрим несколько вариантов в отношении того, как мы могли бы осуществлять повторяемую разбивку, и обсудим достоинства и недостатки каждого из них. Обсудим также технические приемы, которые позволяют эту идею расширять и выполнять не просто разбивку, а повторяемый отбор.

Единый запрос

Для генерирования тренировочного, валидационного и тестового срезов нам не нужны три отдельных запроса. Мы можем это сделать в одном-единственном запросе следующим образом:

```
CREATE OR REPLACE TABLE mydataset.mytable AS
SELECT
  airline,
  departure_airport,
  departure_schedule,
  arrival_airport,
  arrival_delay,
  CASE(ABS(MOD(FARM_FINGERPRINT(date), 10)))
    WHEN 9 THEN 'test'
    WHEN 8 THEN 'validation'
    ELSE 'training' END AS split_col
FROM
  `bigquery-samples`.airline_ontime_data.flights
```

Затем мы можем использовать столбец `split_col`, чтобы решать, в какой из трех наборов данных будет попадать отдельно взятая строка данных. Использование единого запроса сокращает вычислительное время, но требует создания новой таблицы или модифицирования исходной таблицы путем внесения дополнительного столбца `split_col`.

Случайная разбивка

Что делать, если строки не коррелированы? В этом случае мы хотим иметь случайный, повторяющийся срез, но не имеем естественного столбца, по которому следует проводить разбивку. Мы можем хешировать всю строку данных, конвертировав ее в значение типа `String` и хешировав это строковое значение:

```
SELECT
  airline,
  departure_airport,
  departure_schedule,
  arrival_airport,
  arrival_delay
FROM
  `bigquery-samples`.airline_ontime_data.flights f
WHERE
  ABS(MOD(FARM_FINGERPRINT(TO_JSON_STRING(f)), 10)) < 8
```

Обратите внимание, что если у нас есть повторяющиеся строки данных, они всегда будут находиться в одном и том же срезе. Возможно, это именно то, чего мы хотим. В противном случае нам придется добавить столбец с уникальными ID в запрос `SELECT`.

Разбивка по нескольким столбцам

Мы говорили об одном столбце, который улавливает корреляцию между строками. Что делать, если корреляция между двумя строками улавливается комбинацией столбцов? В таких случаях следует просто проконкатенировать поля (мы получаем

синтетический признак) перед вычислением хеша. Например, предположим, что мы просто хотим обеспечить, чтобы рейсы из одного и того же аэропорта в один и тот же день не появлялись в разных срезах. В этом случае мы сделаем следующее:

```
SELECT
  airline,
  departure_airport,
  departure_schedule,
  arrival_airport,
  arrival_delay
FROM
  `bigquery-samples`.airline_ontime_data.flights
WHERE
  ABS(MOD(FARM_FINGERPRINT(CONCAT(date, arrival_airport)), 10)) < 8
```

Если мы выполняем разбивку по гибриду из нескольких столбцов, то в качестве одной из переменных на входе в модель на самом деле *можно* использовать аэропорт прибытия (`arrival_airport`), поскольку примеры любого конкретного аэропорта будут присутствовать как в тренировочном, так и в тестовом наборах. С другой стороны, если мы выполним разбивку только по аэропорту прибытия, то тренировочный и тестовый наборы будут иметь взаимоисключающий набор аэропортов прибытия, и, следовательно, `arrival_airport` не может быть входной переменной для модели.

Повторяемая разбивка

Базового решения будет вполне достаточно, если мы хотим, чтобы 80% всего набора данных было тренировочным. Но что делать, если мы хотим оперировать меньшим набором данных, чем тот, который у нас есть в BigQuery? Этот случай характерен для локальной разработки. Набор данных flights состоит из 70 млн строк данных, и, возможно, нам нужен меньший набор данных из миллиона рейсов. Как бы мы выбрали по 1 из каждых 70 рейсов, а затем 80% из них в качестве тренировочных?

Нельзя сделать что-то вроде этого:

```
SELECT
  date,
  airline,
  departure_airport,
  departure_schedule,
  arrival_airport,
  arrival_delay
FROM
  `bigquery-samples`.airline_ontime_data.flights
WHERE
  ABS(MOD(FARM_FINGERPRINT(date), 70)) = 0
  AND ABS(MOD(FARM_FINGERPRINT(date), 10)) < 8
```

Мы не можем выбирать по 1 из каждых 70 строк, а затем выбирать по 8 из каждых 10. Если мы выбираем числа, которые делятся на 70, то они, конечно же, будут делиться и на 10! Эта вторая операция деления по модулю бесполезна.

Вот более подходящее решение:

```
SELECT
  date,
  airline,
  departure_airport,
  departure_schedule,
  arrival_airport,
  arrival_delay
FROM
  `bigquery-samples`.airline_ontime_data.flights
WHERE
  ABS(MOD(FARM_FINGERPRINT(date), 70)) = 0
  AND ABS(MOD(FARM_FINGERPRINT(date), 700)) < 560
```

В этом запросе 700 равно 70×10 и 560 равно 70×8. Первая операция деления по модулю выбирает по 1 из каждых 70 строк, а вторая операция деления по модулю — по 8 из каждых 10 этих строк.

Для валидационных данных вы бы заменили `< 560` диапазоном:

```
ABS(MOD(FARM_FINGERPRINT(date), 70)) = 0
AND ABS(MOD(FARM_FINGERPRINT(date), 700)) BETWEEN 560 AND 629
```

В приведенном выше фрагменте кода наш миллион рейсов приходится только на 1/70-ю часть дней в наборе данных. Это может оказаться в точности тем, что мы и хотим — например, мы можем моделировать полный спектр рейсов в любой конкретный день, экспериментируя с меньшим набором данных. Однако, если мы хотим получить 1/70-ю часть рейсов в любой конкретный день, то нам придется использовать `RAND()` и сохранить результат в виде новой таблицы для обеспечения повторяемости. Из этой меньшей таблицы мы можем отобрать 80% дат с помощью функции `FARM_FINGERPRINT()`. Поскольку эта новая таблица содержит только 1 млн строк и предназначена лишь для экспериментов, дублирование может стать приемлемым.

Последовательная разбивка

В случае моделей, основанных на временны́х рядах, распространенным подходом является использование срезов данных, которые идут один за другим. Например, для тренировки модели прогнозирования спроса на основе данных за последние 45 дней с целью предсказания спроса в течение следующих 14 дней (полный исходный код см. в репозитории по ссылке[8]) мы извлекаем необходимые данные:

[8] См. **https://github.com/GoogleCloudPlatform/bigquery-oreilly-book/blob/master/blogs/bqml_arima/bqml_arima.ipynb**.

```
CREATE OR REPLACE MODEL ch09eu.numrentals_forecast
OPTIONS(model_type='ARIMA',
        time_series_data_col='numrentals',
        time_series_timestamp_col='date') AS
SELECT
  CAST(EXTRACT(date from start_date) AS TIMESTAMP) AS date
  , COUNT(*) AS numrentals
FROM
  `bigquery-public-data`.london_bicycles.cycle_hire
GROUP BY date
HAVING date BETWEEN
DATE_SUB(CURRENT_DATE(), INTERVAL 45 DAY) AND CURRENT_DATE()
```

Такая последовательная разбивка данных также необходима в быстро меняющихся средах, даже если не преследуется цель предсказания будущего значения временно́го ряда. Например, в модели обнаружения мошенничества плохие агенты быстро адаптируются к алгоритму обнаружения мошенничества, и поэтому модель должна постоянно перетренировываться на новейших данных. Недостаточно просто генерировать оценочные данные из случайного среза исторического набора данных, поскольку цель состоит в том, чтобы предсказывать поведение плохих агентов в будущем. Косвенная цель остается той же, что и у модели, основанной на временно́м ряде, в том смысле, что хорошая модель будет способна тренироваться на исторических данных и предсказывать мошенничество в будущем. Данные должны разбиваться последовательно с точки зрения времени, чтобы их можно было правильно оценивать. Например (полный исходный код см. в репозитории[9]):

```
def read_dataset(client, row_restriction, batch_size=2048):
    ...
    bqsession = client.read_session(
        ...
        row_restriction=row_restriction)
    dataset = bqsession.parallel_read_rows()
    return (dataset.prefetch(1).map(features_and_labels)
                   .shuffle(batch_size*10).batch(batch_size))

client = BigQueryClient()
train_df = read_dataset(client, 'Time <= 144803', 2048)
eval_df = read_dataset(client, 'Time > 144803', 2048)
```

Еще один пример, где требуется последовательная разбивка данных, — это высокие корреляции между интервалами времени, которые следуют друг за другом. Например, в метеопрогнозировании погода в последовательные дни сильно коррелирует. Поэтому нецелесообразно помещать 12 октября в тренировочный набор данных, а 13 октября в тестовый набор данных, потому что будет значительная утечка

[9] См. **https://github.com/GoogleCloudPlatform/training-data-analyst/blob/master/blogs/bigquery_datascience/bigquery_tensorflow.ipynb**.

(представьте, например, что 12 октября был ураган). Кроме того, погода сильно подвержена сезонным колебаниям, и поэтому во всех трех срезах необходимо иметь дни из всех времен года. Один из подходов к надлежащему оцениванию результативности прогностической модели состоит в использовании последовательной разбивки, но при учете сезонности, используя первые 20 дней каждого месяца в тренировочном наборе данных, следующие 5 дней в проверочном наборе данных и последние 5 дней в тестовом наборе данных.

Во всех этих случаях повторяемая разбивка требует только того, чтобы мы поместили логику, используемую для создания среза, в систему управления версиями и обеспечили, чтобы версия модели обновлялась всякий раз, когда эта логика меняется.

Стратифицированная разбивка

Приведенный выше пример в отношении того, как погодные закономерности различаются в разные времена года, является примером ситуации, когда разбивка должна происходить *после стратификации* набора данных. Нам необходимо было обеспечить, чтобы в каждом срезе были примеры всех времен года, и поэтому перед тем, как выполнять разбивку, мы стратифицировали набор данных по месяцам. Мы использовали первые 20 дней каждого месяца в тренировочном наборе данных, следующие 5 дней в проверочном наборе данных и последние 5 дней в тестовом наборе данных. Если бы нас не интересовала корреляция между последовательными днями, мы могли бы разбить даты внутри каждого месяца случайным образом.

Чем больше набор данных, тем меньше мы должны беспокоиться о стратификации. На очень крупных наборах данных весьма высока вероятность того, что значения признаков будут хорошо распределены между всеми срезами. Поэтому в крупномасштабном машинном обучении необходимость стратификации возникает довольно часто только в случае асимметричных наборов данных. Например, в наборе данных flights менее 1% рейсов вылетают до 6 часов утра, и поэтому число рейсов, удовлетворяющих этому критерию, может быть довольно малым. Если для нашего варианта использования крайне важно правильно определять поведение этих рейсов, мы должны стратифицировать набор данных, основываясь на часе вылета, и разбивать каждую стратификацию равномерно.

Время отправления было примером асимметричного признака. В задаче несбалансированной классификации (например, при обнаружении мошенничества, когда количество примеров мошенничества довольно мало) у нас может возникнуть потребность стратифицировать набор данных по метке и разбить каждую стратификацию равномерно. Это также важно, если мы имеем мультиметочную задачу и некоторые метки встречаются реже, чем другие. Указанные задачи обсуждаются в *разд. "Паттерн 10. Перебалансировка" главы 3*.

Неструктурированные данные

Хотя в этом разделе мы сосредоточились на структурированных данных, те же принципы применимы и к неструктурированным данным, таким как снимки, видео,

аудио или текст произвольной формы. Для их разбивки просто следует воспользоваться метаданными. Например, если видео, снятые в один и тот же день, коррелированы, следует использовать дату видеозахвата из их метаданных, чтобы разбить видео между независимыми наборами данных. Если текстовые отзывы от одного и того же человека тяготеют к коррелированию, следует использовать отпечаток Farm Fingerprint идентификатора `user_id` рецензента для многократной разбивки отзывов между наборами данных. Если метаданные недоступны или нет корреляции между экземплярами, нужно закодировать снимок или видео с помощью алгоритма Base64 и вычислить цифровой отпечаток кодировки.

Естественным подходом к разбивке текстовых наборов данных может быть использование хеша самого текста. Однако этот подход подобен случайной разбивке и не решает проблему корреляции между отзывами. Например, если человек часто использует слово "потрясающий" в своих негативных отзывах или оценивает все фильмы "Звездных войн" как плохие, то его отзывы коррелируют. Аналогично естественным подходом к разбивке наборов фото- или аудиоданных может быть использование хеша имени файла, но он не решает проблему корреляции между снимками или видео. Следует старательно продумывать вопрос о наилучшем подходе к разбивке набора данных. По нашему опыту, многие проблемы с низкой результативностью ML можно решить путем надлежащей разбивки (и сбора) данных с учетом потенциальных корреляций.

Во время вычисления векторных вложений или предварительной тренировки автокодировщиков мы должны сначала разбивать данные и выполнять эти предварительные вычисления только на тренировочном наборе данных. По этой причине разбивка не должна выполняться на векторных вложениях снимков, видео или текста, если только эти вложения не были созданы на совершенно отдельном наборе данных.

ПАТТЕРН 23. Мостовая схема

Паттерн "Мостовая схема" (Bridged Schema) предоставляет подходы к адаптированию данных, используемых для тренировки модели, путем наведения моста из старой изначальной схемы в новые и более совершенные данные. Польза от этого паттерна обусловлена тем, что, когда поставщик входных данных вносит улучшения в свой поток данных, часто требуется время для сбора достаточного объема данных улучшенной схемы, чтобы можно было адекватно натренировать заменяющую модель. Паттерн "Мостовая схема" позволяет нам использовать как можно больше новых данных, но при этом дополнять их небольшим числом старых данных с целью повышения точности модели.

Постановка задачи

Рассмотрим приложение кассового терминала, которое предсказывает, сколько чаевых нужно дать курьеру. Приложение может использовать ML-модель, которая предсказывает оплату с учетом суммы заказа, времени доставки, расстояния дос-

тавки и т. д. Такая модель будет тренироваться на фактических оплатах, добавленных клиентами.

Предположим, что одной из пременных оплат на входе в модель является тип платежа. В исторических данных это было записано как "наличные" либо "карта". Однако предположим, что платежная система была обновлена и теперь предлагает более подробную информацию о типе используемой карты (подарочная карта, дебетовая карта, кредитная карта). Эта информация чрезвычайно полезна, потому что поведение оплат за услуги варьируется между тремя типами карт.

Во время предсказания всегда будет доступна более новая информация, т. к. мы всегда предсказываем суммы оплат по транзакциям, проведенным после обновления платежной системы. Поскольку новая информация чрезвычайно ценна и уже доступна в промышленной среде для системы предсказания, мы хотели бы ее использовать в модели как можно скорее.

Мы не можем тренировать новую модель исключительно на новых данных, потому что их объем будет довольно малым, ограниченным тем, что есть, т. е. транзакциями после обновления платежной системы. Поскольку качество ML-модели сильно зависит от объема данных, используемых для ее тренировки, вполне вероятно, что модель, натренированная только с новыми данными, будет демонстрировать слабые результаты.

Решение

Решение состоит в том, чтобы навести мост из схемы старых данных, чтобы те соответствовали новым данным. Затем следует тренировать модель, используя столько новых данных, сколько имеется, и дополнить ее старыми данными. При этом возникают два вопроса. Во-первых, как мы будем соотносить тот факт, что в старых данных присутствует только две категории типа платежа, тогда как в новых данных их четыре? Во-вторых, как будет осуществляться дополнение данными для создания тренировочного, валидационного и тестового наборов данных?

Схема с наведенным мостом

Рассмотрим случай, когда старые данные имеют две категории (наличные и карта). В новой схеме категория карт теперь детализированнее (подарочная карта, дебетовая карта, кредитная карта). Мы знаем лишь то, что транзакция, кодированная как "карта" в старых данных, была бы одним из этих типов, но фактически этот тип не был зафиксирован. Навести мост из схемы можно вероятностно либо статически. Мы рекомендуем статический метод, но его легче понять, если мы сначала обсудим вероятностный метод.

Вероятностный метод. Допустим, глядя на новые тренировочные данные, мы оцениваем, что из карточных транзакций 10% — это подарочные карты, 30% — дебетовые карты и 60% — кредитные карты. Всякий раз, когда старый тренировочный пример загружается в программу-тренер, мы могли бы выбирать тип карты, генерируя равномерно распределенное случайное число в диапазоне [0; 100) и выбирая подарочную карту, когда случайное число меньше 10, дебетовую карту, если она

находится в интервале [10; 40), и кредитную карту в противном случае. При условии, что мы тренируем модель в течение достаточного числа эпох, любой тренировочный пример будет представлен как все три категории, но пропорционально фактической частоте встречаемости. Новые тренировочные примеры, конечно же, всегда будут иметь фактически записанную категорию.

Обоснование для вероятностного подхода строится на том, что мы рассматриваем каждый старый пример как происходивший сотни раз. Когда программа-тренер прокручивает данные, в каждую эпоху мы внедряем один из этих экземпляров. В этой симуляции мы ожидаем, что в 10% случаев использования карты транзакция происходила с подарочной картой. Вот почему мы выбираем "подарочную карту" для значения категориальной входной переменной в 10% случаев. Это, конечно же, упрощение — просто потому, что подарочные карты используются в 10% случаев в целом, вовсе не значит, что подарочные карты будут использоваться в 10% случаев для какой-либо конкретной транзакции. Вообще-то может оказаться, что таксомоторные компании запрещают использование подарочных карт в поездках в аэропорт (и обратно), и поэтому для некоторых исторических примеров подарочная карта даже не имеет юридической ценности. Однако в отсутствие какой-либо дополнительной информации будем считать, что распределение частот является одинаковым для всех исторических примеров.

Статический метод. Категориальные переменные обычно кодируются с одним активным состоянием. Если мы будем следовать описанному выше вероятностному подходу и тренировать достаточно долго, то кодированное с одним активным состоянием среднее значение, поданное в тренирующую программу "карты" в старых данных, будет равно [0; 0,1; 0,3; 0,6]. Первое число, 0, соответствует категории наличных. Второе число равно 0,1, потому что в 10% случаев в карточных операциях это число будет равно 1, а во всех остальных случаях — нулю. Подобным же образом у нас будет 0,3 для дебетовых карт и 0,6 для кредитных карт.

В целях наведения моста из старых данных в новую схему мы можем преобразовать старые категориальные данные в это представление, в котором мы вставляем априорную вероятность новых классов, оцененную по тренировочным данным. Новые данные будут иметь [0, 0, 1, 0] для транзакции, о которой известно, что она была оплачена дебетовой картой.

Мы рекомендуем использовать статический метод, а не вероятностный, потому что они практически эквиваленты, если вероятностный метод работает достаточно долго. К тому же он намного проще реализуется, т. к. каждый платеж по карте из старых данных будет иметь точно такое же значение (массив из 4 элементов [0; 0,1; 0,3; 0,6]). Вместо того чтобы писать сценарий для генерирования случайных чисел, как в вероятностном методе, мы можем обновлять старые данные в одной строке кода. Кроме того, статический метод вычислительно намного дешевле.

Аугментированные данные

Для того чтобы максимизировать использование новых данных, следует иметь только два среза данных, которые обсуждаются в *разд. "Паттерн 12. Контрольные*

точки" главы 4. Предположим, что у нас 1 млн примеров в старой схеме, но только 5000 примеров в новой схеме. Как в этом случае создать тренировочный и оценочный наборы данных?

Давайте сначала возьмем оценочный набор данных. Важно понимать, что цель тренировки ML-модели — делать предсказания на ранее не встречавшихся данных. Невстречавшиеся данные в нашем случае будут исключительно теми данными, которые соответствуют новой схеме. Следовательно, нам нужно выделить достаточное число примеров из новых данных, чтобы адекватно оценивать результативность обобщения. Вполне возможно, что для полной уверенности в том, что модель будет показывать хорошую результативность в производстве, нам понадобятся 2000 примеров в оценочном наборе данных. Оценочный набор данных не будет содержать никаких старых примеров, которые были соединены мостом, чтобы соответствовать новой схеме.

Как узнать, сколько нам нужно примеров в оценочном наборе данных: 1000 или 2000? Для получения этого количества следует вычислить метрику оценивания в текущей производственной модели (которая была натренирована на старой схеме) на подмножествах ее оценочного набора данных и определить величину подмножества до того, как метрика оценивания станет устоявшейся.

Вычислить метрику оценивания на разных подмножествах можно следующим образом (как обычно, полный исходный код[10] находится на GitHub в репозитории исходного кода этой книги):

```
for subset_size in range(100, 5000, 100):
    sizes.append(subset_size)
    # Вычислять изменчивость метрики оценивания
    # в этом размере подмножества на протяжении 25 испытаний
    scores = []
    for x in range(1, 25):
        indices = np.random.choice(N_eval,
                                   size=subset_size, replace=False)
        scores.append(
               model.score(df_eval[indices],
                           df_old.loc[N_train+indices, 'tip'])
        )
    score_mean.append(np.mean(scores))
    score_stddev.append(np.std(scores))
```

В приведенном выше фрагменте кода мы пробуем размеры 100, 200, ..., 5000. В каждом размере подмножества мы оцениваем модель 25 раз, всякий раз на другом, случайно отбираемом подмножестве полного оценочного набора. Поскольку мы имеем дело с оценочным набором текущей производственной модели (которую мы смогли натренировать миллионом примеров), оценочный набор данных здесь может содержать сотни тысяч примеров. Затем мы можем вычислить стандартное

[10] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/06_reproducibility/bridging_schema.ipynb**.

отклонение метрики оценивания над 25 подмножествами, повторить это на разных размерах оценивания и построить график этого стандартного отклонения относительно размера оценивания. Результирующий график будет выглядеть примерно так, как показано на рис. 6.3.

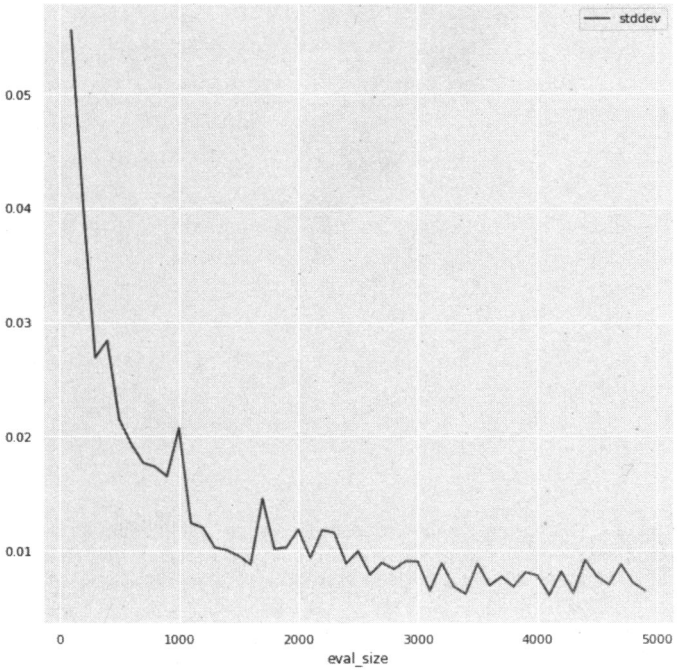


Рис. 6.3. Определить число необходимых оценочных примеров путем оценивания производственной модели на подмножествах разных размеров и отслеживания изменчивости метрики оценивания по размеру подмножества. Здесь стандартное отклонение начинает достигать плато примерно на 2000 примерах

Из рис. 6.3 видно, что число оценочных примеров должно быть не менее 2000, а в идеале — 3000 и более. Для остальной части обсуждения этой темы давайте предположим, что мы решили оценивать на 2500 примерах.

Тренировочный набор будет содержать оставшиеся 2500 новых примеров (объем новых данных, оставшихся после удержания 2500 для оценивания), дополненных некоторым числом старых примеров, которые были соединены мостом, чтобы соответствовать новой схеме. Откуда мы узнаем, сколько старых примеров нам нужно? А мы и не узнаем. Это гиперпараметр, который нам придется настраивать. Например, в задаче с оплатой за услуги (чаевыми), используя поиск о сетке, мы видим из рис. 6.4 (блокнот[11] в репозитории на GitHub содержит полную информацию), что метрика оценивания резко падает до 20 000 примеров, а затем начинает достигать плато.

[11] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/06_reproducibility/bridging_schema.ipynb**.

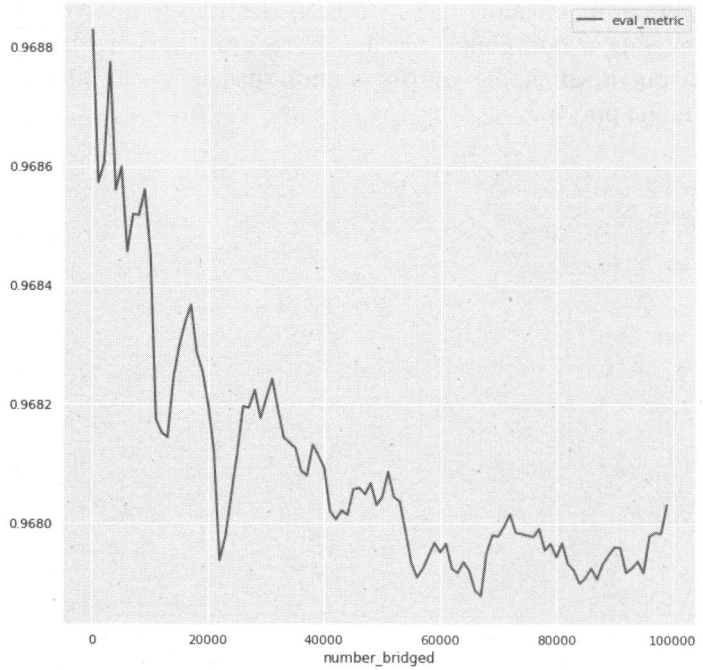


Рис. 6.4. Определить число старых примеров для наведения моста, выполнив гиперпараметрическую настройку. В этом случае очевидно, что после 20 000 мостовых примеров отдача уменьшается

Для достижения наилучших результатов мы должны выбрать наименьшее число старых примеров, которые нам могут сойти с рук — в идеале, со временем, по мере роста числа новых примеров, мы будем полагаться на мостовые примеры все меньше и меньше. И в какой-то момент мы сможем избавиться от старых примеров полностью.

Стоит отметить, что в этой задаче наведение моста действительно приносит пользу, потому что, когда мы не используем мостовые примеры, метрика оценивания оказывается хуже. Если это не так, то метод вменения (т. е. метод выбора статического значения, используемого для наведения моста) должен быть пересмотрен. В следующем разделе мы предлагаем альтернативный метод (каскад).

Чрезвычайно важно проводить сравнение результативности новой модели, натренированной на мостовых примерах, с более старой, неизмененной моделью на оценочном наборе данных. Может оказаться, что новая информация еще не обладает адекватной ценностью.

Поскольку мы будем использовать оценочный набор данных для тестирования на наличие ценности мостовой модели, крайне важно, чтобы оценочный набор данных не использовался во время тренировки или гиперпараметрической настройки. Поэтому следует избегать таких технических приемов, как досрочная остановка или подбор контрольной точки. Вместо этого используется регуляризация, которая помогает держать переподгонку под контролем. Тренировочная потеря должна будет служить метрикой гиперпараметрической настройки. Более подробная информация о том, как сохранять данные, используя только два среза, содержится *в главе 4*, в которой обсуждается паттерн "Контрольные точки".

Компромиссы и альтернативы

Давайте рассмотрим часто предлагаемый подход, который не работает, а также сложную альтернативу наведению моста и расширение для аналогичной задачи.

Объединенная схема

Может возникнуть соблазн принять простое решение — создать объединение старой и новой схем. Например, мы могли бы определить схему для типа платежа как имеющую пять возможных значений: наличные, карта, подарочная карта, дебетовая карта и кредитная карта. Это сделает валидными как исторические данные, так и новые данные, и именно такой подход мы бы использовали в хранилищах данных, чтобы справляться с подобного рода изменениями. При таком подходе и старые, и новые данные валидны и без каких-либо изменений.

Однако обратно совместимый подход с объединением схем не работает в случае машинного обучения.

Во время предсказания мы ни разу не получим значение "карта" для заданного типа оплаты, потому что все поставщики входных данных были обновлены. По сути, все эти тренировочные примеры будут напрасны. Для обеспечения воспроизводимости (именно по этой причине данный паттерн классифицируется как паттерн обеспечения воспроизводимости) мы нуждаемся в наведении моста из старой схемы в новую схему и не можем выполнять объединение двух схем.

Каскадный метод

В статистике вменение (или импутация, от *англ.* imputation) — это совокупность технических приемов, которые используются для замены отсутствующих данных некоторым валидным значением. Распространенный технический прием вменения состоит в замене значения `NULL` на среднее значение этого столбца в тренировочных данных. Почему мы выбираем среднее? Потому что при отсутствии дополнительной информации и исходя из того, что значения распределены нормально, наиболее вероятным значением является среднее.

Обсуждавшийся в главном решении статический метод назначения априорных частот также является методом вменения. Мы исходим из того, что категориальная переменная распределена в соответствии с частотной диаграммой (которую мы оцениваем по тренировочным данным), и вменяем среднее значение, кодированное с одним активным состоянием (в соответствии с этим частотным распределением) "отсутствующей" категориальной переменной.

Знаем ли мы какой-либо другой подход к оцениванию неизвестных значений при наличии нескольких примеров? Ну, разумеется! Машинное обучение. Мы можем натренировать каскад моделей (*см. разд. "Паттерн 8. Каскад" главы 3*). Первая модель использует любые новые примеры, на которых мы должны натренировать модель машинного обучения, чтобы предсказывать тип карты. Если изначальная модель предсказания оплаты за услуги tips имела пять входных переменных, то эта модель будет иметь четыре переменные. Пятая переменная (тип оплаты) будет мет-

кой для этой модели. Тогда результат на выходе из первой модели будет использоваться для тренировки второй модели.

На практике паттерн "Каскад" добавляет слишком много сложности сременному обходному пути до тех пор, пока у вас не будет достаточно новых данных. Статический метод фактически является самой простой ML-моделью — это модель, которую мы получили бы, если бы у нас были неинформативные входные данные. Мы рекомендуем статический подход, а паттерн "Каскад" применять только в том случае, если статический метод окажется недостаточно хорошим.

Манипулирование новыми признаками

Еще одна ситуация, в которой может потребоваться наведение моста, — это случай, когда поставщик входных данных вносит во входной канал дополнительную информацию. Например, в нашем примере со стоимостью проезда в такси мы можем начать получать данные о том, включены ли дворники либо движется автомобиль или стоит. Из этих данных мы можем составить признак о том, шел ли дождь в момент начала поездки в такси, какую долю времени поездки такси простаивает и т. д.

Если у нас есть новые входные признаки, которые мы хотим начать использовать немедленно, нам следует навести мост из старых данных (в которых этот новый признак будет отсутствовать), вменяя им значение для нового признака. Рекомендуемые варианты для вменяемого значения таковы:

- среднее значение признака, если признак является числовым и нормально распределенным;
- медианное значение признака, если признак является числовым и асимметричным или имеет много выбросов;
- медианное значение признака, если признак является категориальным и сортируемым;
- мода признака, если признак является категориальным и несортируемым;
- частота истинного признака, если он является булевым

Если признак говорит о том, шел ли дождь, то он является булевым, и поэтому вмененное значение будет примерно равно 0,02, если в тренировочном наборе данных идет дождь в 2% случаев. Если признак представляет собой долю минут простоя, то мы могли бы использовать медианное значение. Подход на основе паттерна "Каскад" остается жизнеспособным для всех этих случаев, но статическое вменение проще, и его часто достаточно.

Манипулирование увеличениями прецизионности

Когда поставщик входных данных увеличивает гранулярность своего потока данных, следует принять подход с наведением моста, который позволит создавать тренировочный набор данных, состоящий из данных более высокой разрешающей способности, дополненной некоторым количеством старых данных.

В случае значений с плавающей точкой наводить мост из старых данных в явной форме, чтобы те соответствовали прецизионности новых данных, нет никакой необходимости. В целях ознакомления с причиной рассмотрим случай, когда некоторые данные изначально были представлены одним десятичным знаком (например, 3,5 или 4,2), но теперь представляются с двумя десятичными знаками (например, 3,48 или 4,23). Если мы допустим, что 3,5 в старых данных состоит из значений, которые в новых данных были бы равномерно распределены[12] в промежутке [3,45; 3,55], то статически вмененное значение будет равно 3,5, что является в точности тем значением, которое хранится в старых данных.

В случае категориальных значений — например, если старые данные хранили местоположение в виде кода штата или провинции, а новые данные предоставляли код округа или района — следует использовать распределение частот округов внутри штатов, как описано в главном решении, для выполнения статического вменения.

ПАТТЕРН 24. Оконный предсказательный вывод

Паттерн "Оконный предсказательный вывод" (Windowed Inference) манипулирует моделями, которые для выполнения предсказательного вывода требуют непрерывно обновляемой последовательности экземпляров. Указанный паттерн работает путем экстернализации состояния модели и вызова модели из конвейера потоковой аналитики. Этот паттерн также полезен, когда модель машинного обучения требует признаков, которые должны вычисляться из агрегатов над временны́ми окнами. Экстернализируя состояние в потоковый конвейер, паттерн "Оконный предсказательный вывод" обеспечивает правильное повторение признаков, вычисленных динамическим, зависящим от времени путем, между тренировкой и обслуживанием. Он обеспечивает способ предотвращения асимметрии между тренировкой и обслуживанием в случае вре́менных агрегатных признаков.

Постановка задачи

Давайте взглянем на задержки прибытия рейсов в аэропорт Даллас Форт-Уэрт (Dallas Fort Worth, DFW), изображенные для пары дней в мае 2010 года на рис. 6.5 (полный код[13] находится на GitHub).

---

[12] Обратите внимание, что совокупная функция распределения вероятностей не обязательно должна быть равномерной — нам лишь нужно, чтобы изначальные корзины были узкими настолько, чтобы иметь возможность аппроксимировать функцию распределения вероятностей лестничной функцией. Это допущение не работает тогда, когда мы имеем сильно асимметричным распределением, которое было неадекватно отобрано в старых данных. В таких случаях возможно, что 3,46 будет более вероятным, чем 3,54, и это должно быть отражено в мостовом наборе данных.

[13] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/06_reproducibility/stateful_stream.ipynb**.

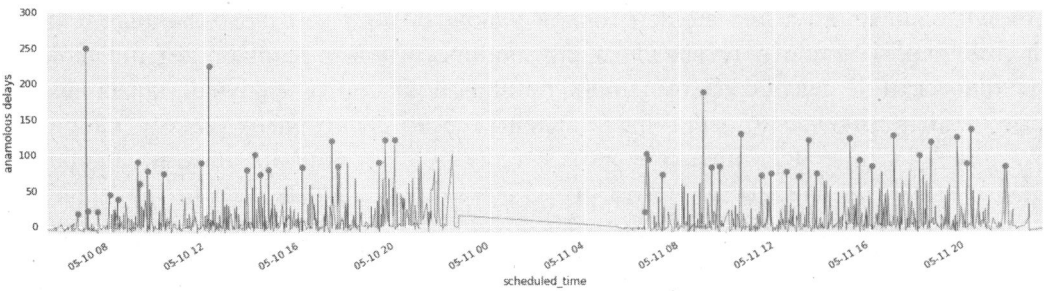


Рис. 6.5. Задержки прибытия рейсов в аэропорт Даллас Форт-Уэрт 10–11 мая 2010 года. Аномальные задержки прибытия отмечены точками

Задержки прибытия рейсов демонстрируют значительную изменчивость, но все же можно отметить необычно большие задержки прибытия (отмеченные точками). Обратите внимание, что значение слова "необычный" варьируется в зависимости от контекста. Ранним утром (левая область участка) большинство рейсов выполняются вовремя, поэтому даже малый всплеск является аномальным. К середине дня (после 12 часов дня 10 мая) изменчивость возрастает, и 25-минутные задержки довольно распространены, но 75-минутная задержка все-таки является необычной.

Является ли отдельно взятая задержка аномальной, зависит от временно́го контекста, например от задержек прибытия, наблюдаемых в течение последних 2 часов. Для определения аномальности задержки нам необходимо сначала отсортировать кадр данных, основываясь на времени (как показано на графике рис. 6.5 и продемонстрировано ниже в исходном коде pandas):

```
df = df.sort_values(by='scheduled_time').set_index('scheduled_time')
```

Затем нам нужно применить функцию обнаружения аномалии к скользящим двухчасовым окнам:

```
df['delay'].rolling('2h').apply(is_anomaly, raw=False)
```

Функция обнаружения аномалии `is_anomaly` может быть довольно изощренной. Но давайте возьмем простой случай отбрасывания экстремумов и будем называть значение данных аномалией, если оно превышает четыре стандартных отклонения от среднего значения в двухчасовом окне:

```
def is_anomaly(d):
    outcome = d[-1]    # последний элемент

    # Отбросить минимальное & максимальное значение & текущий (последний) элемент
    xarr = d.drop(index=[d.idxmin(), d.idxmax(), d.index[-1]])
    prediction = xarr.mean()
    acceptable_deviation = 4 * xarr.std()
    return np.abs(outcome - prediction) > acceptable_deviation
```

Это работает на исторических (тренировочных) данных, потому что весь кадр данных находится под рукой. Разумеется, при выполнении предсказательного вывода на нашей модели у нас не будет всего кадра данных целиком. В промышленной

среде мы будем получать информацию о прибытии рейса один за другим, по мере их прибытия. Поэтому в метке времени у нас будет лишь одно значение задержки:

```
2010-02-03 08:45:00,19.0
```

Видно, что приведенный выше рейс (в 08:45 3 февраля) опаздывает на 19 минут, но является ли этот факт необычным или нет? Как правило, чтобы выполнить предсказательный вывод о рейсе, нам нужны только признаки этого рейса. Однако в данном случае модель требует информации обо всех рейсах в аэропорт DFW между 06:45 и 08:45:

```
2010-02-03 06:45:00,?
2010-02-03 06:?:00,?
...
2010-02-03 08:45:00,19.0
```

Выполнять предсказательный вывод по одному рейсу за раз не получится. Нам нужно каким-то образом предоставлять модели информацию обо всех предыдущих рейсах.

Как выполнять предсказательный вывод, если модель требует не одного экземпляра, а последовательности экземпляров?

Решение

Решение состоит в том, чтобы осуществлять потоковую передачу с поддержкой состояния, т. е. потоковую передачу, в которой отслеживается состояние модели во времени.

- К данным о прибытии рейса применяется скользящее окно. Его необходимо будет рассчитывать над двухчасовым промежутком, но оно может закрываться чаще, например каждые 10 минут. В этом случае агрегированные значения будут рассчитываться каждые 10 минут в течение предыдущих 2 часов.
- Внутреннее состояние модели (это может быть список рейсов) обновляется рейсовой информацией каждый раз, когда прибывает новый рейс, таким образом, собирая двухчасовую историческую запись рейсовых данных.
- Всякий раз, когда окно закрывается (в нашем примере каждые 10 минут), построенная на временно́м ряде ML-модель тренируется на двухчасовом списке рейсов. В дальнейшем указанная модель используется для предсказания будущих задержек рейсов и границ уверенности в таких предсказаниях.
- Параметры построенной на временно́м ряде модели преобразуются во внешнюю переменную состояния. Мы могли бы применить такую модель, как авторегрессионное интегрированное скользящее среднее (autoregressive integrated moving average, ARIMA) или долгая краткосрочная память (long short-term memory, LSTM), и в этом случае параметры модели были бы коэффициентами модели ARIMA или весами модели LSTM. Для того чтобы не усложнять исходный код, мы будем использовать нуль-порядковую регрессионную модель[14], и поэтому

[14] Другими словами, мы вычисляем среднее.

нашими модельными параметрами будут средняя задержка рейса и дисперсия задержек рейса в двухчасовом окне.

- Когда рейс прибывает, его задержку прибытия можно классифицировать как аномальную или не использующую экстернализированное состояние модели — нет необходимости иметь полный список рейсов за последние 2 часа.

Для потоковой передачи конвейеров можно задействовать набор инструментальных средств Apache Beam, потому что тогда один и тот же исходный код будет работать и на исторических, и на вновь поступающих данных. В Apache Beam скользящее окно настраивается следующим образом (полный исходный код[15] находится в репозитории на GitHub):

```
windowed = (data
        | 'window' >> beam.WindowInto(
                beam.window.SlidingWindows(2 * 60 * 60, 10*60))
```

Модель обновляется путем совмещения всех рейсовых данных, собранных за последние 2 часа, и передачи их в функцию `ModelFn`:

```
model_state = (windowed
        | 'model' >> beam.transforms.CombineGlobally(ModelFn()))
```

Функция `ModelFn` обновляет внутреннее состояние модели рейсовой информацией. Здесь внутреннее состояние модели будет содержать кадр данных pandas, который обновляется вместе с рейсами в окне:

```
class ModelFn(beam.CombineFn):
    def create_accumulator(self):
        return pd.DataFrame()

    def add_input(self, df, window):
        return df.append(window, ignore_index=True)
```

Всякий раз, когда окно закрывается, извлекается результат, который в нашем случае (мы называем его экстернализованным состоянием модели) состоит из модельных параметров:

```
def extract_output(self, df):
    if len(df) < 1:
        return {}
    orig = df['delay'].values
    xarr = np.delete(orig, [np.argmin(orig), np.argmax(orig)])
    return {
        'prediction': np.mean(xarr),
        'acceptable_deviation': 4 * np.std(xarr)
    }
```

[15] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/06_reproducibility/find_anomalies_model.py**.

Экстернализованное состояние модели обновляется каждые 10 минут на основе двухчасового скользящего окна (табл. 6.1).

