Машинное обучение

Хенрик Бринк
Джозеф Ричардс
Марк Феверолф

Real-World Machine Learning

HENRIK BRINK
JOSEPH W. RICHARDS
MARK FETHEROLF

MANNING
SHELTER ISLAND

Хенрик Бринк
Джозеф Ричардс
Марк Феверолф

Машинное обучение

ПИТЕР®
Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону • Самара • Минск

2017

ББК 32.973.236
УДК 004.85
Б87

Бринк Хенрик, Ричардс Джозеф, Феверолф Марк

Б87 Машинное обучение. — СПб.: Питер, 2017. — 336 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-496-02989-6

В последние годы машинное обучение вышло на уровень большого бизнеса: компании активно используют его для зарабатывания денег, прикладные исследования бурно развиваются, а неугомонные разработчики используют любую возможность повысить свой уровень владения этой тематикой.

Данная книга рассчитана на тех, кто хочет решать самые разнообразные задачи при помощи машинного обучения. Как правило, для этого нужен Python, поэтому в примерах кода используется этот язык, а также библиотеки pandas и scikit-learn. Вы познакомитесь с основными понятиями ML, такими как сбор данных, моделирование, классификация и регрессия, а главное, получите практический опыт обработки реальных данных.

12+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.236
УДК 004.85

ISBN 978-1617291920 англ.
ISBN 978-5-496-02989-6

Краткое содержание

Оглавление

Предисловие

В последние годы машинное обучение (ML — machine learning) превратилось в большой бизнес — фирмы используют его, чтобы заработать денег, прикладные исследования бурно развиваются как в индустриальной, так и в академической среде, а любопытные разработчики везде ищут возможность поднять свой уровень владения темой. Но возникший *спрос* намного превышает скорость *появления* хороших методик для изучения применяемых на практике техник. Наша книга призвана удовлетворить данный спрос.

Прикладное машинное обучение совмещает в себе равные доли математических принципов и полученных эмпирическим путем приемов, — другими словами, это настоящее искусство. Слишком сильная концентрация только на одном из этих аспектов в ущерб другому — проигрышная стратегия. Тут важен баланс.

Долгое время самым лучшим и единственным способом постижения машинного обучения было получение ученой степени в одной из областей, которые (по большей части независимо друг от друга) развивали статистические методы и техники оптимизации. Основной упор эти программы делали на ключевые алгоритмы, в том числе на их теоретические свойства и ограничения, а также на характерные особенности относящихся к данной сфере задач. Впрочем, параллельно не менее ценные знания накапливались неофициальным образом — в процессе неформального общения на конференциях, обмена информацией и сценариями обработки данных между коллегами из исследовательских лабораторий. Именно эти знания, по большому счету, и позволили установить, какие алгоритмы больше всего подходят в каждой ситуации, как обрабатывать данные на каждом этапе и как связать между собой различные этапы рабочего процесса.

Сейчас мы живем в эпоху открытого исходного кода, с готовыми к использованию высококачественными алгоритмами, доступными на сайте

GitHub, и универсальными, хорошо спроектированными фреймворками, позволяющими связать все фрагменты друг с другом. Но даже среди этого изобилия неофициальные практические знания неизменно оказываются недоступными. Авторы данной книги оказывают вам большую услугу, сведя, наконец, эту скрытую информацию воедино; именно этого ключевого фрагмента не хватает для превращения машинного обучения из понятной лишь посвященным академической дисциплины в набор знаний и навыков по проектированию программного обеспечения.

Стоит подчеркнуть еще один момент. Большинство широко используемых сегодня методов машинного обучения далеки от совершенства и содержат ряд пробелов, которые можно было бы восполнить, будь мы в состоянии спроектировать идеальное решение. Современные методы имеют высокие требования к данным. Они, по большому счету, рады снабдить нас чрезмерно уверенными предсказаниями, если не принять необходимых мер. Небольшие изменения входных данных могут привести к крупным и странным изменениям в обнаруживаемых шаблонах. Получаемые результаты порой сложно интерпретировать и исследовать. Современная инженерия машинного обучения может рассматриваться как упражнение на сглаживание этих (и других) острых углов в методах оптимизации и статистического обучения.

В книге мы постепенно готовим читателя к упомянутым реалиям. Разговор начинается с типичных вариантов рабочего процесса, а затем мы переходим к более сложным примерам, демонстрирующим применение базовых знаний в реальных (читайте: запутанных) ситуациях. В книге крайне мало уравнений (потому что с ними можно ознакомиться где угодно, в том числе в классических учебниках), зато много неизвестных ранее сведений о подходах к реализации продуктов и решений на базе машинного обучения.

Без сомнения, сейчас самое лучшее время для изучения данной темы, и эта книга послужит существенным дополнением к изобилию повсеместно доступных математических и формальных сведений. Это принципиально *новая* книга, которую люди, давно работающие в данной области, хотели бы получить много лет назад.

Бо Кронин,
директор по данным компании 21 Inc.
Беркли, Калифорния

Вступление

Как студент, изучавший физику и астрономию, я много времени отдал работе с результатами измерений и моделирования, чтобы путем анализа, визуализации и дальнейшего моделирования извлечь из этих результатов полезную информацию. Я быстро сообразил, как мои навыки программирования могут помочь в решении повседневных задач. Первая же встреча с миром машинного обучения показала, что это не только потенциально крайне полезный новый инструмент, но и превосходная комбинация двух наиболее интересовавших меня дисциплин — теории анализа и обработки данных и программирования.

Машинное обучение стало важной частью моих исследований в области физики и привело меня на факультет астрономии Калифорнийского университета в Беркли, где статистики, физики и ученые, занимающиеся теорией и практикой вычислительных машин и систем, совместно работали над изучением Вселенной, используя ML как важный инструмент познания.