Таблица 6.1. *Фрагмент обновляемых данных*

Время закрытия окна	prediction	acceptable_deviation
2010-05-10T06:35:00	-2.8421052631578947	10.48412597725367
2010-05-10T06:45:00	-2.6818181818181817	12.083729926046008
2010-05-10T06:55:00	-2.9615384615384617	11.765962341537781

Показанный выше исходный код извлечения параметров модели аналогичен исходному коду для случая с использованием pandas, но он выполняется в конвейере Beam. За счет этого обеспечивается возможность работы исходного кода в потоковом режиме, но состояние модели доступно только в контексте скользящего окна. Для выполненния предсказательного вывода по каждому прибывающему рейсу нам нужно экстернализовать состояние модели (аналогично тому, как мы экспортируем модельные веса в файл в паттерне "Функции обслуживания без поддержки состояния", чтобы отцепить его от контекста программы тренировки, в которой эти веса вычисляются):

```
model_external = beam.pvalue.AsSingleton(model_state)
```

Указанное экстернализованное состояние можно использовать для обнаружения аномальности данного рейса:

```
def is_anomaly(flight, model_external_state):
    result = flight.copy()
    error = flight['delay'] - model_external_state['prediction']
    tolerance = model_external_state['acceptable_deviation']
    result['is_anomaly'] = np.abs(error) > tolerance
    return result
```

Затем функция `is_anomaly` применяется к каждому элементу в последней секции скользящего окна:

```
anomalies = (windowed
        | 'latest_slice' >> beam.FlatMap(is_latest_slice)
        | 'find_anomaly' >> beam.Map(is_anomaly, model_external))
```

Компромиссы и альтернативы

Предложенное выше техническое решение является вычислительно эффективным в случае потоков данных с высокой пропускной способностью, но может быть улучшено, если параметры ML-модели способны обновляться в онлайновом режиме. Этот паттерн также применим к ML-моделям с поддержкой состояния, таким как рекуррентные нейронные сети, и когда модель без поддержки состояния требует входных признаков с поддержкой состояния.

Сокращение вычислительных затрат

В *разд. "Постановка задачи"* мы использовали следующий код библиотеки pandas:

```
dfw['delay'].rolling('2h').apply(is_anomaly, raw=False);
```

тогда как в *разд. "Решение"* исходный код библиотеки Beam был следующим:

```
windowed = (data
        | 'window' >> beam.WindowInto(
                beam.window.SlidingWindows(2 * 60 * 60, 10*60))
model_state = (windowed
        | 'model' >> beam.transforms.CombineGlobally(ModelFn()))
```

Между скользящим окном в pandas и скользящим окном в Apache Beam существуют значимые различия в том, как часто вызывается функция `is_anomaly` и как часто необходимо вычислять модельные параметры (среднее значение и стандартное отклонение). Обсудим это.

Поэлементно либо за интервал времени. В коде pandas функция `is_anomaly` вызывается на каждом экземпляре набора данных. Исходный код обнаружения аномалии вычисляет параметры модели и немедленно применяет их к последнему элементу в окне. В конвейере Beam состояние модели тоже создается на каждом скользящем окне, но скользящее окно в этом случае основано на времени. Поэтому параметры модели вычисляются только один раз в 10 минут.

Само обнаружение аномалии осуществляется на каждом экземпляре:

```
anomalies = (windowed
        | 'latest_slice' >> beam.FlatMap(is_latest_slice)
        | 'find_anomaly' >> beam.Map(is_anomaly, model_external))
```

Обратите внимание, что за счет этого вычислительно дорогостоящая тренировка старательно отделяется от вычислительно дешевого предсказательного вывода. Вычислительно дорогостоящая часть выполняется только один раз в 10 минут, позволяя классифицировать каждый экземпляр как аномалию или норму.

Высокая пропускная способность потоков данных. Объемы данных продолжают расти, и бóльшая часть этого процесса происходит за счет реально-временны́х данных. Следовательно, этот паттерн нужно применять к потокам данных высокой пропускной способности — потокам, в которых число элементов может превышать тысячи элементов в секунду. Подумайте, например, о потоках щелчков мышью с веб-сайтов или потоках машинной активности от компьютеров, носимых устройств или автомобилей.

Предлагаемое техническое решение с использованием потокового конвейера выгодно тем, что позволяет избегать перетренировки модели на каждом экземпляре, т. е. того, что делает исходный код pandas в *разд. "Постановка задачи"*. Однако оно возвращает эти преимущества, создавая в памяти кадр данных всех полученных записей. Если мы получаем 5000 элементов в секунду, то размещенный в памяти кадр данных за 10 минут будет содержать 3 млн записей. Поскольку в любой мо-

мент времени необходимо поддерживать 12 скользящих окон (10-минутные окна, каждое за 2 часа), требования к памяти могут стать значительными.

Хранение всех полученных записей (для вычисления модельных параметров в конце окна) может вылиться в проблему. Когда поток данных имеет высокую пропускную способность, важной становится возможность обновлять параметры модели с каждым элементом. Это можно сделать путем изменения функции `ModelFn` следующим образом (полный исходный код[16] находится на GitHub):

```
class OnlineModelFn(beam.CombineFn):
    ...
    def add_input(self, inmem_state, input_dict):
        (sum, sumsq, count) = inmem_state
        input = input_dict['delay']
        return (sum + input, sumsq + input*input, count + 1)

    def extract_output(self, inmem_state):
        (sum, sumsq, count) = inmem_state
        ...
            mean = sum / count
            variance = (sumsq / count) - mean*mean
            stddev = np.sqrt(variance) if variance > 0 else 0
            return {
                'prediction': mean,
                'acceptable_deviation': 4 * stddev
            }
        ...
```

Ключевое отличие заключается в том, что в памяти хранится не весь кадр данных полученных экземпляров, а только три числа с плавающей точкой (`sum`, `sum2`, `count`), необходимые для извлечения выходного состояния модели. Обновление параметров модели по одному экземпляру за раз называется *онлайновым обновлением* и может выполняться лишь в том случае, если тренировка модели не требует итерации по всему набору данных. Поэтому в приведенной выше реализации дисперсия вычисляется путем поддержания суммы x^2, вследствие чего после вычисления среднего второй обход данных не требуется.

Потоковый SQL

Если наша инфраструктура состоит из высокопроизводительной базы данных SQL, способной обрабатывать потоковые данные, то можно реализовать паттерн "Оконный предсказательный вывод" альтернативным способом, используя окно агрегации (полный исходный код[17] находится на GitHub).

---

[16] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/06_reproducibility/find_anomalies_model.py**.

[17] См . **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/06_reproducibility/find_anomalies_model.py**.

Мы вытаскиваем рейсовые данные из BigQuery:

```
WITH data AS (
  SELECT
    PARSE_DATETIME('%Y-%m-%d-%H%M',
                   CONCAT(CAST(date AS STRING),
                   '-', FORMAT('%04d', arrival_schedule))
                   ) AS scheduled_arrival_time,
      arrival_delay
  FROM `bigquery-samples.airline_ontime_data.flights`
  WHERE arrival_airport = 'DFW' AND SUBSTR(date, 0, 7) = '2010-05'
),
```

Затем создаем состояние модели `model_state`, вычисляя параметры модели в течение временно́го окна, заданного как предшествующие 2 часа и предшествующая 1 секунда (`RANGE BETWEEN 7200 PRECEDING AND 1 PRECEDING`):

```
model_state AS (
  SELECT
    scheduled_arrival_time,
    arrival_delay,
    AVG(arrival_delay) OVER (time_window) AS prediction,
    4*STDDEV(arrival_delay) OVER (time_window) AS acceptable_deviation
  FROM data
  WINDOW time_window AS
    (ORDER BY UNIX_SECONDS(TIMESTAMP(scheduled_arrival_time))
    RANGE BETWEEN 7200 PRECEDING AND 1 PRECEDING)
)
```

Наконец, применяем алгоритм обнаружения аномалий к каждому экземпляру:

```
SELECT
  *,
  (ABS(arrival_delay - prediction) > acceptable_deviation) AS is_anomaly
FROM model_state
```

Результат выглядит так, как представлено в табл. 6.2, в которой 54-минутная задержка прибытия отмечена как аномалия, учитывая, что все предыдущие рейсы прибыли рано.

***Таблица 6.2.** Результат запроса BigQuery, определяющего аномальность входящих рейсовых данных*

scheduled_arrival_time	arrival_delay	prediction	acceptable_deviation	is_anomaly
2010-05-01T05:45:00	-18.0	-8.25	62.51399843235114	false
2010-05-01T06:00:00	-13.0	-10.2	56.878818553131005	false
2010-05-01T06:35:00	-1.0	-10.666	51.0790237442599	false
2010-05-01T06:45:00	-9.0	-9.28576	48.86521793473886	false
2010-05-01T07:00:00	54.0	-9.25	45.24220532707422	true

В отличие от решения на основе Apache Beam, эффективность распределенного SQL позволит нам рассчитывать двухчасовое временно́е окно, центрируемое на каждом экземпляре (а не с разрешающей способностью 10-минутных окон). Однако недостатком является тяготение BigQuery к относительно высокой задержке (порядка секунд), и поэтому его нельзя использовать для приложений управления в режиме реального времени.

Модели на основе последовательностей

Паттерн "Оконный предсказательный вывод" передачи скользящего окна, состоящего из предыдущих экземпляров, в функцию предсказательного вывода полезен за пределами обнаружения аномалий или даже моделей на основе временны́х рядов. В частности, он полезен в любом классе моделей, таких как модели на основе последовательностей, т. е. моделей, которые требуют исторического состояния. Например, прежде чем модель машинного перевода сможет выполнить перевод, она должна увидеть ряд из нескольких последовательных слов для учета контекста слова в переводе. Ведь перевод слов left, Chicago и road варьируется среди предложений "I left Chicago by road" (я уехал из Чикаго по дороге) и "Turn left on Chicago Road" (поверните налево на Чикагской дороге).

По соображениям производительности модель машинного перевода будет настроена как не поддерживающая состояние и требовать от пользователя предоставления контекста. Например, если модель не поддерживает состояние, то экземпляры модели могут автоматически масштабироваться в ответ на увеличение трафика и вызываться параллельно для получения более быстрых переводов. Таким образом, перевод знаменитого монолога из "Гамлета" Шекспира на немецкий язык мог бы следовать этим шагам, выхватывая из середины там, где выделенное жирным шрифтом слово подлежит переводу (табл. 6.3).

Таблица 6.3. *Перевод отдельных слов*

Вход (9 слов, по 4 по обе стороны)	**Выход**
The undiscovered country, from **whose** bourn No traveller returns	dessen
undiscovered country, from whose **bourn** No traveller returns, puzzles	Bourn
country, from whose bourn **No** traveller returns, puzzles the	Kein
from whose bourn No **traveller** returns, puzzles the will,	Reisender

Следовательно, клиенту потребуется потоковый конвейер. Конвейер мог бы принимать входной английский текст, лексемизировать его, посылать по 9 лексем за раз, собирать результаты на выходе и конкатенировать их в немецкие предложения и абзацы.

Большинство моделей на основе последовательностей, таких как рекуррентные нейронные сети и LSTM, требуют потоковых конвейеров в целях обеспечения высокопроизводительного предсказательного вывода.

Признаки с поддержкой состояния

Паттерн "Оконный предсказательный вывод" может быть полезен, если подаваемый в модель признак требует состояния, даже если сама модель не поддерживает состояние. Например, предположим, что мы тренируем модель предсказывать задержки прибытия, и одной из входных модельных переменных является задержка вылета. У нас вполне может возникнуть потребность включить среднюю задержку вылета рейсов из этого аэропорта за последние 2 часа в качестве входной переменной модели.

Во время тренировки мы можем создать набор данных с помощью оконной функции SQL:

```
WITH data AS (
  SELECT
    SAFE.PARSE_DATETIME('%Y-%m-%d-%H%M',
                        CONCAT(CAST(date AS STRING), '-',
                        FORMAT('%04d', departure_schedule))
                        ) AS scheduled_depart_time,
    arrival_delay,
    departure_delay,
    departure_airport
  FROM `bigquery-samples.airline_ontime_data.flights`
  WHERE arrival_airport = 'DFW'
),

  SELECT
    * EXCEPT(scheduled_depart_time),
    EXTRACT(hour from scheduled_depart_time) AS hour_of_day,
    AVG(departure_delay) OVER (depart_time_window) AS avg_depart_delay
  FROM data
  WINDOW depart_time_window AS
    (PARTITION BY departure_airport ORDER BY
     UNIX_SECONDS(TIMESTAMP(scheduled_depart_time))
     RANGE BETWEEN 7200 PRECEDING AND 1 PRECEDING)
```

Тренировочный набор данных теперь включает среднюю задержку в качестве еще одного признака (табл. 6.4).

Таблица 6.4. *Тренировочный набор данных*

Строка	arrival_delay	departure_delay	departure_airport	hour_of_day	avg_depart_delay
1	-3.0	-7.0	LFT	8	-4.0
2	56.0	50.0	LFT	8	41.0
3	-14.0	-9.0	LFT	8	5.0
4	-3.0	0.0	LFT	8	-2.0

Правда, во время предсказательного вывода нам понадобится потоковый конвейер, который будет вычислять эту среднюю задержку вылета, чтобы мы смогли переда-

вать ее в модель. В целях ограничения асимметрии между тренировкой и обслуживанием предпочтительнее использовать тот же самый SQL в функции окна-перевертыша[18] в потоковом конвейере, а не пытаться переводить SQL на язык Scala, Python или Java.

Упаковывание предсказательных запросов

Еще один сценарий, в котором мы могли бы использовать паттерн "Оконный предсказательный вывод", даже если модель не поддерживает состояния, — это случай, когда модель развернута в облаке, но клиентская часть встроена в периферийное устройство или размещается локально. В таких случаях сетевая задержка отправки предсказательных запросов друг за другом в развернутую в облаке модель может привести к переполнению. В этой ситуации можно применить паттерн "Двухфазные предсказания" из *главы 5*, где в первой фазе используется конвейер по сбору серии запросов, а во второй фазе они отправляются в службу одним пакетом.

Этот вариант подходит только для терпимых к задержке случаев использования. Если мы собираем входные экземпляры в течение 5 минут, клиент должен терпимо относиться к пятиминутной задержке в получении предсказаний.

ПАТТЕРН 25. Конвейер рабочего потока

В паттерне "Конвейер рабочего потока" (Workflow Pipeline) мы решаем задачу создания сквозного воспроизводимого конвейера путем контейнеризации и оркестрирования шагов в нашем рабочем потоке процессов машинного обучения. Контейнеризация может выполняться в явной форме либо с использованием каркаса, который упрощает выполнение этой работы.

Постановка задачи

Исследователь данных может выполнять шаги предобработки данных, тренировки и развертывания модели индивидуально от начала до конца (рис. 6.6) в рамках одного скрипта или блокнота. Однако по мере того, как каждый шаг в рабочем потоке процессов ML становится все сложнее, и все больше людей в организации хотят вносить свой вклад в эту кодовую базу, выполнение этих шагов из общего блокнота перестанет масштабироваться.

Рис. 6.6. Шаги типичного сквозного рабочего потока процессов ML. Показанное не означает, что поток является всеобъемлющим, однако он охватывает наиболее распространенные шаги в ходе разработки модели ML

[18] Оконо-перевертыш (tumbling window), или переворачивающееся окно, — это фиксированный, неперекрывающийся интервал, смежный с предыдущим и последующим интервалами. — *Прим. перев.*

В традиционном программировании в монолитных приложениях вся логика обрабатывается одной-единственной программой. В целях тестирования небольшого признака в монолитном приложении мы должны выполнять всю программу целиком. То же самое относится и к развертыванию или отладке монолитных приложений. Развертывание исправления небольшого дефекта в одной части программы требует развертывания всего приложения, что быстро может превратиться в громоздкую работу. Когда вся кодовая база целиком неразрывна, у индивидуальных разработчиков возникают трудности в отлаживании ошибок и независимой работе над разными частями приложения. В последние годы монолитные приложения стали вытесняться приложениями с архитектурой на основе микросервисов. В такой архитектуре отдельные части бизнес-логики строятся и развертываются в виде изолированных (микро) пакетов кода. С помощью микросервисов большое приложение разбивается на меньшие, более управляемые части, благодаря чему разработчики могут строить, отлаживать и развертывать части приложения самостоятельно.

Тема "монолит против микросервиса" аналогична обсуждению масштабирования рабочего потока процессов ML, обеспечения возможности совместной работы и того, что шаги ML должны быть воспроизводимыми и готовыми к многоразовому использованию в разных рабочих потоках. Когда кто-то строит ML-модель самостоятельно, вполне возможно, что итеративное применение "монолитного" подхода будет быстрее. Данный подход нередко работает также и потому, что в разработку и техническое сопровождение каждого шага: сбор и предобработка данных, разработка, тренировка и развертывание модели, — активно вовлечен только один человек. Однако, когда этот рабочий поток масштабируется, за разные шаги могут начать отвечать разные люди или группы в организации. В целях масштабирования рабочего потока процессов ML нам нужен подход, с помощью которого коллектив разработчиков модели мог бы проводить испытания независимо от шага предобработки данных. Нам также понадобятся отслеживание производительности каждого шага конвейера и управление выходными файлами, генерируемыми каждой частью процессов.

В дополнение к этому, когда начальная разработка по каждому шагу будет завершена, мы захотим планировать запуск таких операций, как перетренировка, по расписанию, или инициировать запуски конвейера по наступлению событий, которые вызываются в ответ на изменения в среде, наподобие добавления в корзину новых тренировочных данных. В таких случаях необходимо, чтобы техническое решение позволяло нам выполнять весь рабочий поток целиком, от начала до конца, одним вызовом, сохраняя при этом возможность отслеживать получаемые на выходе данные и выполнять трассировку ошибок на отдельных шагах.

Решение

В целях урегулирования проблем, возникающих при масштабировании рабочего потока процессов машинного обучения, мы можем сделать каждый шаг в нашем рабочем потоке отдельной контейнеризированной службой. Контейнеры гаранти-

руют, что мы сможем выполнять один и тот же исходный код в разных средах, и между этими выполнениями мы увидим согласованное поведение. Такие отдельные контейнеризированные шаги позднее все вместе выстраиваются в цепочку, создавая конвейер, который может выполняться с помощью вызова REST API. Поскольку шаги конвейера выполняются в контейнерах, мы можем запускать их на ноутбуке разработчика с помошью локально установленной инфраструктуры либо посредством внешней облачной службы. Указанный конвейерный рабочий поток позволяет членам команды выстраивать шаги конвейера самостоятельно. Контейнеры также обеспечивают воспроизводимый способ запуска всего конвейера целиком, от начала до конца, поскольку они гарантируют согласованность между версиями библиотек, от которых зависит его работа, и средами выполнения. Кроме того, поскольку шаги контейнеризированного конвейера допускают разделение обязанностей, на отдельных шагах могут использоваться разные среды выполнения и версии языков программирования.

Для создания конвейеров существует масса инструментов как с вариантом локальной установки, так и с вариантом размещения в облаке, включая платформы Cloud AI Platform Pipelines[19], TensorFlow Extended[20] (TFX), Kubeflow Pipelines[21] (KFP), MLflow[22] и Apache Airflow[23]. В целях демонстрации паттерна "Конвейер рабочего потока" здесь мы определим конвейер с помощью платформы TFX и запустим его во внешней службе Cloud AI Platform Pipelines для выполнения конвейеров ML в облачной инфраструктуре Google Cloud с использованием вычислительного механизма Google Kubernetes Engine (GKE) в качестве инфраструктуры контейнеров.

Шаги в конвейерах TFX называются *компонентами*, они бывают двух видов: предварительно построенные и индивидуально настраиваемые. Как правило, первым в конвейере TFX является компонент, который принимает данные из внешнего источника. Он называется ExampleGen, где слово example в названии относится к терминологии машинного обучения и обозначает помеченый экземпляр, используемый для тренировки. Составляющие компонент ExampleGen[24] варианты позволяют получать данные из источников, таких как CSV-файлы, файлы TFRecord, запросы BigQuery или из реального прикладного источника. Например, компонент BigQueryExampleGen позволяет подключать данные, хранящиеся в BigQuery, к конвейеру с помощью запроса, который будет извлекать эти данные. Затем он сохраняет эти данные в виде файла TFRecord в корзине хранилища Google Cloud Storage (GCS), чтобы следующий компонент смог их использовать. Это как раз тот компонент, который мы будем настраивать, передавая ему запрос. Указанные варианты компонента ExampleGen относятся к фазе сбора данных в рамках описанного на рис. 6.6 рабочего потока процессов ML.

---

[19] См. **https://oreil.ly/nJo1p**.

[20] См. **https://oreil.ly/OznI3**.

[21] См. **https://oreil.ly/BoegQ**.

[22] См. **https://mlflow.org/**.

[23] См. **https://oreil.ly/63_GG**.

[24] См. **https://oreil.ly/Sjx9F**.

Следующий шаг указанного рабочего потока — это валидация данных. После того как мы получим и импортируем данные, мы сможем передать их другим компонентам для преобразования или анализа перед тренировкой модели. Компонент StatisticsGen[25] берет данные, принятые на шаге ExampleGen, и генерирует сводную статистику по предоставленным данным. SchemaGen выводит вычисленную схему из импортированных данных. Задействуя результаты работы SchemaGen[26], компонент ExampleValidator[27] обнаруживает аномалии на наборе данных и проверяет данные на наличие дрейфа данных или потенциальной асимметрии между тренировкой и обслуживанием[28]. Компонент Transform[29] также берет результат компонента SchemaGen, и здесь мы выполняем инженерию признаков для преобразования входных данных в правильный для нашей модели формат. Сюда может входить конвертирование входного текста произвольной формы в векторные вложения, нормализация числовых входных переменных и многое другое.

После того как данные будут готовы к подаче в модель, мы можем передать их в компонент тренировки TFX Trainer[30]. Настраивая компонент Trainer, мы указываем на функцию, которая определяет наш исходный код модели. Мы также можем отметить место, где мы хотели бы тренировать модель. Здесь мы покажем принцип использования тренировки в службе Cloud AI Platform Training из этого компонента. Наконец, компонент Pusher[31] занимается развертыванием модели. TFX предлагает много других уже построенных компонентов[32] — мы включили сюда лишь несколько из них и только те, которые будем использовать в примере конвейера.

В этом примере мы будем использовать набор данных NOAA об ураганах из BigQuery, чтобы построить модель, которая выполняет предсказательный вывод, продуцируя код SSHS[33] урагана. Мы будем поддерживать список признаков, компонентов и объем модельного кода относительно короткими, чтобы сосредоточиться на инструментах конвейера. Шаги нашего конвейера описаны ниже и примерно соответствуют рабочему потоку, представленному на рис. 6.6.

1. Сбор данных: выполнить запрос на получение данных об ураганах из BigQuery.
2. Валидация данных: применить компонент ExampleValidator для выявления аномалий и проверки на дрейф данных.

---

[25] См. **https://oreil.ly/kX1QY**.

[26] См. **https://oreil.ly/QpBlu**.

[27] См. **https://oreil.ly/UD7Uh**.

[28] Дополнительные сведения о валидации данных *см. в разд. "Паттерн 30. Призма объективности" главы 7.*

[29] См. **https://oreil.ly/xsJYT**.

[30] См. **https://oreil.ly/XFtR_**.

[31] См. **https://oreil.ly/qP8GU**.

[32] См. **https://oreil.ly/gHv_z**.

[33] SSHS расшифровывается как шкала ураганов Саффира — Симпсона (**https://oreil.ly/62kf3**) и представляет собой шкалу от 1 до 5, используемую для измерения силы и серьезности урагана. Обратите внимание, что ML-модель не предсказывает степень серьезности урагана в более позднее время. Вместо этого она просто усваивает пороговые значения скорости ветра, используемые в шкале Саффира — Симпсона.

3. Анализ и предобработка данных: сгенерировать немного статистики на данных и определить схему.
4. Тренировка модели: натренировать модель tf.keras на платформе AI Platform.
5. Развертывание модели: развернуть натренированную модель в службе AI Platform Prediction[34].

Когда наш конвейер будет завершен, мы сможем вызывать весь описанный выше рабочий поток целиком с помощью одного-единственного вызова API. Начнем с обсуждения вопросов сооружения строительных лесов для типичного конвейера TFX и процесса его выполнения на платформе AI Platform.

Разработка конвейера TFX

Для создания и вызова конвейера будем использовать инструменты командной строки tfx. Новые вызовы конвейера называются *исполнениями* и отличаются от обновлений, вносимых нами в сам конвейер, например добавления нового компонента. И то и другое можно делать с помощью интерфейса командной строки (command line interface, CLI) платформы TFX. Мы можем определять базовые элементы для конвейера в одном-единственном скрипте Python, который состоит из двух ключевых частей:

- экземпляра tfx.orchestration.pipeline[35], в котором мы определяем сам конвейер и компоненты, которые он содержит;
- экземпляра kubeflow_dag_runner[36] из библиотеки tfx[37]. Мы будем использовать его для создания и исполнения конвейера. В дополнение к Kubeflow runner существует также API для исполнения конвейеров TFX с инструментарием Apache Beam[38], который мы могли бы использовать для исполнения конвейера локально.

Наш конвейер (полный исходный код[39] см. на GitHub) будет иметь пять перечисленных выше шагов или компонентов, и мы можем определить наш конвейер следующим образом:

```
pipeline.Pipeline(
      pipeline_name='huricane_prediction',
      pipeline_root='path/to/pipeline/code',
      components=[
          bigquery_gen, statistics_gen, schema_gen, train, model_pusher
      ]
  )
```

[34] Хотя в примере конвейера развертывание является последним шагом, производственные конвейеры часто включают дополнительные шаги, такие как хранение модели в совместном хранилище или выполнение отдельного конвейера обслуживания запросов, который выполняет CI/CD и тестирование.

[35] См. **https://oreil.ly/62kf3**.

[36] См. **https://oreil.ly/62kf3**.

[37] См. **https://oreil.ly/62kf3**.

[38] См. **https://oreil.ly/hn0vF**.

[39] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/tree/master/06_reproducibility/workflow_pipeline**.

В целях использования поставляемого платформой TFX компонента BigQueryExampleGen мы отправляем запрос, который принесет наши данные. Этот компонент можно определить в одной строке кода, где *query* — это SQL-запрос к BigQuery в виде следующей инструкции:

```
bigquery_gen = BigQueryExampleGen(query=query)
```

Еще одним преимуществом использования конвейеров является то, что они предоставляют инструменты для отслеживания входных переменных, выходных артефактов и журналов для каждого компонента. Результатом работы компонента `statistics_gen`, например, является сводка о наборе данных, которую мы видим на рис. 6.7. `statistics_gen`[40] — это предварительно построенный компонент платформы TFX, в котором используется компонент валидации данных TF Data Validation для генерирования сводной статистики на нашем наборе данных.

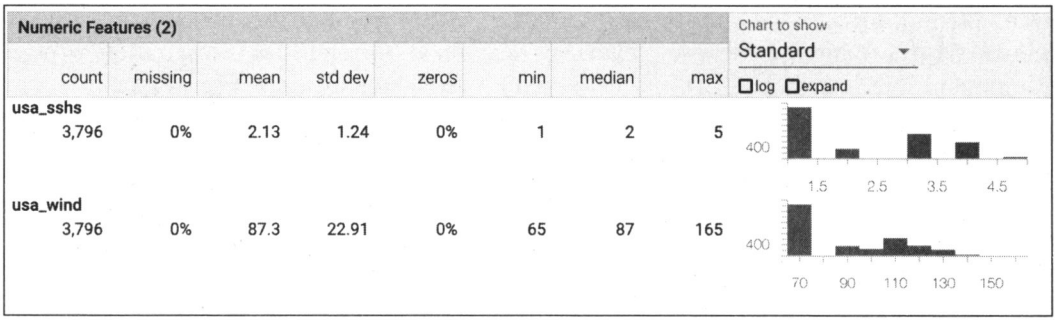


Рис. 6.7. Выходной артефакт из компонента `statistics_gen` в конвейере TFX

Исполнение конвейера на платформе Cloud AI Platform

Конвейер TFX можно исполнять в службе Cloud AI Platform Pipelines, которая будет управлять низкоуровневыми деталями инфраструктуры за нас. В целях развертывания конвейера на платформе AI Platform мы упакуем конвейерный код в контейнер Docker[41] и разместим его в реестре контейнеров GCR (Google Cloud Container Registry)[42, 43]. После перенесения контейнеризированного кода в реестр GCR создадим конвейер, используя интерфейс CLI платформы TFX:

```
tfx pipeline create \
--pipeline-path=kubeflow_dag_runner.py \
--endpoint='your-pipelines-dashboard-url' \
--build-target-image='gcr.io/your-pipeline-container-url'
```

---

[40] См. **https://oreil.ly/wvq9n**.

[41] См. **https://oreil.ly/rdXeb**.

[42] См. **https://oreil.ly/m5wqD**.

[43] Обратите внимание, что для запуска конвейеров TFX на платформе AI Platform в настоящее время вам необходимо разместить свой код на GCR, и вы не можете использовать другую службу реестра контейнеров, такую как DockerHub.

В приведенной выше команде адрес сервиса соответствует URL-адресу нашей панели мониторинга в службе AI Platform Pipelines. Когда она завершится, на панели мониторинга конвейеров мы увидим только что созданный конвейер. Команда `create` создает ресурс конвейера, который мы можем вызывать, инициировав исполнение (`run create`):

```
tfx run create --pipeline-name='your-pipeline-name' --endpoint='pipeline-url'
```

После выполнения этой команды мы сможем увидеть график, который обновляется в режиме реального времени по мере прохождения нашего конвейера через каждый шаг. На панели мониторинга службы Pipelines мы можем дополнительно проинспектировать отдельные шаги и увидеть любые дефекты, которые они генерируют, метаданные и многое другое. Пример результата для отдельного шага показан на рис. 6.8.

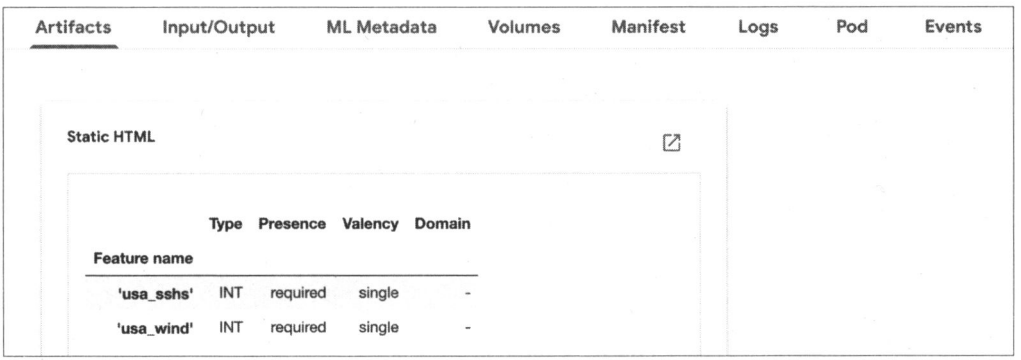


Рис. 6.8. Результат работы компонента `schema_gen` для конвейера ML. В верхнем горизонтальном меню отображаются данные по каждому отдельному шагу конвейера

Мы могли бы натренировать модель непосредственно в контейнерезированном конвейере в вычислительном механизме GKE, но платформа TFX предоставляет утилиту для использования облачной службы AI Platform Training в рамках нашего процесса. TFX также имеет расширение для развертывания натренированной модели в службе AI Platform Prediction. В нашем конвейере мы будем использовать обе интеграции. Служба AI Platform Training позволяет экономически эффективно использовать преимущества оборудования, заточенного под тренировку моделей, такого как графические или тензорные процессоры. Она также предоставляет возможность осуществлять распределенную тренировку, которая снижает время тренировки и минимизирует затраты на нее. Индивидуальные тренировочные задания и их результаты можно отслеживать в консоли платформы AI Platform.

Одним из преимуществ создания конвейера с помощью платформ TFX или Kubeflow является то, что мы не заперты в инфраструктуре Google Cloud. Мы можем выполнять приведенный здесь исходный код, используя службы Google AI Platform Pipelines в Azure ML Pipelines[44], на платформе Amazon SageMaker[45] либо локально.

[44] См. **https://oreil.ly/A5Rxe**.

[45] См. **https://oreil.ly/H3p3Y**.

В целях реализации тренировочного шага в TFX будем применять компонент Trainer и передавать ему информацию о тренировочных данных для использования на входе в модель вместе с исходным кодом тренировки модели. TFX предоставляет расширение, которое предназначено для выполнения тренировочного шага на платформе AI Platform. Его можно задействовать, импортировав tfx.extensions.google_cloud_ai_platform.trainer и предоставив подробную информацию о конфигурации тренировки, настроенной в платформе AI Platform. Указанная информация включает название проекта, регион и местоположение контейнера в реестре контейнеров Google Cloud (GCR) с тренировочным кодом.

Кроме того, TFX имеет компонент AI Platform Pusher[46], который предназначен для развертывания натренированных моделей в службе AI Platform Prediction. Для того чтобы воспользоваться компонентом Pusher с платформой AI Platform, мы предоставляем подробную информацию о названии и версии нашей модели, а также функцию обслуживания, сообщающую платформе AI Platform формат входных данных, которые та ожидает для нашей модели. С учетом всего этого мы получаем полный конвейер, который принимает данные, анализирует их, выполняет их преобразование и, наконец, тренирует и развертывает модель с помощью платформы AI Platform.

Почему это работает

Без выполнения нашего кода ML в качестве конвейера другим людям было бы трудно надежно воспроизводить нашу работу. Им бы приходилось брать наш исходный модельный код предобработки, разработки, тренировки и обслуживания и пытаться воспроизводить ту же среду, в которой мы его выполняли, учитывая все библиотечные зависимости, аутентификацию и многое другое. Если еще есть код, который управляет выбором нижестоящих компонентов, основываясь на данных из вышестоящих компонентов, то этот код тоже должен быть надежно реплицирован. Паттерн "Конвейер рабочего потока" позволяет другим пользователям запускать и выполнять мониторинг всего рабочего потока процессов ML от начала до конца как в локальной, так и в облачной среде, сохраняя при этом возможность отладки данных на выходе из отдельных шагов. Контейнеризация каждого шага конвейера позволяет другим людям воспроизводить как среду, которую мы использовали для его строительства, так и весь рабочий поток целиком, зафиксированный в конвейере. Она также дает нам возможность потенциального воспроизведения окружающей среды позже при необходимости для поддержания нормативных потребностей. Кроме того, с конвейерами TFX и AI Platform Pipelines панель мониторинга предлагает нам пользовательский интерфейс для отслеживания выходных артефактов, полученных в результате каждого исполнения конвейера. Эта тема обсуждается в *разд. "Компромиссы и альтернативы" далее в этой главе*.

Когда каждый компонент конвейера находится в собственном контейнере, разные члены команды могут параллельно строить и тестировать отдельные части конвейе-

[46] См. **https://oreil.ly/bJavO**.

ра. Это позволяет ускорять разработку и минимизировать риски, связанные с более монолитным рабочим потоком процессов ML, в котором шаги неразрывно связаны друг с другом. Например, зависимости пакетов и исходный код, необходимые для создания шага предобработки данных, могут существенно отличаться от тех, которые используются для развертывания модели. Выстраивая эти этапы в рамках конвейера, мы можем каждую деталь построить в отдельном контейнере со своими зависимостями и включить в более крупный конвейер после завершения.

Итак, помимо уже построенных компонентов, которые поставляются с конвейерными каркасами, такими как TFX, паттерн "Конвейер рабочего потока" дает нам преимущества, которые приходят вместе с ориентированным ациклическим графом (directed acyclic graph, DAG). Поскольку конвейер представляет собой ориентированный ациклический граф, у нас есть возможность выполнять отдельные шаги или исполнять весь конвейер от начала до конца. Кроме того, у нас появляется возможность журналирования и мониторинга каждого шага конвейера в разных исполнениях, а также отслеживания артефактов из каждого шага и исполнения конвейера в централизованном месте. Предварительно построенные компоненты обеспечивают автономные, готовые к использованию шаги для распространенных компонентов рабочего потока, включая тренировку, оценивание и предсказательный вывод. Эти компоненты работают как отдельные контейнеры везде, где мы решим исполнить наш конвейер.

Компромиссы и альтернативы

Главной альтернативой использованию конвейерного каркаса является выполнение шагов рабочего потока процессов ML с помощью паллиативного подхода через отслеживание блокнотов и результатов, ассоциированных с каждым шагом. Несомненно, с конвертированием различных частей рабочего потока процессов ML в организованный конвейер связаны накладные расходы. В этом разделе мы рассмотрим некоторые вариации и расширения паттерна "Конвейер рабочего потока": создание контейнеров вручную, автоматизацию конвейера с помощью инструментов непрерывной интеграции и непрерывной доставки (continuous integration/continuous delivery, CI/CD), процессы перехода от конвейера рабочего потока разработки к конвейеру рабочего потока производства и альтернативные инструменты для строительства и оркестровки конвейеров. Мы также рассмотрим способы использования конвейеров для отслеживания метаданных.

Разработка компонентов

Вместо использования уже разработанных и настроенных компонентов TFX для строительства конвейера мы можем определять наши собственные контейнеры, которые будут выступать в качестве компонентов либо использовать функцию на языке Python как компонент.

В целях использования контейнерно-ориентированных компонентов[47], предлагаемых платформой TFX, мы применяем метод `create_container_component`, передавая

[47] См. **https://oreil.ly/5ryEn**.

ему входные и выходные параметры нашего компонента и базовый образ Docker вместе с любыми относящимися к контейнеру командами точки входа. Например, следующий ниже контейнерно-ориентированный компонент вызывает инструмент командной строки bq для скачивания набора данных BigQuery:

```
component = create_container_component(
    name='DownloadBQData',
    parameters={
        'dataset_name': string,
        'storage_location': string
    },
    image='google/cloud-sdk:278.0.0',
,
    command=[
        'bq', 'extract', '--compression=csv', '--field_delimiter=,',
        InputValuePlaceholder('dataset_name'),
        InputValuePlaceholder('storage_location'),
    ]
)
```

Лучше всего использовать базовый образ, который уже имеет большинство необходимых нам зависимостей. Мы выбрали образ Google Cloud SDK, который имеет в своем составе инструмент командной строки bq.

Помимо этого, с помощью декоратора `@component` можно конвертировать функцию Python в компонент TFX. В целях демонстрации предположим, что у нас есть шаг для подготовки ресурсов, используемых во всем конвейере, в котором мы создаем корзину облачного хранилища Google Cloud Storage. Этот шаг можно определить, используя следующий фрагмент исходного кода:

```
from google.cloud import storage
client = storage.Client(project="your-cloud-project")

@component
def CreateBucketComponent(
        bucket_name: Parameter[string] = 'your-bucket-name',
        ) -> OutputDict(bucket_info=string):
    client.create_bucket('gs://' + bucket_name)
    bucket_info = storage_client.get_bucket('gs://' + bucket_name)
    return {
        'bucket_info': bucket_info
    }
```

Затем мы можем добавить этот компонент в определение конвейера:

```
create_bucket = CreateBucketComponent(
    bucket_name='my-bucket')
```

Интеграция CI/CD с конвейерами

В дополнение к вызову конвейеров через панель мониторинга или программно через интерфейс командной строки (CLI) или API, скорее всего, по мере того как мы будем разворачивать модель, у нас возникнет потребность в автоматизировании исполнения конвейера. Например, возможно, понадобится вызывать конвейер всякий раз, когда появляется некоторое количество новых тренировочных данных. Как вариант — у нас может возникнуть потребность исполнять конвейер, когда его исходный код изменяется. Добавление CI/CD в конвейер рабочего потока помогает соединять триггерные события с исполнениями конвейера.

Существует масса управляемых служб по настройке триггеров, которые исполняют конвейер при возникновении потребности в перетренировывании модели на новых данных. Мы можем использовать управляемую службу планирования для вызова конвейера по расписанию. С другой стороны, мы можем использовать бессерверную событийно-управляемую платформу, такую как Cloud Functions[48], для вызова конвейера при добавлении новых данных в хранилище. В нашей функции мы можем указать условия — например, порог количества новых добавленных данных, обусловливающий необходимость перетренировки, — для инициирования нового исполнения конвейера. Как только у нас появилось достаточно новых тренировочных данных, мы можем пересоздать конвейер для перетренировки и переразвертывания модели (рис. 6.9).

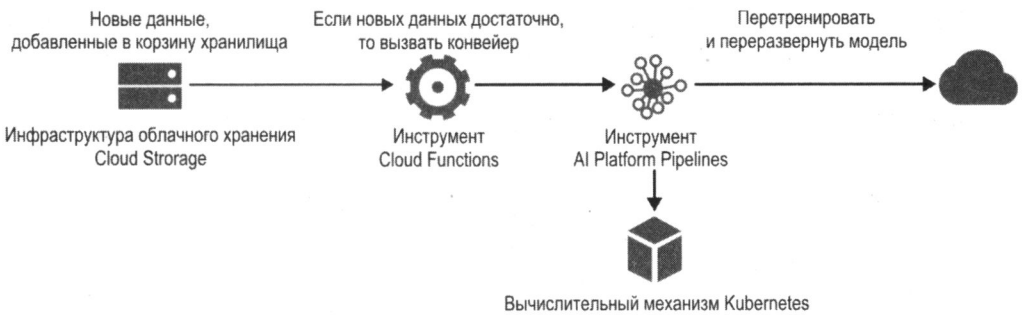


Рис. 6.9. Рабочий поток процессов CI/CD с использованием инструмента Cloud Functions для вызова конвейера при наличии достаточного количества новых данных в хранилище

Если мы хотим исполнять конвейер, основываясь на изменениях в исходном коде, нам поможет инструмент CI/CD, такой как Cloud Build. Когда Cloud Build[49] исполняет исходный код, он отрабатывает его в виде серии контейнеризированных шагов. Этот подход хорошо вписывается в контекст конвейеров. Мы можем присоединить службу Cloud Build к действиям GitHub[50] или триггерам GitLab[51] в репози-

[48] См. **https://oreil.ly/rVyzX**.

[49] См. **https://oreil.ly/kz8Aa**.

[50] См. **https://oreil.ly/G2Xwv**.

[51] См. **https://oreil.ly/m_dYr**.

тории, где находится конвейерный код. Когда код будет зафиксирован в репозитории, служба Cloud Build, основываясь на новом коде, построит контейнеры, которые ассоциированы с нашим конвейером, и инициирует его исполнение.

Платформы Apache Airflow и Kubeflow Pipelines

В дополнение к TFX альтернативными платформами для реализации паттерна "Конвейер рабочего потока" являются платформы Apache Airflow[52] и Kubeflow Pipelines[53]. Как и TFX, платформы Airflow и KFP рассматривают конвейеры как ориентированный ациклический граф, в котором рабочий поток для каждого шага определяется в скрипте Python. Затем они берут этот скрипт и предоставляют API-интерфейсы для манипуляций с запуском по расписанию и оркестрированием графа в указанной инфраструктуре. И Airflow, и KFP имеют открытый исходный код и поэтому могут работать локально либо в облаке.

Обычно Apache Airflow используется для инженерии данных, поэтому его следует рассматривать для задач извлечения, преобразования и загрузки (ETL) корпоративных данных. Однако, хотя Airflow и обеспечивает надежный инструментарий для

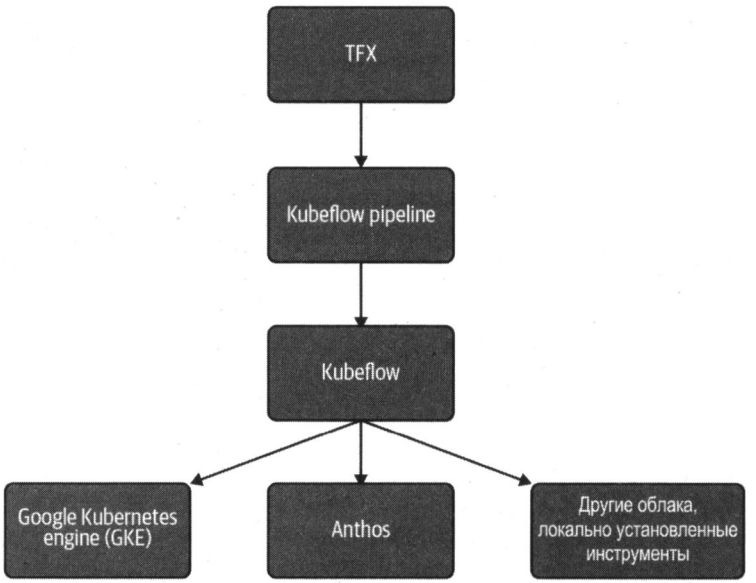


Рис. 6.10. Взаимосвязь между платформами TFX, Kubeflow Pipelines, Kubeflow и инфраструктурой. Платформа TFX работает на самом высоком уровне поверх платформы Kubeflow Pipelines, с предварительно построенными компонентами, предлагающими те или иные подходы к распространенным шагам рабочего потока. Платформа Kubeflow Pipelines предлагает API для определения и организации конвейера ML, обеспечивая бóльшую гибкость в реализации каждого шага. И TFX, и KFP работают на Kubeflow — платформе для выполнения контейнерно-ориентированных рабочих нагрузок ML в Kubernetes. Все инструменты на этой диаграмме имеют открытый исходный код, поэтому инфраструктура, в которой работают конвейеры, зависит от пользователя — некоторые варианты включают GKE, Anthos, Azure, AWS либо локально установленную инфраструктуру

[52] См. **https://oreil.ly/rQlqK**..

[53] См. **https://oreil.ly/e_7zJ**.

выполнения текущих заданий, она была построена как общецелевое техническое решение и не была рассчитана на рабочие нагрузки ML. Платформа KFP, с другой стороны, была создана специально для ML и работает на более низком уровне, чем TFX, обеспечивая бóльшую гибкость в том, как определяются шаги конвейера. В то время как TFX реализует собственный подход к оркестровке, KFP позволяет нам выбирать способ оркестровки конвейеров через свой API. Взаимосвязь между TFX, KFP и Kubeflow представлена на рис. 6.10.

Конвейер разработки и промышленный конвейер

Принцип вызова конвейера часто меняется по мере перехода от разработки к промышленному использованию. Скорее всего, мы захотим строить и прототипировать конвейер из блокнота, из которого сможем вызывать конвейер многократно, выполняя ячейку блокнота, отлаживать ошибки и обновлять код из одной и той же среды. После того как мы будем готовы к производству, мы сможем перенести наш компонентный код и определение конвейера в единый скрипт. Располагая определением конвейера в скрипте, мы сможем планировать запуски на исполнение по расписанию и облегчим другим сотрудникам нашей организации вызов конвейера воспроизводимым способом. Одним из инструментов для создания таких конвейеров является Kale[54], который берет код блокнота Jupyter и конвертирует его в скрипт, используя API платформы Kubeflow Pipelines.

Промышленный конвейер также допускает *оркестровку* рабочего потока процессов ML. Под оркестровкой мы подразумеваем добавление в конвейер логики, которая определяет типы исполняемых шагов и результатов этих шагов. Например, мы могли бы принять решение развертывать только те модели, которые имеют точность 95% или выше. Когда свежие данные запустят конвейер на исполнение и начнут тренировку обновленной модели, мы сможем добавлять логику проверки данных на выходе из компонента оценивания и исполнять компонент развертывания, если точность превышает установленный порог, либо завершать исполнение конвейера в противном случае. Рассмотренные ранее в этом разделе платформы Airflow и Kubeflow Pipelines предоставляют API для оркестровки конвейеров.

Отслеживание линии преемственности в конвейерах ML

Еще одной особенностью конвейеров является их использование для отслеживания модельных метаданных и артефактов. Такое отслеживание называется *отслеживанием линии преемственности* (lineage). Всякий раз при вызове конвейера генерируется ряд артефактов. Они могут включать сводки о наборах данных, экспортированные модели, результаты модельного оценивания, метаданные о тех или иных вызовах конвейера и многое другое. Отслеживание линии преемственности позволяет нам визуализировать историю версий модели вместе с другими ассоциированным модельными артефактами. Например, в платформе AI Platform Pipelines мы можем использовать панель мониторинга конвейеров, чтобы знакомиться с данны-

[54] См. **https://github.com/kubeflow-kale/kale**.

ми, разбитыми по схеме данных и по дате, на которых та или иная версия модели была натренирована. На рис. 6.11 показана панель мониторинга проводника по линии преемственности для конвейера TFX, работающего на платформе AI Platform. Она позволяет нам отслеживать входные данные и выходные артефакты, ассоциированные с той или иной моделью.

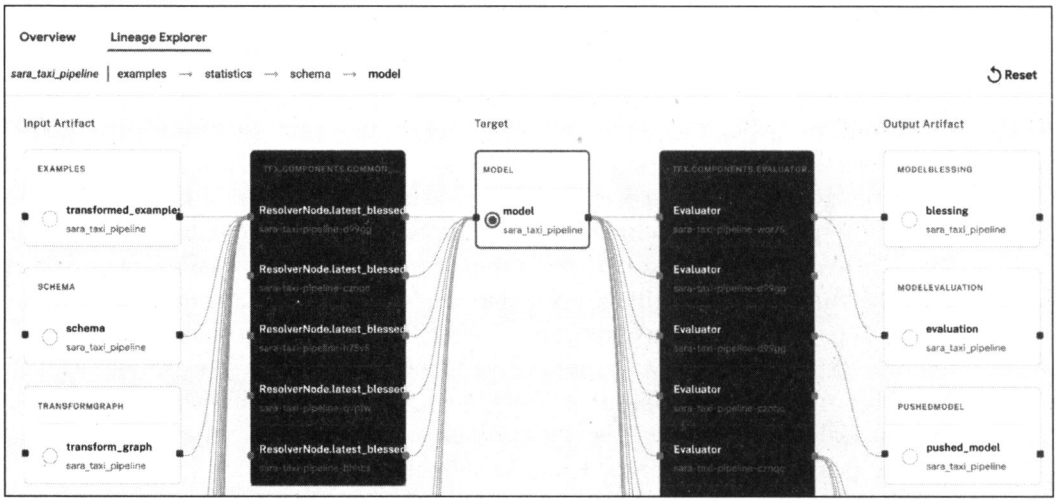


Рис. 6.11. Секция **Lineage Explorer** в панели мониторинга платформы AI Platform Pipelines для конвейера TFX

Одним из преимуществ отслеживания линии преемственности для управления артефактами, генерируемыми во время исполнения конвейера, является поддержка им как облачных, так и локально установленных сред. Фунционал отслеживания придает гибкость, обеспечивая информацией о том, где модели тренируются и развертываются и где хранятся модельные метаданные. Отслеживание линии преемственности также является важным аспектом обеспечения воспроизводимости конвейеров ML, поскольку оно позволяет сравнивать метаданные и артефакты из разных исполнений конвейера.

ПАТТЕРН 26. Хранилище признаков

Паттерн "Хранилище признаков" (Feature Store) упрощает управление признаками и многоразовое их использование в разных проектах, устраняя сцепленность между процессом создания признаков и разработкой моделей, в которых эти признаки используются.

Постановка задачи

Хорошая инженерия признаков имеет решающее значение для успеха многих технических решений машинного обучения. Однако этот процесс также является одной из самых трудоемких частей разработки модели. Для правильного расчета

некоторых признаков требуются значительные знания предметной области — изменения в деловой стратегии могут повлиять на то, каким образом признак следует вычислять. В целях обеспечения согласованного вычисления таких признаков рекомендуется, чтобы эти признаки находились под контролем экспертов предметной области, а не инженеров ML. Некоторые входные поля вполне могут допускать разные варианты представления данных (*см. главу 2*), делая их более удобными для машинного обучения. Прежде чем принимать решение о том, какие признаки будут использоваться в окончательной модели, инженер ML или исследователь данных обычно будет экспериментировать с несколькими разными преобразованиями, чтобы определять, какие из них полезны, а какие можно проигнорировать. Во многих случаях используемые для модели ML данные не берутся из одного источника. Одни данные могут поступать со склада данных, другие могут находиться в корзине хранилища в виде неструктурированных данных, а третьи же могут собираться в реальном времени посредством потоковой передачи. Структура данных также может варьироваться между любым из этих источников, требуя, чтобы каждая входная переменная проходила собственные шаги генерирования признака перед тем, как ее можно будет подать в модель. Эта работа нередко выполняется на виртуальной либо персональной машине, в результате чего создание признаков привязывается к программной среде, в которой строится модель, и чем сложнее становится модель, тем сложнее эти конвейеры данных.

Нерегламентированный подход, при котором признаки создаются в проектах ML по мере необходимости, возможно, и будет работать для разовой разработки и тренировки модели, но по мере масштабирования организаций этот метод инженерии признаков становится непрактичным и начинают возникать значительные проблемы.

- Нерегламентированные признаки нелегко использовать многократно. Признаки создаются снова и снова, отдельными пользователями или внутри коллективов, либо никогда не покидают конвейеры (или блокноты), в которых они создаются. Такая ситуация вызывает особые проблемы для признаков более высокого уровня, которые сложно вычислять. Это может быть связано с тем, что их получают посредством дорогостоящих процессов, таких как предварительно натренированные векторные вложения пользователей или позиций каталога. В других случаях это может быть связано с тем, что признаки улавливаются из вышестоящих процессов, таких как приоритеты предприятия, наличие контрактов или сегментации рынка. Еще один источник сложности кроется в том, что признаки более высокого уровня, такие как число заказов клиента за последний месяц, предусматривают агрегирование во времени. Усилия и время тратятся впустую, создавая одни и те же признаки с нуля для каждого нового проекта.
- Руководство данными усложняется, если каждый ML-проект вычисляет признаки из чувствительных данных по-разному.
- Нерегламентированные признаки непросто использовать совместно в разных командах или проектах. Во многих организациях одни и те же сырые данные используются несколькими командами, но команды могут определять признаки по-своему, при этом нелегко получить доступ к документации признаков. Всё

это также препятствует эффективному взаимному сотрудничеству, приводит к разрозненной работе и ненужному дублированию усилий.

- Нерегламентированные признаки, используемые для модельной тренировки и обслуживания запросов, не согласованы, т. е. имеется асимметрия между тренировкой и обслуживанием. Тренировка обычно проводится на основе исторических данных с пакетными признаками, которые создаются в офлайновом режиме. Однако обработка запросов моделью, как правило, осуществляется в онлайновом режиме. Если признаковый конвейер для модельной тренировки вообще отличается от конвейера, используемого в промышленной среде, для обработки запросов (например, разные библиотеки, исходный код предобработки или языки), то мы рискуем получить асимметрию между тренировкой и обработкой.
- Производство признаков затруднено. При переходе к промышленному использованию не существует стандартизированного способа доставки признаков для онлайновых ML-моделей и доставки пакетных признаков для офлайновой тренировки моделей. Модели тренируются в офлайновом режиме с признаками, созданными в пакетных процессах, но при обработке запросов в продуктиве эти признаки часто создаются с акцентом на низкую задержку и в меньшей степени на высокую пропускную способность. Фреймворк генерирования и хранения признаков не является гибким для манипулирования обоими этими сценариями.

Иными словами, нерегламентированный подход к инженерии признаков замедляет разработку модели и приводит к дублированию усилий и неэффективности рабочего потока. Более того, процессы создания признаков несовместимы между тренировкой и предсказательным выводом, что приводит к риску асимметрии между тренировкой и обработкой или утечки данных из-за непреднамеренного внесения неточной информации о метках в конвейер ввода данных для модели.

Решение

Решение состоит в создании совместного хранилища признаков, централизованного места для хранения и документирования наборов признаковых данных, которые будут задействоваться при разработке моделей машинного обучения и могут использоваться совместно между проектами и командами. Хранилище признаковых данных выступает в качестве интерфейса между конвейерами создания признаков (компетенция инженера данных) и рабочих потоков разработки моделей с использованием этих признаков (компетенция исследователя данных) (рис. 6.12). При таком подходе существует центральное хранилище для размещения предварительно вычисленных признаков, что ускоряет время разработки и помогает в обнаружении признаков. Указанное хранилище также позволяет применять к создаваемым признакам базовые принципы инженерии программного обеспечения — управление версиями, документирование и контроль доступа.

Типичное хранилище признаков строится с двумя ключевыми характеристиками их дизайна: инструментарием для быстрой обработки крупных наборов признаковых данных и методикой хранения признаков, которая поддерживает как доступ с низкой задержкой (для предсказательного вывода), так и крупнопакетный доступ (для

тренировки модели). Существует также слой метаданных, который упрощает документирование и управление версиями разных наборов признаков, а также API, который управляет загрузкой и извлечением признаковых данных.

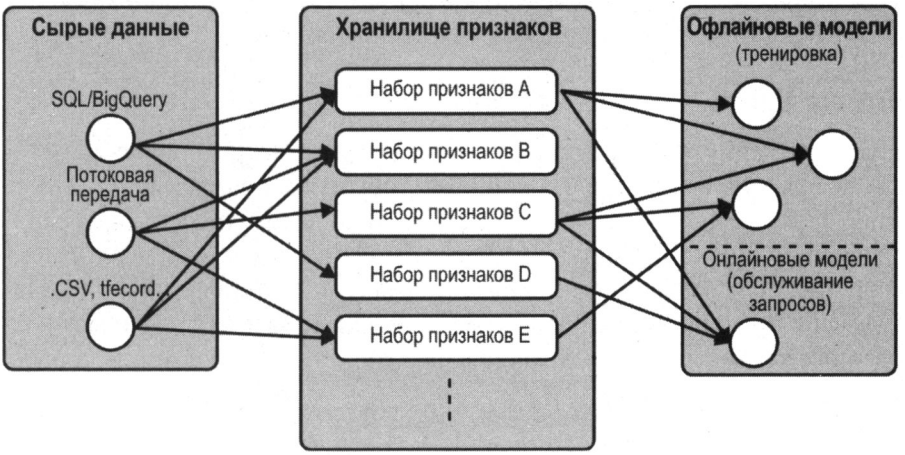


Рис. 6.12. Хранилище признаков обеспечивает мост между источниками сырых данных и модельной тренировкой и обслуживанием запросов

Типичный рабочий поток исследователя данных или инженера ML состоит в чтении сырых данных (структурированных или потоковых) из источника данных, применении различных преобразований на данных посредством любимого фреймворка и сохранении преобразованного признака внутри хранилища признаков. Вместо того чтобы создавать признаковые конвейеры для поддержания одной-единственной модели ML, паттерн "Хранилище признаков", наоборот, устраняет сцепленность между генерированием признаков и разработкой модели. В частности, такие инструменты, как Apache Beam, Flink или Spark, часто используются при строительстве хранилища признаков, поскольку они могут обрабатывать данные как в пакетном режиме, так и в потоковом. Он также снижает вероятность асимметрии между тренировкой и обработкой, т. к. признаковые данные заполняются одинаковыми конвейерами создания признаков.

После создания признаки помещаются в хранилище данных, откуда они извлекаются для модельной тренировки и обработки запросов. Его скорость оптимизирована под обработку запросов на извлечение признаков. Промышленная модель, которая поддерживает какое-либо онлайновое приложение, возможно, должна будет выдавать предсказания в режиме реального времени в пределах миллисекунд, а значит, необходимо обеспечить низкую задержку. Однако в случае тренировки более высокая задержка не является проблемой. Вместо этого акцент делается на высокой пропускной способности, поскольку исторические признаки извлекаются для тренировки крупными пакетами. Хранилище признаков обращается к обоим этим вариантам, используя разные хранилища данных для доступа к онлайновым и офлайновым признакам. Например, хранилище признаков может использовать Cassandra или Redis как хранилища данных для онлайнового извлечения признаков, а Hive или BigQuery — для доставки исторических наборов признаков.

В конечном счете типичное хранилище признаков будет вмещать большое число разных наборов признаков, созданных из несметного числа источников сырых данных. Внутри него будет встроен слой метаданных, служащий для документирования наборов признаков и обеспечения реестра с целью легкого обнаружения признаков и перекрестного сотрудничества между коллективами.

Хранилище Feast

В качестве примера этого паттерна в действии рассмотрим хранилище данных с открытым исходным кодом для машинного обучения под названием Feast[55], разработанное в Google Cloud и Gojek[56]. Оно построено вокруг облачных служб Google Cloud[57], использующих BigQuery для офлайновой тренировки моделей и Redis для обработки запросов с низкой задержкой (рис. 6.13). Набор инструментальных средств Apache Beam применяется для создания признаков, что позволяет создавать согласованные конвейеры данных как для пакетной, так и для потоковой передачи.

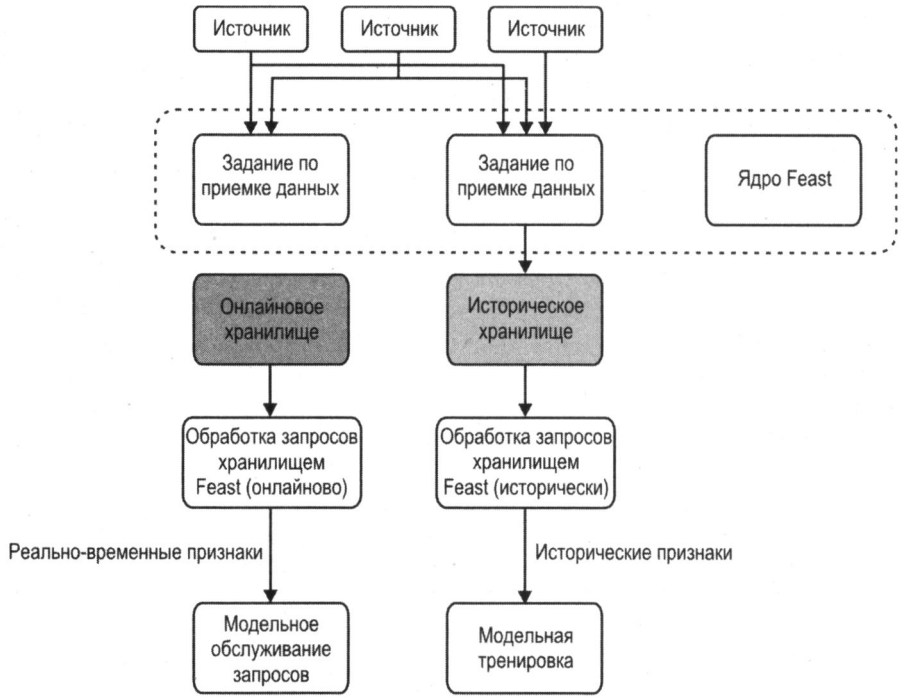


Рис. 6.13. Высокоуровневая архитектура хранилища признаков Feast. Она строится вокруг Google BigQuery, Redis и Apache Beam

[55] См. **https://github.com/feast-dev**.

[56] См. **https://oreil.ly/PszIn**.

[57] См. **https://oreil.ly/ecJou**.

В целях ознакомления с тем, как это хранилище работает на практике, мы воспользуемся публичным набором данных BigQuery, содержащим информацию о поездках в нью-йоркском такси[58]. Каждая строка таблицы содержит метку времени посадки, широту и долготу места посадки, широту и долготу места высадки, число пассажиров и стоимость проезда в такси. Целью ML-модели будет предсказание стоимости проезда в такси, обозначаемой `fare_amount`, с использованием этих характеристик.

Указанная модель пользуется преимуществом генерирования дополнительных признаков из сырых данных. Например, поскольку поездки в такси основаны на расстоянии и продолжительности поездки, предварительно вычисленное расстояние между местом посадки и высадки является полезным признаком. После того как этот признак будет вычислен на наборе данных, мы сможем сохранить его в наборе признаков для использования в будущем.

Добавление признаковых данных в хранилище Feast. Данные хранятся в Feast, используя специальный для наборов признаков класс `FeatureSets`. Это класс содержит схему данных и информацию об источнике данных независимо от того, откуда она поступает: из кадра данных pandas или же из темы Kafka. Он позволяет хранилищу Feast знать о том, из каких источников выбирать необходимые для признака данные, как их принимать (получать и импортировать), а также некоторые базовые характеристики типов данных. Группы признаков могут приниматься и храниться вместе, а наборы признаков обеспечивают эффективное хранение и логически организованную разбивку данных на пространства имен внутри этих хранилищ.

После регистрации нашего набора признаков хранилище Feast запустит задание Apache Beam для заполнения хранилища признаков данными из источника. Набор признаков используется для генерирования хранилищ офлайновых признаков и хранилищ онлайновых признаков, что позволяет разработчикам тренировать свои модели и обрабатывать запросы на модельное предсказание с помощью одних и тех же данных. Feast обеспечивает, чтобы данные из источников соответствовали ожидаемой схеме набора признаков.

Как показано на рис. 6.14, приемка признаковых данных в хранилище Feast состоит из четырех шагов.

Эти четыре шага заключаются в следующем.

1. Создать экземпляр `FeatureSet`. Набор признаков содержит сущности, признаки и источник.
2. Добавить сущности и признаки в экземпляр `FeatureSet`.
3. Зарегистрировать экземпляр `FeatureSet`. В результате будет создан именованный набор признаков внутри хранилища Feast. Набор признаков не содержит признаковых данных.
4. Загрузить признаковые данные в экземпляр `FeatureSet`.

---

[58] Данные доступны в таблице BigQuery: bigquery-public-data.new_york_taxi_trips.tlc_yellow_trips_2016.

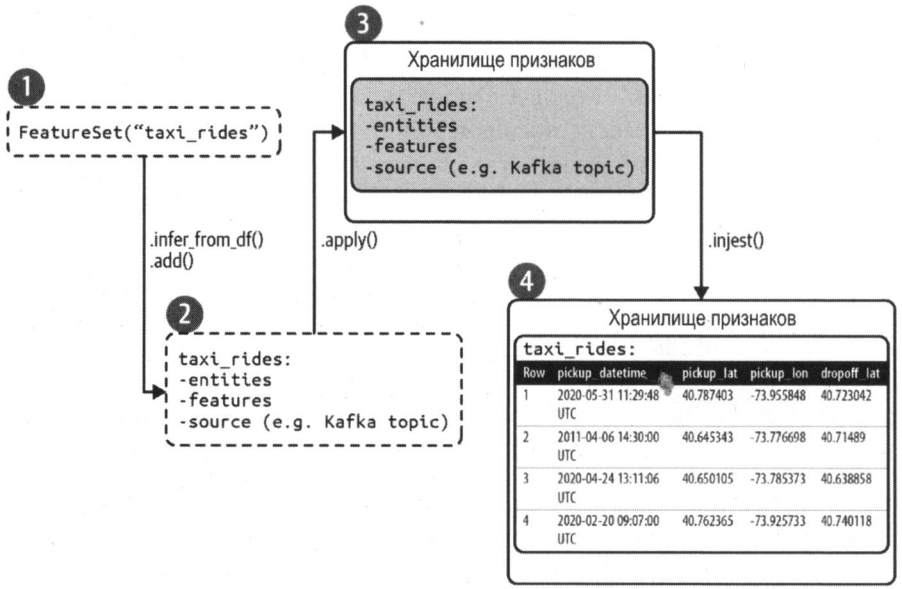


Рис. 6.14. Приемка признаковых данных в хранилище Feast состоит из четырех шагов: создать набор признаков, добавить сущности и признаки, зарегистрировать экземпляр FeatureSet и импортировать признаковые данные в FeatureSet

Блокнот с полным исходным кодом[59] этого примера можно найти в репозитории, прилагаемом к этой книге.

Создание экземпляра `FeatureSet`. Мы подключаемся к развернутому хранилищу Feast, настраивая клиента с помощью SDK на языке Python:

```
from feast import Client, FeatureSet, Entity, ValueType

# Подключиться к существующему развернутому хранилищу Feast
client = Client(core_url='localhost:6565')
```

Мы можем проверить подключение клиента, распечатав существующие наборы признаков с помощью команды `client.list_feature_sets()`. Если выполняется новое развертывание, она вернет пустой список. В целях создания нового набора признаков следует вызвать класс `FeatureSet` и указать имя набора признаков:

```
# Создать набор признаков
taxi_fs = FeatureSet("taxi_rides")
```

Добавление сущностей и признаков в экземпляр `FeatureSet`. В контексте хранилища Feast класс `FeatureSets` состоит из сущностей и признаков. Сущности используются в качестве ключей для поиска значений признаков и для соединения признаков между разными наборами признаков при создании наборов данных для тренировки модели или обработки запросов. Сущность служит идентификатором для любой релевантной характеристики, которая у вас есть в наборе данных. Он пред-

[59] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/06_reproducibility/feature_store.ipynb**.

ставляет собой объект, который можно моделировать и куда можно сохранять информацию. В контексте службы найма автомобиля с водителем или доставки еды релевантной сущностью может быть `customer_id`, `order_id`, `driver_id` или `restaurant_id`. В контексте модели оттока клиентов сущностью может быть `customer_id` или `segment_id`. В нашем случае сущностью является `taxi_id`, уникальный идентификатор поставщика такси для каждой поездки.

На этом этапе созданный нами набор признаков под названием `taxi_rides` (поездки в такси) не содержит никаких сущностей или признаков. Мы можем использовать ядерного клиента Feast, чтобы указать их из кадра данных pandas, который содержит сырые входные данные и сущности (табл. 6.5).

Таблица 6.5. *Набор данных `taxi_ride`, содержащий информацию о поездках в нью-йоркском такси. Сущностью является `taxi_id`, уникальный идетификатор поставщика такси для каждой поездки*

Строка	pickup_datetime	pickup_lat	pickup_lon	dropoff_lat	dropoff_lon	num_pass	taxi_id	fare_amt
1	2020-05-31 11:29:48 UTC	40.787403	-73.955848	40.723042	-73.993106	2	0	15.3
2	2011-04-06 14:30:00 UTC	40.645343	-73.776698	40.71489	-73.987242	2	0	45.0
3	2020-04-24 13:11:06 UTC	40.650105	-73.785373	40.638858	-73.9678	2	2	32.1
4	2020-02-20 09:07:00 UTC	40.762365	-73.925733	40.740118	-73.986487	2	1	21.3

Определение источников потоковых данных при создании экземпляра FeatureSet

При создании набора признаков пользователи могут определять источники потоковых данных. После того как набор признаков будет зарегистрирован с источником, хранилище Feast автоматически начнет заполнять свои хранилища данными из этого источника. Ниже приведен пример набора признаков с предоставленным пользователем источником, который извлекает потоковые данные из темы Kafka:

```
feature_set = FeatureSet(
    name="stream_feature",
    entities=[
        Entity("taxi_id", ValueType.INT64)
    ],
    features=[
        Feature("traffic_last_5min", ValueType.INT64)
    ],
    source=KafkaSource(
        brokers="mybroker:9092",
        topic="my_feature_topic"
    )
)
```

Здесь важна метка времени pickup_datetime, т. к. она необходима для извлечения пакетных признаков и используется для обеспечения соединений пакетных признаков, правильных на определенный момент времени. В целях создания дополнительного признака, такого как евклидово расстояние, следует загрузить набор данных в кадр данных pandas и вычислить этот признак:

```
# Загрузить кадр данных
taxi_df = pd.read_csv("taxi-train.csv")

# Сконструировать признак, евклидово расстояние
taxi_df['euclid_dist'] = taxi_df.apply(compute_dist, axis=1)
```

Мы можем добавлять сущности и признаки в набор признаков с помощью функции .add(...). С другой стороны, метод .infer_fields_from_df(...) создаст сущности и признаки для нашего экземпляра FeatureSet непосредственно из фрейма данных pandas. Мы просто указываем имя столбца, который представляет сущность. Схема и типы данных для признаков экземпляра FeatureSet позднее вычисляются из фрейма данных:

```
# Вычислить признаки из кадра данных pandas
  taxi_fs.infer_fields_from_df(taxi_df,
        entities=[Entity(name='taxi_id', dtype=ValueType.INT64)],
replace_existing_features=True)
```

Регистрация экземпляра FeatureSet. После того как набор признаков будет создан, мы можем зарегистрировать его в хранилище Feast с помощью метода client.apply(taxi_fs). В целях подтверждения правильности регистрации набора признаков или инспектирования содержимого еще одного набора признаков мы можем извлечь его с помощью метода .get_feature_set(...):

```
print(client.get_feature_set("taxi_rides"))
```

Приведенная выше инструкция возвращает объект JSON, содержащий схему данных для набора признаков taxi_rides:

```
{
  "spec": {
    "name": "taxi_rides",
    "entities": [
      {
        "name": "key",
        "valueType": "INT64"
      }
    ],
    "features": [
      {
        "name": "dropoff_lon",
        "valueType": "DOUBLE"
      },
```

```
      {
        "name": "pickup_lon",
        "valueType": "DOUBLE"
      },
      ...
    ...
    ],
    }
}
```

Внесение признаковых данных в экземпляр `FeatureSet`. Убедившись в правильности нашей схемы, мы можем направить признаковые данные кадра данных в хранилище Feast для их приемки, используя метод `.ingest(...)`. Здесь мы укажем экземпляр `FeatureSet` под названием `taxi_fs` и кадр данных `taxi_df`, из которого будут заполняться признаковые данные.

```
# Загрузить признаковые данные в Feast для этого конкретного набора признаков
client.ingest(taxi_fs, taxi_df)
```

На этом шаге приемки на экран выводится индикатор хода выполнения, показывая, что мы внесли 28 247 строк данных в набор признаков `taxi_rides` внутри хранилища Feast:

```
100%|██████████|28247/28247 [00:02<00:00,
2771.19rows/s] Ingestion complete!

Ingestion statistics:
Success: 28247/28247 rows ingested
```

На этом этапе вызов метода `client.list_feature_sets()` теперь перечислит только что созданный нами набор признаков `taxi_rides` и вернет `[default/taxi_rides]`. Здесь `default` относится к области видимости набора признаков внутри хранилища Feast в текущем проекте. Его можно изменить при инстанцировании набора признаков, чтобы удерживать некоторые наборы признаков в пределах доступа проекта.

С течением времени наборы данных могут измениться, что также приводит к модификации наборов признаков. После создания набора признаков в хранилище Feast можно внести лишь несколько изменений. Например, допускаются следующие изменения:

- добавление новых признаков;
- удаление существующих признаков (обратите внимание, что признаки помечаются на удаление и остаются в реестре, поэтому они не удаляются полностью; это повлияет на способность новых признаков принимать имена ранее удаленных признаков);
- изменение схем признаков;
- изменение источника набора признаков или максимального возраста `max_age` примеров набора признаков.

Следующие изменения не допускаются:

- изменение в имени набора признаков;
- изменения в сущностях;
- изменение имен существующих признаков.

Получение данных из хранилища Feast

После того как набор признаков будет заполнен признаками из источника, мы можем извлечь исторические или онлайновые признаки. Пользователи и промышленные системы извлекают признаковые данные через слой доступа к данным, обрабатывающий запросы на получение признаков хранилища Feast. Поскольку хранилище Feast поддерживает как офлайновый, так и онлайновый типы хранилищ, общепринято иметь развернутые версии Feast для обоих типов (рис. 6.15). Одни и те же признаковые данные содержатся в двух хранилищах признаков, обеспечивая согласованность между тренировкой и обслуживанием.

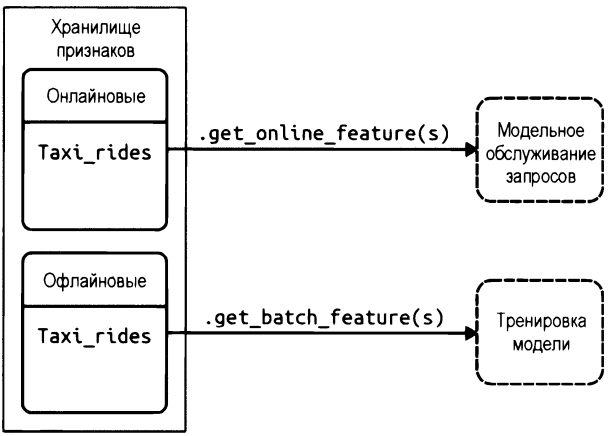


Рис. 6.15. Признаковые данные могут быть получены либо в офлайновом режиме, используя исторические признаки для тренировки модели, либо в онлайновом режиме для обработки запросов

Доступ к этим развернутым вариантам осуществляется через отдельного онлайнового и пакетного клиента:

```
_feast_online_client = Client(serving_url='localhost:6566')
_feast_batch_client = Client(serving_url='localhost:6567',
                             core_url='localhost:6565')
```

Пакетное обслуживание. В случае тренировки модели извлечение исторических признаков поддерживается в BigQuery и доступно с помощью метода `.get_batch_features(...)` с клиентом пакетного обслуживания. В этом случае мы предоставляем в хранилище Feast кадр данных pandas, содержащий сущности и временны́е метки, с которыми будут соединены признаковые данные. Это позволяет хранилищу Feast продуцировать правильный на определенный момент времени набор данных, основываясь на запрошенных признаках:

```
# Создать сущность df из всех сущностей и меток времени
entity_df = pd.DataFrame(
    {
        "datetime": taxi_df.datetime,
        "taxi_id": taxi_df.taxi_id,
    }
)
```

В целях получения исторических признаков доступ к признакам в наборе осуществляется по имени набора признаков и имени признака, разделенными двоеточием, например `taxi_rides:pickup_lat`:

```
    FS_NAME = taxi_rides
model_features = ['pickup_lat',
                  'pickup_lon',
                  'dropoff_lat',
                  'dropoff_lon',
                  'num_pass',
                  'euclid_dist']
    label = 'fare_amt'

    features = model_features + [label]

# Извлечь тренировочный набор данных из Feast
dataset = _feast_batch_client.get_batch_features(
    feature_refs=[FS_NAME + ":" + feature for feature in features],
    entity_rows=entity_df).to_dataframe()
```

Набор данных из кадра данных теперь содержит все признаки и метку для нашей модели, извлеченные непосредственно из хранилища признаков.

Обработка онлайн. В случае обработки запросов хранилище Feast хранит только самые последние значения сущностей, в отличие от исторической обработки запросов, при которой хранятся все исторические значения. В основу онлайновой обработки с использованием хранилища Feast заложена очень низкая задержка, и Feast предоставляет gRPC API, поддерживаемый в Redis. В целях получения онлайновых признаков, например при генерировании онлайновых предсказаний с помощью натренированной модели, мы используем метод `.get_online_features(...)`, указывая признаки, которые хотим получить, и сущность:

```
# Извлечь онлайновые признаки для одного taxi_id
online_features = _feast_online_client.get_online_features(
    feature_refs=["taxi_rides:pickup_lat",
                  "taxi_rides:pickup_lon",
                  "taxi_rides:dropoff_lat",
                  "taxi_rides:dropoff_lon",
                  "taxi_rides:num_pass",
                  "taxi_rides:euclid_dist"],
    entity_rows=[
        GetOnlineFeaturesRequest.EntityRow(
            fields={
                "taxi_id": Value(
                    int64_val=5)
            }
        )
    ]
)
```

Приведенный выше фрагмент кода сохраняет онлайновые признаки `online_features` в виде списка пар, в котором каждая позиция содержит самые последние значения признаков для предоставленной сущности, здесь `taxi_id = 5`:

```
field_values {
  fields {
    key: "taxi_id"
    value {
      int64_val: 5
    }
  }
  fields {
    key: "taxi_rides:dropoff_lat"
    value {
      double_val: 40.78923797607422
    }
  }
  fields {
    key: "taxi_rides:dropoff_lon"
    value {
      double_val: -73.96871948242188
    }
...
```

В целях генерирования онлайнового предсказания для этого примера мы передаем значения полей объекта, возвращаемого в `online_features`, в метод `model.predict` в виде кадра данных `predict_df` библиотеки pandas:

```
predict_df = pd.DataFrame.from_dict(online_features_dict)
model.predict(predict_df)
```

Почему это работает

Хранилища признаков работают, потому что они устраняют сцепленность между генерированием признаков и их использованием, позволяя процессам разработки и создания признаков происходить независимо от использования признаков во время разработки модели. Когда признаки добавляются в хранилище признаков, они сразу же становятся доступными как для тренировки модели, так и для обслуживания запросов, и хранятся в одном месте. За счет этого обеспечивается согласованность между тренировкой модели и обслуживанием запросов.

Например, модель, эксплуатируемая в качестве клиент-ориентированного приложения, может получать от клиента только 10 входных значений, но они перед отправкой в модель могут нуждаться в преобразовании в ряд других признаков посредством методики генерирования признаков. Поддержание этих сгенерированных признаков обеспечивается внутри хранилища признаков. Очень важно, чтобы конвейер извлечения признаков во время разработки был таким же, что и при обслуживании запросов. Хранилище данных предоставляет такую согласованность (рис. 6.16).

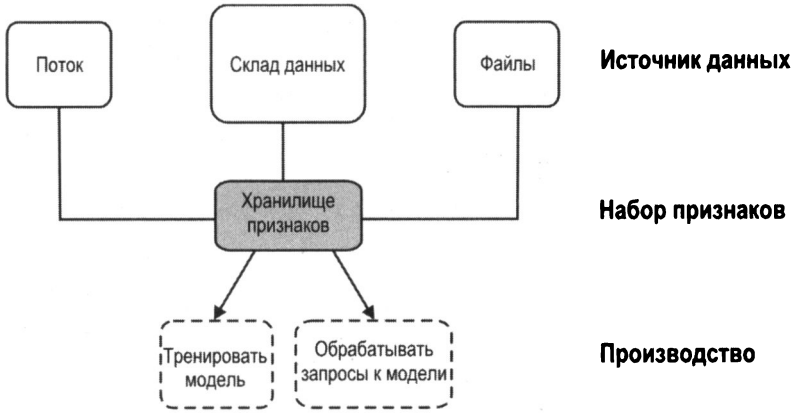


Рис. 6.16. Хранилище признаков обеспечивает согласованность конвейеров генерирования признаков между тренировкой модели и обработкой запросов. См. также **https://docs.feast.dev/**

Хранилище Feast выполняет это, используя Beam на стороне сервера (бэкенда) для конвейеров приемки признаков, которые записывают значения признаков в наборы, и применяет Redis и BigQuery (соответственно) для извлечения онлайновых и офлайновых признаков (рис. 6.17)[60]. Как и в любом хранилище признаков, конвейер приемки также занимается частичными отказами или ситуациями гонки, которые

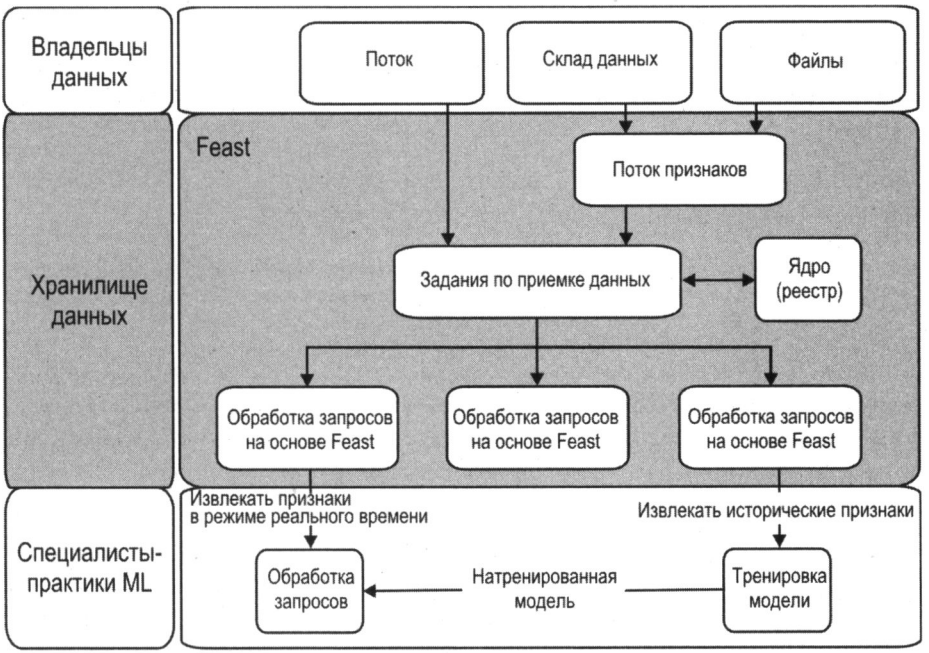


Рис. 6.17. Хранилище Feast использует Beam на стороне сервера для приемки признаков и Redis и bigQuery для извлечения онлайновых и офлайновых признаков

[60] См. пост в блоге Годжека (Gojek) "Feast: наведение моста между моделями ML и данными" (Feast: Bridging ML Models and Data, **https://oreil.ly/YVta5**).

могут приводить к тому, что некоторые данные окажутся не в том хранилище, в каком надо.

Разные системы могут производить данные с различной скоростью, и хранилище признаков является достаточно гибким в манипулировании этими разными ритмами как во время импортирования, так и во время извлечения (рис. 6.18). Например, данные с датчиков могут поступать в режиме реального времени, каждую секунду, либо может существовать ежемесячный файл, который генерируется из внешней системы, сообщающей сводку транзакций за последний месяц. Все это необходимо обрабатывать и принимать в хранилище признаков. Точно так же могут существовать разные временны́е горизонты и для извлечения данных из хранилища признаков. Например, клиент-ориентированное онлайновое приложение может работать с очень низкой задержкой, используя признаки с точностью до секунды, тогда как во время тренировки модели признаки извлекаются в офлайновом режиме как крупный пакет, но с более высокой задержкой.