В центре исследования данных, меняющихся во времени (CTDI — Center for Time Domain Informatics), я встретил статистика и будущего соавтора этой книги Джозефа Ричардса. Мы ощутили не только возможность применения техник анализа данных и машинного обучения к научным исследованиям, но и возрастающий интерес к данной теме в компаниях и отраслях, не принадлежащих к академическим кругам. В результате совместно с Дамианом Идсом, Дэном Старром и Джошуа Блумом мы основали компанию Wise.io, предоставляющую машинное обучение в распоряжение бизнеса.

За последние четыре года Wise.io успела поработать над оптимизацией, расширением и автоматизацией посредством машинного обучения рабочих процессов многих компаний. Мы строили для наших клиентов масштабные прикладные платформы, делающие сотни миллионов пред-

сказаний в месяц, и узнали, насколько запутанными и неупорядоченными зачастую являются реальные данные. Мы надеемся передать вам знания, позволяющие работать с такими данными и строить с помощью машинного обучения интеллектуальное программное обеспечение следующего поколения.

Наш третий соавтор, Марк Феверолф, — основатель и технический директор нескольких компаний в области систем управления и бизнес-аналитики, применяющих в работе традиционные статистические и количественные методы. При разработке систем измерения и оптимизации процессов нефтепереработки он и его группа поняли, что техники, пригодные для непрерывного производства, можно применить также к производительности баз данных, компьютерных систем и сетей. Их технологии управления распределенными системами встроены в ведущие инструменты управления. Следующие попытки были связаны с системами, отвечающими за изменения и оптимизацию телекоммуникаций и взаимодействие с клиентами.

Через несколько лет он увлекся конкурсами на платформе Kaggle и обратил внимание на машинное обучение. Он вел проект с рекомендациями для кабельного телевидения, попутно много узнав о больших данных, адаптации вычислительных алгоритмов для параллельных расчетов и вариантах реакции людей на рекомендации, данные машиной. В последние годы он дает консультации по применению машинного обучения и прогностического анализа к приложениям для цифровой рекламы, телекоммуникаций, производства полупроводников, систем управления и оптимизации пользовательского опыта.

Хенрик Бринк

Благодарности

Мы хотим поблагодарить издательство Manning Publications и всех, кто внес свой вклад в создание и публикацию этой книги, в частности редактора Сюзанну Клайн, которая терпеливо и последовательно консультировала нас в процессе работы.

Спасибо Бо Кронину за написанное им предисловие. Спасибо Валентину Креттасу за полную корректуру всех глав. И всем остальным, кто в рабочем порядке давал нам свои бесценные советы: Алену Куньо, Алессандрини Альфредо, Алексу Айверсону, Артуру Зубареву, Дэвиду Клементсу, Дину Айверсону, Якобу Кванту, Яну Гойвертсу, Костасу Пассадису, Лейфу Сингеру, Луи Луангзорну, Массимо Иларио, Майклу Лунду, Морану Корену, Пабло Доминику Вассели, Патрику Тухи, Равишанкару Раягопалану, Рэю Луго, Рэю Морхеду, Рису Моррисону, Ризвану Пателю, Роберту Диана и Урсину Стауссу.

Марк Феверолф благодарит Крэйга Кармайкла за их совместную одержимость идеей машинного обучения, а также свою жену Патрисию и дочь Эми за многолетнее терпение.

Хенрик Бринк хотел бы сказать спасибо основателям и всей команде проекта Wise.io за энтузиазм в применении машинного обучения к решению различных задач. Он благодарит своих родителей Эдит и Джонни, а также своего брата и сестру, которые передали ему страсть к знаниям и информации, и, что самое важное, свою жену Иду и своего сына Харальда за любовь и поддержку.

Джозеф Ричардс также хотел бы поблагодарить команду Wise.io за их общую увлеченность идеей машинного обучения и за бесконечную работоспособность и энергию, благодаря которым каждый рабочий день был истинным удовольствием. Особенно он хотел бы выразить благодарность своим родителям Сюзан и Карлу, которые научили его испытывать радость от возможности постоянно узнавать что-то новое, привили любовь к упорному труду и умение сопереживать. Кроме того, он благодарит свою жену Тришу за бесконечную любовь, сострадание и поддержку.

О книге

Книга «*Машинное обучение на практике*» предназначена для тех, кто хочет применять машинное обучение к решению различных задач. В ней описываются и объясняются процессы, алгоритмы и инструменты, относящиеся к основным принципам машинного обучения. Внимание акцентируется не на способах написания популярных алгоритмов, а на их практическом применении. Каждый этап построения и использования моделей машинного обучения иллюстрируется примерами, сложность которых варьируется от простого до среднего уровня.

Структура книги

Часть I «Последовательность действий при машинном обучении» знакомит с пятью этапами основной последовательности машинного обучения:

- В главе 1 «Что такое машинное обучение?» рассказывается, что представляет собой машинное обучение и для чего оно нужно.
- В главе 2 «Реальные данные» подробно рассматриваются характерные стадии подготовки данных для моделей с машинным обучением.
- Глава 3 «Моделирование и прогнозирование» обучает с помощью распространенных алгоритмов и библиотек созданию простых ML-моделей и генерированию прогнозов.
- В главе 4 «Оценка и оптимизация модели» ML-модели подробно рассматриваются с целью оценки и оптимизации их производительности.
- В главе 5 «Основы проектирования признаков» рассказывается о том, как увеличить количество необработанных данных, используя информацию из поставленной перед нами задачи.

В части II «Практическое применение» вводятся техники масштабирования моделей, а также техники извлечения признаков из текста, изображений и временных рядов, увеличивающие эффективность решения многих современных задач с машинным обучением. Эта часть содержит три главы с практическими примерами.

- Глава 6 «Пример: чаевые для таксистов» — первая, полностью посвященная рассмотрению примера. Мы попытаемся предсказать шансы таксиста на получение чаевых.
- Глава 7 «Усовершенствованное проектирование признаков» знакомит с более сложными техниками проектирования признаков, предназначенными для извлечения значений из текстов, изображений и временных рядов.
- В главе 8 «Пример обработки естественного языка» усовершенствованные техники проектирования признаков используются для предсказания тональности рецензий на фильмы.
- Глава 9 «Масштабирование процесса машинного обучения» знакомит с техниками, дающими ML-системам возможность работать с большими объемами данных, обеспечивающими более высокую скорость прогнозирования и уменьшающими время их ожидания.
- В главе 10 «Пример с цифровой рекламой» на большом объеме данных строится модель, предсказывающая вероятность перехода по рекламному баннеру.

Как читать эту книгу

Тех, кто пока не имеет опыта в области машинного обучения, главы с 1-й по 5-ю познакомят с процессами подготовки и исследования данных, проектированием признаков, моделированием и оценкой моделей. В примерах кода на языке Python используются такие популярные библиотеки, как pandas и scikit-learn. Главы с 6-й по 10-ю включают в себя три практических примера машинного обучения наряду с такими продвинутыми темами, как проектирование признаков и оптимизация. Так как основная вычислительная сложность инкапсулирована в библиотеках, приведенные фрагменты кода легко адаптировать к вашим собственным ML-приложениям.

Целевая аудитория

Эта книга позволит программистам, аналитикам данных, статистикам, специалистам по обработке данных и всем остальным применить машинное обучение к решению реальных задач или хотя бы просто понять, что оно собой представляет. Читатели, не прибегая к глубокому теоретическому изучению конкретных алгоритмов, получат практический опыт обработки реальных данных, моделирования, оптимизации и развертки систем машинного обучения. Для тех, кому интересна теория, мы обсуждаем математическую основу машинного обучения, объясняем некоторые алгоритмы и даем ссылки на материалы для дополнительного чтения. Основной акцент делается на практических результатах при решении поставленных задач.

Формат кода, загрузки и требования к ПО

Книга содержит множество примеров кода, как в виде листингов, так и в виде обычного текста. В обоих случаях код форматируется `вот таким шрифтом`, что позволяет легко отделить его от остального текста.

Представленные в листингах фрагменты кода написаны на языке Python с использованием библиотек pandas и scikit-learn. Записная книжка iPython с материалами книги доступна на сайте GitHub по *адресу https://github.com/brinkar/real-world-machine-learning*. Каждая глава помещена в отдельный ipynb-файл. Примеры данных добавлены в каталоги репозитория, так что все содержимое записных книжек можно запустить на выполнение, если вместе с интерактивной средой iPython вы установили необходимые библиотеки. Графики создавались с помощью модуля pyplot из библиотек matplotlib и Seaborn.

Часть графики, сгенерированной в записных книжках iPython, помещена в текст в виде рисунков. (Некоторые из них модифицированы для улучшения визуального восприятия в печатном издании и электронной книге.)

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция). Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Об авторах

Хенрик Бринк — специалист по обработке и анализу данных и разработчик программного обеспечения, имеющий огромный практический опыт машинного обучения как в области производства, так и в сфере научной деятельности.

Джозеф Ричардс — старший научный сотрудник в области прикладной статистики и предсказательной аналитики. Хенрик и Джозеф совместно основали компанию Wise.io, которая занимается разработкой решений с машинным обучением для промышленности.

Марк Феверолф — основатель и президент компании Numinary Data Science, специализирующейся в области управления данными и предсказательной аналитики. Он работал статистиком и разработчиком аналитических баз данных в области социальных наук, химической инженерии, производительности информационных систем, планирования объема производства, кабельного телевидения и приложений для рекламы в Интернете.

ЧАСТЬ I

Последовательность действий при машинном обучении

Первая часть книги познакомит вас с основами процесса машинного обучения. В каждой главе рассматривается один из этапов.

Глава 1 рассказывает о предназначении машинного обучения и о том, зачем вам читать эту книгу.

В главе 2 будет детально рассмотрен этап обработки данных, а также распространенные способы их очистки и извлечения из них полезной информации.

В главе 3 вы приступите к созданию простых ML-моделей, попутно познакомившись с некоторыми алгоритмами и способами их применения в распространенных реализациях.

Глава 4 даст возможность пристально рассмотреть наши ML-модели с целью оценки и оптимизации их производительности.

Глава 5 посвящена основам проектирования признаков. Извлечение признаков из данных является крайне важной частью построения и оптимизации производительности любой ML-системы.

1 Что такое машинное обучение?

В этой главе:

- ✓ основы машинного обучения;
- ✓ преимущества машинного обучения перед традиционными методами;
- ✓ базовые этапы машинного обучения;
- ✓ усовершенствованные методы повышения эффективности моделей.

В 1959 г. специалист по вычислительной технике из компании IBM Артур Самуэль написал компьютерную программу для игры в шашки. Каждому положению на доске присваивался некий вес, базирующийся на вероятности выигрыша. Изначально вероятность определялась по формуле, в которой учитывались такие факторы, как количество шашек на каждой стороне и количество дамок. Подход работал, но Самуэль придумал, каким образом можно повысить его эффективность. Сыграв с программой тысячу партий, он использовал их результаты для уточнения позиционных весов. К середине 1970-х гг. программа достигла уровня хорошо подготовленного непрофессионального игрока.[1]

Самуэль написал компьютерную программу, которая могла по мере накопления опыта улучшать собственные результаты. Программа училась — так зародилось машинное обучение (ML — machine learning).