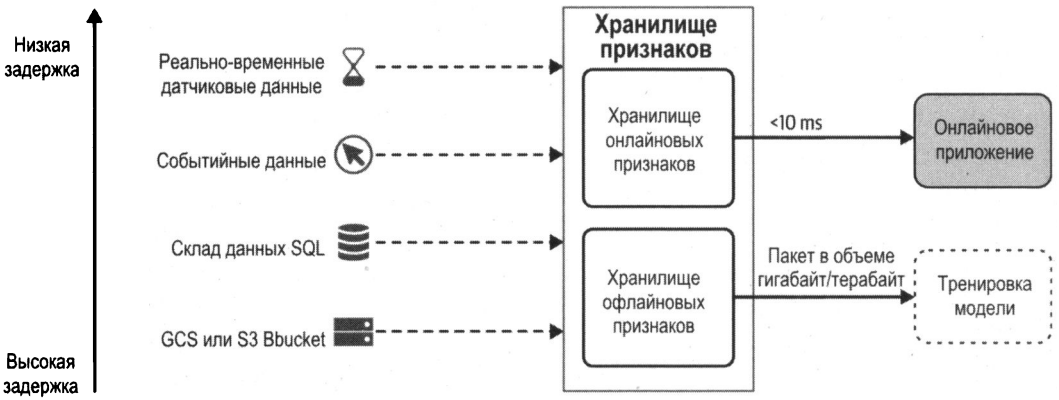


Рис. 6.18. Паттерн "Хранилище признаков" справляется с требованиями к высокой масштабируемости данных для крупных пакетов во время тренировки и чрезвычайно низкой задержкой для обработки

Нет ни одной базы данных, которая могла бы справляться и с масштабированием теоретически до терабайт данных, и с чрезвычайно низкой задержкой порядка миллисекунд. Хранилище признаков достигает этого при помощи отдельных хранилищ онлайновых и офлайновых признаков и обеспечивает, чтобы признаки обрабатывались согласованно в обоих сценариях.

Наконец, хранилище признаков действует как инструмент управления версиями признаковых данных, позволяя применять те же практические приемы CI/CD разработки кода и моделей к процессу инженерии признаков. Это означает, что вместо генерирования признаков с нуля новые ML-проекты стартуют с процесса отбора признаков из каталога, что позволяет организациям достигать эффекта экономии за счет масштаба — по мере создания и добавления новых признаков в хранилище признаков разработка новых моделей, в которых эти признаки используются многократно, становится проще и быстрее.

Компромиссы и альтернативы

Рассмотренный нами инструмент Feast построен на основе Google BigQuery, Redis и Apache Beam. Однако есть хранилища признаков, которые опираются на другие инструменты и технологические стеки. И хотя хранилище признаков является рекомендуемым подходом к управлению признаками в требуемом масштабе, библиотека tf.transform предоставляет альтернативное техническое решение, которое избавляет от проблемы асимметрии между тренировкой и обслуживанием, но не многоразовости использования признаков. Кроме того, мы еще подробно не описали несколько альтернативных вариантов использования хранилищ признаков, например способы манипулирования хранилищем данными из разных источников и данными, поступающими с разными ритмами.

Альтернативные реализации

Многие крупные технологические компании, такие как Uber, LinkedIn, Airbnb, Netflix и Comcast, имеют собственные версии хранилища признаков, хотя архитектура и инструменты различаются. Хранилище Michelangelo Palette (Палитра Микеланджело) компании Uber построено вокруг Spark/Scala с использованием Hive для создания офлайновых признаков и Cassandra для онлайновых признаков. Хранилище Hopsworks представляет собой еще одно хранилище признаков с открытым исходным кодом как альтернатива хранилищу Feast и построено вокруг кадров данных с использованием Spark и pandas с Hive для доступа к офлайновым признакам и MySQL Cluster для доступа к онлайновым признакам. В Airbnb было построено собственное хранилище признаков в рамках их ML-платформы под названием Zipline. В нем Spark и Flink используются для заданий по генерированию признаков и Hive — для хранения признаков.

Какой бы технологический стек ни использовался, первостепенные компоненты хранилища признаков остаются одинаковыми:

- инструмент для быстрой обработки крупных заданий по генерированию признаков, такой как Spark, Flink или Beam;
- компонент хранилища для размещения созданных наборов признаков, такой как Hive, облачное хранилище (Amazon S3, Google Cloud Storage), BigQuery, Redis, BigTable и/или Cassandra. Комбинация, которая используется в Feast (BigQuery и Redis), оптимизирована под извлечение офлайновых и онлайновых (с низкой задержкой) признаков;
- слой метаданных для ведения информации о версии признаков, документации и реестра признаков с целью упрощения обнаружения и совместного использования наборов признаков;
- API для признаков в хранилища и извлечения признаков из них.

Паттерн "Преобразователь"

Если код генерирования признаков не совпадает во время тренировки и предсказательного вывода, то существует риск того, что два источника кода не будут со-

гласованы. Это приводит к асимметрии между тренировокой и обработкой, и модельные предсказания могут стать ненадежными, поскольку признаки могут быть разными. Хранилища признаков решают эту проблему, имея задания по генерированию признаков, которые пишут признаковые данные как в онлайновую, так и в офлайновую базу данных. И хотя хранилище признаков само по себе не выполняет преобразования признаков, оно предоставляет подход к отделению вышестоящих шагов по генерированию признаков от обработки запросов и обеспечению правильности на определенный момент времени.

Рассмотренный в этой главе паттерн "Преобразователь" также обеспечивает подход к поддержанию преобразований признаков в раздельной и воспроизводимой форме. Например, библиотека tf.transform может применяться для предобработки данных с использованием одинакового исходного кода во время тренировки модели и во время обработки запросов в производстве, что устраняет асимметрию между тренировкой и обработкой. За счет этого обеспечивается согласованность конвейеров генерирования признаков времени тренировки и времени обработки.

Однако хранилище признаков обеспечивает дополнительное преимущество многоразового использования признаков, которого нет в библиотеке tf.transform. Хотя конвейеры tf.transform обеспечивают воспроизводимость, признаки создаются и разрабатываются только для этой модели и не могут легко использоваться совместно или многоразово другими моделями и конвейерами.

С другой стороны, библиотека tf.transform уделяет особое внимание обеспечению того, чтобы создание признаков во время обработки запросов выполнялось с аппаратным ускорением, поскольку оно является частью графа обслуживания. Сегодня хранилища признаков, как правило, такой возможности не предлагают.

ПАТТЕРН 27. Управление версиями

В паттерне "Управление версиями" (Model Versioning) обратная совместимость достигается путем развертывания измененной модели как микросервиса с REST API. Указанный паттерн является необходимым условием для многих других, обсуждаемых в этой главе.

Постановка задачи

Как мы убедились на примере *дрейфа данных* (представленного в *главе 1*), модели со временем могут приобретать черты застарелости и нуждаться в регулярном обновлении с целью обеспечения отражения изменяющихся целей организации и среды, ассоциированной с их тренировочными данными. Развертывание обновлений моделей неизбежно будет влиять на то, как модели ведут себя на новых данных, что создает свою трудность — нам нужен подход для поддержания промышленных моделей в актуальном состоянии, обеспечивая при этом обратную совместимость для существующих пользователей моделей.

Обновления существующей модели могут включать изменение ее архитектуры с целью повышения точности или перетренировку модели на более недавних дан-

ных с целью устранения дрейфа. Хотя эти типы изменений, скорее всего, не потребуют другого формата данных на выходе из модели, они будут влиять на результаты предсказания, которые пользователи получают из модели. В качестве примера представим, что мы строим модель, которая предсказывает жанр книги по ее описанию и использует предсказанные жанры для рекомендаций пользователям. Мы натренировали первоначальную модель на наборе данных о старых классических книгах, но теперь имеем доступ к новым данным о тысячах более поздних книг, которые можно использовать для тренировки. Тренировка на этом обновленном наборе данных повышает совокупную точность нашей модели, но немного снижает точность на старых "классических" книгах. В целях урегулирования этого случая нам понадобится техническое решение, которое позволит пользователям выбирать более старую версию модели, если они захотят.

С другой стороны, у конечных пользователей модели, возможно, появится потребность иметь больше информации о том, как модель приходит к тому или иному предсказанию. В случае медицинского приложения врач, вероятно, захочет увидеть на рентгеновском снимке участки, которые побудили модель предсказать наличие болезни, не полагаясь только на предсказанную метку. В этом случае ответ из развернутой модели должен быть обновлен, чтобы можно было включить эти выделенные участки. Указанный процесс называется *обеспечением объяснимости* и обсуждается в *главе 7*.

При развертывании обновления модели нам также, вероятно, понадобится какой-то способ отслеживать результативность ее работы в производстве и проводить ее сравнение с предыдущими итерациями. Нам также может потребоваться способ тестирования новой модели только с подмножеством пользователей. Задачи мониторинга результативности и раздельного тестирования, наряду с другими возможными изменениями модели, будет трудно решить, заменяя модель всякий раз, когда мы делаем обновления. Такой подход будет нарушать работу приложений, которые опираются на соответствие результата работы модели тому или иному формату. Для урегулирования этого случая нам хорошо бы иметь техническое решение, которое позволит регулярно обновлять модель, не нарушая работу существующих пользователей.

Решение

В целях плавного манипулирования обновлением модели следует разворачивать несколько ее версий с разными REST API. За счет этого обеспечивается обратная совместимость — поддерживая несколько версий модели, развернутых в определенный момент времени, те пользователи, которые опираются на более старые версии, смогут по-прежнему пользоваться службой. Управление версиями также позволяет осуществлять глубокий мониторинг результативности и аналитическое отслеживание разных версий. Мы можем сравнивать статистику точности и использования и задействовать ее для определения времени, когда следует перевести ту или иную версию в офлайновый режим. Если у нас есть обновление модели, которое мы хотим протестировать только с малым подмножеством пользователей, то

паттерн "Управление версиями" позволяет это делать, например для A/B-тестирования.

Вдобавок, при управлении версиями каждая развернутая версия модели является микросервисом, а следовательно, устраняет сцепленность изменений в модели с внешним интерфейсом приложения. В целях добавления поддержки новой версии разработчики приложений из нашей команды будут лишь изменять API, который указывает на модель. Разумеется, если новая версия модели вносит изменения в формат ответа модели, то нам понадобится модифицировать и само приложение, чтобы это урегулировать, но модель и исходный код приложения остаются попрежнему разделенными. И поэтому исследователи данных или инженеры ML смогут самостоятельно развертывать и тестировать новую версию модели, не беспокоясь о том, что работа промышленного приложения будет нарушена.

Типы пользователей модели

Когда мы говорим о "конечных пользователях" модели, мы имеем в виду две разные группы людей. Если мы делаем API модели доступной разработчикам приложений вне нашей организации, то этих разработчиков можно рассматривать как один тип пользователей модели. Они создают приложения, которые опираются на нашу модель, для предоставления предсказаний другим людям. Для этих пользователей выгода от обратной совместимости, которая проистекает из управления версиями модели, является наиболее важной. Если формат ответа нашей модели изменится, то разработчики приложений, возможно, захотят пользоваться более старой версией модели до тех пор, пока не обновят свой прикладной код для поддержания самого последнего формата ответа.

Другая группа конечных пользователей относится к тем, кто пользуется приложением, вызывающим нашу развернутую модель. Это может быть врач, который опирается на нашу модель в предсказании болезни по снимку, некто, кто использует наше приложение для рекомендаций книг, структурная единица нашей организации, которая анализирует результат из построенной нами модели предсказания выручки, и др. Эта группа пользователей с меньшей вероятностью столкнется с проблемами обратной совместимости, но, вполне возможно, захочет выбирать время начала использования новой функциональности в нашем приложении. Кроме того, если мы можем разбить пользователей на отличимые группы (т. е. опираясь на использование ими приложения), то можем обслуживать каждую группу разными версиями моделей, основываясь на их предпочтениях.

Управление версиями модели с помощью сервиса

В целях демонстрации управления версиями построим модель, которая предсказывает задержки рейсов, и развернем эту модель в Cloud AI Platform Prediction[61]. Поскольку мы рассматривали формат SavedModel платформы TensorFlow в предыдущих главах, здесь будем использовать модель XGBoost.

---

[61] См. **https://oreil.ly/-GAVQ**.

Натренировав модель, мы сможем ее экспортировать, чтобы подготовить к обработке запросов:

```
model.save_model('model.bst')
```

Для развертывания этой модели на платформе AI Platform нам понадобится создать версию модели, которая будет указывать на эту модель в формате model.bst в корзине облачного хранилища Cloud Storage.

На платформе AI Platform с модельным ресурсом может быть связано много версий. В целях создания новой версии мы выполним следующую ниже команду в терминале, используя интерфейс командой строки gcloud:

```
gcloud ai-platform versions create 'v1' \
  --model 'flight_delay_prediction' \
  --origin gs://your-gcs-bucket \
  --runtime-version=1.15 \
  --framework 'XGBOOST' \
  --python-version=3.7
```

После развертывания этой модели она будет доступна по URL-адресу /models/flight_delay_predictions/versions/v1 https-протокола, привязанному к нашему проекту. Поскольку эта версия является единственной, которую мы развернули до сих пор, она используется по умолчанию. Это означает, что если мы не указываем версию в запросе API, то сервис предсказания будет использовать v1. Теперь мы можем делать предсказания с помощью нашей развернутой модели, отправляя ей примеры в том формате, который модель ожидает, — в данном случае 110-элементный массив из кодов аэропортов, кодированных с использованием фиктивных переменных (полный исходный код см. в блокноте[62] в репозитории на GitHub). Модель возвращает сигмоидный результат, вещественное значение между 0 и 1, указывающее на вероятность того, что данный рейс был задержан более чем на 30 минут.

В целях отправки предсказательного запроса в развернутую модель мы используем следующую ниже команду gcloud, где input.json — это файл с примерами, разделенными символом новой строки, готовыми к отправке для предсказания:

```
gcloud ai-platform predict --model 'flight_delay_prediction'
  --version 'v1'
  --json-request 'input.json'
```

Если мы отправим пять примеров для предсказания, то обратно получим пятиэлементный массив, соответствующий сигмоидному результату для каждого тестового примера, как показано ниже:

```
[0.019, 0.998, 0.213, 0.002, 0.004]
```

Теперь, когда у нас есть рабочая модель, давайте представим, что наш коллектив исследователей данных решает поменять модель с XGBoost на TensorFlow, поскольку это приведет к повышению точности и даст им доступ к дополнительным

[62] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/06_reproducibility/model_versioning.ipynb**.

инструментам в экосистеме TensorFlow. Модель имеет тот же формат данных на входе и выходе, но ее архитектура и формат экспортируемых активов изменились. Вместо файла в формате BST наша модель теперь находится в формате SavedModel платформы TensorFlow. В идеале мы можем держать наши базовые активы модели отдельно от интерфейса приложения — это позволит разработчикам приложений сосредоточиваться не на изменении формата модели, который не повлияет на характер взаимодействия конечных пользователей с моделью, а на функциональности самого приложения. Именно здесь поможет управление версиями модели. Мы развернем нашу модель TensorFlow в качестве второй версии под тем же ресурсом модели flight_delay_prediction. Конечные пользователи смогут перейти на новую версию для получения повышенной результативности, просто изменив имя версии в API.

В целях развертывания второй версии мы выполним экспорт модели и скопируем ее в новый подраздел в корзине, которую мы использовали ранее. Мы можем применить ту же команду развертывания, что и выше, заменив имя версии на v2 и указав местоположение облачного хранилища новой модели. Как показано на рис. 6.19, теперь в нашей облачной консоли мы можем увидеть обе развернутые версии.

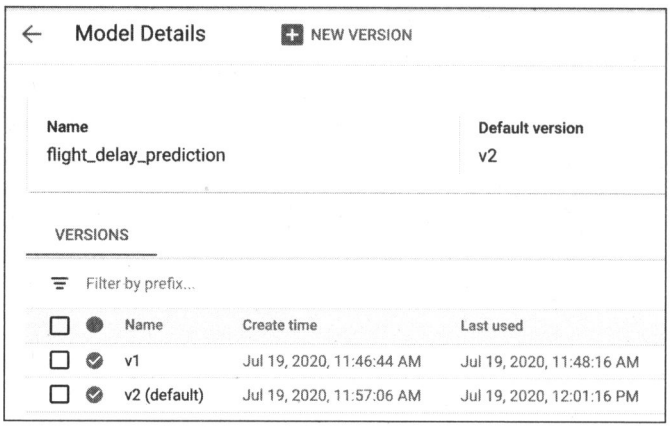


Рис. 6.19. Панель мониторинга для управления моделями и версиями в консоли Cloud AI Platform

Обратите внимание, что мы также установили v2 в качестве новой версии, которая используется по умолчанию, поэтому если пользователи не укажут версию, то получат ответ от v2. Поскольку формат данных на входе нашей модели и выходе из нее одинаков, клиенты могут обновлять свои приложения, не беспокоясь о нарушающих работу изменениях.

И Azure, и AWS имеют аналогичные службы управления версиями моделей. В Azure модельное развертывание и версионирование доступны с помощью служб Azure Machine Learning[63]. В AWS эти службы доступны в SageMaker[64].

[63] См. **https://oreil.ly/Q7NWh**.

[64] См. **https://oreil.ly/r98Ve**.

Инженер ML, развертывающий новую версию модели в качестве API, может захотеть использовать шлюз API, такой как Apigee, который определяет версию модели для вызова. Для этого есть разные причины, в том числе и раздельное тестирование новой версии. В случае раздельного тестирования, возможно, специалист захочет протестировать обновление модели со случайно выбранной группой из 10% пользователей приложения, чтобы проследить за тем, как это влияет на их совокупное взаимодействие с приложением. Шлюз API определяет версию развернутой модели, которую следует вызывать с учетом идентификатора пользователя или IP-адреса.

При развертывании нескольких версий модели платформа AI Platform позволяет осуществлять мониторинг результативности и вести аналитику в разных версиях. Так можно выполнять трассировку ошибок вплоть до определенной версии, вести мониторинг трафика и объединять все это с дополнительными данными, которые мы собираем в нашем приложении.

Управление версиями для манипулирования новыми данными

В дополнение к манипулированию изменениями в самой модели еще одна причина использовать управление версиями вызвана появлением новых тренировочных данных. Если эти новые данные следуют той же схеме, которая использовалась для тренировки изначальной модели, важно отслеживать время, когда данные были захвачены для каждой новой натренированной версии. Один из подходов к такого рода слежению заключается в кодировании диапазона временны́х меток каждого тренировочного набора данных в имени версии модели. Например, если последняя версия модели тренируется на данных 2019 года, мы могли бы назвать версию v20190101_20191231.

Этот подход можно использовать в сочетании с паттерном "Непрерывное оценивание модели" (который обсуждается в *главе 5*), чтобы определять время, когда следует переводить старые версии модели в офлайновый режим или насколько старыми должны быть тренировочные данные. Непрерывное оценивание, вероятно, поможет нам установить, что наша модель показывает наилучшую результативность во время тренировки на данных за последние два года. А это, возможно, даст нам подсказку в отношении версий, которые мы решим удалить, и того, сколько данных использовать во время тренировки новых версий.

Компромиссы и альтернативы

Хотя мы и рекомендуем использовать паттерн "Управление версиями" вместо поддержания одной версии модели, в реализации описанного выше технического решения есть несколько альтернатив. Здесь мы рассмотрим другие бессерверные инструменты с открытым исходным кодом для указанного паттерна и подход к созданию нескольких функций обслуживания. Мы также обсудим вопрос о том, когда вместо версии следует создавать совершенно новый ресурс модели.

Другие бессерверные инструменты управления версиями

Мы использовали инструмент, специально разработанный для управления версиями моделей, но смогли бы добиться аналогичных результатов с другими бессерверными решениями. Под капотом каждая версия модели представляет собой функцию без поддержки состояния с заданным форматом данных на входе и выходе REST API. Поэтому мы могли бы использовать такой сервис, как Cloud Run[65], например, для сборки и развертывания каждой версии в отдельном контейнере. Каждый контейнер имеет уникальный URL-адрес и может вызваться запросом API. Этот подход дает нам бóльшую гибкость в конфигурировании среды развернутой модели, позволяя добавлять такую функциональность, как предобработка на стороне сервера для входных переменных модели. В приведенном выше примере с авиарейсами мы вполне можем отказаться от того, чтобы требовать от клиентов кодировать категориальные значения с использованием одного активного состояния. Вместо этого мы могли бы позволить клиентам передавать категориальные значения в виде строковых значений и манипулировать предобработкой в нашем контейнере.

Почему мы должны использовать именно специализированный инструмент, такой как AI Platform Prediction, а не более обобщенный бессерверный инструмент? Поскольку инструментальная платформа AI Platform была создана специально для развертывания моделей, она имеет встроенную поддержку развертывания моделей с графическими процессорами, оптимизированными под ML. Она также работает с управлением зависимостями. Когда выше мы развертывали модель XGBoost, нам не пришлось беспокоиться об инсталлировании правильной версии XGBoost или других библиотечных зависимостей.

Инструмент Tensorflow Serving

Вместо облачной платформы Cloud AI Platform или другого облачного бессерверного технического решения для управления версиями моделей мы могли бы использовать инструмент с открытым исходным кодом, такой как TensorFlow Serving[66]. Рекомендуемый подход для реализации своей задачи посредством системы TensorFlow Serving заключается в применении контейнера Docker посредством самого последнего образа Docker, tensorflow/serving[67]. С помощью контейнера Docker можно будет обрабатывать запросы на модельное предсказание, используя любое аппаратное обеспечение по нашему усмотрению, включая графические процессоры. API системы TensorFlow Serving имеет встроенную поддержку управления версиями моделей, следуя аналогичному подходу, описанному в *разд. "Решение"*.

---

[65] См. **https://oreil.ly/KERBV**.

[66] См. **https://oreil.ly/NzDA9**.

[67] См. **https://oreil.ly/G0_Z7**.

В дополнение к TensorFlow Serving существуют и другие варианты инструментов с открытым исходным кодом для обработки запросов, включая Seldon[68] и MLflow[69].

Несколько функций обработки

Еще одной альтернативой развертыванию нескольких версий является определение нескольких функций обработки для одной версии экспортированной модели. В *разд. "Паттерн 16. Функция обслуживания без поддержки состояния" главы 5* даны пояснения по экспортированию натренированной модели в качестве функции без поддержки состояния для обработки запросов. Это особенно полезно, когда данные, подаваемые на вход модели, требуют предобработки с целью преобразования отправленных клиентом данных в ожидаемый моделью формат.

Для урегулирования требований в отношении разных групп конечных пользователей модели мы можем определить несколько функций обработки во время экспортирования модели. Эти функции обработки являются частью версии одной экспортированной модели и имеют один REST API. В TensorFlow функции обслуживания реализуются с использованием модельных сигнатур, которые задают формат данных, ожидаемый моделью на входе и выходе. Мы можем определить несколько функций обслуживания с помощью декоратора `@tf.function` и каждой функции передавать входную сигнатуру.

В прикладном коде, из которого мы вызываем развернутую модель, мы определяем используемую функцию обработки, основываясь на данных, отправленных от клиента. Например, такой запрос, как:

```
{"signature_name": "get_genre", "instances": ...}
```

будет отправлен в экспортированную сигнатуру под названием `get_genre` (получить жанр), тогда как запрос:

```
{"signature_name": "get_genre_with_explanation", "instances": ... }
```

будет отправлен в экспортированную сигнатуру под названием `get_genre_with_explanation` (получить жанр с пояснением).

Таким образом, развертывание нескольких сигнатур может решить проблему обратной совместимости. Однако существует весомая разница — есть только одна модель, и при развертывании этой модели все сигнатуры обновляются одновременно. В нашем изначальном примере, в котором изменение модели состояло в том, что предоставление только одного жанра было заменено предоставлением нескольких жанров, изменилась модельная архитектура. С этим примером подход с несколькими сигнатурами не будет работать, потому что у нас две разные модели. Решение с несколькими сигнатурами также не подходит, когда мы хотим хранить разные версии модели отдельно и со временем отказаться от более старой версии.

---

[68] См. **https://oreil.ly/Cddpi**.

[69] См. **https://mlflow.org/**.

Использовать нескольких сигнатур лучше, чем несколько версий, если вы хотите сохранить *обе* сигнатуры модели в будущем. В сценарии, в котором некоторые клиенты хотят получать лишь наилучший ответ, а другие клиенты — наилучший ответ и объяснение вместе, имеется дополнительная выгода от обновления всех сигнатур новой моделью вместо необходимости обновлять версии всякий раз, когда модель перетренировывается и переразвертывается.

Каковы сценарии, в которых у нас могла бы возникнуть потребность поддерживать обе версии модели? В случае модели классифицирования текста у нас могут быть клиенты, которые будут нуждаться в отправке в модель сырого текста, и другие, которые могут преобразовывать сырой текст в матрицы перед тем, как получать предсказание. Основываясь на данных клиентского запроса, серверная инфраструктура модели может определять тип запроса, который должна применять. Передача матриц с векторным вложением текста в модель обходится дешевле, чем предобработка сырого текста, и, значит, это пример, когда несколько функций обработки могут сократить время обработки на стороне сервера. Стоит также отметить, что мы можем иметь несколько функций *с несколькими* версиями модели, хотя есть риск, что это создаст слишком большую сложность.

Новые модели против новых версий моделей

Иногда трудно определиться с тем, что именно следует создавать: еще одну версию модели либо совершенно новую модель. Мы рекомендуем создавать новую модель при изменении задачи предсказания. Новая задача предсказания обычно приводит к другому формату данных на выходе из модели, и изменение этого формата может приводить к нарушению работы существующих клиентов. Если мы не уверены в том, что именно использовать — новую версию либо модель, следует подумать о том, хотим ли мы, чтобы существующие клиенты обновлялись. Если ответ будет "да", то есть шанс, что мы улучшили модель без изменения задачи предсказания, и создания новой версии будет достаточно. Если же мы изменили модель настолько, что теперь пользователи должны решать, должны ли они обновляться, то мы, скорее всего, захотим создать новую модель.

В целях ознакомления с этой дилеммой на практике давайте вернемся к нашей модели предсказания авиарейсов. Текущая модель дала определение того, что она считает задержкой (опоздание на 30 минут и более), но наши конечные пользователи могут иметь по этому поводу разные мнения. Некоторые считают, что опоздание всего на 15 минут уже считается задержкой, в то время как другие уверены, что рейс задерживается только в том случае, если он опаздывает более чем на час. Давайте представим, что теперь мы хотим, чтобы пользователи могли брать не наше определение задержки, а включать собственное. В этом случае мы используем паттерн "Переформулировка" (*см. главу 3*), чтобы поменять модель на регрессионную. Входной формат для этой модели тот же, но на выходе теперь мы имеем числовое значение, представляющее предсказание задержки.

Характер интерпретации этого модельного ответа пользователями, очевидно, будет отличаться от первой версии. С нашей новейшей регрессионной моделью разработчики приложений, возможно, пожелают показывать предсказанную задержку, когда

пользователи ищут рейсы, заменив нечто вроде "Этот рейс обычно задерживается более чем на 30 минут" из первой версии. В указанном сценарии наилучшим решением является создание новой модели, скажем, под названием flight_model_regression, чтобы отразить изменения. При таком подходе разработчики приложений смогут выбирать, что им использовать, а мы сможем продолжать обновлять результативность каждой модели, развертывая новые версии.

Резюме

В этой главе в центре внимания находились паттерны, которые затрагивают разные аспекты обеспечения воспроизводимости. Начав с паттерна "Преобразователь" (Transform), мы ознакомились с тем, как он используется для обеспечения воспроизводимости зависимостей процесса подготовки данных между конвейером тренировки модели и конвейером обработки запросов. Это достигается путем явного захвата преобразований, применяемых для конвертирования данных на входе в модель, в модельные признаки. Паттерн "Повторяемая разбивка" (Repeatable Splitting) фиксирует подход, на основе которого данные разбиваются между тренировочным, валидационным и тестовым наборами данных для обеспечения того, чтобы используемый в тренировке пример никогда не подвергался оцениванию или тестированию даже по мере роста набора данных.

Паттерн "Мостовая схема" (Bridged Schema) обращается к вопросу обеспечения воспроизводимости, когда тренировочный набор данных представляет собой гибрид новых данных и старых данных с другой схемой. Этот паттерн позволяет согласованно совмещать два набора данных с разными схемами в целях тренировки. Далее мы обсудили паттерн "Оконный предсказательный вывод" (Windowed Inference), который обеспечивает, чтобы при динамическом, зависящем от времени вычислении признаков они могли бы повторяться правильно между тренировкой и обслуживанием. Указанный паттерн особенно полезен, когда ML-модели требуют признаков, вычисляемых из агрегатов над временны́ми окнами.

Паттерн "Конвейер рабочего потока" (Workflow Pipeline) решает задачу создания сквозного воспроизводимого конвейера путем контейнеризации и оркестровки шагов рабочего потока процессов машинного обучения. Затем мы увидели способы применения паттерна "Хранилище признаков" (Feature Store) для решения задачи обеспечения воспроизводимости и многократного использования признаков в разных заданиях по машинному обучению. Наконец, мы рассмотрели паттерн "Управление версиями" (Versioning), в котором обратная совместимость достигается путем развертывания измененной модели как микросервиса с отличным от предыдущей версии REST API.

В следующей главе мы рассмотрим паттерны, которые помогают ответственно подходить к решениям на базе искусственного интеллекта.

ГЛАВА 7

Ответственный искусственный интеллект

До этого момента все наше внимание было сосредоточено на паттернах, предназначенных для оказания помощи командам исследователей данных и инженеров ML в подготовке, разработке, тренировке и масштабировании моделей для промышленного использования. Указанные паттерны в основном касались команд, непосредственно участвующих в процессе разработки ML-модели. После внедрения модели в производство ее влияние распространяется далеко за пределы команд, которые ее создавали. В этой главе мы рассмотрим других участников процесса, имеющих отношение к модели, как внутри организации, так и за ее пределами. Это могут быть руководители, чьи бизнес-цели обусловливают цели создания модели, конечные пользователи модели, аудиторы и регулирующие агентства по соблюдению нормативных требований.

В этой главе мы поговорим о нескольких группах.

- *Разработчики моделей.*

  Исследователи данных и ученые-исследователи ML непосредственно участвуют в разработке моделей ML.

- *Инженеры ML.*

  Члены коллективов по интеграции процессов ML в производство (MLOps) непосредственно участвуют в развертывании моделей ML.

- *Лица, принимающие решения.*

  Решают, следует ли включать ML-модель в свои бизнес-процессы, и принимают участие в оценивании пригодности модели для этих целей.

- *Конечные пользователи систем ML.*

  Используют предсказания из ML-модели. Существует целый ряд типов конечных пользователей моделей: клиенты, сотрудники и люди, выступающие одновременно и теми и другими. В качестве примеров можно привести клиента, получающего от модели рекомендацию по фильму, сотрудника в производственном цехе, использующего модель визуального осмотра для определения неисправности продукта, или практикующего врача, которому модель помогает в диагностике.

- *Регулирующие и надзорные агентства.*

  Люди и организации, которые нуждаются в резюме исполнительного уровня в отношении того, как модель принимает решения с точки зрения соблюдения

нормативных требований. Это могут быть финансовые аудиторы, правительственные учреждения или управленческие коллективы внутри организации.

В этой главе мы рассмотрим паттерны, которые учитывают воздействие модели на отдельных людей и их группы за пределами создающих модель коллектива и организации. Паттерн "Эвристический эталон" (Heuristic Benchmark) обеспечивает способ представления производительности модели в контексте, понятном конечным пользователям и лицам, принимающим решения. Паттерн "Объяснимые предсказания" (Explainable Predictions) обеспечивает подходы к повышению доверия к системам ML, разъясняя сигналы, которые модель использует для предсказания. Паттерн "Призма объективности" (Fairness Lens) нацелен на обеспечение одинаковости поведения моделей среди разных подмножеств пользователей и сценариев предсказания.

Взятые в совокупности паттерны этой главы подпадают под практику ответственного искусственного интеллекта[1]. Предметом исследований этой области активных научных исследований являются наилучшие подходы к встраиванию объективности, интерпретируемости, приватности и безопасности в системы искссственного интеллекта (ИИ). Рекомендуемые практические приемы ответственного ИИ включают применение подхода, который основан на дизайне, нацеленном на человека, с вовлечением разнообразных групп пользователей и наборов вариантов использования на протяжении всей разработки проекта, понимание пределов наборов данных и моделей, а также продолжение мониторинга и обновления ML-систем после развертывания. Паттерны ответственного ИИ не ограничиваются только теми тремя, которые мы обсуждаем в этой главе, — многие паттерны из предыдущих глав (например, "Непрерывное оценивание модели", "Повторяемая разбивка" и "Нейтральный класс", и это лишь некоторые) предоставляют методы для реализации указанных рекомендуемых практических приемов и достижения цели встраивания объективности, интерпретируемости, приватности и безопасности в системы ИИ.

ПАТТЕРН 28. Эвристический эталон

Паттерн "Эвристический эталон" (Heuristic Benchmark) сравнивает ML-модель с простой и понятной эвристикой, чтобы объяснять результативность модели лицам, принимающим решения.

Постановка задачи

Предположим, что агентство по велопрокату хочет использовать ожидаемую продолжительность проката для создания технического решения динамического ценового предложения. После тренировки ML-модели для предсказания продолжительности периода велопроката в агентстве оценивают модель на тестовом наборе данных и определяют, что средняя абсолютная ошибка (mean absolute error, MAE)

[1] См. **https://oreil.ly/MlJkM**.

натренированной ML-модели составляет 1200 секунд. Когда эксперты модели представят ее лицам, принимающим решения, то, скорее всего, услышат вопрос о том, насколько хорошим или плохим является показатель MAE, равный 1200? Мы должны быть готовы отвечать на него всякий раз, когда разрабатываем модель и представляем ее для бизнес-применения. Если мы тренируем модель классифицирования снимков на товарах из каталога, а среднее значение усредненной прецизионности (mean average precision, MAP)[2] составляет 95%, то мы должны быть готовы к вопросу о том, насколько хорошим или плохим является 95%-ный показатель MAP?

Махнуть рукой и сказать, что, дескать, все зависит от задачи, не выход. Конечно же, зависит. Так каким же все-таки должен быть хороший показатель MAE для задачи велопроката в Нью-Йорке? А как быть с Лондоном? А каким будет хороший показатель MAP для задачи классифицирования снимков каталога товаров?

Результативность модели обычно выражается в терминах холодных, конкретных чисел, которые конечным пользователям трудно воспринять. Объяснение формулы MAP, MAE и аналогичных не дает интуитивного понимания, о котором просят лица, принимающие решения.

Решение

Если это новая версия ML-модели или новый способ решения, который разрабатывается для решаемой задачи, нужно просто сравнить результативность модели с текущей версией решения. Теперь довольно легко сказать, что MAE на 30 секунд меньше или MAP на 1% больше. Это работает, даже если текущий производственный процесс не использует ML. Коль скоро эта задача уже выполняется в производстве и собираются метрики оценивания, мы можем сравнивать результативность новой ML-модели с текущей производственной реализацией.

Но что делать, если в настоящее время не существует производственного воплощения модели и мы строим самую первую модель для неосвоенной задачи? В таких случаях решение заключается в создании простого эталона с единственной целью сравнивать его с нашей недавно разработанной моделью ML. Мы называем такой эталон *эвристическим*.

Хороший эвристический эталон должен быть интуитивно простым для понимания и относительно тривиальным для вычисления. Если мы обнаруживаем, что защищаем или отлаживаем алгоритм, который используется в эталоне, мы должны подыскать более простой и понятный. Хорошими примерами эвристического эталона являются константы, эмпирические правила или совокупные статистические величины (например, среднее, медиана или мода). Следует избегать соблазна натренировать даже простую модель машинного обучения, такую как линейная регрессия, на наборе данных и использовать ее как эталон — линейная регрессия, скорее все-

[2] Среднее значение усредненной прецизионности (mean average precision, MAP) вычисляется как среднее значение от средней прецизионности по каждому классу. — *Прим. перев.*

го, будет не совсем понятной, в особенности когда мы начинаем включать в ее состав категориальные переменные, несколько входных переменных либо сгенерированные признаки.

Не следует применять эвристический эталон, если уже существует работающее решение. Вместо этого мы должны сравнивать нашу модель с ним. Существующее решение не нуждается в использовании ML — это просто некая методика, которая в настоящее время используется для решения задачи.

Примеры хороших эвристических эталонов и ситуаций, в которых мы могли бы их использовать, приведены в табл. 7.1. Пример исходного кода для реализаций этих эвристических эталонов находится в репозитории на GitHub[3] этой книги.

Таблица 7.1. *Эвристические эталоны для нескольких отобранных сценариев (исходный код[4] см. в репозитории на GitHub)*

Сценарий	Эвристический эталон	Пример задачи	Реализация для примера задачи
Регрессионная задача, в которой признаки и взаимодействия между признаками не очень хорошо понимаемы на предприятии	Среднее или медианное значения метки по всем тренировочным данным. Выбирать медиану при наличии большого числа выбросов	Интервал времени до момента получения ответа на вопрос Stack Overflow	Предсказывать, что ответ на вопрос всегда будет занимать 2120 секунд. 2120 секунд — это медианное время первого ответа по всему тренировочному набору данных
Двоичная классификационная задача, в которой признаки и взаимодействия между признаками не очень хорошо понимаемы на предприятии	Совокупная доля положительных результатов в тренировочных данных	Будет ли отредактирован принятый ответ в Stack Overflow или нет	Предсказывать 0,36 в качестве выходной вероятности для всех ответов. 0,36 — это доля принятых ответов в целом, которые редактируются
Мультиметочная классификационная задача, в которой признаки и взаимодействия между ними не очень хорошо понимаемы на предприятии	Распределение значения метки по тренировочным данным	Страна, из которой будет дан ответ на вопрос Stack Overflow	Предсказывать 0,03 для Франции, 0,08 для Индии и т. д. Это доли ответов, написанных людьми из Франции, Индии и других стран
Регрессионная задача, в которой есть один-единственный очень важный числовой признак	Линейная регрессия, основанная на том, что интуитивно воспринимается как один-единственный самый важный признак	Предсказывать стоимость проезда в такси с учетом мест посадки и высадки. Расстояние между этими двумя точками является интуитивно понятным ключевым признаком	Стоимость проезда равна $4,64 за километр. $4,64 рассчитывается из тренировочных данных по всем поездкам

[3] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/07_responsible_ai/heuristic_benchmark.ipynb**.

[4] См. **https://oreil.ly/WoESU**.

***Таблица 7.1** (продолжение)*

Сценарий	Эвристический эталон	Пример задачи	Реализация для примера задачи
Регрессионная задача с одним или двумя важными признаками. Признаки могут быть числовыми или категориальными, но должны выступать в качестве широко используемых эвристик	Справочная таблица, в которой строки и столбцы соответствуют ключевым признакам (при необходимости дискретизированным), и предсказание для каждой ячейки является средней меткой в этой ячейке, оцененной по тренировочным данным	Предсказывать продолжительность велопроката. Здесь два ключевых признака — это станция, на которой велосипед берется в прокат, и признак, является ли часом пик для поездок время дачи в прокат	Справочная таблица средней продолжительности проката с каждой станции в зависимости от часа пик и не часа пик
Классификационная задача с одним или двумя важными признаками. Признаки могут быть числовыми или категориальными	Как и выше, за исключением того, что предсказание для каждой ячейки представляет собой распределение меток в этой ячейке. Если цель состоит в предсказании одного-единственного класса, то вычислять моду метки в каждой ячейке	Предсказывать получение ответа на вопрос Stack Overflow в течение одного дня. Наиболее важный признак здесь — первичный тег	Для каждого тега вычислять долю вопросов, на которые были получены ответы в течение одного дня
Регрессионная задача, которая предусматривает предсказание будущего значения временно́го ряда	Постоянство или линейный тренд. Учитывать сезонность. Для получения годовых данных сравнивать их с тем же днем/неделей/кварталом предыдущего года	Предсказывать недельные объемы продаж	Предсказывать, что: • либо продажи на следующей неделе равны s_0, где s_0 — продажи на этой неделе; • либо продажи на следующей неделе равны $s_0 + (s_0 - s_{-1})$, где s_{-1} — продажи на прошлой неделе; • либо продажи на следующей неделе равны s_{-1y}, где s_{-1y} — продажи соответствующей недели прошлого года. Избегать соблазна комбинировать эти три варианта, поскольку значение относительных весов не является интуитивно понятным
Классификационная задача в настоящее время решается экспертами-людьми. Это типичная ситуация для снимковых, видео и текстовых задач; она включает сценарии, в которых рутинное решение задачи экспертами-людьми обходится непомерно дорого	Результативность экспертов-людей	Обнаружение глазных заболеваний по сканам сетчатки	Организовать работу так, чтобы каждый снимок исследовался группой из трех или более врачей. Трактовать решение большинства врачей как правильное и обращаться к процентильному рангу ML-модели среди экспертов-людей

Таблица 7.1 (окончание)

Сценарий	Эвристический эталон	Пример задачи	Реализация для примера задачи
Превентивное или предсказательное техническое обслуживание	Выполнять техническое обслуживание по фиксированному плану	Превентивное техническое обслуживание автомобиля	Привозить автомобили на техобслуживание раз в 3 месяца. Три месяца — это медианное время до отказа автомобилей с даты последнего обслуживания
Обнаружение аномалий	Значение 99-го процентиля, полученное из тренировочного набора данных	Определять DoS-атаку (отказ в обслуживании) на основе сетевого трафика	Находить в исторических данных 99-й процентиль числа запросов в минуту. Если за какой-либо одноминутный период число запросов превысит это число, отмечать его как DoS-атаку
Рекомендательная модель	Рекомендовать самую популярную позицию в категории последней покупки клиента	Рекомендовать пользователям кинофильмы	Если пользователь только что посмотрел научно-фантастический фильм "Начало" и он ему понравился, порекомендовать зрителю фильм "Икар" (самый популярный научно-фантастический фильм, который он еще не смотрел)

Во многих приведенных в табл. 7.1 сценариях упоминаются "важные признаки". Они являются важными в том смысле, что широко приняты на предприятии как имеющие хорошо понятное воздействие на задачу предсказания. В частности, они не являются признаками, установленными с помощью методов определения важности признаков на вашем тренировочном наборе данных. Например, в таксомоторной индустрии принято считать, что наиболее важным фактором, определяющим стоимость проезда в такси, является расстояние, и более длительные поездки стоят дороже. Именно это, а не результат исследования важности признаков, делает расстояние важным признаком.

Компромиссы и альтернативы

Мы часто обнаруживаем, что эвристический эталон полезен не только для объяснения результативности модели. В некоторых случаях эвристический эталон может потребовать специального сбора данных. Иногда эвристический эталон может оказаться недостаточным, потому что само сравнение нуждается в контексте.

Проверка разработки

Нередко эвристический эталон оказывается полезным не только для объяснения результативности ML-моделей. Во время разработки он также помогает в диагностике проблем с конкретной моделью.

Например, предположим, что мы создаем модель для предсказания продолжительности проката, и наш эталон — это справочная таблица средней продолжительно-

сти проката с учетом названия станции и признака, является ли время сдачи в прокат часом пик для поездок:

```
CREATE TEMPORARY FUNCTION is_peak_hour(start_date TIMESTAMP) AS
    EXTRACT(DAYOFWEEK FROM start_date) BETWEEN 2 AND 6 -- рабочий день
    AND (
       EXTRACT(HOUR FROM start_date) BETWEEN 6 AND 10
       OR
       EXTRACT(HOUR FROM start_date) BETWEEN 15 AND 18)
;

SELECT
   start_station_name,
    is_peak_hour(start_date) AS is_peak,
    AVG(duration) AS predicted_duration,
FROM `bigquery-public-data.london_bicycles.cycle_hire`
GROUP BY 1, 2
```

По мере разработки нашей ML-модели полезно сравнивать ее результативность с этим эталоном. Для этого мы будем оценивать результативность модели на разных стратификациях оценочного набора данных. Здесь оценочный набор данных будет стратифицирован `по start_station_name` (названию станции начала проката) и `is_peak` (является ли время часом пик). Поступая так, мы можем легко диагностировать факт чрезмерного акцента модели на занятых популярных станциях и игнорирования нечастых станций в тренировочных данных. Если это происходит, мы можем поэкспериментировать с увеличением сложности модели или балансировкой набора данных для придания избыточного веса менее популярным станциям.

Эксперты-люди

Общая рекомендация для классификационных задач, таких как диагностика глазных заболеваний, в которых оценивание выполняется экспертами-людьми, — привлекать к работе группу таких экспертов в качестве эталона. Когда каждый снимок исследуют три или более врача, можно определять степень, в которой врачи-люди допускают ошибки, и сравнивать частоту ошибок модели с частотой ошибок экспертов-людей. В случае таких задач классифицирования снимков этот подход является естественным продолжением фазы разметки, потому что метки для глазных заболеваний создаются с помощью разметки, выполняемой людьми.

Иногда полезно использовать экспертов-людей, даже если у нас есть эмпирическое наблюдение. Например, во время разработки модели для предсказания стоимости ремонта автомобиля после аварии мы можем обратиться к историческим данным и найти фактическую стоимость ремонта. Для решения этой задачи мы не будем использовать экспертов-людей, потому что эмпирическое наблюдение доступно непосредственно из исторического набора данных. Однако для целей донесения эталона бывает полезно, чтобы страховые агенты оценивали автомобили на предмет величины ущерба, и сравнивать оценки нашей модели с оценками агентов.

Использование экспертов-людей не должно ограничиваться неструктурированными данными, как в случае определения глазных заболеваний или оценивании величины ущерба. Например, если мы создаем модель, чтобы предсказывать возможность рефинансирования кредита в течение года, то данные будут табличными, а эмпирическое наблюдение будет иметься в исторических данных. Однако даже в этом случае мы могли бы попросить экспертов-людей определить кредиты, которые будут рефинансированы, для понимания того, насколько часто кредитные агенты в этой области правы.

Величина полезности

Даже если у нас есть операционная модель или отличная эвристика для сравнения, нам все равно придется объяснять последствия улучшения, которое наша модель предлагает. Довести информацию о том, что MAE на 30 секунд меньше или что MAP на 1% больше, бывает недостаточно. Следующий вопрос вполне может звучать так: является ли улучшение на 1% хорошим? Стоит ли запускать ML-модель в промышленное использование в противовес простому эвристическому правилу?

По возможности следует перевести улучшение результативности модели в величину полезности модели. Это значение может быть денежным, но оно также может соответствовать другим мерам полезности, таким как более качественные результаты поиска, более раннее обнаружение заболеваний или меньший объем отходов в результате повышения эффективности производства. Применение указанной величины полезности становится целесообразным при принятии решения о развертывании данной модели, поскольку развертывание или изменение модели всегда сопряжено с некоторыми затратами с точки зрения надежности и бюджета ошибок. Например, если модель классифицирования снимков используется для предварительного заполнения формы заказа, то мы можем рассчитывать, что улучшение на 1% приведет к уменьшению числа заброшенных заказов на 20 штук в день и, следовательно, будет стоить определенной суммы денег. Если оно превысит порог, установленный нашим коллективом инженеров по обеспечению надежности вебсайта, то мы бы развернули модель.

В нашей задача о велопрокате можно было бы измерить последствия от использования этой модели для предприятия. Например, мы могли бы рассчитать увеличение доступности велосипедов или увеличение прибыли, опираясь на использование модели в техническом решении по динамическому ценообразованию.

ПАТТЕРН 29. Объяснимые предсказания

Паттерн "Объяснимые предсказания" (Explainable Predictions) повышает доверие пользователей к ML-системам, позволяя понять, как и почему модели делают те или иные предсказания. В отличие от таких моделей, как деревья решений, которые поддаются интерпретации по своему дизайну, объяснение архитектуры глубоких нейронных сетей вызывает затруднения. Полезно иметь возможность интерпретировать предсказания всех моделей, чтобы понимать комбинации признаков, влияющих на поведение каждой отдельно взятой модели.

Постановка задачи

Во время оценивания модели машинного обучения, чтобы определить ее готовность к промышленному использованию, такие показатели, как точность, прецизионность, полнота и корень из среднеквадратической ошибки, рассказывают только одну часть истории. Они предоставляют данные о степени правильности предсказания модели относительно значений эмпирических наблюдений в тестовом наборе, но не дают представления *о причине*, по которой модель пришла к таким предсказаниям. Во многих ML-сценариях пользователи могут сомневаться в надежности предсказаний модели.

Давайте рассмотрим модель[5], которая предсказывает тяжесть диабетической ретинопатии (ДР — diabetic retinopathy, DR) по снимку сетчатки[6]. Модель возвращает результат на основе функции мягкого максимума, показывающий вероятность того, что отдельный снимок принадлежит к одной из 5 категорий, обозначающих степень тяжести заболевания на снимке — в диапазоне от 1 (отсутствие ДР) до 5 (пролиферативная ДР, наихудшая форма). Представим, что для данного снимка модель возвращает 95%-ную уверенность в том, что снимок содержит пролиферативную ДР. На первый взгляд такой результат может гарантировать высокую уверенность и иметь точность, но если профессиональный медик будет полагаться исключительно на результаты этой модели в постановке диагноза пациенту, то он не получит никакого представления о том, как модель пришла к такому предсказанию. Возможно, модель определила на снимке правильные участки, которые указывают на ДР, но есть и шанс, что предсказание модели основывается на пикселах снимка, которые не отражают никаких признаков заболевания. Например, вполне возможно, что некоторые снимки в наборе данных содержат заметки врача или аннотации. При генерировании предсказания модель могла неправильно использовать аннотации на снимке вместо пораженных участков[7]. В текущей форме модель не имеет способа приписывать предсказание участкам снимка, что создает препятствие в укреплении доверия к модели со стороны врача.

Медицинская визуализация — это только один пример: существует много отраслей, сценариев и типов моделей, в которых отсутствие понимания процесса принятия решений в модели может подрывать к ней доверие со стороны ее пользователей. Если ML-модель используется для предсказания кредитного балла человека или другой метрики "финансового здоровья", то люди, скорее всего, захотят знать причину, почему они получили тот или иной балл. Была ли это просрочка платежа? Слишком много кредитов? Короткая кредитная история? Возможно, модель в сво-

---

[5] См. **https://oreil.ly/5W-2n**.

[6] Диабетическая ретинопатия (diabetic retinopathy, DR) — это заболевание глаз, поражающее миллионы людей по всему миру. Оно может приводить к слепоте, но если его выявлять на ранней стадии, то оно успешно лечится. Для того чтобы узнать больше и получить набор данных, обратитесь сюда: **https://oreil.ly/ix21h**.

[7] В исследовании, которое можно найти по ссылке **https://oreil.ly/qowNO**, объяснения использовались для выявления присутствующих на рентгенологических снимках аннотаций и внесения соответствующих поправок.

их предсказаниях опирается исключительно на демографические данные и впоследствии вносит искаженность без нашего ведома. Невозможно узнать, каким образом модель пришла к своему предсказанию, имея лишь кредитный балл.

Помимо конечных пользователей моделей в группу заинтересованных лиц входят те, кто занимается регуляторными и нормативными стандартами для ML-моделей, поскольку модели в некоторых отраслях могут требовать аудита или дополнительной прозрачности. Специалисты, участвующие в аудите моделей, вероятно, будут нуждаться в более подробном резюме относительно того, каким образом модель приходит к своим предсказаниям, чтобы оправдать ее использование и воздействие. Такие метрики, как точность, в данном случае бесполезны — без глубокого понимания причины, почему модель делает предсказания, ее использование может стать проблематичным.

Наконец, как исследователи данных и инженеры ML, без понимания признаков, на которые наша модель опирается в своих предсказаниях, мы можем улучшать ее качество только до определенной степени. Нам нужна методика получения подтверждений относительно того, что модели работают именно так, как мы и ожидаем. Например, предположим, что мы тренируем модель на табличных данных, чтобы предсказывать возможность задержки рейса. Модель тренируется на 20 признаках. Под капотом, возможно, она опирается только на 2 признака из этих 20, и если бы мы удалили остальные, то смогли бы значительно улучшить производительность нашей системы. Или же, все-таки, для достижения необходимой нам степени точности необходим каждый из этих 20 признаков? Без более подробной информации о том, что именно модель использует, это трудно понять.

Решение

В целях урегулирования проблемы с неотъемлемыми для ML неизвестностями нам нужен подход к пониманию того, каким образом модели работают. Подходы к пониманию и донесению информации о том, как и почему ML-модель делает предсказания, являются областью активных научных исследований. Объяснимость, также именуемая интерпретируемостью и пониманием модели, — это новая и быстро эволюционирующая область в рамках ML, способная принимать различные формы в зависимости от архитектуры и типа данных, на которых она тренируется. Объяснимость также помогает выявлять искаженность в моделях ML, что мы и рассмотрим при обсуждении паттерна "Призма объективности" далее в этой главе. Здесь же мы сосредоточимся на объяснении глубоких нейронных сетей с помощью признаковых атрибуций. В целях понимания атрибуций в контексте сначала мы рассмотрим объяснимость моделей с менее сложными архитектурами.

Более простые модели, такие как деревья решений, проще объяснять, чем глубокие модели, поскольку они часто поддаются *интерпретации по своему дизайну*. Это означает, что их усвоенные веса дают прямое представление о том, каким образом модель делает предсказания. Если у нас линейная регрессионная модель с независимыми числовыми входными признаками, то веса иногда могут поддаваться интерпретации. Возьмем, к примеру, линейную регрессионную модель, которая

предсказывает топливную эффективность автомобиля[8]. В scikit-learn[9] усвоенные коэффициенты линейной регрессионной модели можно получить следующим образом:

```
model = LinearRegression().fit(x_train, y_train)
coefficients = model.coef_
```

Результирующие коэффициенты для каждого признака в нашей модели показаны на рис. 7.1.

число цилиндров	-0.926610
литраж	0.037055
лошадиные силы	-0.017953
вес	-0.007286
ускорение	0.164976
год модели авто	0.723584
происхождение_1	-1.779775
происхождение_2	0.781041
происхождение_3	0.998735

Рис. 7.1. Коэффициенты, усвоенные из нашей линейной регрессионной модели топливной эффективности, которая предсказывает пробег автомобиля в милях на галлон топлива. Мы использовали метод `get_dummies()` из библиотеки pandas для преобразования изначального признака в булев столбец, поскольку он является категориальным

Коэффициенты показывают нам связь между каждым признаком и результатом на выходе из модели, т. е. предсказываемым милям пробега на галлон топлива (miles per gallon, MPG). Например, из этих коэффициентов мы можем заключить, что для каждого дополнительного цилиндра в автомобиле показатель MPG, который наша модель предсказывает, будет уменьшаться. Наша модель также усвоила, что по мере появления новых автомобилей (обозначаемых признаком "model year" (год выпуска модели авто)) они часто имеют более высокую топливную эффективность. Из этих коэффициентов узнать о связях между признаками и результатом нашей модели можно гораздо больше, чем из усвоенных весов скрытого слоя в глубокой нейронной сети. Вот почему модели, подобные приведенной выше, часто называются *интерпретируемыми по своему дизайну*.

Хотя и заманчиво придавать значительный смысл усвоенным весам в линейных регрессионных моделях или моделях на основе деревьев решений, все-таки мы должны проявлять крайнюю осторожность. Заключения, к которым мы пришли ранее, по-прежнему верны (т. е. обратная зависимость между числом цилиндров и топливной эффективностью), но мы не можем из величины коэффициентов, например, заключить, что для нашей модели категориальный признак происхождения (origin) или число цилиндров (cylinders) важнее, чем мощность в лошадиных силах

[8] Обсуждаемая здесь модель тренируется на публичном наборе данных UCI (см. **https://oreil.ly/cNixp**).

[9] См. **https://oreil.ly/V9GT5**.

(horsepower) или вес (weight). Прежде всего, каждый из этих признаков представлен в другой единице измерения. Один цилиндр не эквивалентен одному фунту — автомобили в этом наборе данных имеют максимум 8 цилиндров, но весят более 3000 фунтов. Вдобавок, категориальный признак происхождения представлен фиктивными значениями, поэтому каждое значение происхождения может быть равно только 0 или 1. Коэффициенты также ничего не говорят нам и о связи *между* признаками в нашей модели. Большее число цилиндров часто коррелирует с большей мощностью, но мы не можем сделать такой вывод из усвоенных весов[10].

При более высокой сложности модели мы используем *постфактумные* методы обеспечения объяснимости, служащие для аппроксимирования связей между признаками модели и результатами ее работы. Как правило, постфактумные методы выполняют этот анализ, не полагаясь на внутренние детали модели, такие как усвоенные веса. Эта область является областью постоянных научных исследований, и предлагается целый ряд методов объяснения наряду с инструментарием для добавления этих методов в ваш рабочий поток ML. Методы объяснения, которые мы обсудим, называются *признаковой атрибуцией*. Они ориентированы на то, чтобы приписывать результаты работы модели — будь то снимок, классификация или числовое значение — ее признакам, назначая каждому признаку атрибуционные значения, указывающие на степень вклада, который этот признак внес в данные на выходе из модели. Существуют два типа признаковой атрибуции.

- *Экземплярный уровень.*

  Признаковые атрибуции, которые объясняют данные на выходе из модели для отдельного предсказания. Например, в модели, предсказывающей, должен ли заявитель получить одобрение на кредитную линию, признаковая атрибуция уровня экземпляра даст представление о том, почему заявка конкретного человека была отклонена. В снимковой модели атрибуция уровня экземпляра может выделять пикселы снимка, которые побудили ее предсказать, что на нем изображена кошка.

- *Глобальный уровень.*

  Глобальные признаковые атрибуции анализируют поведение модели по всему агрегату, чтобы делать заключения о том, как модель ведет себя в целом. В типичной ситуации это делается путем усреднения признаковых атрибуций уровня экземпляра из тестового набора данных. В модели, предсказывающей возможную задержку рейса, глобальные атрибуции могли бы сообщать нам о том, что в целом экстремальные погодные условия являются наиболее важным признаком во время предсказания задержек.

Два метода признаковой атрибуции, которые мы рассмотрим[11] далее, описаны в табл. 7.2 и предлагают разные подходы, которые могут использоваться как для объяснения на уровне экземпляра, так и для объяснения на глобальном уровне.

---

[10] Документация scikit-learn более подробно описывает процедуру правильного интерпретирования усвоенных весов в линейных моделях (см. **https://oreil.ly/DAmIm**).

[11] Мы сосредоточимся на этих двух методах обеспечения объяснимости, поскольку они широко используются и охватывают разные типы моделей, но в этот анализ не включен ряд других методов и каркасов, таких как LIME (см. **https://oreil.ly/0c4uB**) и ELI5 (**https://github.com/TeamHG-Memex/eli5**).

Таблица 7.2. *Описания разных методов объяснения и ссылки на посвященные им научно-исследовательские работы*

Название	Описание	Исследование
Отобранное значение Шепли (sampled Shapley)	Основываясь на концепции значения Шепли (Shapley)[12], этот подход определяет предельный вклад признака, вычисляя степень, с которой добавление и удаление этого признака влияет на предсказание, анализируемое по многочисленным комбинациям значений признаков	**https://oreil.ly/ubEjW**
Интегрированные градиенты (integrated gradients, IG)	Используя предопределенный базовый уровень модели, показатель интегрированных градиентов вычисляет производные (градиенты) вдоль пути от этого базового уровня к конкретному значению на входе	**https://oreil.ly/sy8f8**

Хотя мы могли бы реализовать эти подходы с нуля, существует инструментарий, который специально предназначен для упрощения процесса получения признаковых атрибуций. Имеющиеся инструменты обеспечения объяснимости с открытым исходным кодом и облачные средства позволяют нам сосредоточиваться на отладке, совершенствовании и обобщении наших моделей.

Базовый уровень модели

Для того чтобы воспользоваться этими инструментами, нам сначала нужно понять идею *базового уровня* (baseline) в части его применения к объяснению моделей с признаковыми атрибуциями. Цель любого метода обеспечения объяснимости состоит в том, чтобы ответить на вопрос: почему модель предсказала X? Признаковые атрибуции пытаются сделать это путем предоставления числовых значений для каждого признака, указывающих на величину вклада, который этот признак внес в конечный результат работы. Возьмем, к примеру, модель, предсказывающую наличие у пациента сердечных заболеваний, учитывая некоторые демографические и медицинские данные. В качестве одного примера в нашем тестовом наборе данных давайте представим, что атрибуционное значение для признака "холестерин" равно 0,4, а атрибуция для его артериального давления равна –0,2. Без контекста эти атрибуционные значения не имеют большого смысла, и наш первый вопрос, скорее всего, будет таким: 0,4 и –0,2 относительно чего? Это "чего" является *базовым уровнем модели.*

Всякий раз, когда мы получаем значения признаковых атрибуций, все они относятся к значению предопределенного для нашей модели базово-уровневого предсказания. Базово-уровневые предсказания могут быть как *информативными*, так и *неинформативными.* Неинформативные базовые уровни обычно сравниваются с некоторым средним случаем в тренировочном наборе данных. В снимковой модели неинформативным базовым уровнем может быть сплошной черно-белый снимок.

[12] Значение Шепли было введено в статье Ллойда Шепли (см. **https://oreil.ly/xCrqU**) в 1951 году и основано на концепциях теории игр.

В текстовой модели неинформативным базовым уровнем могут быть нулевые значения для матриц векторного вложения либо стоп-слова, такие как the, is или and. В модели с числовыми входными данными распространенный подход к выбору базового уровня заключается в генерировании предсказания с использованием медианного значения для каждого модельного признака.

Определение базовых уровней

Наше представление базового уровня будет отличаться в зависимости от того, какую задачу, регрессионную или классификационную, наша модель выполняет. В регрессионной задаче модель будет иметь *ровно одно* числовое значение базовоуровневого предсказания. В примере с пробегом автомобиля давайте представим, что для расчета базового уровня мы решили использовать медианный подход. Медиана для восьми признаков в нашем наборе данных представляет собой следующий массив:

```
[151.0, 93.5, 2803.5, 15.5, 76.0, 1.0, 0.0, 0.0]
```

Когда мы отправляем его в нашу модель, предсказанный показатель MPG будет равен `22.9`. Следовательно, для каждого предсказания, который мы делаем в этой модели, мы будем использовать MPG `22.9` в качестве базового уровня для сравнения предсказаний.

Теперь давайте представим, что мы следуем паттерну "Переформулировка", чтобы поменять эту задачу с регрессионной на классификационную. С этой целью для топливной эффективности мы определим корзины "низкая", "средняя" и "высокая", и наша модель, следовательно, выведет трехэлементную матрицу на основе функции мягкого максимума, показывающую вероятность того, что данный автомобиль соответствует каждому классу. Беря те же самые медианные базовоуровневые данные на входе, что и выше, наша классификационная модель теперь возвращает следующее в качестве базовоуровневого предсказания:

```
[0.1, 0.7, 0.2]
```

Теперь у нас есть *другое* значение базовоуровневого предсказания для каждого класса. Допустим, мы генерируем новое предсказание на примере из тестового набора, и наша модель выводит следующий массив, предсказывая 90%-ную вероятность того, что этот автомобиль имеет "низкую" топливную эффективность:

```
[0.9, 0.06, 0.04]
```

Результирующие значения признаковых атрибуций должны объяснять причину, почему модель предсказала `0.9` по сравнению со значением базовоуровневого предсказания `0.1` для "низкого" класса. Мы также можем посмотреть на значения признаковых атрибуций у других классов, чтобы понять, например, почему наша модель предсказала, что тот же автомобиль имеет 6%-ный шанс принадлежать к "среднему" классу топливной эффективности.

На рис. 7.2 показаны признаковые атрибуции уровня экземпляра для модели, которая предсказывает продолжительность велосипедной поездки. Неинформативным

базовым уровнем для этой модели является продолжительность поездки в 13,6 минуты, которую мы получаем, генерируя предсказание с использованием медианного значения по каждому признаку в нашем наборе данных. Когда предсказание модели меньше, чем значение базовоуровневого предсказания, мы должны ожидать, что большинство атрибуционных значений будут отрицательными, и наоборот. В этом примере мы получаем предсказанную продолжительность 10,71, которая меньше модельного базового уровня и объясняет причину, почему многие атрибуционные значения отрицательные. Мы можем определить наиболее важные признаки, взяв абсолютное значение признаковых атрибуций. В этом примере расстояние поездки было самым важным признаком и привело к тому, что предсказание нашей модели уменьшилось на 2,4 минуты от базового уровня. Вдобавок, в качестве проверки на здравость мы должны обеспечить, чтобы значения признаковых атрибуций в сумме примерно составляли разницу между текущим и базовоуровневым предсказаниями.

Базовоуровневое предсказание: 13.61
Предсказанная продолжительность: 10.71

Имя	Значение признака	Атрибуционное значение
distance	1395.51	-2.44478
start_hr	18	-1.29039
max_temp	20.7239	0.690506
temp	16.168	0.12629
dew_point	7.83396	0.0110318
prcp	0.03	-0.00134132
weekday	1	0
wdsp	0	0
rain_drizzle	0	0

Рис. 7.2. Значения признаковой атрибуции для одного примера в модели, предсказывающей продолжительность велосипедной поездки. Модельный базовый уровень, рассчитанный с использованием медианы каждого признакового значения, составляет 13,6 минуты, а атрибуционные значения показывают, насколько каждый признак повлиял на предсказание

Информативные базовые уровни, с другой стороны, сравнивают предсказание модели с конкретным альтернативным сценарием. В модели, идентифицирующей мошеннические транзакции, информативный базовый уровень может отвечать на вопрос: почему эта транзакция была помечена именно как мошенническая, а не как не мошенническая? Вместо того чтобы для расчета базового уровня использовать медианные значения признаков по всему тренировочному набору данных, мы возьмем медиану только не мошеннических значений. В снимковой модели тренировочные фото, возможно, содержат значительную часть сплошных черно-белых пикселов, и их использование в качестве базового уровня приведет к неточным предсказаниям. В этом случае нам пришлось бы придумать другой *информативный* базовоуровневый снимок.

Эвристические эталоны и модельные базовые уровни

Как модельные базовые уровни соотносятся с паттерном дизайна "Эвристический эталон"? Эвристический эталон должен служить отправной точкой для обобщения модели на глобальном уровне, нередко до реализации объяснимости. При использовании методов обеспечения объяснимости выбор типа базового уровня (информативный или неинформативный) и способ его расчета зависят от нас. Подходы, описанные в паттерне "Эвристический эталон", также могут использоваться с целью определения базового уровня модели для применения с методом обеспечения объяснимости.

И эвристические эталоны, и модельные базовые уровни обеспечивают базу для ответа на вопрос: почему модель сделала X, а не Y? Эвристические эталоны являются первым шагом в модельном анализе и представляют собой один из возможных подходов к расчету базового уровня. Когда в этом разделе мы используем термин "базовый уровень", мы имеем в виду именно значение, используемое в качестве точки отсчета в методах обеспечения объяснимости.

Библиотека SHAP

Библиотека с открытым исходным кодом SHAP[13] обеспечивает API на языке Python для получения признаковых атрибуций на многих типах моделей и основана на концепции значения Шепли, представленной в табл. 7.2. Для определения значений признаковых атрибуций библиотека SHAP вычисляет величину, с которой добавление или удаление каждого признака способствует предсказанию на выходе из модели. Она выполняет этот анализ в разных комбинациях признаковых значений и результата работы модели на выходе.

Библиотека SHAP независима от фреймворка и работает с моделями, натренированными на снимках, текстах или табличных данных. Для ознакомления с тем, как SHAP работает на практике, воспользуемся ранее приведенным набором данных о топливной эффективности. На этот раз мы построим глубокую модель с помощью API Sequential в Keras:

```
model = tf.keras.Sequential([
  tf.keras.layers.Dense(16, input_shape=(len(x_train.iloc[0]))),
  tf.keras.layers.Dense(16, activation='relu'),
  tf.keras.layers.Dense(1)
])
```

В целях использования библиотеки SHAP мы сначала создадим объект объяснителя `DeepExplainer`, передав ему нашу модель и подмножество примеров из тренировочного набора. Затем получим атрибуционные значения для первых 10 примеров из тестового набора:

```
import shap
explainer = shap.DeepExplainer(model, x_train[:100])
attribution_values = explainer.shap_values(x_test.values[:10])
```

[13] См. **https://github.com/slundberg/shap**.

Библиотека SHAP имеет несколько встроенных методов визуализации, которые облегчают понимание результирующих атрибуционных значений. Мы воспользуемся методом `force_plot()` и построим график атрибуционных значений для первого примера в тестовом наборе с помощью следующего фрагмента исходного кода:

```
shap.force_plot(
  explainer.expected_value[0],
  shap_values[0][0,:],
  x_test.iloc[0,:]
)
```

В приведенном выше фрагменте кода ожидаемое значение объяснителя `explainer.expected_value` является базовым уровнем нашей модели. Библиотека SHAP вычисляет базовый уровень как среднее значение данных на выходе из модели по всему набору данных, который мы передали при создании объяснителя (в данном случае `x_train[:100]`), хотя в метод `force_plot()` можно также передать наше собственное базовоуровневое значение. Значение эмпирического наблюдения для этого примера составляет 14 миль пробега на галлон топлива, а наша модель предсказывает 13,16. Следовательно, наше объяснение будет пояснять модельное предсказание 13,16 с помощью атрибуционных значений. В данном случае атрибуционные значения соотносятся с базовым уровнем модели, равным 24,16 миль пробега на галлон топлива. Следовательно, атрибуционные значения должны составлять в целом примерно 11, т. е. разницу между базовым уровнем модели и предсказанием для этого примера. Наиболее важные признаки можно определить, рассматривая те из них, которые имеют наибольшее абсолютное значение. На рис. 7.3 показан результирующий график для атрибуционных значений этого примера.

Рис. 7.3. Значения признаковых атрибуций для одного примера из нашей модели предсказания топливной эффективности. В данном случае вес автомобиля является наиболее значимым для MPG индикатором со значением признаковой атрибуции примерно 6. Если бы предсказание нашей модели было выше базового уровня 24,16, то вместо этого мы бы увидели главным образом отрицательные атрибуционные значения

В этом примере наиболее важным для топливной эффективности индикатором является вес, толкающий предсказание нашей модели вниз примерно на 6 миль пробега на галлон топлива от базового уровня. За ним следуют лошадиные силы, объем двигателя, а затем год выпуска модели автомобиля. Сводка (или глобальное объяснение) атрибуционных значений для первых 10 примеров из тестового набора может быть получена с помощью следующего фрагмента исходного кода:

```
shap.summary_plot(
  shap_values,
```

```
    feature_names=data.columns.tolist(),
    class_names=['MPG']
)
```

Его выполнение приведет к сводному графику, показанному на рис. 7.4.

На практике у нас был бы более крупный набор данных, и мы хотели бы рассчитывать глобальные атрибуции на большем числе примеров. Тогда мы могли бы использовать этот анализ для обобщения поведения нашей модели перед другими заинтересованными лицами внутри и за пределами нашей организации.

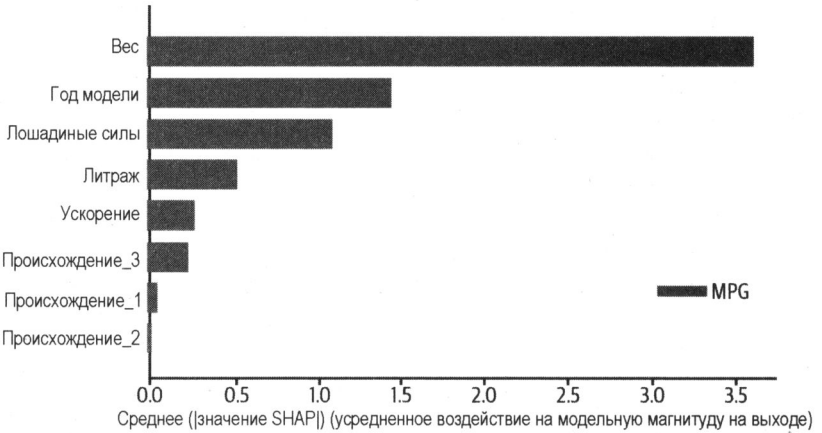


Рис. 7.4. Пример признаковых атрибуций глобального уровня для модели топливной эффективности, рассчитанной на первых 10 примерах из тестового набора данных

Объяснения из развернутых моделей

Библиотека SHAP предлагает интуитивно понятный API для получения атрибуций на языке Python, как правило, используемого в скриптах или блокнотах. Это хорошо работает во время разработки модели, но есть сценарии, в которых желательно получать объяснения в отношении развернутой модели в дополнение к результату ее предсказания. В этом случае облачные инструменты обеспечения объяснимости являются наилучшим вариантом. Здесь мы продемонстрируем способы получения признаковых атрибуций на развернутой модели с помощью инструментария интерпретации предсказаний Explainable AI[14] инфрастуктуры Google Cloud. На момент написания этой книги инструментарий Explainable AI работал с пользовательскими моделями TensorFlow и моделями на основе табличных данных, построенными с помощью службы AutoML.

Мы развернем снимковую модель на платформе AI Platform, чтобы показать объяснения, но мы могли бы применить инструментарий Explainable AI и с моделями TensorFlow, натренированными на табличных или текстовых данных. Для начала

[14] См. **https://oreil.ly/lDocn**.

мы развернем модель из TensorFlow Hub[15], натренированную на наборе данных ImageNet. Для того чтобы сосредоточиться на задаче получения объяснений, мы не будем выполнять никакого трансферного обучения на модели, а просто воспользуемся изначальной 1000 классов меток ImageNet:

```
model = tf.keras.Sequential([
    hub.KerasLayer(".../mobilenet_v2/classification/2",
                   input_shape=(224,224,3)),
    tf.keras.layers.Softmax()
])
```

В целях развертывания модели на платформе AI Platform с объяснениями нам сначала нужно создать файл метаданных, который будет использоваться службой объяснений для расчета признаковых атрибуций. Эти метаданные предоставляются в файле JSON и включают информацию о базовом уровне, которую мы хотели бы использовать, и частях модели, которые мы хотим объяснять. Для упрощения этого процесса инструментарий Explainable AI предоставляет SDK, который будет генерировать метаданные с помощью следующего фрагмента кода:

```
from explainable_ai_sdk.metadata.tf.v2 import SavedModelMetadataBuilder

model_dir = 'path/to/savedmodel/dir'

model_builder = SavedModelMetadataBuilder(model_dir)
model_builder.set_image_metadata('input_tensor_name')
model_builder.save_metadata(model_dir)
```

В этом фрагменте кода базовый уровень модели не задан, следовательно, будет использоваться значение по умолчанию (для снимковых моделей это черно-белый снимок). Мы можем опционально добавить параметр `input_baselines` в `set_image_metadata`, чтобы указать пользовательский базовый уровень. Выполнение метода `save_metadata()` в приведенном выше фрагменте кода создает файл explanation_metadata.json в каталоге модели (полный исходный код находится в репозитории на GitHub[16]).

При использовании этого SDK посредством управляемой службы блокнотов AI Platform Notebooks у нас также есть возможность генерировать объяснения локально внутри экземпляра блокнота без развертывания нашей модели в облаке. Это делается с помощью метода `load_model_from_local_path()`.

Имея в корзине хранилища Cloud Storage экспортированную модель и файл explanation_metadata.json, мы готовы создать новую версию модели. При ее создании мы указываем метод объяснения, который хотели бы использовать.

В целях развертывания нашей модели на платформе AI Platform мы можем скопировать наш каталог моделей в корзину Cloud Storage и использовать интерфейс

---

[15] См. **https://oreil.ly/Ws8jx**.

[16] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/07_stakeholder_management/explainability.ipynb**.

командной строки gcloud для создания версии модели. Платформа AI Platform имеет три возможных метода объяснения на выбор.

- *Интегрированные градиенты (IG).*

  В этом методе реализован подход, представленный в статье об интегрированных градиентах[17]. Он работает с любой дифференцируемой моделью TensorFlow — снимковой, текстовой или табличной. Для снимковых моделей, развернутых на платформе AI Platform, метод интегрированных градиентов возвращает снимок с выделенными пикселами, указывая на участки, которые сигнализировали о модельном предсказании.

- *Отобранное значение Шепли.*

  Этот метод основан на статье о механизме отбора для получения значения Шепли[18]. В нем используется подход, аналогичный принятому в библиотеке SHAP с открытым исходным кодом. На платформе AI Platform мы можем использовать этот метод с табличными и текстовыми моделями TensorFlow. Для расчета признаковых атрибуций по всем моделям службой AutoML Tables используется метод отобранного значения Шепли, поскольку метод интегрированных градиентов работает только с дифференцируемыми моделями.

- *XRAI.*

  Этот подход[19] строится на методе интегрированных градиентов. В нем применяется сглаживание для возврата атрибуций на основе участков. Метод XRAI работает только со снимковыми моделями, развернутыми на платформе AI Platform.

В команде gcloud мы задаем желаемый метод объяснения, а также число интегральных шагов или путей, которые хотим использовать при вычислении атрибуционных значений[20]. Параметр `steps` относится к числу признаковых комбинаций, отбираемых для каждого результата. В общем случае увеличение этого числа будет повышать точность объяснения:

```
!gcloud beta ai-platform versions create $VERSION_NAME \
--model $MODEL_NAME \
--origin $GCS_VERSION_LOCATION \
--runtime-version 2.1 \
--framework TENSORFLOW \
--python-version 3.7 \
--machine-type n1-standard-4 \
--explanation-method xrai \
--num-integral-steps 25
```

---

[17] См. **https://oreil.ly/FJhMd**.

[18] См. **https://oreil.ly/EAS8T**.

[19] См. **https://oreil.ly/niGVQ**.

[20] Более подробную информацию об этих методах объяснения и их имплементации см. в белой книге инструментария Explainable AI по адресу **https://oreil.ly/PYn8P**.

Развернув модель, мы можем получать объяснения с помощью SDK инструментария Explainable AI:

```
model = explainable_ai_sdk.load_model_from_ai_platform(
  GCP_PROJECT,
  MODEL_NAME,
  VERSION_NAME
)
request = model.explain([test_img])

# Распечатать снимок с пиксельными атрибуциями
request[0].visualize_attributions()
```

На рис. 7.5 мы видим сравнение объяснений методов IG и XRAI, возвращаемых из Explainable AI для нашей модели ImageNet. Выделенные участки пикселов показывают пикселы, которые внесли наибольший вклад в предсказание "хаски" нашей моделью.

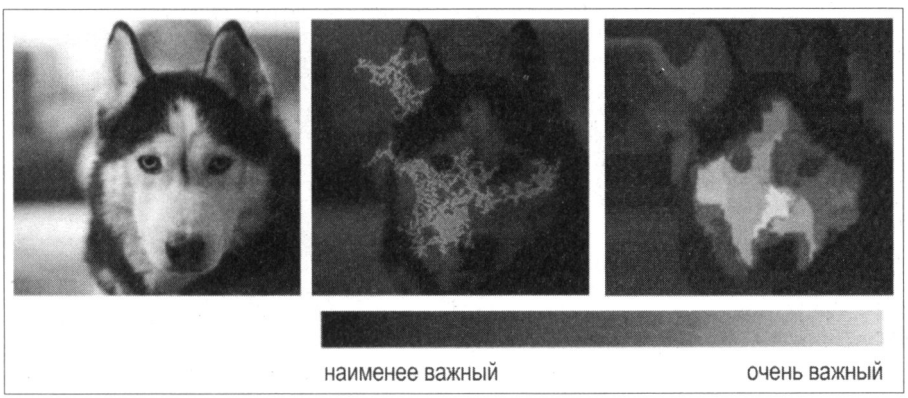


Рис. 7.5. Признаковые атрибуции, возвращенные из инструментария Explainable AI для модели ImageNet, развернутой на платформе AI Platform. Слева — оригинальный снимок. Атрибуции метода IG представлены посередине, а атрибуции метода XRAI — справа. Ключ снизу показывает, чему соответствуют участки в XRAI — более светлые области являются наиболее важными, а более темные участки представляют наименее важные участки

В типичной ситуации метод интегрированных градиентов (IG) рекомендуется для "неестественных" снимков, например сделанных в медицинской, заводской или лабораторной среде. Метод XRAI обычно лучше всего работает для снимков, сделанных в естественных условиях и для настоящих объектов, таких как эта хаски. Для того чтобы понять причины, почему метод IG предпочтительнее для неестественных снимков, см. атрибуции IG для снимка диабетической ретинопатии на рис. 7.6. В случаях, подобных приведенному выше медицинскому, он помогает увидеть атрибуции на мелкозернистом, пиксельном уровне. На снимке собаки, в принципе, знать точные пикселы, которые заставили нашу модель предсказать "хаски", не так важно, и метод XRAI дает нам более высокоуровневую сводку важных участков.

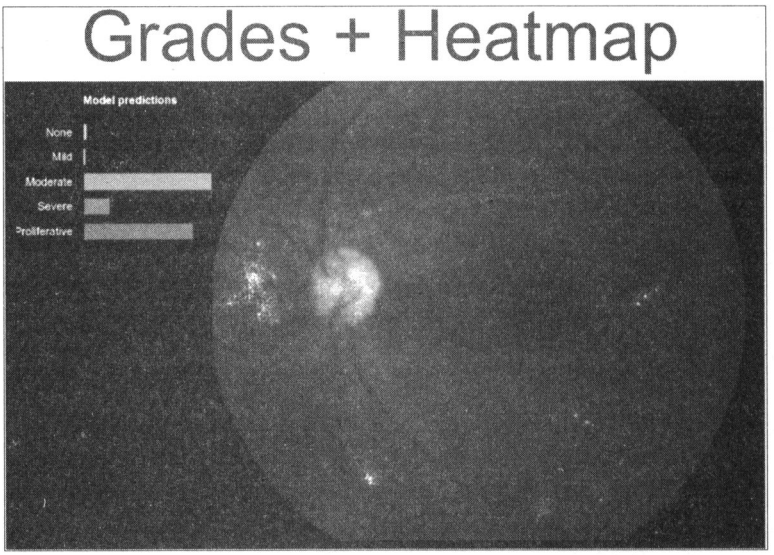


Рис. 7.6. В рамках научного исследования[21], проведенного Рори Сайресом (Rory Sayres) и его коллегами в 2019 году, разным группам офтальмологов было предложено оценить степень диабетической ретинопатии на снимке в трех сценариях: автономный снимок без предсказаний, снимок с предсказаниями моделью и снимок с предсказаниями и пиксельными атрибуциями (показан здесь). На снимке можно увидеть, как пиксельные атрибуции помогают повышать уверенность в модельном предсказании

Инструментарий Explainable AI также функционирует в службе AutoML Tables[22], инструменте для тренировки и развертывания моделей, работающих на табличных данных. Служба AutoML Tables выполняет предобработку данных и отбирает наилучшую для наших данных модель, а значит, нам не нужно писать какой-либо модельный исходный код. Признаковая атрибуция с помощью Explainable AI для моделей, натренированных в AutoML Tables, включена по умолчанию. Кроме того, предоставляются объяснения как глобального уровня, так и уровня экземпляра.

Компромиссы и альтернативы

Хотя объяснения обеспечивают важное представление о том, каким образом модель принимает решения, они хороши лишь настолько, насколько хороши модельные тренировочные данные, качество вашей модели и выбранный базовый уровень. В этом разделе мы обсудим некоторые пределы объяснимости, а также некоторые альтернативы признаковой атрибуции.

Искаженность отбора данных

Нередко говорят, что машинное обучение работает по принципу "мусор вошел, мусор вышел". Другими словами, модель хороша лишь настолько, насколько хороши

[21] См. **https://oreil.ly/Xp_vp**.

[22] См. **https://oreil.ly/CSQly**.

данные, используемые для ее тренировки. Если мы натренируем снимковую модель идентифицировать 10 разных пород кошек, то эти 10 пород кошек будут всем, что она знает. Если мы покажем модели снимок собаки, то она лишь попытается классифицировать собаку как одну из 10 категорий кошек, на которых была натренирована. Она даже сможет сделать это с большой уверенностью, т. е. модели являются прямым представлением своих тренировочных данных.

Если мы не улавливаем дисбалансы данных до тренировки модели, то методы обеспечения объяснимости, такие как признаковая атрибуция, помогают высвечивать искаженность отбора данных. В качестве примера предположим, что мы строим модель, чтобы предсказывать тип лодки на снимке. Допустим, модель правильно помечает снимок из нашего тестового набора как "каяк", но, используя признаковую атрибуцию, мы обнаруживаем, что в своем предсказании "каяк" модель опирается не на контур лодки, а на ее весло. Это сигнал о том, что в нашем наборе данных может быть недостаточно вариативных тренировочных снимков по каждому классу — нам, вероятно, придется вернуться и добавить больше снимков каяков под разными углами, как с веслами, так и без них.

Контрфактический анализ и объяснения на основе примеров

В дополнение к признаковой атрибуции, описанной в *разд. "Решение"*, существуют и другие самые разные подходы к объяснению данных на выходе из ML-моделей. Данный раздел не предназначен для того, чтобы давать исчерпывающий перечень всех подоходов к обеспечению объяснимости, поскольку эта область быстро эволюционирует. Мы лишь кратко опишем еще два подхода: контрфактический анализ и объяснение на примерах.