Мы не собираемся вдаваться в запутанные и сложные математические подробности алгоритмов машинного обучения (хотя и «снимем несколько верхних листьев с этого капустного кочана», чтобы дать вам представление о функционировании наиболее распространенных алгоритмов). Но, по сути, основной целью книги является предоставление неспециалистам информации о важных аспектах и распространенных проблемах, с которыми приходится сталкиваться при интеграции машинного обучения в приложения и конвейеры данных. В этой главе мы рассмотрим реальную экономическую задачу — обзор кредитных заявок, которая покажет преимущество машинного обучения перед большинством существующих альтернатив.

1.1. Как обучаются машины

У людей мы различаем механическое заучивание и интеллектуальное осмысление. Зазубривание телефонных номеров или инструкций, без сомнения, тоже относится к процессу *обучения*. Но, как правило, под этим понятием мы подразумеваем кое-что другое.

Ребенок, играющий с друзьями, наблюдает реакцию других членов группы на свои действия. Этот опыт влияет на его будущее поведение

[1] Jonathan Schaeffer. *One Jump Ahead: Computer Perfection at Checkers* (New York: Springer, 2009).

в социуме. Но он не вспоминает и не проигрывает заново свое прошлое, а опирается на определенные, легко опознаваемые характеристики прошлых взаимодействий: детская площадка, класс, мама, папа, сестры и братья, друзья, незнакомцы, взрослые, дети, в помещении или на улице. Оценка новой ситуации базируется на признаках, с которыми ему доводилось сталкиваться раньше. Обучение при этом является не просто сбором информации. Формируется то, что можно назвать *аналитической оценкой*.

Представьте, как вы по картинкам учите ребенка отличать собаку от кошки. Показанная картинка кладется в одну из двух стопок, в зависимости от правильности полученного ответа. Чем дольше продолжается процесс, тем выше эффективность распознавания. Что интересно, нет необходимости специально учить ребенка отличать собаку от кошки. Человеческое сознание обладает встроенными механизмами классификации. Ему требуются только *образцы*. Научившись работать с картинками, ребенок сможет опознать практически любое изображение кошки или собаки, не говоря уже о реальных животных. Эта способность *обобщать*, применяя полученные в процессе тренировок знания к новым, ранее не встречавшимся образцам, является ключевой характеристикой как человеческого, так и машинного обучения.

Разумеется, процесс получения знаний человеком превосходит своей сложностью самые совершенные алгоритмы машинного обучения, но у компьютера есть преимущество в виде большей емкости для запоминания, извлечения и обработки данных. Накапливаемый им опыт представлен в форме данных за длительный период времени, обработанных с помощью описанных в этой книге техник, причем это представление позволяет получать и оптимизировать алгоритмы, реализующие если не аналитическую оценку, то хотя бы способность к обобщениям.

Аналогия между человеческим и машинным обучением закономерно заставляет вспомнить такое явление, как *искусственный интеллект* (AI — artificial intelligence). При этом естественным образом возникает вопрос: «Чем искусственный интеллект отличается от машинного обучения?». По этому вопросу нет единого мнения, но большинство соглашается с тем, что ML — это одна из форм AI, так как AI представляет собой куда более обширную область, включающую в числе прочего робототехнику, обработку лингвистической информации и системы машинного зрения. Неоднозначность терминологии усиливает тот факт, что машинное обу-

чение все чаще применяется во многих сопутствующих областях AI. Можно сказать, что такая дисциплина, как машинное обучение, относится к *специализированной совокупности знаний и связанным с ней техникам.* Легко определить, что относится, а что не относится к машинному обучению, в то время как в случае искусственного интеллекта далеко не всегда можно провести такую же четкую границу. Перефразируя часто цитируемое определение Тома Митчелла, скажем, что компьютерная программа обучается, если ее производительность при выполнении определенной задачи, выраженная в измеряемых единицах, увеличивается по мере накопления опыта.[1]

Компания Kaggle объявила конкурс на алгоритм, максимально точно отличающий собак от кошек.[2] Для тренировки участникам предоставили 25 000 изображений с метками, указывающими, кто именно изображен на картинке. После обучения каждый алгоритм должен был классифицировать 12 500 не имеющих меток тестовых изображений.

Те, кому мы рассказывали об этом конкурсе, зачастую задумывались о признаках, по которым можно отличить собаку от кошки. У кошек треугольные и стоячие уши, а у собак они висят, но бывают и исключения. Представьте, что вы должны, не прибегая к иллюстрациям, объяснить разницу между кошкой и собакой человеку, который никогда не видел ни того ни другого животного.

Для обучения и обобщения люди используют различные данные из примеров, включая формы, цвета, текстуры, пропорции и другие характеристики. Машинное обучение также применяет множество стратегий в различных комбинациях в зависимости от поставленной задачи.

Эти стратегии нашли свое воплощение в наборе алгоритмов, разработанных в течение последних десятилетий как учеными, так и практиками в самых разных дисциплинах — от статистики, компьютерной науки, робототехники и прикладной математики до поиска в Интернете, развлекательной сферы, цифровой рекламы и переводов с одного языка на другой. Алгоритмы крайне разнообразны и имеют свои сильные

[1] Tom Mitchell, *Machine Learning* (McGraw Hill, 1997): «Говорят, что компьютерная программа обучается на основе опыта E по отношению к некоторому классу задач T и меры качества P, если качество решения задач из T, измеренное на основе P, улучшается с приобретением опыта E».