Контрфактический анализ — это методика обеспечения объяснимости уровня экземпляра, которая призвана отыскивать примеры из нашего набора данных с аналогичными признаками, приведшие к разным предсказаниям из нашей модели. Один из подходов к проведению такого анализа обеспечивается инструментом с открытым исходным кодом What-If[23], служащим для оценивания и визуализации результатов работы ML-моделей. Мы предоставим более подробный обзор инструмента What-If в паттерне "Призма объективности" — здесь же мы сосредоточимся конкретно на его функциональности по контрфактическому анализу. При визуализации точек данных из нашего тестового набора в инструменте What-If у нас есть возможность показывать ближайшую контрфактическую точку данных к той, которую мы выбираем. Это позволяет нам сравнивать значения признаков и предсказания модели для этих двух точек данных, помогая лучше понимать признаки, на которые наша модель опирается больше всего. На рис. 7.7 мы видим контрфактическое сравнение двух точек данных из набора данных об ипотечных заявках. Жирным шрифтом выделены признаки, в которых эти две точки данных различаются, а снизу — результат работы модели для каждой из них.

В процедуре объяснения на уровне примеров новые примеры и соответствующие им предсказания сравниваются с аналогичными примерами из нашего тренировоч-

[23] См. **https://oreil.ly/Vf3D-**.

ного набора данных. Этот тип объяснения особенно полезен для понимания степени влияния нашего тренировочного набора данных на поведение модели. Объяснения на основе примеров лучше всего работают на снимковых или текстовых данных и бывают информативнее, чем признаковые атрибуции или контрфактический анализ, поскольку они соотносят модельное предсказание непосредственно с данными, используемыми для тренировки.

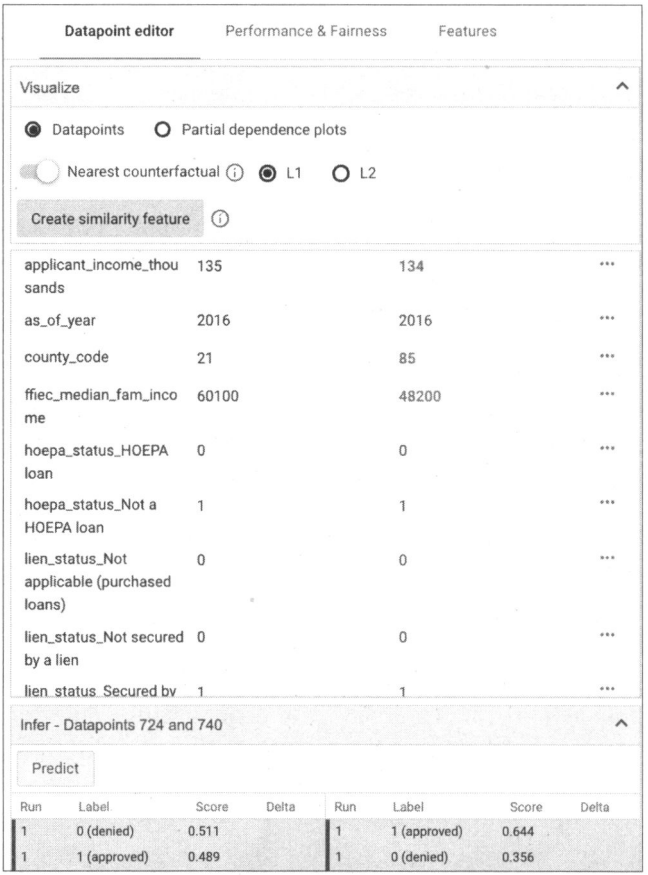


Рис. 7.7. Контрфактический анализ в инструменте What-If для двух точек данных из набора данных об ипотечных заявках США. Различия между двумя точками данных выделены жирным шрифтом. Более подробную информацию об этом наборе данных можно найти в обсуждении паттерна "Призма объективности" в этой главе

Для более глубокого понимания этого подхода давайте посмотрим на игру "Quick, Draw!"[24, 25]. Игра просит игрока нарисовать предмет и в режиме реального времени угадывает фигуру, которую он рисует, используя глубокую нейронную сеть, натренированную на тысячах рисунков других людей. После того как игрок закончит

[24] См. **https://oreil.ly/-QsHl**.

[25] Более подробную информацию о "Quick, Draw!" и объяснения, основанные на примерах, см. в статье **https://oreil.ly/Yvexy**.

рисовать, он может увидеть, каким образом нейронная сеть пришла к своему предсказанию, посмотрев на примеры из тренировочного набора данных. На рис. 7.8 мы видим примеры объяснений рисунка картофеля фри, который модель успешно распознала.

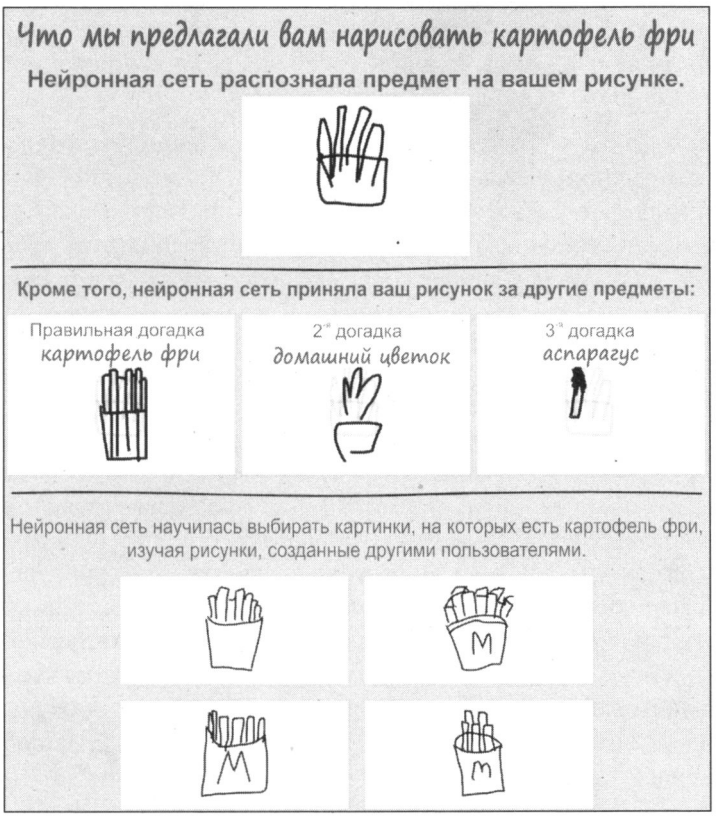


Рис. 7.8. Объяснения на основе примеров из игры "Quick, Draw!", показывающие ход генерирования моделью правильного предсказания "картофель фри" для данного рисунка на примерах из тренировочного набора данных

Пределы объяснений

Методика, обеспечивающая объяснения, позволяет значительно лучше понять работу моделей, но мы должны соблюдать осторожность, не слишком доверяя объяснениям со стороны нашей модели или не слишком полагаясь на допущения, что они обеспечивают совершенное понимание сути модели. Объяснения в любой форме являются прямым отражением наших тренировочных данных, модели и отобранного базового уровня. Иначе говоря, мы не можем ожидать, что наши объяснения будут качественными, если набор тренировочных данных является неточным представлением групп, отраженных нашей моделью, или если выбранный нами базовый уровень плохо подходит для решаемой нами задачи.

Вдобавок связь, которую объяснения могут выявлять между модельными признаками и результатом, представляет только наши данные и модель, и не обязательно

окружающую среду вне этого контекста. В качестве примера предположим, что мы тренируем модель идентифицировать мошеннические операции по кредитным картам, и она находит в качестве признаковой атрибуции глобального уровня, что сумма транзакции является признаком, который наилучшим образом свидетельствует о мошенничестве. Исходя из этого, было бы неверно делать вывод о том, что сумма *всегда* является самым большим показателем мошенничества с кредитными картами — это имеет место только в контексте нашего тренировочного набора данных, модели и заданного значения базового уровня.

Мы можем рассматривать объяснения как важное дополнение к точности, ошибке и другим метрикам, используемым для оценивания ML-моделей. Они полезны для формулировки мнения о качестве модели и о потенциальной искаженности, но они не должны быть единственным определяющим детерминантом высококачественной модели. Мы рекомендуем использовать объяснения как один из критериев оценивания в дополнение к оцениванию данных и модели, а также применять другие паттерны, описанные в этой и предыдущих главах.

ПАТТЕРН 30. Призма объективности

Паттерн "Призма объективности" (Fairness Lens) предполагает использование технических приемов предобработки и постобработки с целью обеспечения справедливости и объективности предсказаний модели для разных групп пользователей и сценариев. Объективность в машинном обучении — это постоянно эволюционирующая область научных исследований — ни единого всеохватывающего решения, которое делает модель "объективной", ни определения ему пока что нет. Оценивание всего сквозного рабочего потока ML-процессов целиком — от сбора данных до развертывания модели — через призму объективности имеет большое значение для создания успешных, высококачественных моделей.

Постановка задачи

Видя слово "машинный" в названии, легко допустить, что ML-модели не могут быть искаженными. В конце концов, модели — это результат усвоенных компьютером закономерностей, не так ли? Проблема такого рассуждения заключается в том, что наборы данных, на которых модели учатся, создаются не машинами, а людьми, а люди полны искаженных взглядов и пониманий. Эта присущая человеку искаженность неизбежна, но не обязательно всегда плоха. Возьмем, к примеру, набор данных, используемый для тренировки модели обнаружения финансового мошенничества — эти данные, скорее всего, будут сильно несбалансированы, т. к. содержат очень малое число примеров мошенничества, поскольку оно в большинстве случаев является относительно редким. Это пример естественной искаженности, поскольку она является отражением статистических свойств изначального набора данных. Искаженность становится *вредной*, когда она по-разному влияет на разные группы людей. Она называется *проблемной искаженностью* (problematic bias), и именно на этой теме мы сосредоточимся в настоящем разделе. Если данный

тип искаженности не учитывается, то он может найти свой путь в модели, создавая неблагоприятные эффекты, поскольку промышленные модели непосредственно отражают искаженности, присутствующее в данных.

Проблемная искаженность присутствует даже в тех ситуациях, когда вы ее не ожидаете. Например, представьте, что мы создаем модель, которая будет идентифицировать разные типы одежды и аксессуаров. Нам было поручено собрать все снимки обуви для тренировочного набора данных. Когда мы думаем об обуви, мы обращаемся к первому, что приходит на ум. Это теннисный ботинок? Кеды? Шлепанцы? А как насчет туфли на каблуке-шпильке? Давайте представим, что мы живем в теплом климате круглый год и большинство наших знакомых постоянно носят сандалии. Когда мы думаем о ботинке, сандалия — это первое, что приходит на ум. Как следствие, мы собираем неоднородные по своему составу снимки сандалий с различными типами ремешков, толщиной подошвы, цвета и многими другими признаками. Мы вносим их в большой набор данных одежды, и когда тестируем модель на тестовом наборе снимков обуви нашей подруги, модель достигает 95%-ной точности на метке "обувь". Модель выглядит многообещающе, но проблемы возникают тогда, когда наши коллеги из разных географических мест тестируют модель на снимках своих каблуков и кроссовок. Для их снимков метка "обувь" вообще не появляется.

Этот пример с обувью демонстрирует искаженность распределения тренировочных данных, и хотя он, возможно, кому-то покажется чрезмерно упрощенным, такой тип искаженности часто встречается в производственных условиях. Искаженность распределения данных происходит, когда собираемые нами данные не точно отражают всю популяцию, которая будет использовать нашу модель. Если наш набор данных ориентирован на человека, то этот тип искаженности может быть особенно очевиден, если наш набор данных не включает равное представительство возрастов, рас, полов, религий, сексуальной ориентации и других характеристик личности[26].

Даже когда наш набор данных выглядит сбалансированным по отношению к этим характеристикам личности, он все равно подвержен искаженности в том, как эти группы представлены в данных. Предположим, мы тренируем модель анализа настроений классифицировать отзывы о ресторанах по шкале от 1 (крайне негативный) до 5 (крайне позитивный). Мы позаботились о том, чтобы получить сбалансированное представление разных типов ресторанов в данных. Однако оказывается, что большинство отзывов о ресторанах морепродуктов являются позитивными, в то время как большинство отзывов о ресторанах вегетарианской пищи негативны. Эта искаженность представления данных будет непосредственно отображена нашей

[26] Более подробно о том, как расовое и гендерное искаженные представления могут найти свой путь в моделях классифицирования снимков, см. в статье "Гендерные оттенки: межсекторальные различия в точности коммерческой гендерной классификации" (Joy Buolamwini, Timmit Gebru. Gender Shades: Intersectional Accuracy Differences in Commercial Gender Classification // Proceedings of Machine Learning Research. — 2018. — № 81. — P. 1–15. — URL: **https://oreil.ly/1zw3e**).

моделью. Всякий раз когда добавляются новые отзывы о вегетарианских ресторанах, у них гораздо больше шансов на классификацию в качестве негативных, а это может оттолкнуть потенциальных клиентов. Эта искаженность также называется *искаженностью отчетности* (reporting bias), поскольку набор данных (здесь "отчетные" данные) не точно отражает реальный мир.

Распространенная ошибочность при решении проблем искаженности данных заключается в том, что удаление областей искаженности из набора данных устранит проблему. Допустим, мы строим модель для предсказания вероятности невыплаты по кредиту. Если мы обнаружим, что модель необъективно относится к людям разных рас, мы готовы допустить, что это можно исправить, просто удалив расу как признак из набора данных. Проблема с этим решением заключается в том, что из-за систематической искаженности такие характеристики, как раса и пол, часто неявно отражаются в других характеристиках, таких как почтовый индекс или доход. Этот эффект называется *неявной* или *косвенной искаженностью* (proxy bias). Удаление очевидных признаков с потенциальной искаженностью, таких как раса и пол, часто имеет худшие последствия, чем их оставление, поскольку это затрудняет выявление и исправление случаев искаженности в модели.

При сборе и подготовке данных еще одной областью, в которую может быть внесена искаженность, является принцип разметки данных. Коллективы часто передают разметку крупных наборов данных на аутсорсинг, но важно понимать то, как разметчики могут вносить искаженность в набор данных, в особенности если разметка имеет субъективный характер. Это называется *искаженностью экспериментатора* (experimenter bias). Представьте себе, что мы строим модель анализа настроений и передаем ее на аутсорсинг группе из 20 человек — их работа состоит в том, чтобы помечать каждый фрагмент текста по шкале от 1 (негативный) до 5 (позитивный). Этот тип анализа чрезвычайно субъективен и может зависеть от культуры, воспитания и многих других факторов. Прежде чем использовать эти данные для тренировки нашей модели, мы должны обеспечить, чтобы в эту группу из 20 разметчиков входили разные индивидумы.

В дополнение к данным искаженность также может вноситься во время тренировки модели выбранной нами целевой функцией. Например, если мы оптимизируем нашу модель под метрику совокупной точности, то она может не точно отражать результативность модели во всех срезах данных. В тех случаях когда наборы данных изначально не сбалансированы, использование точности как единственной метрики может упускать случаи, когда наша модель недостаточно результативна или принимает необъективные решения на миноритарных классах в наших данных.

На протяжении всего обсуждения в этой книге мы видели, что ML обладает способностью улучшать результативность, повышать ценность бизнеса и автоматизировать задачи, которые ранее выполнялись вручную. Как исследователи данных и инженеры ML, мы несем совместную ответственность за то, чтобы собираемые нами модели не оказывали неблагоприятного воздействия на эксплуатирующих их людей.

Решение

В целях урегулирования проблемной искаженности в ML нам нужны технические решения как для выявления областей вредной искаженности данных перед тренировкой модели, так и для оценивания нашей натренированной модели через призму объективности. Паттерн "Призма объективности" предоставляет подходы для создания наборов данных и моделей, которые относятся ко всем группам пользователей одинаково. Мы продемонстрируем технические приемы для обоих типов анализа с помощью инструмента с открытым исходным кодом What-If[27], который может выполняться в любых окружениях, работающих с блокнотами Python.

Прежде чем приступать к работе с описанными в этом разделе инструментами, стоит проанализировать и набор данных, и задачу предсказания, чтобы определить наличие потенциала для проблемной искаженности. Это требует более пристального рассмотрения конкретных групп людей, на которых модель будет оказывать воздействие, и того, каким образом на эти группы людей будет оказываться воздействие. Если проблемная искаженность выглядит вероятной, то описанные в этом разделе технические подходы обеспечивают хорошую отправную точку для смягчения этого типа искаженности. Если же перекос в наборе данных содержит естественную искаженность, которая не будет иметь неблагоприятных последствий для разных групп людей, то паттерн "Перебалансировка" (*см. главу 3*) предлагает технические решения для манипулирования данными, которые по своей сути являются несбалансированными.

На протяжении всего этого раздела мы будем ссылаться на публичный набор данных[28] об американских ипотечных заявках. Кредитные агентства в США обязаны сообщать информацию о заявке физического лица, такую как тип кредита, доход заявителя, обрабатывающее кредит агентство и статус заявки. Мы натренируем модель одобрения кредитной заявки на этом наборе данных, чтобы продемонстрировать разные аспекты объективности. Насколько нам известно, этот набор данных не используется ни одним кредитным агентством для тренировки ML-моделей, и поэтому поднимаемые нами красные флажки объективности являются лишь гипотетическими.

as_of_year	agency_code	loan_type	property_type	loan_purpose	occupancy	loan_amt_thousands	preapproval	county_code	applicant_income_thousands	purchaser_type	hoepa_status	lien_status	population
2016	Consumer Financial Protection Bureau (CFPB)	Conventional (any loan other than FHA, VA, FSA...	One to four-family (other than manufactured ho...	Refinancing	1	110.0	Not applicable	119.0	55.0	Freddie Mac (FHLMC)	Not a HOEPA loan	Secured by a first lien	5930.0
2016	Department of Housing and Urban Development (HUD)	Conventional (any loan other than FHA, VA, FSA...	One to four-family (other than manufactured ho...	Home purchase	1	480.0	Not applicable	33.0	270.0	Loan was not originated or was not sold in cal...	Not a HOEPA loan	Secured by a first lien	4791.0
2016	Federal Deposit Insurance Corporation (FDIC)	Conventional (any loan other than FHA, VA, FSA...	One to four-family (other than manufactured ho...	Refinancing	2	240.0	Not applicable	59.0	96.0	Commercial bank, savings bank or savings assoc...	Not a HOEPA loan	Secured by a first lien	3439.0
2016	Office of the Comptroller of the Currency (OCC)	Conventional (any loan other than FHA, VA, FSA...	One to four-family (other than manufactured ho...	Refinancing	1	76.0	Not applicable	65.0	85.0	Loan was not originated or was not sold in cal...	Not a HOEPA loan	Secured by a subordinate lien	3952.0

Рис. 7.9. Внешний вид нескольких столбцов из набора данных об ипотечных заявках в США, на который мы будем ссылаться в этом разделе

[27] См. **https://oreil.ly/Sk36z**.

[28] См. **https://oreil.ly/azFUV**.

Мы создали подмножество этого набора данных и провели некоторую предобработку, чтобы превратить его в двоичную классификационную задачу — заявка одобрена либо отклонена. На рис. 7.9 представлен внешний вид набора данных.

До тренировки

Поскольку ML-модели являются прямым представлением данных, используемых для их тренировки, существует возможность уменьшить значительный объем искаженности до строительства или тренировки модели, выполнив скрупулезный анализ данных и использовав результаты этого анализа для корректировки наших данных. В этой фазе следует сосредоточиться на выявлении искаженности сбора данных или представления данных, описанных в *разд. "Постановка задачи"*. В табл. 7.3 приведено несколько вопросов, о которых следует подумать при рассмотрении каждого типа искаженности в зависимости от типа данных.

Таблица 7.3. *Описание разных типов искаженности данных*

Тип	Определение	Соображения для анализа
Искаженность распределения данных	Данные, не содержащие равной представленности всех возможных групп, которые будут использовать модель в производстве	• Содержат ли данные сбалансированный набор примеров по всем соответствующим демографическим срезам (пол, возраст, раса, религия и т. д.)? • Содержит ли каждая метка в данных сбалансированную разбивку всех возможных вариантов этой метки? (См. пример с обувью в *разд. "Постановка задачи"*.)
Искаженность представления данных	Данные, которые хорошо сбалансированы, но не представляют разные срезы данных одинаково	• В случае классификационных моделей сбалансированы ли метки по соответствующим признакам? Например, содержатся ли в наборе данных, предназначенном для предсказания кредитоспособности, данные о равной представленности по полу, расе и другим характеристикам личности людей, отмеченных как имеющие малую вероятность возврата кредита? • Есть ли искаженность в представленности разных демографических групп в данных? Это особенно актуально для моделей, предсказывающих настроения или значение оценки. • Внесена ли разметчиками субъективная искаженность?

Рассмотрев наши данные и внеся исправления с учетом искаженности, мы должны принять в расчет те же самые соображения при разбивке данных на тренировочный, тестовый и валидационный наборы. То есть сразу после того, как наш полный набор данных будет сбалансирован, очень важно, чтобы тренировочный, тестовый и валидационный срезы поддерживали бы тот же баланс. Возвращаясь к примеру со снимками обуви, давайте представим, что мы улучшили набор данных, включив в него разные снимки 10 типов обуви. Тренировочный набор должен содержать такой же процент каждого типа обуви, как тестовый и валидационный наборы. За счет этого обеспечивается отражение моделью реальных практических сценариев и ее оценивание на таких сценариях.

Для ознакомления с тем, как этот анализ набора данных выглядит на практике, воспользуемся инструментом What-If на представленном выше наборе данных об ипо-

теке. Он позволит нам визуализировать текущий баланс наших данных по разным срезам. Инструмент What-If работает и с моделью, и без нее. Поскольку мы еще не построили модель, можем инициализировать виджет инструмента What-If, передав ему только данные:

```
config_builder = WitConfigBuilder(test_examples, column_names)
WitWidget(config_builder)
```

На рис. 7.10 мы видим окно указанного инструмента при загрузке переданных ему 1000 примеров из набора данных. Первая вкладка называется **Datapoint editor** (Редактор точек данных), она показывает внешний вид наших данных и позволяет нам просматривать отдельные примеры. В этой визуализации точки данных окрашены меткой — заявка на ипотеку одобрена или нет. Отдельный пример также выделен, и мы можем увидеть значения ассоциированных с ним признаков.

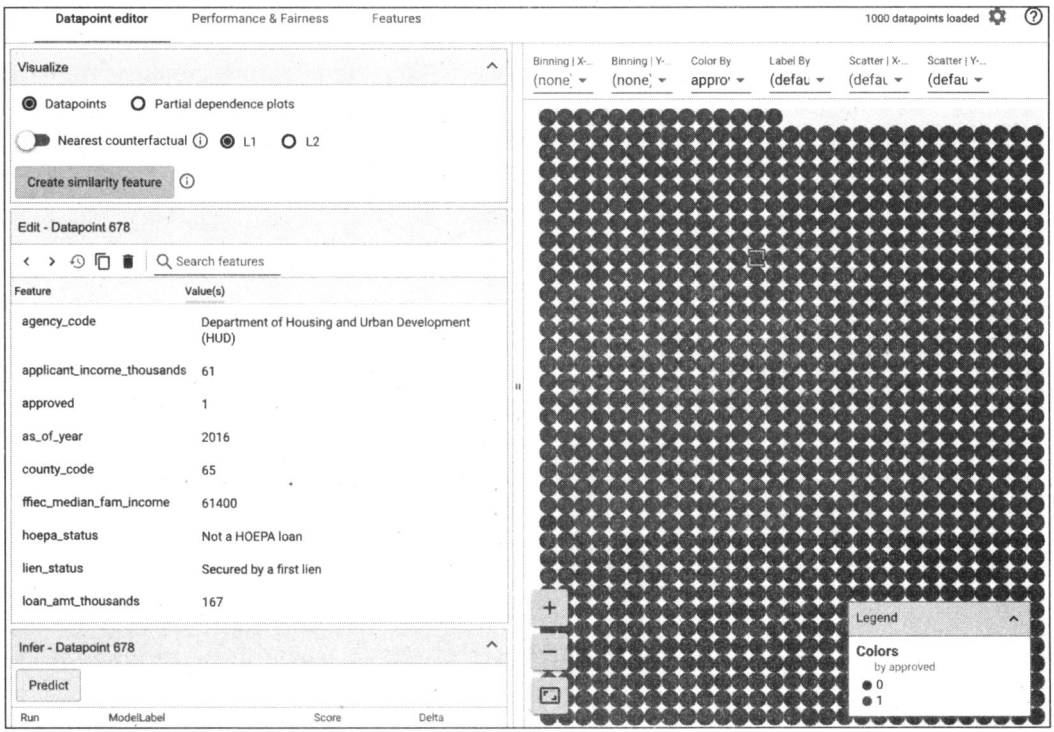


Рис. 7.10. Редактор точек данных инструмента What-If, в котором мы можем увидеть характер разбивки данных по классам меток и проинспектировать признаки в отдельных примерах из набора данных

В редакторе точек данных имеется много вариантов индивидуальной настройки визуализации, которая помогает понять, каким образом набор данных разбит на разные срезы. Сохраняя ту же цветовую кодировку по метке, после выбора столбца `agency_code` из раскрывающегося списка **Binning | Y-Axis** (Группировка в корзины | Ось Y) инструмент покажет диаграмму степени сбалансированности данных по отношению к агентству, гарантирующему кредит по каждой заявке. Это показано на рис. 7.11. Если допустить, что эти 1000 точек данных являются хорошим представ-

лением остальной части набора данных, то имеется несколько примеров потенциальной искаженности, которые можно обнаружить на рис. 7.11.

- *Искаженность представления данных.*

  Процент неодобренных заявок агентства HUD выше, чем у других агентств, представленных в наших данных. Модель, скорее всего, это усвоит, побудив ее чаще предсказывать "не одобрено" для заявок, происходящих из агентства HUD.

- *Искаженность сбора данных.*

  У нас может оказаться недостаточно данных о кредитах, происходящих из агенств FRS, OCC, FDIC или NCUA, для точного использования кода агентства (`agency_code`) в качестве модельного признака. Мы должны обеспечить, чтобы процент заявок по каждому агентству в наборе данных отражал реальные практические тренды. Например, если аналогичное число кредитов проходит через FRS и HUD, то мы должны иметь равное число примеров по каждому из этих агентств в наборе данных.

Мы можем повторить этот анализ в других столбцах данных и использовать свои выводы для добавления примеров и улучшения данных. В инструменте What-If имеется много вариантов создания пользовательских визуализаций — дополнительные идеи можно получить, обратившись к полному исходному коду[29] в репозитории на GitHub.

Рис. 7.11. Подмножество набора данных об ипотечных кредитах в США, привязанное к столбцу `agency_code` в наборе данных

[29] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/07_responsible_ai/fairness.ipynb**.

Еще один подход к пониманию данных с помощью инструмента What-If состоит в использовании вкладки **Features** (Признаки), приведенной на рис. 7.12. Она показывает несбалансированность данных по каждому столбцу в наборе данных. Глядя из нее, мы можем понять, где данные нужно добавить или удалить или изменить задачу предсказания[30]. Например, возможно, мы захотим ограничить нашу модель выполнением предсказаний только по кредитам на рефинансирование или на покупку жилья, поскольку в столбце `loan_purpose` (назначение кредита) может оказаться недостаточно данных для других возможных значений.

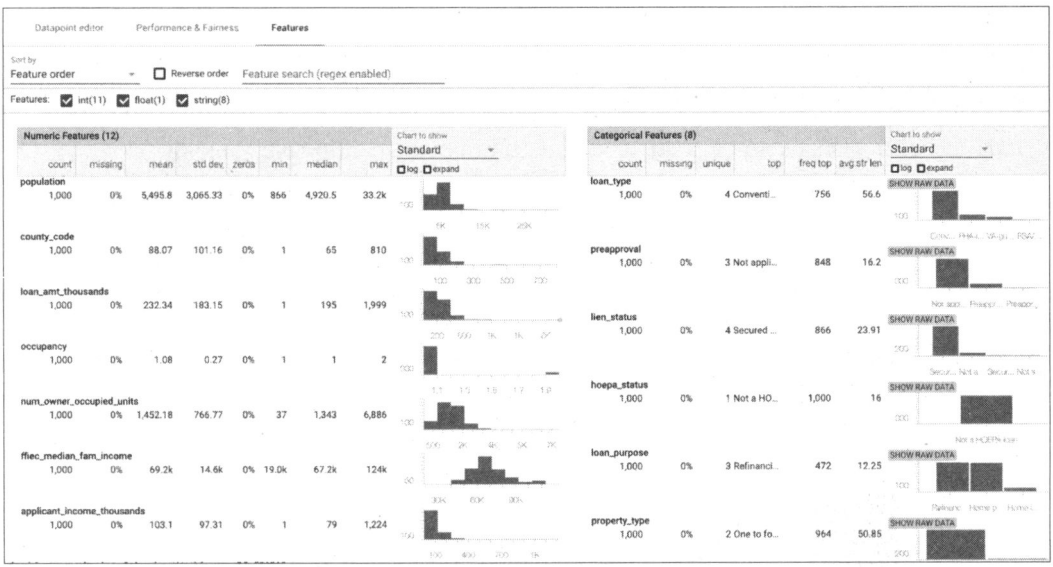


Рис. 7.12. Вкладка **Features** в инструменте What-If, которая показывает гистограммы сбалансированности набора данных по каждому столбцу

После того как мы уточнили набор данных и задачу предсказания, можем рассмотреть все остальное, доступное для оптимизации во время тренировки модели. Например, может быть, нас больше всего интересует точность модели на заявках, которые она предсказывает как "одобрено". Во время тренировки мы хотели бы выполнять оптимизацию под метрику AUC (или другую метрику) на классе "одобрено" в этой двоичной классификационной модели.

Если мы сделали все возможное, чтобы устранить искаженность сбора данных и обнаружили, что для того или иного класса недостаточно данных, то можем применить паттерн "Перебалансировка" (*см. главу 3*). В этом паттерне обсуждаются технические подходы к разработке моделей для манипулирования несбалансированными данными.

[30] Дополнительные сведения об изменении задачи предсказания *см. в разд. "Паттерн 5. Переформулировка"* и *"Паттерн 9. Нейтральный класс" главы 3.*

Искаженность в других формах данных

Хотя мы здесь показали только табличный набор данных, искаженность одинаково часто встречается и в других типах данных. Набор данных Civil Comments[31], предоставленный американским онлайновым каталогом организаций Jigsaw, является хорошим примером областей, в которых мы можем обнаружить искаженность текстовых данных. Этот набор данных помечает комментарии в соответствии с их негативностью (в интервале от 0 до 1) и используется в создании моделей для пометки негативных онлайновых комментариев. Каждый комментарий в наборе данных помечается в зависимости от присутствия одного из атрибутов личности, например упоминания религии, расы или сексуальной ориентации. Если мы планируем использовать эти данные для тренировки модели, то важно обратить внимание на искаженность представления данных. То есть термины, относящиеся к личности человека в комментарии, *не* должны влиять на негативность этого комментария, и любая такая искаженность должна быть учтена до тренировки модели.

Возьмем в качестве примера следующий вымышленный комментарий: "Mint chip is their best ice cream flavor, hands down" ("Мятные кусочки — их лучший вкус мороженого, без преувеличения"). Если бы мы заменили "Mint chip" на "Rocky road" (Каменистая дорогая), то комментарий должен быть помечен тем же баллом негативности (в идеале 0). Подобным же образом, если бы вместо этого был комментарий: "Mint chip is the worst. If you like this flavor you're an idiot" ("Мятные кусочки — худший вкус. Если он вам нравится, то вы — идиот"), то ожидаем более высокий балл негативности, и этот балл должен быть таким всякий раз, когда мы меняем "Mint chip" на другое название вкуса. В этом примере мы использовали мороженое, но легко представить, как всё обернется с более спорными терминами относительно личности человека, в особенности в наборе данных, ориентированном на человека, — хороший пример применения идеи, именуемой контрфактуальной объективностью.

После тренировки

Искаженность сможет найти свой путь в натренированную модель даже при скрупулезном анализе данных. Это может произойти вследствие архитектуры модели, метрик оптимизации или искаженности данных, которая не была выявлена до тренировки. В целях отыскания решения этой проблемы важно оценить модель с точки зрения объективности и углубиться в метрики, отличные от совокупной точности модели. Цель этого посттренировочного анализа состоит в том, чтобы понять компромиссы между точностью модели и влиянием предсказаний модели на разные группы людей.

Инструмент What-If является одним из таких вариантов постмодельного анализа. Для иллюстрации его использования на натренированной модели мы будем отталкиваться от нашего примера набора данных об ипотечных кредитах. Основываясь на нашем предыдущем анализе, мы уточнили набор данных, включив в него креди-

[31] См. **https://oreil.ly/xaocx**.

ты только для целей рефинансирования или покупки жилья[32], и натренировали модель XGBoost предсказывать возможность одобрения заявки. Поскольку мы используем XGBoost, мы конвертировали все категориальные признаки в булевы столбцы с помощью метода `get_dummies()` библиотеки pandas.

Мы внесем несколько дополнений в приведенный выше исходный код инициализации инструмента What-If, на этот раз передав функцию, которая вызывает натренированную модель наряду с конфигурацией, задающей столбец меток и имя для каждой метки:

```
def custom_fn(examples):
    df = pd.DataFrame(examples, columns=columns)
    preds = bst.predict_proba(df)
    return preds

config_builder = (WitConfigBuilder(test_examples, columns)
    .set_custom_predict_fn(custom_fn)
    .set_target_feature('mortgage_status')
    .set_label_vocab(['denied', 'approved']))
WitWidget(config_builder, height=800)
```

Теперь, когда мы передали нашу модель в инструмент, результирующая визуализация строит график тестовых точек данных в соответствии с показанной на оси *y* уверенностью модели в предсказании (рис. 7.13).

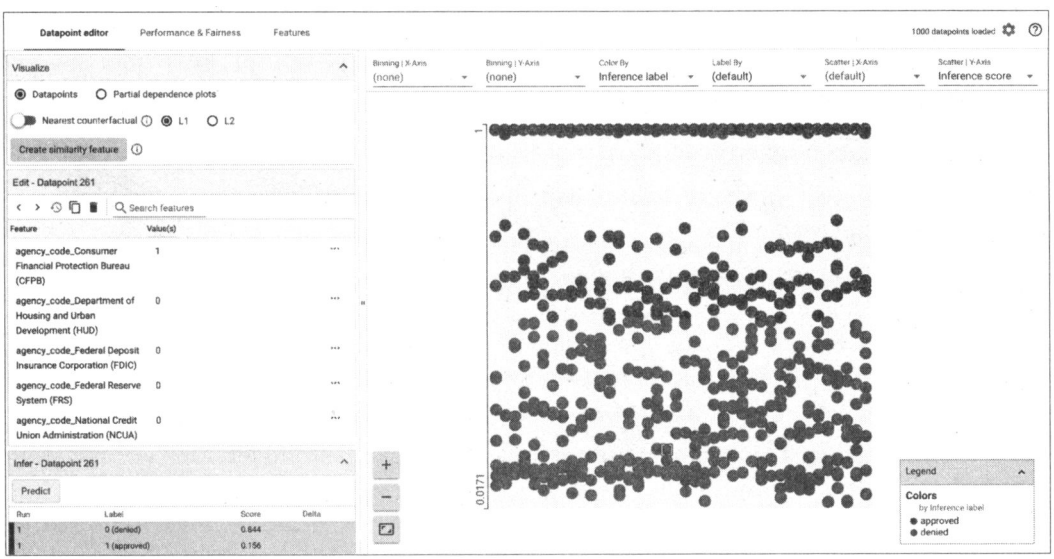


Рис. 7.13. Редактор точек данных инструмента What-If для двоичной классификационной модели. Ось *y* — это результат предсказания модели для каждой точки данных в диапазоне от 0 (отклонено) до 1 (одобрено)

[32] На этом наборе данных можно было бы выполнить еще много других предтренировочных оптимизаций. Мы выбрали только одну из них в качестве демонстрации того, что возможно.

Вкладка **Performance & Fairness** (Результативность и объективность) инструмента What-If позволяет оценивать объективность модели по разным срезам данных. Выбрав один из признаков модели для **Slice by** (Нарезать по), мы можем сравнить результаты работы модели для различных значений этого признака. На рис. 7.14 показано, что мы нарезали набор данных по признаку `agency_code_HUD` — булеву значению, указывающему на факт гарантирования заявки агентством HUD (0 для кредитов не под надзором HUD, 1 для кредитов под надзором HUD).

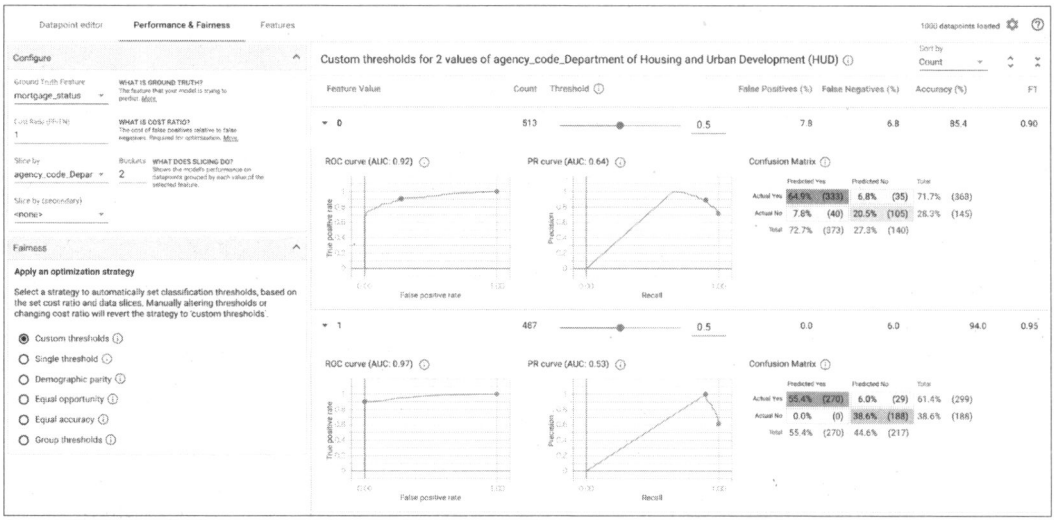


Рис. 7.14. Вкладка **Performance & Fairness** инструмента What-If показывает результативность нашей модели XGBoost в условиях разных значений признака

Из этих графиков результативности и объективности можно увидеть следующее:

- точность модели на кредитах под надзором агентства HUD значительно выше — 94% по сравнению с 85%;
- согласно матрице путаницы кредиты не под надзором агентства HUD утверждаются с более высокой частотой — 72% по сравнению с 55%. Это, вероятно, связано с искаженностью представления данных, выявленной в предыдущем разделе (мы намеренно оставили набор данных таким, чтобы показать, каким образом модели могут усиливать искаженность данных).

На эти сведения можно реагировать, используя несколько подходов, как показано в блоке стратегии оптимизации[33] на рис. 7.14. Указанные методы оптимизации предусматривают изменение *классификационного порога* модели — порога, при котором модель будет выдавать положительную классификацию. В контексте этой модели возникает вопрос: какой порог уверенности нас устроит для пометки заявки как "одобренной"? Если наша модель более чем на 60% уверена в том, что заявка должна быть одобрена, должны ли мы ее одобрить? Или же мы одобряем заявки

[33] В левом нижнем углу на рис. 7.14 (**Apply an optimizaion strategy**). — *Прим. ред.*

только тогда, когда модель уверена более чем на 98%? Это решение в значительной степени зависит от контекста модели и задачи предсказания. Если мы предсказываем наличие на снимке кошки, то нас, возможно, устроит, если будет возвращаться метка "кошка", даже если модель уверена только на 60%. Однако если у нас модель, которая предсказывает наличие заболевания на медицинском снимке, то, вероятно, наш порог должен быть намного выше.

Инструмент What-If помогает нам выбирать порог, основываясь на разных оптимизациях. Выбор в пользу "демографического паритета", например, обеспечил бы, чтобы наша модель соответствовала одинаковому проценту заявок как на кредиты под надзором агентства HUD, так и на кредиты под надзором любого иного агентства, кроме HUD[34]. С другой стороны, использование метрики равенства возможностей[35] как метрики объективности будет обеспечивать, чтобы точкам данных из среза HUD и из среза не-HUD со значением эмпирического наблюдения "одобрено" в тестовом наборе данных давались равные шансы быть предсказанными моделью "одобрено".

Обратите внимание, что изменение предсказательного порога модели является лишь одним из подходов к реагированию на метрики оценивания объективности. Существует много других подходов, включая перебалансировку тренировочных данных, перетренировку модели для оптимизации под иную метрику и многое другое.

Инструмент What-If не зависит от модели и может использоваться для любого типа модели, без учета архитектуры или фреймворка. Он работает с моделями, загруженными в блокнот или в инструмент визуализации TensorBoard[36], моделями, запросы к которым обслуживаются посредством TensorFlow Serving, и моделями, развернутыми на платформе Cloud AI Platform Prediction. Коллектив разработчиков What-If также создал инструмент обеспечения языковой интерпретируемости для текстовых моделей под названием Language Interpretability Tool (LIT)[37].

Еще одним важным аспектом посттренировочного оценивания является тестирование модели на сбалансированном наборе примеров. Если есть какие-либо срезы данных, которые, как мы предполагаем, будут проблемными для модели — например, входные данные, на которые может повлиять искаженность сбора данных или представления, — то мы должны обеспечить, чтобы наш тестовый набор включал достаточное число таких случаев. После разбивки данных мы будем использовать тот же тип анализа, который мы использовали в *разд. "До тренировки" ранее в этой главе* по каждому срезу наших данных: тренировочному, валидационному и тестовому.

---

[34] В этой статье (см. **https://oreil.ly/wFx_W**) более подробно рассматриваются настройки инструмента What-If для стратегий оптимизации объективности.

[35] Более подробную информацию о равенстве возможностей как метрике объективности можно найти по ссылке **https://oreil.ly/larIS**.

[36] См. **https://oreil.ly/xWV4_**.

[37] См. **https://oreil.ly/CZ60B**.

Как видно из этого анализа, универсального технического решения или метрики оценивания для обеспечения объективности модели не существует. Это непрерывный, итеративный процесс, который должен использоваться на протяжении всего рабочего ML-потока целиком — от сбора данных до развернутой модели.

Компромиссы и альтернативы

В дополнение к обсуждаемым в *разд. "Решение"* пред- и посттренировочным техническим приемам обеспечения объективности есть много других подходов. Здесь мы представим несколько альтернативных инструментов и процессов для достижения объективных моделей. Объективность ML — это быстро эволюционирующая область научных исследований — инструменты, включенные в этот раздел, вовсе не означают, что они будут представлять исчерпывающий список. Это лишь несколько технических приемов и инструментов, доступных в настоящее время для улучшения объективности моделей. Мы также обсудим различия между паттернами "Призма объективности" и "Объяснимые предсказания", поскольку они связаны между собой и часто используются вместе.