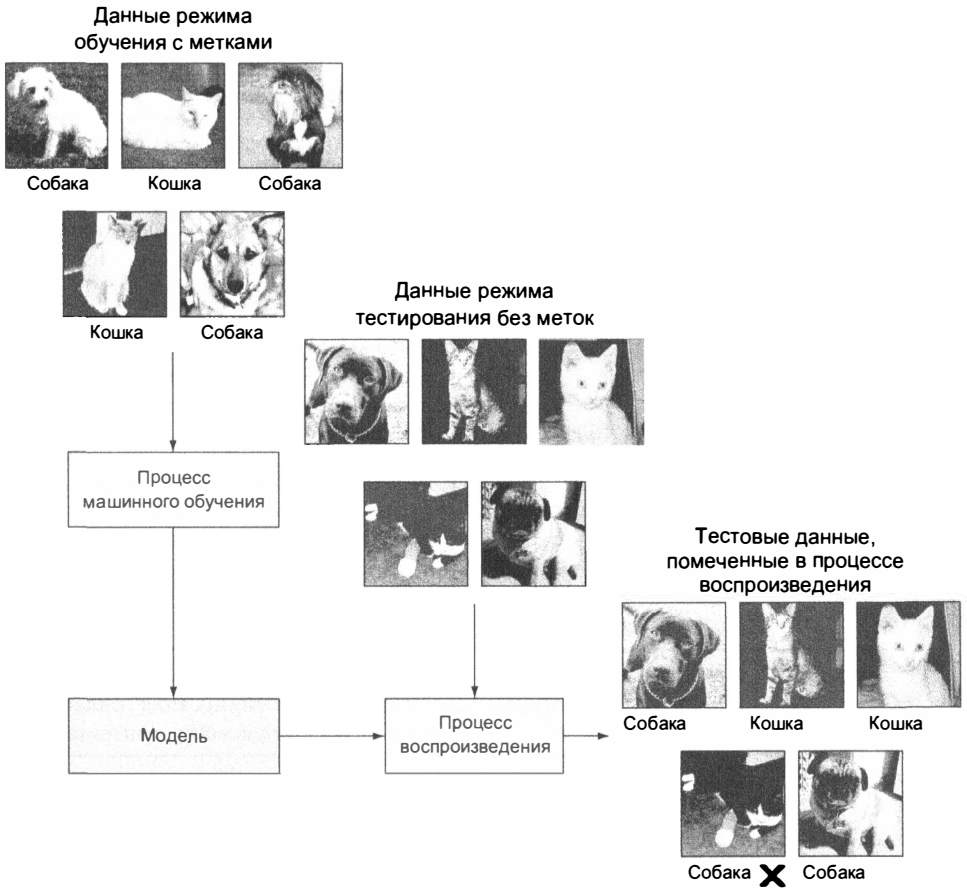
[2] См. страницу «Собаки против кошек» на сайте *www.kaggle.com/c/dogs-vs-cats.*

и слабые стороны. Некоторые относят объекты к определенному классу, другие предсказывают числовые значения. Существуют и алгоритмы, определяющие сходства и различия допускающих сравнение сущностей (например, людей, машин, процессов, кошек, собак). При этом все алгоритмы обучаются на примерах (опыте) и умеют применять полученные знания к новым, ранее не встречавшимся случаям, то есть способны к обобщению.

Заявленные на конкурс «Собаки против кошек» программы на этапе обучения раз за разом пытались корректно выполнить классификацию, используя множество алгоритмов. На каждой из миллионов обучающих итераций программа производила классификацию, измеряла полученный результат и затем хотя бы немного корректировала процесс в поисках постепенного улучшения. Победитель смог корректно распознать 98,914% ранее не демонстрировавшихся тестовых изображений. Это замечательный результат, если учесть, что у людей частота появления ошибки составляет примерно 7%. Процедура показана на илл. 1.1. Процесс машинного обучения анализирует изображения с метками и строит модель, которая в свою очередь используется процессом *воспроизведения* (предсказания) для классификации новых изображений. В примере вы видите, что одно изображение с кошкой было распознано некорректно.

Обратите внимание, что в данном случае мы рассматриваем так называемое *обучение с учителем* (supervised machine learning), но существуют и другие типы машинного обучения. Позже мы поговорим и о них.

Машинное обучение применяется к широкому кругу экономических задач — от обнаружения мошенничества до выбора целевой аудитории и рекомендаций товара, наблюдения за производством в реальном времени, анализа тональности текстов и медицинской диагностики. Оно может взять на себя задачи, которые невозможно выполнить вручную из-за огромного количества подлежащих обработке данных. В случае больших наборов данных машинное обучение иногда обнаруживает неочевидные зависимости, которые невозможно распознать при сколь угодно скрупулезном ручном рассмотрении. При этом комбинация множества таких «слабых» соотношений дает прекрасно работающие механизмы прогнозирования.



Илл. 1.1. Процесс машинного обучения для алгоритма, отличающего кошку от собаки

Процесс обучения на основе данных и последующего применения полученных знаний для обоснования будущих решений — чрезвычайно мощный инструмент. Машинное обучение быстро превращается в двигатель современной экономики, управляемой данными.

В табл. 1.1 перечислены широко распространенные техники машинного обучения с учителем и варианты их практического применения. Список далеко не исчерпывающий, так как потенциальные варианты использования могут занять несколько страниц.

Таблица 1.1. Варианты применения машинного обучения с учителем, систематизированные по типам задач

Задача	Описание	Пример применения
Классификация	На основе данных определяется дискретный класс для каждого объекта	Фильтрация спама, анализ тональности текстов, обнаружение мошенничества, рассылка целевых рекламных объявлений, прогнозирование оттока клиентов, обработка заявок на техническую поддержку, персонализация контента, выявление производственных дефектов, сегментация потребителей, обнаружение событий, изучение геномов и эффективности лекарственных средств
Регрессия	На основе данных предсказывается фактическое значение параметра	Прогнозы на рынке ценных бумаг, прогноз спроса, прогноз цены, оптимизация аукциона реальных объявлений, управление рисками, управление активами, прогнозы погоды, спортивные предсказания
Рекомендация	Предсказывается альтернатива, которую предпочтет пользователь	Предложения продуктов, подбор персонала, конкурс Netflix Prize, онлайн-знакомства, предложение контента
Заполнение пропусков	Вывод значений отсутствующих входных данных	Неполные истории болезни, отсутствующая информация о клиентах, данные переписей

1.2. Принятие решений на основе данных

В качестве примера рассмотрим реальную экономическую задачу, решение которой упрощается с помощью машинного обучения. Мы перечислим распространенные альтернативы и покажем преимущества ML-подхода.

Представьте, что вы сотрудник организации, которая предоставляет кредиты физическим лицам для открытия малого бизнеса в неблагоприятных районах. На начальном этапе вы получали несколько заявлений в неделю, поэтому могли вручную прочитать каждое и выполнить все проверки для принятия решения об одобрении кредита. Этот процесс схематично представлен на илл. 1.2. Заемщиков устраивала скорость

Илл. 1.2. Схема одобрения кредита в микрофинансовой организации

вашей работы и индивидуальный подход, и они везде рекомендовали вашу фирму.

По мере роста популярности фирмы количество заявлений увеличивается, их уже сотни в неделю. Даже сверхурочная работа не позволяет справиться с таким наплывом, клиенты устают ждать и отправляются в конкурирующую фирму. Очевидно, что обрабатывать заявления вручную нерационально, к тому же вы начинаете испытывать сильный стресс из-за отставания от графика.

Как выйти из этой ситуации? Сейчас мы рассмотрим несколько способов, позволяющих ускорить процесс анализа заявлений, к которым обычно прибегают в подобных случаях.

1.2.1. Традиционные подходы

Исследуем два традиционных подхода к анализу данных при рассмотрении заявлений — анализ вручную и бизнес-правила. Внимательно изучим реализацию обеих техник и покажем, что именно мешает расширению бизнеса.

Увеличение штата

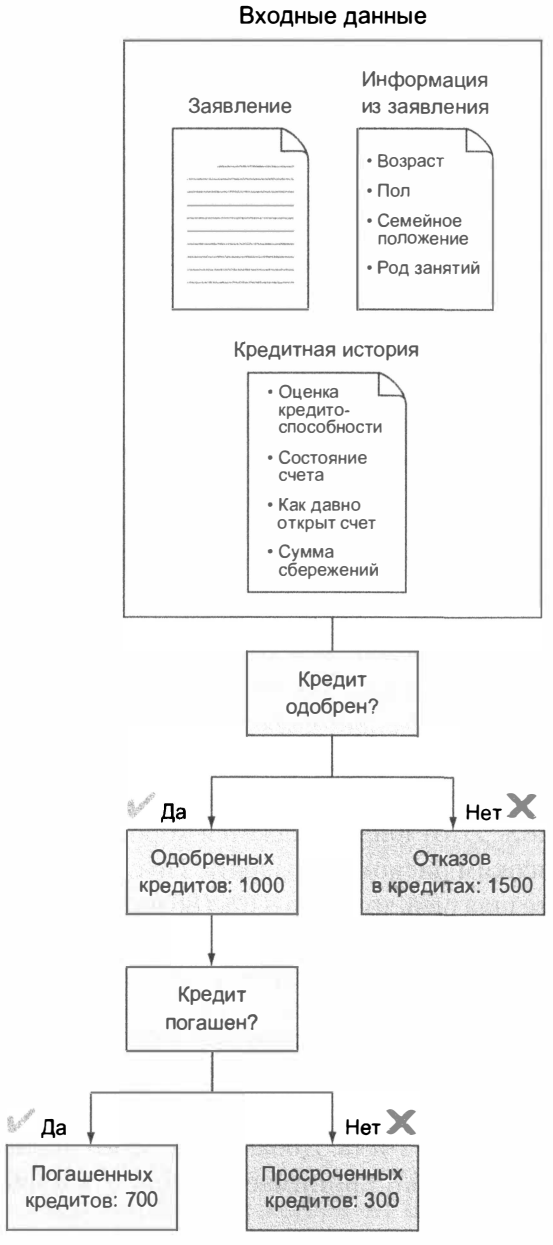
Вы нанимаете для работы с заявлениями еще одного аналитика. Тот факт, что часть прибыли придется потратить на зарплату нового сотрудника, не вызывает особого восторга, зато вдвоем можно за то же время сделать в два раза больше. В результате за неделю вы справляетесь с очередью заявок.

Пару недель ваш дуэт успевает за спросом. Но количество потенциальных клиентов продолжает расти и в следующем месяце достигает 1000 в неделю. Чтобы справиться с нагрузкой, нужны еще два аналитика. Вы делаете вывод, что в долгосрочной перспективе такая схема не работает: доход от новых заемщиков пойдет на зарплату новым сотрудникам, а не в фонд кредитования. *Увеличение штата по мере роста спроса препятствует развитию бизнеса.* Более того, сам процесс найма — длительное и дорогое развлечение, лишающее бизнес изрядной части дохода. Наконец, новый сотрудник имеет меньше опыта и обрабатывает заявки медленнее, а вы начинаете чувствовать стресс из-за необходимости управлять рабочей группой.

Кроме такого явно негативного фактора, как рост издержек, вы столкнетесь с тем, что люди добавляют в процесс принятия решений собственные сознательные и подсознательные представления. Для обеспечения согласованности потребуется детально проработать всю процедуру одобрения, а также придумать обширную программу обучения для новых аналитиков, но это еще больше увеличит издержки и вовсе не факт, что решит проблему.

Использование бизнес-правил

Представим, что из 1000 кредитов с просроченной датой погашения только 70% погашены вовремя. Эта ситуация представлена на илл. 1.3.