Инструментарий Fairness Indicators

Fairness Indicators[38] (FI) — это комплект инструментов с открытым исходным кодом, предназначенных для понимания распределения набора данных до тренировки и оценивания результативности модели с использованием метрик объективности. Fairness Indicators содержит инструмент валидации данных TensorFlow Data Validation (TFDV) и инструмент модельного анализа TensorFlow Model Analysis (TFMA). Указанный инструментарий чаще всего используется в качестве компонентов в конвейерах TFX (подробности *см. в разд. "Паттерн 25. Конвейер рабочего потока" главы 6*) или посредством инструмента визуализации TensorBoard. В TFX есть два предварительно построенных компонента, в которых задействуются инструменты комплекта Fairness Indicators:

- ExampleValidator для анализа данных, обнаружения дрейфа и асимметрии между тренировкой и обслуживанием с TFDV;
- оценщик, который использует библиотеку TFMA для оценивания модели по всем подмножествам набора данных. Пример интерактивной визуализации, сгенерированной из TFMA, показан на рис. 7.15. Он берет один признак из данных (высоту) и разбивает частоту ложноотрицательных результатов модели для каждого возможного категориального значения этого признака.

Библиотека TFMA из Python-пакета Fairness Indicators[39] также может использоваться в качестве автономного инструмента, который работает как с тензорными, так и с нетензорными моделями.

---

[38] См. **https://github.com/tensorflow/fairness-indicators**.

[39] См. **https://oreil.ly/pYM1j**.

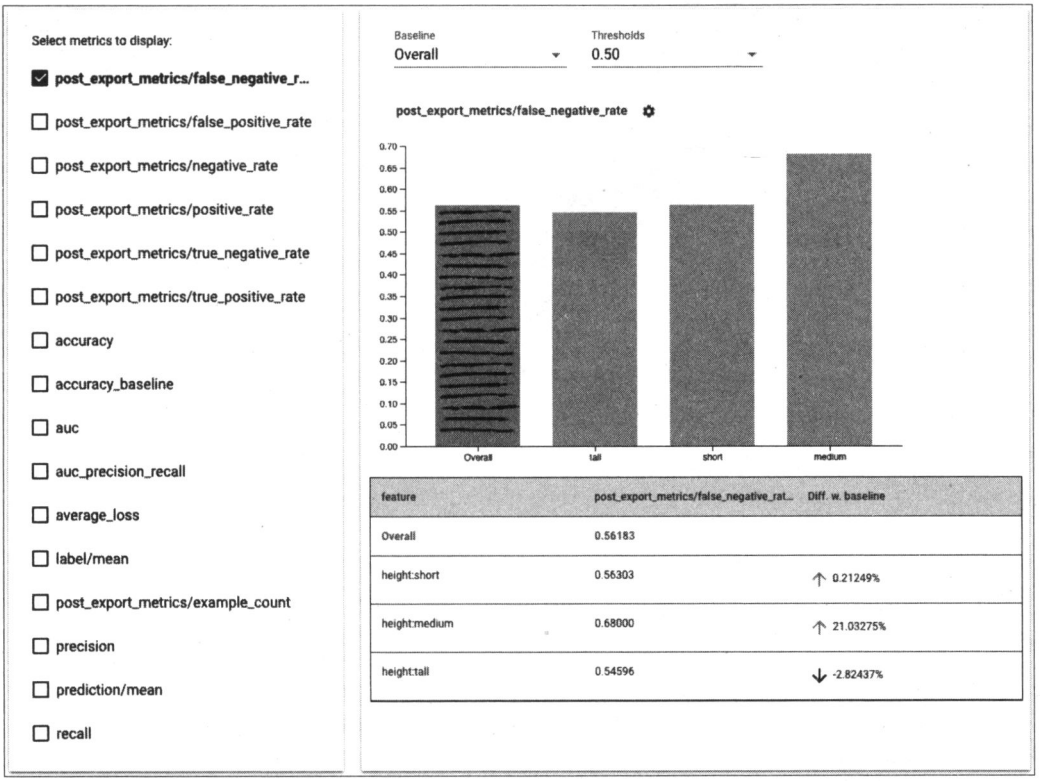


Рис. 7.15. Сравнение частоты ложноотрицательных результатов модели по разным подмножествам данных

Автоматизирование оценивания данных

Методы оценивания объективности, которые мы обсуждали в *разд. "Решение"*, были сосредоточены на ручном, интерактивном анализе данных и моделей. Этот тип анализа важен, в особенности в начальных фазах разработки модели. По мере того как мы вводим нашу модель в действие и смещаем акцент на ее техническое сопровождение и совершенствование, отыскание подходов к автоматизированию оценивания объективности будет повышать эффективность и обеспечит интеграцию объективности в наш рабочий поток ML. Это можно делать посредством обсуждаемого паттерна "Непрерывное оценивание модели" (*см. главу 5*) либо с помощью паттерна "Конвейер рабочего потока" (*см. главу 6*), используя компоненты, подобные тем, которые предоставлены в TFX для анализа данных и оценивания моделей.

Списки разрешений и запретов

Когда мы не можем напрямую отыскать подход к исправлению проблемы внутренне присущей искаженности данных или модели, есть возможность жестко закодировать правила поверх нашей производственной модели, используя списки разрешений и запретов. Это относится главным образом к классификационным или генеративным моделям, когда есть метки или слова, которые, по нашему мнению,

модель не должна возвращать. Например, гендерные слова, такие как "мужчина" и "женщина", были удалены[40] из функционала API обнаружения меток службы Google Cloud Vision. Поскольку гендер невозможно определить только по внешнему виду, возвращение этих меток, когда модельное предсказание основывается исключительно на визуальных признаках, только усиливало бы необъективные искажения. Вместо этого Vision API возвращает "лицо" (person). Аналогично функционал "Умный ввод" (Smart Compose) в почтовом клиенте Gmail позволяет избегать использования гендерных местоимений[41] при завершении предложений, как в предложении "Я встречаюсь с инвестором на следующей неделе. Вы хотите встретиться с ___?"

Эти списки разрешений и запретов могут применяться в одной из двух фаз рабочего потока ML.

- *Сбор данных.*

  Во время тренировки модели с нуля или использовании паттерна "Трансферное обучение" с целью добавления собственного классификационного слоя мы можем определить набор меток модели в фазе сбора данных, до того как модель будет натренирована.

- *После тренировки.*

  Если в предсказаниях мы опираемся на предварительно натренированную модель и используем те же модельные метки, то список разрешений и запретов может быть реализован в производстве — после того, как модель вернет предсказание, но до того, как эти метки будут показаны конечным пользователям. Это также может применяться к моделям генерации текста, в которых мы не имеем полного контроля над всеми возможными результатами работы модели.

Аугментация данных

В дополнение к рассмотренным ранее техническим решениям по распределению и представлению данных еще один подход к минимизированию искаженности модели заключается в аугментировании данных. Согласно этому подходу данные перед тренировкой изменяются с целью устранения потенциальных источников искаженности. Один специфический тип аугментации данных называется абляцией (размыванием) и особенно применим в текстовых моделях. Например, в текстовой модели анализа настроений мы могли бы удалить из текста термины личности человека для устранения их влияния на предсказания нашей модели. Отталкиваясь от примера с мороженым, который мы использовали ранее в этом разделе, предложение "Мятные кусочки — это их лучший вкус мороженого" после применения абляции станет "ПРОБЕЛ — их лучший вкус мороженого". Далее мы заменили бы все другие слова в наборе данных, которые, по нашему мнению, не должны влиять на модельное предсказание настроений, одним и тем же словом (здесь мы использовали ПРОБЕЛ,

---

[40] См. **https://oreil.ly/WY2vp**.

[41] См. **https://oreil.ly/dtMhK**.

но будет работать все, что не присутствует в остальных текстовых данных). Обратите внимание, что хотя указанный метод абляции хорошо работает для многих текстовых моделей, важно быть осторожным при удалении областей искаженности из табличных наборов данных, как упоминалось в *разд. "Постановка задачи"*.

Еще один подход на основе аугментации данных предусматривает генерирование новых данных, и он был использован в Google Переводчике для минимизирования гендерной искаженности[42] при переводе текста на гендерно-нейтральные и гендерно-специфичные языки. Техническое решение предусматривало переписывание данных перевода таким образом, чтобы при необходимости предоставляемый перевод предлагался как в женском, так и в мужском варианте. Например, гендерно-нейтральное английское предложение "We are doctors" ("Мы — врачи") даст два результата при переводе на испанский (рис. 7.16). В испанском языке слово we может иметь как женский, так и мужской род.

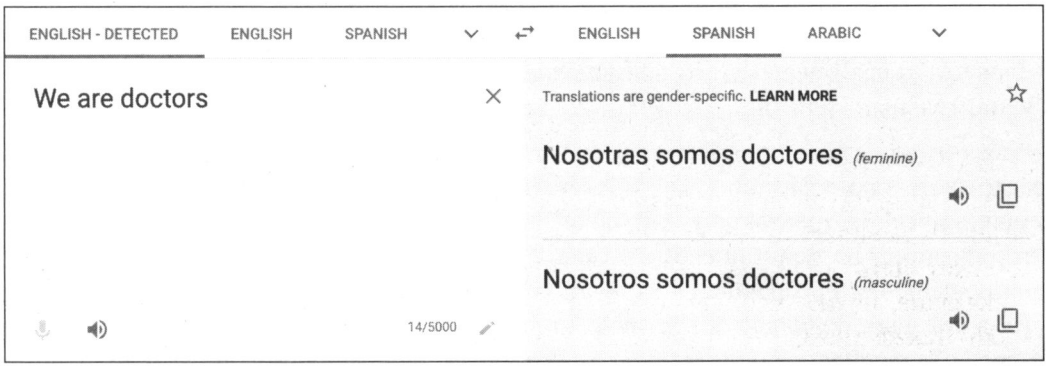


Рис. 7.16. При переводе гендерно-нейтрального слова на одном языке (здесь слово we в английском языке) на язык, где это слово является гендерно-специфичным, Google Переводчик теперь предоставляет несколько переводов, чтобы можно было свести к минимуму гендерную искаженность

Модельные карточки

В одной научно-исследовательской работе[43] были продемонстрированы модельные карточки, которые обеспечивают каркас для представления информации о возможностях и ограничениях модели. Цель модельных карточек состоит в том, чтобы повышать прозрачность модели, предоставляя подробную информацию о сценариях, в которых модель должна и не должна использоваться, поскольку смягчение проблемной искаженности работает только в том случае, если модель используется так, как она была задумана. При таком подходе модельные карточки поощряют ответственность за использование модели в правильном контексте.

Первые выпущенные модельные карточки[44] содержат сводки и метрики объективности для признаков распознавания лиц и обнаружения объектов в API службы

[42] См. **https://oreil.ly/3Rkdr**.

[43] См. **https://oreil.ly/OAIcs**.

[44] См. **https://oreil.ly/OwiJY**.

Vision инфраструктуры Google Cloud. В целях генерирования модельных карточек для наших собственных ML-моделей фреймворк TensorFlow предоставляет инструментарий Model Card Toolkit[45] (MCT), который может выполняться как автономная библиотека Python или в рамках конвейера TFX. Указанный инструментарий читает экспортированные модельные активы и генерирует серию диаграмм с различными метриками результативности и объективности.

Объективность против объяснимости

Понятия объективности и объяснимости в ML иногда путают, поскольку они нередко используются вместе и являются частью более широкой инициативы под названием "ответственный искусственный интеллект" (responsible AI). Объективность относится конкретно к выявлению и устранению искаженности из моделей, а объяснимость — это один из подходов к диагностированию наличия искаженности. Например, применение объяснимости к модели анализа настроений может показать, что в своем выведении предсказаний модель опирается на термины личности человека, когда вместо этого она должна использовать такие слова, как "худший", "удивительный" или "не".

Объяснимость также может использоваться вне контекста объективности, чтобы раскрывать такие факторы, как объяснения, почему модель помечает те или иные мошеннические транзакции или пикселы, которые побудили модель предсказать "заболевание" на медицинском снимке. Следовательно, объяснимость — это метод для повышения прозрачности модели. Иногда прозрачность может выявлять области, в которых модель необъективно относится к некоторым группам, но она также может обеспечивать более глубокое понимание процесса принятия решений моделью.

Резюме

Хотя Питер Паркер (Peter Parker), возможно, не имел в виду машинное обучение, когда сказал, что "с большой мощью приходит большая ответственность", эта цитата, безусловно, здесь применима. ML обладает способностью разрушать отрасли, повышать продуктивность и генерировать новые идеи на основе данных. Учитывая этот потенциал, особенно важно понимать, как наши модели будут влиять на разные группы заинтересованных лиц. Такими лицами могут быть различные демографические срезы пользователей моделей, регуляторные группы, команды исследователей данных или бизнес-группы внутри организации.

Описанные в этой главе паттерны ответственного ИИ являются существенной частью каждого рабочего потока ML — они помогают нам лучше понимать генерируемые нашими моделями предсказания и улавливать потенциальное неблагоприятное поведение до того, как модели будут запущены в промышленное исполь-

[45] См. **https://github.com/tensorflow/model-card-toolkit**.

зование. Начав с паттерна "Эвристический эталон" (Heuristic Benchmark), мы рассмотрели принцип выявления начальной метрики, которая служит для оценивания модели. Эта метрика полезна в качестве точки сравнения для понимания последующих версий модели и резюмирования поведения модели для лиц, принимающих решения. В паттерне "Объяснимые предсказания" (Explainable Predictions) мы продемонстрировали способы использования признаковых атрибуций, чтобы ознакомиться с тем, какие признаки являются наиболее важными в подаче сигналов о предсказании модели. Признаковая атрибуция является одним из видов метода обеспечения объяснимости и может использоваться для оценивания предсказания как на одном примере, так и по группе тестовых входных данных. Наконец, в паттерне "Призма объективности" (Fairness Lens) были представлены инструменты и метрики для обеспечения того, чтобы предсказания модели относились ко всем группам пользователей в ключе, который является объективным, равноправным и неискаженным.

ГЛАВА 8

Взаимосвязанность паттернов

Мы задались целью сформировать каталог паттернов машинного обучения, т. е. технических решений задач, повторяющихся при создании архитектуры и выполнении тренировки, развертывания моделей и конвейеров машинного обучения. В этой главе представлен краткий справочник по указанному перечню паттернов.

Мы сгруппировали паттерны в книге с точки зрения того, где они будут использоваться в типичном рабочем потоке процессов ML. Именно поэтому у нас была глава о представлении входных данных и еще одна — об отборе модели. Затем мы обсудили паттерны, которые модифицируют типичный цикл тренировки и делают предсказательный вывод отказоустойчивее. Мы закончили паттернами, которые способствуют ответственному использованию систем ML. Такое изложение сравнимо с тем, как организована книга рецептов, в которой есть отдельные разделы о закусках, супах, первых блюдах и десертах. Подобная организация, однако, может вызывать затруднения при определении того, когда и какой суп выбирать и какие десерты хорошо сочетаются с тем или иным основным блюдом. Поэтому в данной главе мы также прольем свет на то, каким образом паттерны связаны друг с другом. Наконец, мы также составим "диеты", обсудив тему взаимодействия паттернов в распространенных категориях задач ML.

Справочник паттернов

Мы обсудили много разных паттернов и принципы их использования для решения распространенных задач, которые возникают в машинном обучении из раза в раз. Вот краткое их описание (табл. 8.1).

Таблица 8.1. *Перечень представленных паттернов*

Глава	Паттерн	Решаемая задача	Решение
Паттерны для представления данных	Хешированный признак (Hashed Feature)	Задачи, связанные с категориальными признаками, такими как неполный словарь, размер модели вследствие кардинальности и холодный пуск	Группировать детерминированный и переносимый хеш строкового значения на корзины и делать уступку в пользу наличия коллизий в представлении данных

***Таблица 8.1** (продолжение)*

Глава	Паттерн	Решаемая задача	Решение
	Векторные вложения (Embeddings)	Признаки высокой кардинальности, в которых важно сохранить отношения близости	Усваивать представление данных, которое отображает данные высокой кардинальности, в низкоразмерное пространство таким образом, что сохранится информация, относящаяся к задаче обучения
	Синтетический признак (Feature Cross)	Сложность модели недостаточна для обучения связей между признаками	Помогать моделям быстрее усваивать связи между входными переменными, в явной форме делая каждую комбинацию входных значений отдельным признаком
	Мультимодальный вход (Multimodal Input)	Как выбрать между несколькими потенциальными представлениями данных	Конкатенировать все имеющиеся представления данных
Паттерны для представления задачи	Переформулировка (Reframing)	Несколько задач, включая уверенность в численном предсказании, порядковых категориях, ограничении диапазона предсказания и мультиметочном обучении	Менять представление выходных данных в задаче машинного обучения; например, представлять регрессионную задачу в качестве классификационной (и наоборот)
	Мультиметка (Multilabel)	К данному тренировочному примеру применяется более одной метки	Кодировать метку, используя массив значений с несколькими активными состояниями, и использовать *k* сигмоид в качестве выходного слоя
	Ансамбли (Ensembles)	Компромисс между смещенностью и дисперсией в задачах малого и среднего масштаба	Совмещать несколько моделей машинного обучения и агрегировать их результаты в целях генерирования предсказаний
	Каскад (Cascade)	Проблемы технической сопроводимости или дрейфа, когда ML-задача разбивается на ряд ML-подзадач	Трактовать систему машинного обучения как единый рабочий поток для процессов тренировки, оценивания и предсказания
	Нейтральный класс (Neutral Class)	Метка класса для некоторого подмножества примеров по существу является произвольной	Вводить дополнительную метку для классификационной модели, не пересекающуюся с текущими метками
	Перебалансировка (Rebalancing)	Сильно несбалансированные данные	Выполнять понижающий отбор, повышающий отбор либо использовать функцию взвешенной потери в зависимости от разных соображений
Паттерны для тренировки моделей	Полезное переобучение (Useful Overfitting)	Использование методов машинного обучения для обучения физической модели или динамической системы	Отказываться от обычных технических приемов обобщения ради намеренной переподгонки на тренировочном наборе данных
	Контрольные точки (Checkpoints)	Потеря несохраненных результатов работы во время длительных тренировочных заданий из-за аварийного отказа машины	Периодически сохранять полное состояние модели, чтобы частично натренированные модели были доступны и могли использоваться для возобновления тренировки не с нуля, а с промежуточной точки

***Таблица 8.1** (продолжение)*

Глава	Паттерн	Решаемая задача	Решение
	Трансферное обучение (Transfer Learning)	Отсутствие крупных наборов данных, необходимых для тренировки сложных моделей машинного обучения	Брать часть ранее натренированной модели, замораживать веса и использовать эти нетренируемые слои в новой модели, которая решает аналогичную задачу
	Распределительная стратегия (Distribution Strategy)	Тренировка крупных нейронных сетей может занимать очень много времени, что замедляет экспериментирование	Выполнять цикл тренировки в требуемом масштабе на нескольких воркерах, используя преимущества кэширования, аппаратного ускорения и параллелизации
	Гиперпараметрическая настройка (Hyperparameter Tuning)	Как определять оптимальные гиперпараметры модели машинного обучения	Вставлять цикл тренировки в метод оптимизации с целью отыскания оптимального набора модельных гиперпараметров
Паттерны для отказоустойчивой обработки	Функция обслуживания без поддержки состояния (Stateless Serving Function)	Производственная система машинного обучения должна быть способна синхронно обрабатывать от тысяч до миллионов предсказательных запросов в секунду	Экспортировать ML-модель как функцию без поддержки состояния, чтобы многочисленные клиенты имели возможность ее использовать совместно масштабируемым образом
	Пакетное обслуживание (Batch Serving)	Выполнение модельных предсказаний на крупных объемах данных с использованием конечной точки, предназначенной для обработки запросов по одному за раз, будет переполнять модель	Использовать программно-информационную инфраструктуру, обычно предназначенную для распределенной обработки данных, призванную выполнять предсказательный вывод асинхронно на крупном числе экземпляров одномоментно
	Непрерывное оценивание модели (Continued Model Evaluation)	Результативность развернутых моделей со временем деградирует либо из-за дрейфа данных, либо из-за дрейфа концепции, либо из-за других изменений в конвейерах, которые подают данные в модель	Обнаруживать момент времени, когда развернутая модель больше не соответствует целевому назначению, постоянно мониторя предсказания модели и оценивая ее результативность
	Двухфазные предсказания (Two-Phase Predictions)	Крупные, сложные модели должны оставаться результативными, когда они развертываются на периферии или на распределенных устройствах	Разделять вариант использования на две фазы, при этом на периферии выполняется только более простая фаза
	Предсказания по ключу (Keyed Predictions)	Как соотносить предсказания на выходе из модели с соответствующими данными на входе в модель при отправке крупных заданий по генерированию предсказаний	Разрешать модели во время предсказания пропускать в сквозном порядке поддерживаемый клиентом ключ, который может использоваться для соединения данных на входе в модель с модельными предсказаниями на выходе из нее

***Таблица 8.1** (продолжение)*

Глава	Паттерн	Решаемая задача	Решение
Паттерны обеспечения воспроизводимости	Преобразователь (Transform)	Данные на входе в модель должны быть преобразованы для создания ожидаемых моделью признаков, и этот процесс должен быть согласованным между тренировкой модели и обработкой запросов	В явной форме захватывать и сохранять преобразования, применяемые для конвертирования входных модельных данных в признаки
	Повторяемая разбивка (Repeatable Splitting)	При создании срезов данных важно иметь метод, который остается легковесным и воспроизводимым независимо от языка программирования или случайных начальных чисел	Выявлять столбец, который улавливает корреляционную связь между строками, и использовать алгоритм хеширования Farm Fingerprint для разбивки имеющихся данных на тренировочный, валидационный и тестовый наборы данных
	Мостовая схема (Bridged Schema)	По мере появления новых данных любые изменения в схеме данных могут помешать использованию как новых, так и старых данных для перетренировки	Адаптировать данные из старой, изначальной схемы данных в соответствии со схемой новых, более совершенных данных
	Оконный предсказательный вывод (Windowed Inference)	Некоторые модели требуют непрерывно обновляемой последовательности экземпляров для выполнения предсказательного вывода, либо признаки должны агрегироваться во временно́м окне таким образом, чтобы избегать асимметрии между тренировкой и обработкой запросов	Экстернализировать состояние модели и вызывать модель из конвейера потоковой аналитики с целью обеспечения того, чтобы признаки, вычисленные динамическим, зависящим от времени способом, могли правильно повторяться между тренировкой модели и обработкой запросов
	Конвейер рабочего потока (Workflow Pipeline)	При масштабировании рабочего потока ML-процессов выполнять испытания независимо и отслеживать производительность для каждого шага конвейера	Делать каждый шаг рабочего потока отдельной контейнеризированной службой, пригодной для выстраивания цепочки служб, создавая конвейер, который можно запускать одним вызовом REST API
	Хранилище признаков (Feature Store)	Нерегламентированный подход к генерированию признаков замедляет разработку модели и приводит к дублированию усилий между коллективами, а также к неэффективности потока работ	Создавать хранилище признаков, централизованное место для хранения и документирования наборов признаковых данных, которые будут использоваться во время строительства ML-моделей и могут использоваться совместно в разных проектах и коллективах
	Управление версиями (Versioning)	Трудно проводить мониторинг результативности и раздельное тестирование модели, имея одну-единственную модель в производстве, либо обновлять модели, не нарушая работу существующих пользователей	Развертывать измененную модель в качестве микросервиса с отдельным адресом REST API для достижения обратной совместимости развернутых моделей

Таблица 8.1 (окончание)

Глава	Паттерн	Решаемая задача	Решение
Ответственный искусственный интеллект	Эвристический эталон (Heuristic Benchmark)	Объяснение результативности модели с помощью сложных метрик оценивания не дает интуитивного понимания, необходимого лицам, принимающим бизнес- решения	Сравнивать ML-модель с простой и понятной эвристикой
	Объяснимые предсказания (Explainable Predictions)	Иногда для отладки либо в целях регулирования и надзора за соблюдением стандартов необходимо знать причину, почему модель делает те или иные предсказания	Применять технические приемы обеспечения объяснимости моделей, чтобы понимать, как и почему модели делают предсказания, и повышать доверие пользователей к системам машинного обучения
	Призма объективности (Fairness Lens)	Искаженность может приводить к тому, что ML-модели не будут одинаково относиться ко всем пользователям и могут иметь неблагоприятные последствия для некоторых групп населения	Использовать инструменты выявления искаженностей наборов данных перед тренировкой и оценивать натренированные модели через призму объективности с целью обеспечения объективности модельных предсказаний для разных групп пользователей и разных сценариев

Взаимодействие паттернов

Паттерны не существуют изолированно. Многие из них тесно связаны друг с другом прямо или косвенно и часто дополняют друг друга. Диаграмма взаимодействия на рис. 8.1 резюмирует взаимозависимости и некоторые связи между разными паттернами. Если вы обнаружите, что используете некоторый паттерн, то вы, возможно, только выиграете при мысли о том, как могли бы встроить другие паттерны, которые с ним связаны.

Мы рассмотрим несколько путей, которыми эти паттерны связаны между собой, и приемы их использования вместе при разработке полного технического решения. Например, во время работы с категориальными признаками паттерн "Хешированный признак" (Hashed Feature) можно совместить с паттерном "Векторные вложения" (Embeddings). Оба паттерна сотрудничают в урегулировании проблемы модельных входных данных высокой кардинальности, таких как при работе с текстом. В TensorFlow это демонстрируется обертыванием признакового столбца `categorical_column_with_hash_bucket` (категориальный столбец с хеш-корзиной) признаковым столбцом `embedding` для конвертирования разреженной категориальной текстовой входной переменной в плотное представление:

```
import tensorflow.feature_column as fc
keywords = fc.categorical_column_with_hash_bucket("keywords", hash_bucket_size=10K)
keywords_embedded = fc.embedding_column(keywords, num_buckets=16)
```

При обсуждении векторных вложений мы убедились, что этот технический прием рекомендуется при использовании паттерна "Синтетический признак" (Feature

Cross). Паттерн "Хешированный признак" (Hashed Feature) идет рука об руку с "Повторяемой разбивкой" (Repeatable Splitting), поскольку алгоритм хеширования Farm Fingerprint можно применять для разбивки данных. А при использовании паттерна "Хешированный признак" или "Векторные вложения" принято обращаться к концепциям "Гиперпараметрической настройки" (Hyperparameter Tuning), чтобы определить оптимальное число хеш-корзин или правильную размерность векторного вложения.

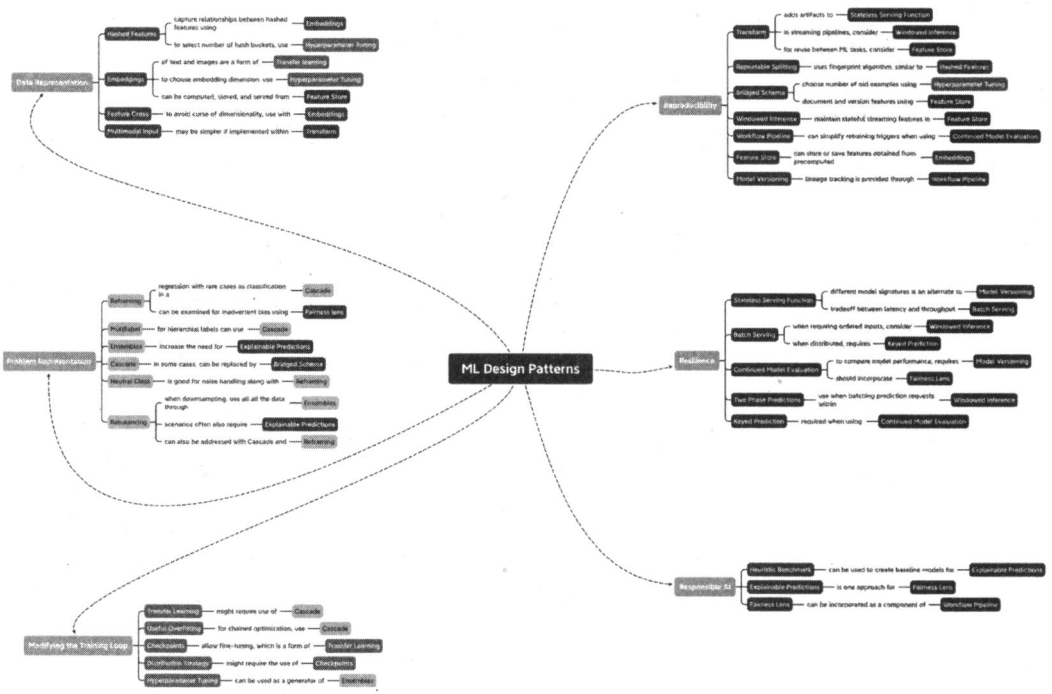


Рис. 8.1. Многие из рассмотренных в этой книге паттернов связаны между собой либо могут использоваться вместе (схема имеется в репозитории на GitHub[1] этой книги)

На самом деле, паттерн "Гиперпараметрическая настройка" является традиционной частью рабочего потока процессов машинного обучения и часто используется в сочетании с другими паттернами. Например, мы могли бы применить гиперпараметрическую настройку для определения числа старых примеров, которые надлежит использовать, если реализуем паттерн "Мостовая схема" (Bridged Schema). При гиперпараметрической настройке важно учитывать и то, как мы задали контрольные точки модели, используя виртуальные эпохи и распределенную тренировку. Между тем, паттерн "Контрольные точки" (Checkpoints) естественным образом связан с паттерном "Трансферное обучение" (Transfer Learning), поскольку более ранние контрольные точки модели часто используются во время тонкой настройки.

[1] См. **https://github.com/GoogleCloudPlatform/ml-design-patterns**.

Векторные вложения возникают в самых разных частях машинного обучения, поэтому существует масса путей взаимодействия паттерна "Векторные вложения" с другими паттернами. Возможно, наиболее примечательным является трансферное обучение, поскольку выходные данные, генерируемые из промежуточных слоев предварительно натренированной модели, по существу являются усвоенными векторными вложениями признаков. Мы также убедились, как встраивание паттерна "Нейтральный класс" (Neutral Class) в классификационную модель естественным путем либо посредством паттерна "Переформулировка" (Reframing) улучшает эти усвоенные векторные вложения. Далее, если эти векторные вложения используются в модели как признаки, то было бы выгодно сохранять их с помощью паттерна "Хранилище признаков" (Feature Store), чтобы обеспечивать их легкую доступность и возможность управления версиями. Как вариант — в трансферном обучении результат на выходе из предварительно натренированной модели можно рассматривать в качестве входных данных паттерна "Каскад" (Cascade).

Мы также увидели, как можно задействовать паттерн "Перебалансировка" (Rebalancing) путем совмещения двух других: "Переформулировка" и "Каскад". Паттерн "Переформулировка" позволил бы нам представлять несбалансированный набор данных как двоичную классификацию между "нормальным" значением и "выбросным". Тогда данные на выходе из этой модели передавались бы во вторичную регрессионную модель, оптимизированную под предсказание на любом распределении данных. Эти паттерны, вероятно, также приведут к паттерну "Объяснимые предсказания" (Explainable Predictions), поскольку во время работы с несбалансированными данными особенно важно получать подтверждения о том, что модель реагирует на правильные для предсказания сигналы. На самом деле, паттерн "Объяснимые предсказания" рекомендуется рассматривать во время строительства технического решения, предусматривающего каскад из нескольких моделей, поскольку тот может ограничивать объяснимость модели. Это компромиссное ухудшение объяснимости модели снова проявляется с паттернами "Ансамбли" (Ensembles) и "Мультимодальный вход" (Multimodal Input), поскольку эти технические приемы также плохо поддаются некоторым методам обеспечения объяснимости.

Паттерн "Каскад" также бывает полезен при использовании паттерна "Мостовая схема" и может применяться как альтернативный паттерн, имея предварительную модель, которая вменяет недостающие значения вторичной схемы. Затем оба паттерна можно совместить с целью сохранения результирующего набора признаков для последующего использования, как описано в паттерне "Хранилище признаков". Это еще один пример, который подчеркивает многогранность паттерна "Хранилище признаков" и частоту его сочетания с другими паттернами. Например, хранилище признаков предоставляет удобный подход к поддержанию и эффективному использованию потоковой передачи признаков модели, которые могут возникать в результате использования паттерна "Оконный предсказательный вывод" (Windowed Inference). Паттерн "Хранилище признаков" также работает рука об руку с паттернами управления разными наборами данных, которые могут появляться в паттерне "Переформулировка" (Reframing) и предоставляют готовую для многоразового применения версию технических приемов, возникающих при использовании пат-

терна "Преобразователь" (Transform). Возможность управления версиями признаков, описанная в паттерне "Хранилище признаков", также играет определенную роль в паттерне "Управление версиями" (Versioning).

С другой стороны, паттерн "Управление версиями" тесно связан с паттернами "Функция обслуживания без поддержки состояния" (Stateless Serving Function) и "Непрерывное оценивание модели" (Continued Model Evaluation). В паттерне "Непрерывное оценивание модели" разные версии модели можно использовать при оценивании степени деградации результативности модели с течением времени. Подобным же образом разные модельные сигнатуры функции обслуживания обеспечивают простое средство создания различных версий модели. Подход к управлению версиями модели посредством паттерна "Функция обслуживания без поддержки состояния" может быть продолжен паттерном "Переформулировка", в котором две разные версии модели могут предоставлять свои REST API для двух разных представлений результата модели на выходе.

Мы также коснулись того, как при использовании паттерна "Непрерывное оценивание модели" нередко бывает полезно провести разведку технических решений, предлагаемых паттерном "Конвейер рабочего потока" (Workflow Pipeline), как для настройки триггеров, которые инициируют конвейер перетренировки, так и для отслеживания преемственности различных версий модели. Паттерн "Непрерывное оценивание модели" также тесно связан с паттерном "Предсказания по ключу" (Keyed Predictions), поскольку он обеспечивает механизм для легкого увязывания эмпирического наблюдения с результатом предсказания на выходе из модели. Паттерн "Предсказания по ключу" в той же степени переплетен с паттерном "Пакетное обслуживание" (Batch Serving). По той же причине паттерн "Пакетное обслуживание" часто используется в соединении с паттерном "Функция обслуживания без поддержки состояния" для выполнения заданий на предсказание в требуемом масштабе, который, в свою очередь, под капотом опирается на паттерн "Преобразователь" (Transform) для поддержания согласованности между тренировкой модели и обработкой запросов.

Паттерны в рамках проектов машинного обучения

Системы машинного обучения позволяют коллективам внутри организации создавать, развертывать и поддерживать технические решения по машинному обучению в требуемом масштабе. Они обеспечивают платформу для автоматизации и ускорения всех этапов жизненного цикла ML — от управления данными до тренировки, оценивания результативности, развертывания моделей, обслуживания предсказательных запросов и мониторинга результативности. Паттерны, о которых мы говорили в этой книге, проявляются в любом проекте машинного обучения. В этом разделе мы опишем этапы жизненного цикла ML и места, где могут возникать многие из этих паттернов.

Жизненный цикл машинного обучения

Строительство технического решения по машинному обучению — это циклический процесс, который начинается с четкого понимания деловых целей и приводит к внедрению модели ML в производство, принося пользу этим целям. Приведенный далее высокоуровневый обзор жизненного цикла ML (рис. 8.2) представляет собой полезную дорожную карту, служащую для того, чтобы обеспечивать ML способностями приносить ценность для бизнеса. Каждый этап одинаково важен, и неспособность выполнить любой из этих шагов увеличивает риск порождения уводящих в сторону идей или не имеющих ценности моделей на более поздних этапах.

Рис. 8.2. Жизненный цикл ML начинается с определения делового варианта использования и приводит к внедрению модели машинного обучения в производство, принося пользу деловым целям

Жизненный цикл ML состоит из трех этапов, как показано на рис. 8.2: обнаружение, разработка и развертывание. Существует канонический порядок отдельных шагов каждого этапа. Однако эти шаги выполняются итеративно, и более ранние шаги могут пересматриваться в зависимости от результатов и идей, полученных на более поздних этапах.

Обнаружение

Машинное обучение существует как инструмент для решения задачи. Этап обнаружения в проекте ML начинается с определения делового варианта использования (шаг 1 на рис. 8.2). Это время является критически важным для верхнего звена предприятия и для специалистов-практиков ML в части согласования специфики задачи и развития понимания пределов возможностей ML для достижения поставленной цели.

На каждом этапе жизненного цикла важно следить за ценностью. На различных этапах приходится выбирать среди большого числа вариантов решений, и нередко единственный "правильный" ответ отсутствует. Скорее даже лучший вариант определяется тем, как модель будет использоваться для реализации бизнес-потребности. В то время как достижимой целью научно-исследовательского проекта может быть повышение точности на 0,1% в сравнении с эталонным набором данных, это неприемлемо в производстве. Успех промышленной модели, разработанной для организации, определяется факторами, более тесно связанными с предприятием, такими как улучшение методов удержания клиентов, оптимизация деловых процессов, повышение вовлеченности клиентов или снижение уровня оттока. Кроме того, могут иметься и косвенные факторы, связанные с бизнесом, которые влияют на выбор направления разработки, такие как скорость предсказательного вывода, размер модели или ее интерпретируемость. Любой ML-проект должен начинаться с глубокого понимания бизнес-потребности и того, как модель машинного обучения могла бы существенно улучшить работу текущих операций.

Успешный этап обнаружения требует сотрудничества между бизнесом и экспертами по машинному обучению в оценивании жизнеспособности подхода на основе ML. Крайне важно, чтобы лица, которые разбираются в бизнесе и данных, сотрудничали с командами, понимающими технические проблемы и задействуемые инженерные усилия. Если совокупные инвестиции в ресурсы разработки перевешивают ценность для организации, то это решение не подходит. Вполне возможно, что технические накладные расходы и стоимость ресурсов на выпуск модели превысят выгоду, предлагаемую моделью, которая в конечном счете улучшит прогноз оттока лишь на 0,1%. А может, и не превысят. Если клиентская база организации насчитывает 1 млрд человек, то 0,1% — это все-таки 1 млн более счастливых клиентов.

Во время фазы обнаружения важно сформулировать целевые установки и задать объем задачи. Это также как раз то время, чтобы определить метрики, которые будут использоваться для измерения или определения успеха. Для разных организаций или даже внутри разных групп одной и той же организации успех может выглядеть по-разному. Обратитесь, например, к обсуждению темы нескольких целевых установок в *разд. "Распространенные проблемы машинного обучения" главы 1*. Создание четко определенных метрик и ключевых индикаторов эффективности (key performance indicators, KPI) в начале ML-проекта поможет обеспечить согласованность всех сторон в достижении общей цели. В идеале уже существует некая процедура, которая обеспечивает удобный базовый уровень, относительно которого можно измерять будущее продвижение вперед. Это может быть и модель, которая уже находится в производстве, и даже просто эвристика на основе правил, которая используется в настоящее время. Машинное обучение — это не ответ на все вопросы, и превзойти эвристику на основе правил иногда бывает трудно. Разработка не должна выполняться ради разработки. Базовоуровневая модель, независимо от того, насколько она проста, полезна при выборе вариантов реализации в будущем и понимания того, как каждый вариант перемещает стрелку в этой предопределенной метрике оценивания. В *главе 7* мы обсудили роль паттерна "Эвристиче-

ский эталон" (Heuristic Benchmark), а также другие темы, связанные с ответственным искусственным ителлектом, которые часто возникают при донесении заинтересованным бизнесменам мысли о воздействии и влиянии машинного обучения.

Разумеется, эти беседы также должны происходить в контексте данных. Глубокое погружение в специфику предприятия должно идти рука об руку с глубоким погружением в анализ данных (шаг 2 на рис. 8.2). Каким бы полезным ни было решение, если нет качественных данных, то нет и проекта. Или же, возможно, данные существуют, но из-за соображений их приватности они не могут использоваться или должны быть очищены от релевантной информации, необходимой для модели. В любом случае жизнеспособность проекта и потенциал успеха зависят от данных. Таким образом, очень важно, чтобы управленцы данными внутри организации были вовлечены в эти дискуссии на раннем этапе.

Данные управляют процессом, и важно понимать качество доступных данных. Каковы распределения ключевых признаков? Сколько там пропущенных значений? Как будут обрабатываться пропущенные значения? Есть ли выбросы? Сильно ли коррелируют какие-либо входные значения? Какие признаки существуют во входных данных и какие признаки должны быть сгенерированы? Для тренировки многих моделей машинного обучения требуются массивные наборы данных. Достаточно ли данных? Каким образом можно увеличить набор данных? Есть ли искаженность в наборе данных? Все эти вопросы важны и затрагивают только поверхность. Одно из возможных решений на данном этапе сводится к необходимости собрать больше данных, или данных конкретного сценария, прежде чем продолжать проект.

Изучение данных — ключевой шаг в ответе на вопрос о том, существуют ли данные достаточного качества. Беседа сама по себе редко заменяет грязную работу и эксперименты с данными. На этом этапе важную роль играет визуализация. Графики плотности и гистограммы полезны для понимания разброса разных входных значений. Блочные диаграммы с ограничителями выбросов (так называемые ящики с усами, box plots) помогут выявлять выбросы. Графики рассеяния полезны для описания и обнаружения двумерных связей. Процентили помогают определять интервал для числовых данных. Средние значения, медианы и стандартные отклонения помогут описывать центральную тенденцию. Эти и другие технические приемы помогут выявлять признаки, которые, вероятно, будут приносить модели пользу, а также позволят глубже понимать характер преобразований, которые потребуются для подготовки данных к моделированию.

Внутри этапа обнаружения бывает полезно провести несколько экспериментов, чтобы убедиться в наличии "сигнала в шуме". На этом этапе было бы разумно провести технико-экономическое обоснование машинного обучения (шаг 3 на рис. 8.2). Как понятно из названия, оно, как правило, представляет собой короткий технический спринт, охватывающий всего несколько недель, цель которого — оценить жизнеспособность данных для решения задачи. Такое обоснование дает возможность изучить варианты постановки задачи машинного обучения, поэкспериментировать с выбором алгоритма и узнать, какие этапы генерирования признаков будут наиболее полезными. Шаг технико-экономического обоснования на этапе обна-

ружения также является хорошей точкой для создания эвристического эталона (*см. главу 7*).