Илл. 1.3. За несколько месяцев работы из 2500 заявлений на кредит одобрена 1000. Из них 700 заявителей погасили кредит вовремя, а остальные 300 просрочили погашение. Этот исходный набор данных крайне важен для автоматизации процесса одобрения кредита

Теперь можно приступить к поиску связей между данными заявителя и количеством погашенных кредитов. В частности, ищется набор правил фильтрации, которые на выходе дают подмножество «хороших» кредитов, оплаченных преимущественно вовремя. Вручную проанализировав сотни заявок, вы приобретете огромный опыт, позволяющий отличать хорошую заявку от плохой.[1] После проверки накопившихся данных о погашениях кредитов вы обнаружите в процедуре проверки кредитной надежности определенные тенденции:[2]

- большинство заемщиков с лимитом кредитования более $7500 не выполняли своих обязательств по кредиту;
- большинство заемщиков, не имеющих контокоррентного счета, погашали кредит в срок.

Теперь можно спроектировать механизм фильтрации, который уменьшит количество заявлений, вручную проверяемых по двум вышеуказанным критериям.

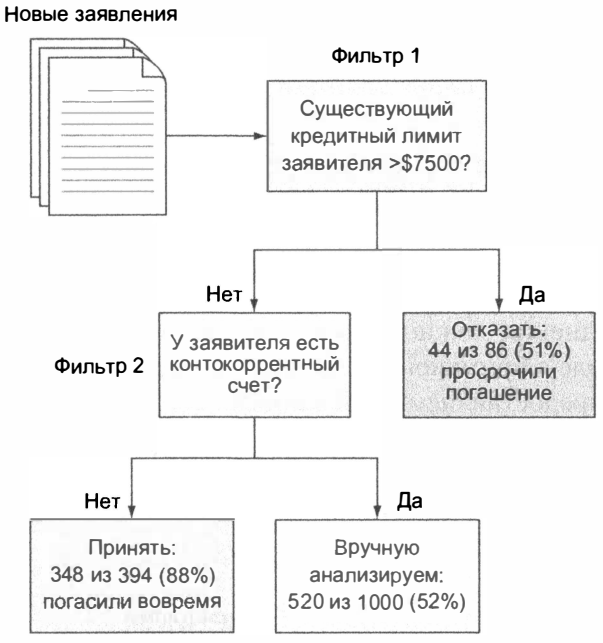
Первый фильтр будет автоматически отказывать всем заемщикам с лимитом кредитования более $7500. Ведь накопленные данные показали, что 44 из 86 заемщиков, взявших кредит, превышающий $7500, просрочили дату его погашения. Примерно 51% из просивших максимально возможный кредит не выполнили своих обязательств по сравнению с остальными 28%. Этот фильтр кажется хорошим способом отсечения заемщиков с высоким риском невозврата. Но следует учесть, что о столь крупном кредите шла речь только в 8,6% (86 из 1000) одобренных заявок. Это значит, что более 90% анкет вам все равно придется обрабатывать вручную. То есть требуется дополнительная фильтрация.

Второй фильтр автоматически принимает любого заявителя, у которого отсутствует контокоррентный счет. Это кажется отличным решением, так как вовремя погасило свой кредит 348 из 394 (88%) заемщиков, не имеющих контокоррентного счета. Добавление этого фильтра увеличивает количество автоматически принимаемых или отклоняемых заявлений

[1] Для определения атрибутов исходных данных, сильнее всего связанных с итоговым событием погашения кредита, можно также использовать методы статистической корреляции.

[2] В этом примере использовался набор данных *German Credit Data*. Он доступен на странице *http://mng.bz/95r4*.

до 45%. Соответственно, вручную остается обработать чуть больше половины анкет. Иллюстрация 1.4 демонстрирует эти правила фильтрации в виде схемы.



Илл. 1.4. Фильтрация по двум бизнес-правилам позволяет уменьшить количество проверяемых вручную заявок до 52%

Два бизнес-правила почти в два раза увеличивают количество обрабатываемых заявок без найма второго аналитика, так как вручную решение принимается только по 52% новых заемщиков. Кроме того, имея данные по 1000 заявок с известным результатом, вы предвидите, что механизм фильтрации будет ошибочно отклонять 42 заявки из каждой 1000 (4,2%) и ошибочно принимать 46 из каждой 1000 (4,6%).

По мере роста бизнеса хотелось бы, чтобы система автоматически принимала или отклоняла все больше заявок, не приводя к росту убытков из-за нарушения долговых обязательств. Для этого потребуются новые бизнес-правила. И скоро начнутся проблемы:

- По мере усложнения системы фильтрации все сложнее находить эффективные фильтры.

- Бизнес-правила становятся настолько запутанными, что их отладка и удаление устаревших, ставших малозначительными правил превращается в практически нереальную задачу.
- Формирование таких правил не является статистически строгим. Вам кажется, что путем более тщательного изучения данных можно найти лучшие «правила», но не факт, что они на самом деле существуют.
- Признаки возвращаемого кредита со временем изменяются — например, из-за изменения контингента заемщиков, — но система к этому не адаптируется. Для поддержания актуального состояния ее нужно постоянно редактировать.

Все эти затруднения можно свести к одному критическому недостатку в самом подходе с применением бизнес-правил: система автоматически не учится на предоставляемых ей данных.

Системы, управляемые данными, от простых статистических моделей до более проработанных обучающихся рабочих процессов, позволяют избежать проблем такого рода.