Разработка

После согласования ключевых метрик оценивания и ключевых индикаторов эффективности предприятия в рамках жизненного цикла машинного обучения начинается этап разработки. Детали разработки ML-модели подробно описаны во многих ресурсах по машинному обучению. Мы же выделим ключевые компоненты.

На этапе разработки всё начинается со строительства конвейеров данных и инженерии признаков (шаг 4 на рис. 8.2) для обработки входных данных, которые будут подаваться в модель. Данные, собранные в реальных практических приложениях, могут иметь много проблем, таких как пропущенные значения, невалидные примеры или повторяющиеся точки данных. Конвейеры данных необходимы для предобработки этих входных значений, чтобы можно было их использовать в модели. Инженерия признаков — это процесс преобразования сырых исходных данных в признаки, которые более тесно связаны с целевой установкой модели и выражены в формате, приемлемом для его передачи в модель с целью ее тренировки. Технические приемы инженерии признаков могут включать группирование входных данных в корзины, преобразование из одного формата данных в другой, лексемизацию и выделение основ слов в тексте, создание категориальных признаков или кодирование входных данных с использованием одного активного состояния, создание синтетического признака и векторных вложений признаков и многие другие. В *главе 2* рассматриваются паттерны для представления данных и затрагиваются многие аспекты данных, возникающие на этом этапе жизненного цикла ML. *Главы 5* и *6* описывают паттерны, связанные с обеспечением отказоустойчивости и воспроизводимости в системах ML и помогающие в создании конвейеров данных.

Этот шаг также может предусматривать инженерию меток для задачи и решения относительно вариантов дизайна, связанных с тем, как задача будет представлена. Например, в задачах на основе временны́х рядов сюда может входить создание признаковых окон и экспериментирование с временны́ми задержками и размером интервалов между метками. Или же, возможно, будет полезнее переформулировать регрессионную задачу в классификационную и полностью изменить представление меток. А может, придется задействовать методы перебалансировки, если распределение выходных классов чрезмерно представлено одним классом. *Глава 3* целиком посвящена представлению задач и рассматривает эти и другие важные паттерны, связанные с постановкой задач.

Следующий шаг (шаг 5 на рис. 8.2) этапа разработки ориентирован на строительство ML-модели. На этом этапе разработки крайне важно придерживаться лучших практических приемов фиксации рабочего ML-потока в конвейере (*см. разд. "Паттерн 25. Конвейер рабочего потока" главы 6*). Сюда входит создание повторяемых срезов для тренировочного/валидационного/тестового наборов до начала разработки какой-либо модели с целью недопущения утечки данных. На этом этапе могут тренироваться разные алгоритмы или комбинации алгоритмов. Это делается для оценивания их результативности на валидационном наборе и для проведения ин-

спекции качества их предсказаний. Также выполняется настройка параметров и гиперпараметров модели, используются методы регуляризации и исследуются граничные случаи. Типичный цикл тренировки ML-модели подробно описан в начале *главы 4*, где мы также рассматриваем полезные паттерны по изменению цикла тренировки с целью достижения тех или иных целевых установок.

Многие шаги жизненного цикла ML являются итеративными, и это особенно верно при разработке модели. Нередко после нескольких экспериментов может возникать необходимость в пересмотре данных, требований и ключевых индикаторов эффективности. На этапе разработки модели собираются новые данные, которые могут пролить дополнительный свет на то, что является возможным (а что невозможным). Нередко фаза разработки модели требует много времени, в особенности в отношении прикладной модели. В *главе 6* рассматриваются многие другие паттерны, служащие для обеспечения воспроизводимости, которые решают проблемы, возникающие в этой итеративной фазе разработки модели.

На протяжении всей разработки модели каждая новая корректировка или подход сравниваются с метриками оценивания, заданными на этапе обнаружения. Поэтому успешное исполнение этапа обнаружения имеет решающее значение, и необходимо иметь согласованные позиции относительно решений, которые были приняты во время этого этапа. В самом конце разработка модели завершается заключительным этапом оценивания (этап 6 на рис. 8.2). В этой точке разработка модели прекращается, и результативность модели оценивается по этим заранее заданным метрикам оценивания.

Одними из ключевых результатов этапа разработки являются интерпретация и представление результатов (шаг 7 на рис. 8.2) заинтересованным лицам и менеджерам в рамках бизнеса. Такое высокоуровневое оценивание имеет решающее значение и необходимо для того, чтобы донести до руководства ценность этапа разработки. Этот шаг сосредоточен на создании числовой и визуальной информации для первоначальных отчетов, которые будут доведены до интересантов внутри организации. В *главе 7* обсуждаются некоторые распространенные паттерны, которые обеспечивают ответственное использование ИИ и помогают заинтересованным лицам в управлении. Как правило, это ключевой момент принятия решения о выделении дополнительных ресурсов на заключительный этап жизненного цикла — выпуск и развертывание ML-модели.

Развертывание

Успешное завершение разработки модели и наличие многообещающих результатов обусловливает переход к следующему этапу, который ориентирован на выпуск модели, при этом первым шагом (шаг 8 на рис. 8.2) является составление плана развертывания.

Тренировка модели машинного обучения требует значительного объема работы, но для полной реализации ценности этих усилий модель должна работать в производстве, выполняя бизнес-критерии, которые она была призвана улучшить. Существует несколько подходов к достижению этой цели, и развертывание может оказаться

индивидуальным в разных организациях в зависимости от варианта использования. Например, вариант релизной версии ML может принимать форму интерактивных панелей мониторинга, статических блокнотов, исходного кода, обернутого в многократно используемую библиотеку, или конечных сервисов.

Для выпуска моделей существует много технических соображений и вариантов реализации. Как и прежде, многие решения, принятые на этапе обнаружения, определяют и этот шаг. Каким образом следует управлять перетренировкой модели? Нужно ли будет подавать входные значения в потоковом режиме? Следует ли проводить тренировку на новых пакетах данных или же в режиме реального времени? А что делать с предсказательным выводом модели? Должны ли мы планировать разовые задания пакетного предсказательного вывода каждую неделю или же необходимо поддерживать предсказание в режиме реального времени? Существуют ли особые проблемы с пропускной способностью или задержкой? Есть ли необходимость учитывать пиковые рабочие нагрузки? Является ли низкая задержка приоритетом? Существуют ли проблемы с сетевым соединением? Паттерны *главы 5* затрагивают некоторые проблемы, возникающие при вводе ML-модели в эксплуатацию.

Для многих предприятий приведенные выше важные соображения и данный заключительный этап, как правило, являются самым большим препятствием, поскольку все это может потребовать сильной координации между разными частями организации и интеграции различных технических компонентов. Указанная трудность также частично связана с тем, что выпуск требует интеграции в существующую систему совершенно нового процесса, который основан на ML-модели. Эта интеграция может предусматривать работу с унаследованными системами, которые были разработаны для поддержания единого подхода, или же сложные процессы управления изменениями и производством для навигации внутри организации. Кроме того, нередко существующие системы не имеют механизма поддержки поступающих из модели машинного обучения предсказаний, поэтому необходимо разрабатывать новые приложения и рабочие потоки. Важно предвидеть эти проблемы, поскольку разработка комплексного решения требует значительных инвестиций со стороны деловых операций, чтобы переход из проектной стадии в эксплуатационную был как можно более легким, а скорость выхода на рынок увеличилась.

Следующим шагом этапа развертывания является ввод в эксплуатацию модели (шаг 9 на рис. 8.2). Эта область практической деятельности обычно называется интеграцией ML в производство (ML Operations, MLOps)[2] и охватывает аспекты, связанные с автоматизацией, мониторингом, тестированием, управлением и поддержанием моделей машинного обучения в промышленной среде. Она является необходимым компонентом для любой компании, надеющейся масштабировать в своей организации приложения, которые приводятся в движение технологией машинного обучения.

---

[2] Интеграция ML в производство (MLOps) — это практика применения наработок по интеграции разработанных программно-информационных продуктов в производство (DevOps) для рабочего потока ML (ML Operations, MLOps). — *Прим. перев.*

Одной из ключевых характеристик введенных в эксплуатацию моделей являются автоматизированные конвейеры рабочих потоков. Этап разработки в рамках жизненного цикла ML представляет собой многоступенчатый процесс. Строительство конвейеров автоматизации этих шагов обеспечивает более эффективные рабочие потоки и повторяемые процессы, которые улучшают будущее развитие моделей, а также повышают гибкость при решении возникающих проблем. Сегодня такие инструменты с открытым исходным кодом, как Kubeflow[3], обеспечивают эту функциональность, а многие крупные софтверные компании разработали собственные сквозные платформы ML, такие как Uber Michelangelo[4] или Google TFX[5], которые тоже имеют открытый исходный код.

Успешный ввод в эксплуатацию включает компоненты непрерывной интеграции и непрерывной доставки (continuous integration/continuous delivery, CI/CD), т. е. хорошо известные лучшие практические приемы разработки программного обеспечения. Указанные методы CI/CD ориентированы на надежность, воспроизводимость, скорость, безопасность и управление версиями в рамках разработки исходного кода. Рабочие потоки процессов ML/ИИ выигрывают от тех же соображений, хотя в них есть некоторые заметные различия. Например, в дополнение к исходному коду, который используется для разработки модели, важно применять принципы CI/CD к данным, включая очистку данных, управление версиями и оркестровку конвейеров данных.

Последний шаг, который следует рассмотреть на этапе развертывания, — это мониторинг и техническое сопровождение модели. После того как модель была введена в эксплуатацию и находится в производстве, необходимо следить за ее результативностью. Со временем распределение данных меняется, что приводит к застарелости модели. Эта застарелость (рис. 8.3) может возникать по многим причинам — от изменений в поведении клиентов до изменений в окружающей среде. По этой причине важно иметь механизмы для эффективного мониторинга модели машинного обучения и всех компонентов, которые способствуют ее работе, от сбора данных до качества предсказаний во время обслуживания запросов. В паттерне "Не-

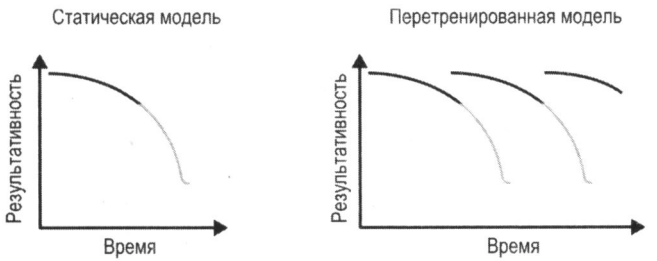


Рис. 8.3. Застарелость модели может возникать по многим причинам. Периодическая перетренировка моделей помогает улучшать их результативность с течением времени

[3] См. **https://oreil.ly/I_cJf**.

[4] См. **https://oreil.ly/se4G9**.

[5] См. **https://oreil.ly/OznI3**.

прерывное оценивание модели" (*см. главу 5*) эта распространенная проблема и ее решение рассматриваются подробно.

Например, важно отслеживать распределение значений признаков для сравнения с распределениями, которые использовались в шагах разработки. Также важно следить за распределением значений меток с целью устранения дисбаланса или сдвига в распределении меток, вызываемых дрейфом данных. Нередко модель машинного обучения опирается на данные, собранные из внешнего источника. Возможно, наша модель опирается на API стороннего трафика в части предсказания времени ожидания найма автомобилей или использует данные из API погоды в качестве входных переменных для модели, которая предсказывает задержки рейсов. Эти API не управляются нашим коллективом. Если данный API выйдет из строя или его выходной формат существенно изменится, то это будет иметь последствия для нашей промышленной модели. В данном случае важно организовать мониторинг с проверкой наличия изменений в этих вышестоящих источниках данных. Наконец, важно выстроить систему мониторинга распределения предсказаний и по возможности измерять качество этих предсказаний в промышленной среде.

После завершения шага мониторинга полезно вернуться к бизнес-целям по использованию и объективно, точно оценить степень влияния ML-модели на результативность. Скорее всего, это приведет к новым озарениям и запуску новых проектов ML, и жизненный цикл начнется снова.

Готовность к искусственному интеллекту

Мы видим, что различные организации, работающие над реализацией решений по машинному обучению, находятся на разных этапах готовности к ИИ. Согласно статье, опубликованной Google Cloud[6], зрелость компании в области внедрения ИИ на предприятии обычно можно охарактеризовать тремя фазами: тактической, стратегической и трансформационной. Использование инструментов машинного обучения в этих трех фазах сводится главным образом к ручной разработке в тактической фазе, к конвейерам в стратегической фазе и к полноавтоматизированным конвейерам в трансформационной фазе.

Тактическая фаза: ручная разработка

Тактическая фаза готовности к ИИ часто наблюдается в организациях, только начинающих изучать потенциал ИИ для достижения целей, с акцентом на краткосрочные проекты. Здесь примеры использования ИИ/ML, как правило, более узкие, они больше сосредоточены на доказательствах концепции или прототипах; прямая связь с бизнес-целями не всегда бывает ясной. На этом этапе организации признают перспективность продвинутой аналитической работы, но ее выполнение осуществляется главным образом отдельными участниками или полностью передается на аутсорсинг партнерам; доступ к крупномасштабным качественным наборам данных внутри организации бывает затруднен.

---

[6] См. **https://oreil.ly/5GljC**.

Как правило, на этом этапе отсутствует процесс согласованного масштабирования технических решений и используемые инструменты ML (рис. 8.4) разрабатываются на нерегламентированной основе. Данные складируются офлайново или на изолированных островах данных и доступны вручную для проведения исследовательского анализа данных. Не существует инструментов для автоматизирования различных фаз цикла разработки ML, и мало внимания уделяется разработке повторяемых процессов рабочего потока. Это затрудняет совместное использование активов внутри членов организации, а специально выделенное для разработки аппаратное обеспечение отсутствует.

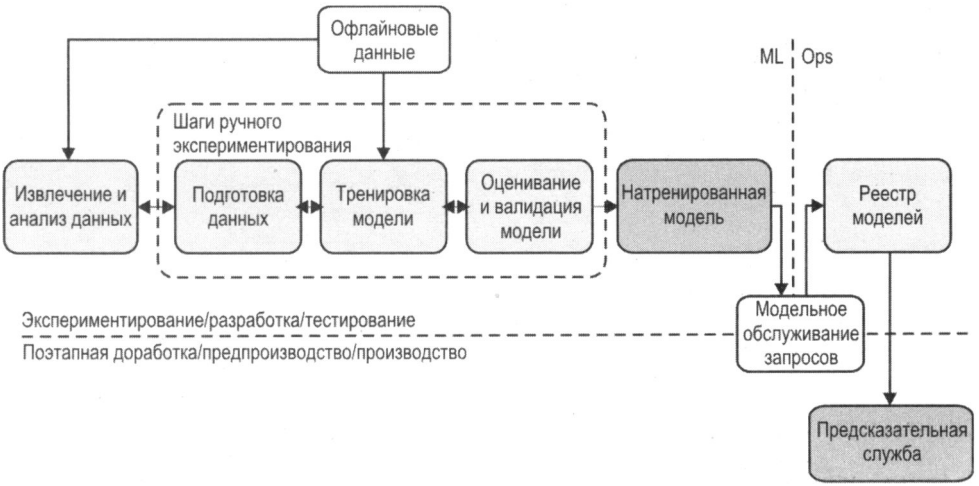


Рис. 8.4. Ручная разработка моделей искусственного интеллекта (рисунок адаптирован из документации Google Cloud[7])

Объем практики MLOps ограничен репозиторием натренированных моделей, и практически нет различий между тестовой средой и промышленной средой, в которой окончательная модель может быть развернута как техническое решение на основе API.

Стратегическая фаза: эффективное использование конвейеров

Организации в стратегической фазе согласовали усилия в области ИИ с бизнес-целями и приоритетами, и ML рассматривается на предприятии как ключевой ускоритель для бизнеса. В силу этого нередко есть поддержка со стороны топ-менеджеров и выделенный бюджет для проектов ML, который исполняется квалифицированными командами и стратегическими партнерами. Есть инфраструктура, позволяющая этим командам легко обмениваться активами и разрабатывать системы ML, задействующие как готовые к использованию, так и прикладные модели. Существует четкое различие между средой разработки и промышленной средой.

[7] См. **https://oreil.ly/aC1HP**.

Команды, как правило, уже имеют навыки работы с данными и опыт в описательной и предсказательной аналитике. Данные хранятся на корпоративном складе данных, и существует единая модель централизованного управления данными и активами ML. Разработка ML-моделей происходит как организованный эксперимент. Активы ML и исходный код для этих конвейеров хранятся в централизованном репозитории исходных текстов и легко используются совместно членами организации.

Конвейеры данных для разработки моделей ML автоматизированы с помощью полноавтоматизированной бессерверной службы данных для приемки и обработки и управляются по расписанию либо по событиям. Вдобавок рабочий поток процессов ML, состоящий из тренировки, оценивания и пакетного предсказания, управляется автоматизированным конвейером, вследствие чего этапы жизненного цикла ML — от валидации и подготовки данных до тренировки и валидации модели (рис. 8.5) — выполняются триггером мониторинга результативности. Эти модели

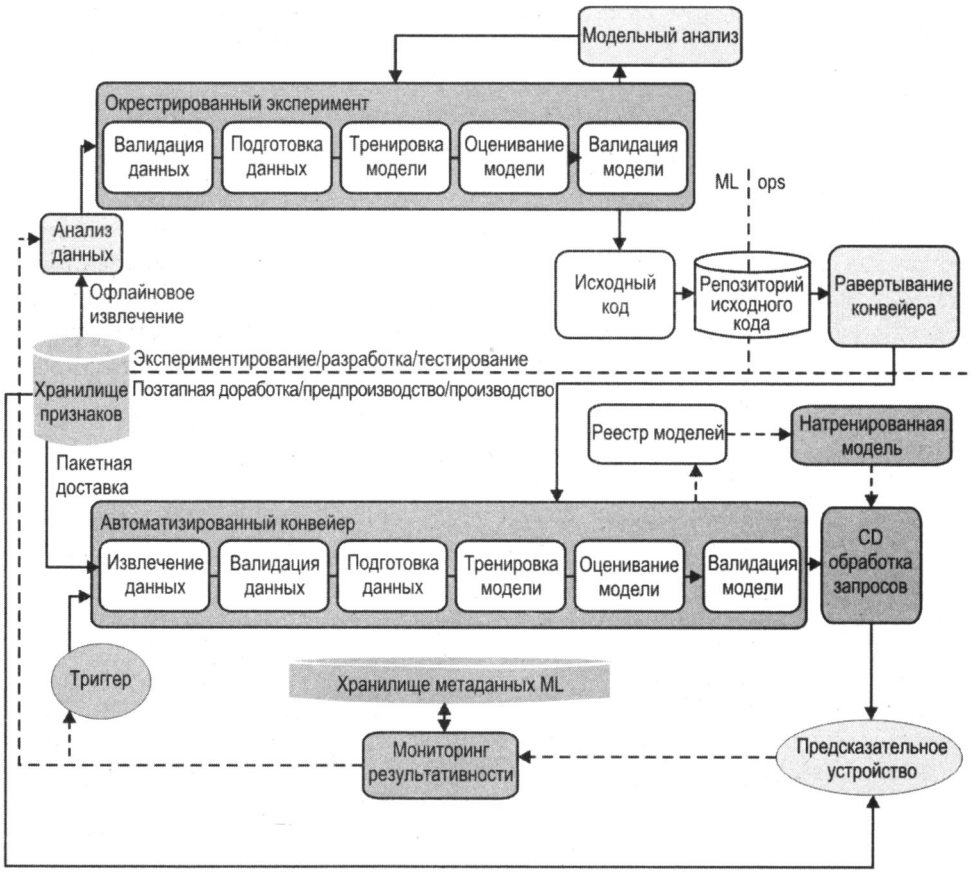


Рис. 8.5. Конвейерная фаза разработки искусственного интеллекта (рисунок адаптирован из документации Google Cloud[8])

[8] См. **https://oreil.ly/sMNo7**.

хранятся в централизованном реестре натренированных моделей и могут развертываться автоматически, основываясь на заранее определенных метриках валидации моделей.

В производстве может быть развернуто и технически сопровождаться несколько систем ML с журналированием, мониторингом результативности и уведомлениями. Системы ML эффективно задействуют API модели, способный манипулировать потоками данных в режиме реального времени как для предсказательного вывода, так и для сбора данных, которые подаются в автоматизированный конвейер ML с целью обновления модели для последующей тренировки.

Трансформационная фаза: полноавтоматизированные процессы

Организации, находящиеся в трансформационной фазе готовности к ИИ, активно используют ИИ для стимулирования инноваций, поддержания гибкости и формирования культуры, в которой процессы экспериментирования и обучения продолжаются непрерывно. Стратегические партнерства используются для инноваций, совместного создания и расширения технических ресурсов внутри компании. В этой фазе готовности к ИИ возникают многие паттерны по обеспечению воспроизводимости и отказоустойчивости, описанные в *главах 5* и *6*.

В этой фазе команды ИИ, ориентированные на конкретные продукты, обычно внедряются в более широкие продуктовые коллективы и поддерживаются коллективом продвинутой аналитики. При таком подходе опыт ML способен распространяться по различным направлениям предприятия внутри организации. Устоявшиеся распространенные паттерны и лучшие практические приемы, а также стандартные инструменты и библиотеки для ускорения проектов ML легко используются совместо между разными группами внутри организации.

Наборы данных хранятся на платформе, доступной для всех команд, что позволяет легко обнаруживать, совместно и многократно использовать наборы данных и ресурсы ML. Существуют стандартизированные хранилища признаков ML, и сотрудничество по всей организации поощряется. Полноавтоматизированные организации используют интегрированную платформу экспериментирования и производства ML, в которой строятся и развертываются модели, а практические приемы ML вседоступны в организации. Указанная платформа поддерживается масштабируемыми и бессерверными вычислениями для пакетного и онлайнового приема и обработки данных. Специализированные ускорители ML, такие как графические и тензорные процессоры, доступны по запросу, и существуют организованные эксперименты для сквозных данных и конвейеров ML.

Среды разработки и производства аналогичны конвейерному этапу (рис. 8.6), но они также встраивают методы CI/CD в каждый этап своего рабочего потока ML. Эти лучшие практические приемы CI/CD сосредоточены на обеспечении надежности, воспроизводимости и контроля версий кода для создания моделей, а также на конвейерах данных и их оркестровке. Это позволяет строить, тестировать и упаковывать различные компоненты конвейеров. Управление версиями поддерживается

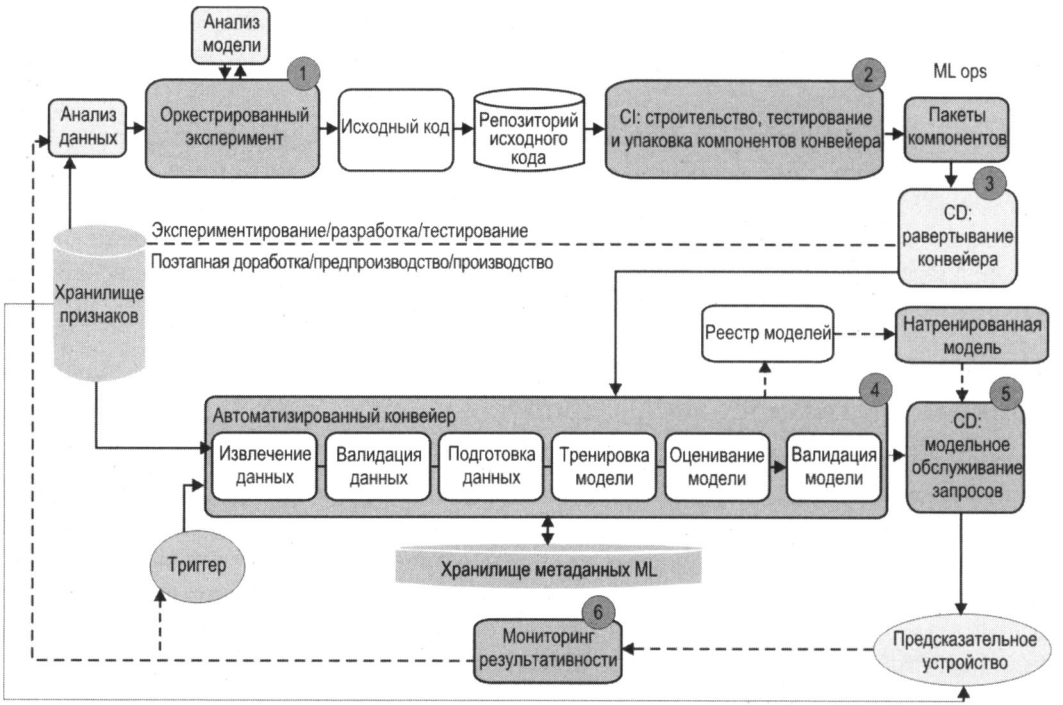


Рис. 8.6. Полноавтоматизированные процессы поддерживают разработку искусственного интеллекта (рисунок адаптирован из документации Google Cloud[9])

реестром моделей, в котором также хранятся необходимые метаданные и артефакты.

Распространенные паттерны с группировкой по варианту использования и типу данных

Многие из рассмотренных в этой книге паттернов используются в течение любого цикла разработки ML-системы и, вероятно, будут эксплуатироваться независимо от производственного варианта, например "Гиперпараметрическая настройка", "Эвристический эталон", "Повторяемая разбивка", "Управление версиями", "Распределенная тренировка", "Конвейер рабочего потока" или "Контрольные точки". Вы, возможно, обнаружите, что другие паттерны особенно полезны для определенных сценариев. Мы сгруппировали часто используемые паттерны в соответствии с популярными примерами машинного обучения.

Понимание естественного языка

Понимание естественного языка (natural language understanding, NLU) — это отрасль ИИ, в центре внимания которой находится тренировка машины понимать

[9] См. **https://oreil.ly/VX31C**.

смысл текста и языка. NLU используется речевыми агентами, такими как Alexa компании Amazon, Siri компании Apple и Помощник компании Google, для понимания предложений типа: "Какой прогноз погоды на эти выходные?" В сферу NLU подпадает большое число вариантов использования, и данная технология может применяться ко многим процессам, таким как классифицирование текста (фильтрация электронной почты), извлечение сущностей, ответы на вопросы, распознавание речи, резюмирование текста и анализ настроений.

- "Векторные вложения" (Embeddings).
- "Хешированный признак" (Hashed Feature).
- "Нейтральный класс" (Neutral Class).
- "Мультимодальный вход" (Multimodal Input).
- "Трансферное обучение" (Transfer Learning).
- "Двухфазные предсказания" (Two-Phase Predictions).
- "Каскад" (Cascade).
- "Оконный предсказательный вывод" (Windowed Inference).

Компьютерное зрение

Компьютерное зрение — это широкое родовое название разновидности ИИ, предметом которой является тренировка машин понимать визуальные входные данные, такие как снимки, видео, значки и все, что связано с пикселами. Модели компьютерного зрения направлены на автоматизацию любой задачи, которая может зависеть от человеческого зрения, от использования МРТ для выявления рака легких до применения беспилотных автомобилей. Классическими приложениями компьютерного зрения являются классифицирование снимков, анализ движения на видео, сегментирование снимков и шумоподавление снимков.

- "Переформулировка" (Reframing).
- "Нейтральный класс" (Neutral Class).
- "Мультимодальный вход" (Multimodal Input).
- "Трансферное обучение" (Transfer Learning).
- "Векторные вложения" (Embeddings).
- "Мультиметка" (Multilabel).
- "Каскад" (Cascade).
- "Двухфазные предсказания" (Two-Phase Predictions).

Предсказательная аналитика

В предсказательном моделировании используются исторические данные для отыскания закономерностей и определения вероятности наступления некоторого события в будущем. Предсказательные модели можно найти во многих самых раз-

нообразных областях промышленности. Например, компании могут использовать предсказательные модели для более точного предсказания выручки или прогнозирования будущего спроса на продукцию. В медицине предсказательные модели могут применяться для оценивания риска развития у пациента хронического заболевания или предсказания ситуаций, когда пациент может не явиться на плановый осмотр. Другие примеры включают прогнозирование потребностей в энергии, предсказание оттока клиентов, финансовое моделирование, метеопрогнозирование и предсказательное техническое обслуживание.

- "Хранилище признаков" (Feature Store).
- "Синтетический признак" (Feature Cross).
- "Векторные вложения" (Embeddings).
- "Преобразователь" (Transform).
- "Переформулировка" (Reframing).
- "Каскад" (Cascade).
- "Мультиметка" (Multilabel).
- "Нейтральный класс" (Neutral Class).
- "Оконный предсказательный вывод" (Windowed Inference).
- "Пакетное обслуживание" (Batch Serving).

В состав предсказательной аналитики также входит широкая категория под названием аналитика Интернета вещей (Internet of thing, IoT-аналитика). Модели IoT опираются на данные, собираемые подключенными к Интернету датчиками, именуемыми IoT-устройствами. Представьте себе коммерческий самолет, на котором установлены тысячи датчиков, собирающих более 2 Тбайт данных в сутки. Машинное обучение данных сенсорных IoT-устройств способно обеспечить работу предсказательных моделей, которые смогут предупреждать об отказе оборудования до того, как он произойдет.

- "Хранилище признаков" (Feature Store).
- "Преобразователь" (Transform).
- "Переформулировка" (Reframing).
- "Хешированный признак" (Hashed Feature).
- "Каскад" (Cascade).
- "Нейтральный класс" (Neutral Class).
- "Двухфазные предсказания" (Two-Phase Predictions)
- "Функция обслуживания без поддержки состояния" (Stateless Serving Function).
- "Оконный предсказательный вывод" (Windowed Inference).

Рекомендательные системы

Рекомендательные системы являются одним из наиболее распространенных приложений машинного обучения в бизнесе, и они появляются всякий раз, когда пользователи взаимодействуют с предметами. Рекомендательные системы улавливают признаки прошлого поведения и похожих пользователей и рекомендуют предметы, наиболее релевантные для данного пользователя. Подумайте о том, как YouTube рекомендует вам ряд видео для просмотра, основываясь на истории ваших просмотров, или как Amazon рекомендует покупки на базе позиций в вашей корзине. Рекомендательные системы распространены во многих компаниях, в особенности в части рекомендаций продуктов, персонализированного и динамического маркетинга, а также на платформах потоковой передачи видео или музыки.

- "Векторные вложения" (Embeddings).
- "Ансамбли" (Ensembles).
- "Мультиметка" (Multilabel).
- "Трансферное обучение" (Transfer Learning).
- "Хранилище признаков" (Feature Store).
- "Хешированный признак" (Hashed Feature).
- "Переформулировка" (Reframing).
- "Преобразователь" (Transform).
- "Оконный предсказательный вывод" (Windowed Inference).
- "Двухфазные предсказания" (Two-Phase Predictions).
- "Нейтральный класс" (Neutral Class).
- "Мультимодальный вход" (Multimodal Input).
- "Пакетное обслуживание" (Batch Serving).

Обнаружение мошенничества и аномалий

Многие финансовые учреждения используют машинное обучение для обнаружения мошенничества, чтобы обезопасить счета своих клиентов. Эти ML-модели учатся помечать транзакции, которые выглядят мошенническими, основываясь на определенных характеристиках или закономерностях, усвоенных ими из данных.

В более широком смысле обнаружение аномалий — это метод, используемый для отыскания аномального поведения или выбросных элементов в наборе данных. Аномалии могут возникать как всплески или провалы, которые отклоняются от нормальных регулярностей, либо они могут быть долгосрочными аномальными трендами. Обнаружение аномалий в машинном обучении появляется во всевозможных вариантах применения и может даже работать в сочетании с отдельным вариантом использования. Например, представьте себе ML-модель, которая идентифицирует аномальные железнодорожные пути, основываясь на снимках.

- "Перебалансировка" (Rebalancing).
- "Синтетический признак" (Feature Cross).
- "Векторные вложения" (Embeddings).
- "Ансамбли" (Ensembles).
- "Двухфазные предсказания" (Two-Phase Predictions).
- "Преобразователь" (Transform).
- "Хранилище признаков" (Feature Store).
- "Каскад" (Cascade).
- "Нейтральный класс" (Neutral Class).
- "Переформулировка" (Reframing).

Предметный указатель

D

E

F

G

H

I

J

K

T

U

V

W

X

Z

A

Б

В

Г

Д

Е

Ж

З

И

К

Л

М

Н

О

П

P

C

T

У

Ф

Х

Ц

Ч

Ш

Э

www.bhv.ru

Уатт Д., Борхани Р., Катсаггелос А.

Машинное обучение: основы, алгоритмы и практика применения

Отдел оптовых поставок:
e-mail: opt@bhv.ru

- Интуитивно понятные объяснения
- Доступный подход к современным методам численной оптимизации
- Комплексное введение в логистическую регрессию и машины опорных векторов
- Представление сложных тем через призму аппроксимации функций
- Уточненное описание глубоких нейронных сетей и методов ядра

Благодаря интуитивно понятному, но строгому подходу к машинному обучению эта книга предоставляет фундаментальные знания и практические инструменты, необходимые для проведения исследований и разработки систем машинного обучения. В книге приведено более 100 углубленных упражнений на языке Python. Дано введение в машинное обучение и математическую оптимизацию, включая методы первого и второго порядков, градиентного спуска и Ньютона. Отдельно рассмотрены продвинутые методы оптимизации. Приведено полное описание обучения с учителем, включая линейную регрессию, двухклассовую и многоклассовую классификацию, а также обучение без учителя и фундаментальные методы генерации признаков. Дано введение в нелинейное обучение с учителем и без. Обсуждается тема автоматизированного отбора подходящих нелинейных моделей, включая перекрестную валидацию, бустирование, регуляризацию и ансамблирование. Рассмотрены фиксированно-контурные ядра, нейронные сети, деревья и другие универсальные аппроксиматоры.

www.bhv.ru

Элбон К.

Машинное обучение с использованием Python. Сборник рецептов

Отдел оптовых поставок:
e-mail: opt@bhv.ru

В книге Вы найдете рецепты для:

- обработки числовых и категориальных данных, текста, изображений, дат и времени;
- уменьшения размерности с использованием методов выделения или отбора признаков;
- оценивания и отбора моделей;
- сохранения и загрузки натренированных моделей.

Научитесь решать задачи с использованием:

- векторов, матриц и массивов;
- линейной и логистической регрессии, деревьев, лесов и k ближайших соседей;
- опорно-векторных машин (SVM), наивных байесовых классификаторов, кластеризации и нейронных сетей.

Книга содержит около 200 рецептов, которые помогут решить задачи машинного обучения, возникающие в повседневной работе практикующего специалиста, такие как загрузка и обработка текстовых или числовых данных, отбор модели, уменьшение размерности и многие другие. Рассмотрена работа с языком Python и его библиотеками, в том числе pandas и scikit-learn. Решения всех задач сопровождаются подробными объяснениями. Каждый рецепт содержит программный код, который можно скопировать и опробовать на игрушечном наборе данных (toy dataset). Затем этот код можно вставлять, объединять и адаптировать, создавая собственные приложения.

Крис Элбон — аналитик данных и политолог с десятилетним опытом применения статистического обучения, искусственного интеллекта и разработки программного обеспечения для политических, социальных и гуманитарных проектов — от мониторинга выборов до оказания помощи в случае стихийных бедствий. В настоящее время является ведущим аналитиком данных в компании BRCK, продвигающей интернет-технологии на африканский рынок.

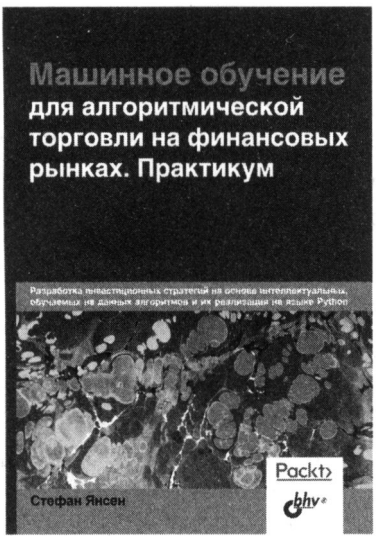
www.bhv.ru

Янсен С.

Машинное обучение для алгоритмической торговли на финансовых рынках. Практикум

Отдел оптовых поставок:
e-mail: opt@bhv.ru

Разработка инвестиционных стратегий на основе интеллектуальных, обучаемых на данных алгоритмов и их реализация на языке Python



Вы научитесь:

- реализовывать технические методы машинного обучения для решения инвестиционных и торговых задач
- использовать рыночные, фундаментальные и альтернативные данные с целью исследования альфа-факторов
- конструировать и тонко настраивать автоматически обучающиеся контролируемые, неконтролируемые и подкрепляемые модели
- оптимизировать портфельный риск и результативность с помощью библиотек pandas, NumPy и scikit-learn
- интегрировать автоматически обучающиеся модели в живую торговую стратегию на платформе Quantopian
- оценивать стратегии с использованием надежных методологий тестирования временных рядов
- конструировать и оценивать глубоко обучающиеся нейронные сети с помощью библиотек Keras, PyTorch и TensorFlow
- работать с подкрепляемым обучением для торговых стратегий на платформе OpenAI Gym

Книга посвящена практике применения машинного обучения с целью создания мощных алгоритмических стратегий для успешной торговли на финансовых рынках. Изложены базовые принципы работы с данными: оценивание наборов данных, доступ к данным через API на языке Python, доступ к финансовым данным на платформе Quandl и управление ошибками предсказания. Рассмотрены построение и тренировка алгоритмических моделей с помощью Python-библиотек pandas, Seaborn, StatsModels и sklearn и построение, оценка и интерпретация моделей AR(p), MA(q) и ARIMA(p, d, q) с использованием библиотеки StatsModels. Описано применение библиотеки PyMC3 для байесового машинного обучения, библиотек NLTK, sklearn (Scikit-learn) и spaCy для назначения отметок финансовым новостям и классифицирования документов, библиотеки Keras для создания, настройки и оценки нейронных сетей прямого распространения, рекуррентных и сверточных сетей. Показано, как применять трансферное обучение к данным спутниковых снимков для предсказания экономической активности и как эффективно использовать подкрепляемое обучение для достижения оптимальных результатов торговли.

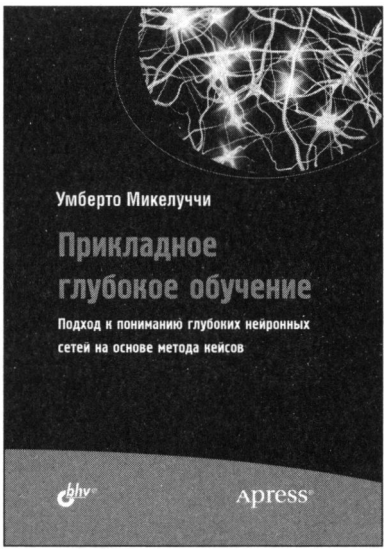
www.bhv.ru

Микелуччи У.

Прикладное глубокое обучение. Подход к пониманию глубоких нейронных сетей на основе метода кейсов

Отдел оптовых поставок:
e-mail: opt@bhv.ru

Для разработчиков от разработчиков



На практике вы освоите:

- передовые технические решения на Python и TensorFlow
- отладку и оптимизацию расширенных методов (таких как отсев и регуляризация)
- проведение анализа ошибок (с целью выяснить наличие проблем со смещением, дисперсией, смещением данных и т.д.)
- настройку проектов машинного обучения, ориентированных на глубокое обучение с использованием сложных наборов данных

Затронуты расширенные темы глубокого обучения: оптимизационные алгоритмы, настройка гиперпараметров, отсев и анализ ошибок, стратегии решения типичных задач во время тренировки глубоких нейронных сетей.

Описаны простые активационные функции с единственным нейроном (ReLu, сигмоида и Swish), линейная и логистическая регрессии с использованием библиотеки TensorFlow, выбор стоимостной функции, а также более сложные нейросетевые архитектуры с многочисленными слоями и нейронами и случайной инициализацией весов. Проведен анализ ошибок нейронной сети с примерами решения проблем, возникающих из-за дисперсии, смещения, переподгонки, а также наборов данных, поступающих из разных распределений. Показано, как реализовывать логистическую регрессию средствами Python и NumPy.

Книга содержит примеры использования по каждому техническому решению с целью реализации на практике всей теоретической информации. Вы также найдете советы и рекомендации по написанию оптимизированного кода Python.

O'REILLY®

Машинное обучение. Паттерны проектирования

Приводимые в этой книге паттерны проектирования отражают лучшие практические подходы к решению типичных задач машинного обучения. Авторы книги, три инженера компании Google, систематизировали проверенные временем методы решений и поместили их в книгу. Указанные паттерны, реализованные в программном коде, превращают опыт сотен экспертов в простые и легкодоступные советы.

В книге вы найдете подробный разбор 30 паттернов, служащих для представления данных и задач, тренировки моделей, отказоустойчивого обслуживания, обеспечения воспроизводимости и искусственного интеллекта. Каждый паттерн включает в себя постановку задачи, ряд потенциальных решений и рекомендации по выбору технического приема, наилучшим образом подходящего к данной ситуации.

«Эта книга с ее превосходными и разнообразными примерами обязательна для прочтения любым исследователем данных или инженером в области машинного обучения, стремящимся понять проверенные временем решения сложных задач».

— Дэвид Кантер, исполнительный директор компании ML Commons

«Если вы хотите, чтобы на вашем пути разработки решений в области машинного обучения было меньше синяков и шишек, то Лак, Сара и Майкл вас прикроют».

— Уилл Граннис, управляющий директор облачного офиса Google

Вы научитесь:

- **Выявлять и преодолевать трудности, встречающиеся во время тренировки, оценивания и развертывания моделей машинного обучения**
- **Представлять данные для разных типов моделей машинного обучения, включая векторные вложения, гибриды признаков и многое другое**
- **Выбирать правильный тип модели для той или иной задачи**
- **Строить надежный цикл тренировки с использованием контрольных точек, распределительной стратегии и гиперпараметрической настройки**
- **Разворачивать масштабируемые модели машинного обучения, которые можно переучивать и обновлять с целью учета новых данных**
- **Интерпретировать предсказания модели понятным конечному пользователю образом**
- **Улучшать точность, воспроизводимость и отказоустойчивость моделей**

Валлиаппа (Лак) Лакшманан (Valliappa (Lak) Lakshmanan) – руководитель отдела аналитики данных и ИИ в коллективе разработчиков Google Cloud.

Сара Робинсон (Sara Robinson) занимает должность developer advocate в коллективе разработчиков Google Cloud со специализацией в машинном обучении.

Майкл Мунн (Michael Munn) – инженер по техническим решениям в области машинного обучения в компании Google, где он помогает клиентам разрабатывать и внедрять модели машинного обучения.

ISBN 978-5-9775-6797-8

9 785977 567978

191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru

ozon
ii340718010б