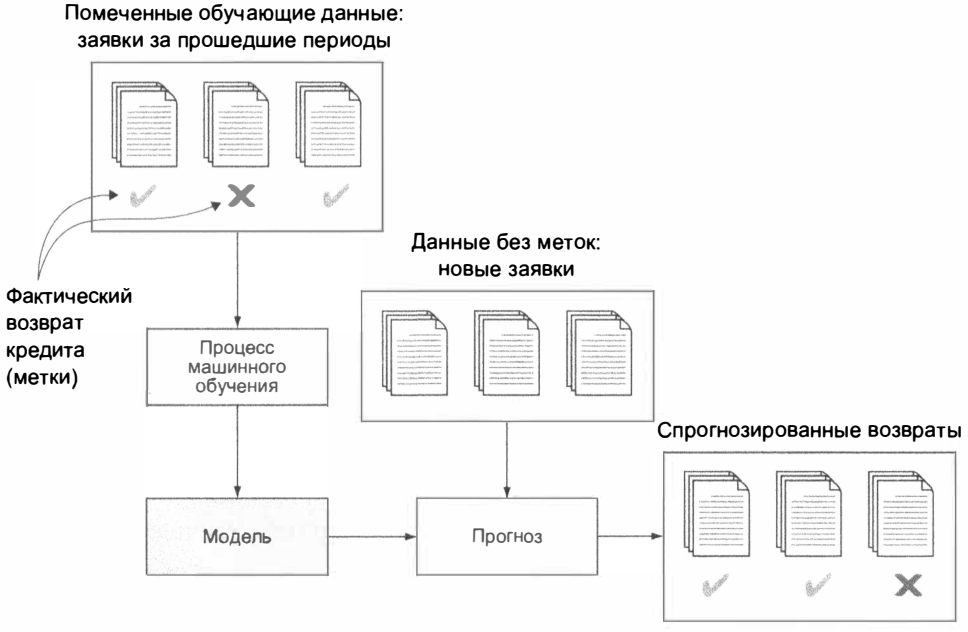
1.2.2. Подход с машинным обучением

Наконец, вы решили сделать процесс рассмотрения кредитных заявок полностью автоматизированным и управляемым данными. Машинное обучение прекрасно подходит для таких случаев, ведь сама его природа позволяет поддерживать нужный темп, как бы быстро ни возрастал приток новых заявлений. Более того, система извлекает оптимальные решения *непосредственно из поступающих данных*, не заставляя заранее жестко программировать произвольным образом выбираемые бизнес-правила. Такой переход от процесса анализа на основе бизнес-правил к машинному обучению означает, что точность решений не только возрастет, но и будет увеличиваться по мере выдачи новых кредитов. Вы будете полностью уверены, что ML-система дает оптимальный результат практически без вашего вмешательства.

При машинном обучении данные служат основой для более глубокого представления о проблеме. Чтобы определить наилучший порядок действий по отношению к каждой новой кредитной заявке, ML-система

использует данные за прошлые периоды, на *которых она обучалась*. Для запуска машинного обучения системы одобрения кредитов берутся данные о 1000 выданных кредитов. Это сведения из заявлений заемщиков и информация о возврате кредита. Сведения из заявлений в свою очередь состоят из набора *признаков* (features) — численных или категориальных показателей, фиксирующих значимые характеристики каждого заявления. К признакам относятся, к примеру, оценка кредитоспособности заявителя, его пол и род занятий.

Иллюстрация 1.5 показывает схему обучения модели на основе данных за прошедший период. При поступлении новой заявки на кредит вероятность будущего корректного погашения мгновенно вычисляется по указанным в заявке сведениям.



Илл. 1.5. Базовая схема машинного обучения на примере выдачи кредитов

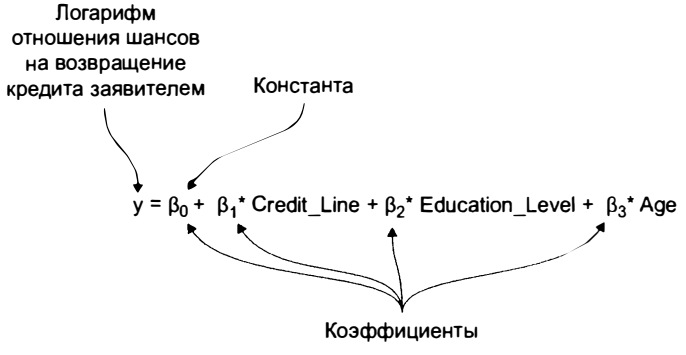
После этого модель с машинным обучением определяет, как можно использовать данные каждого заявления для *наилучшего прогнозирования* ситуации. Обнаруживая в наборе обучающих данных закономерности и применяя их, процедура машинного обучения создает модель (пока ее

можно рассматривать как черный ящик), прогнозирующую поведение каждого заемщика на основе предоставленной им информации.

Следующим шагом является выбор алгоритма. Виды машинного обучения варьируются от простых статистических моделей до нетривиальных подходов. Мы сравним два примера: простую параметрическую модель и непараметрический ансамбль деревьев классификации. Не стоит пугаться сложной терминологии. Скоро вы увидите, что в машинном обучении используется множество алгоритмов и множество способов их категоризации.

Большинство традиционных статистических экономических моделей попадает в первую категорию. В параметрических моделях соотношение между результатом и входными данными выражается через простые фиксированные уравнения. Данные применяются для определения оптимальных значений неизвестных частей уравнения. В эту категорию попадают: модель линейной регрессии, модель логистической регрессии и модель авторегрессии с лаговым оператором L. Все они будут подробно рассматриваться в главе 3.

В нашем примере для моделирования процесса одобрения кредита возьмем логистическую регрессию. В ней *логарифм отношения шансов* (log odds) каждого конкретного кредита на погашение рассматривается как линейная функция входных признаков. К примеру, если каждая новая заявка содержит три значимых признака: кредитный лимит заявителя (`Credit_Line`), уровень его образования (`Education_Level`) и возраст (`Age`), — логистическая регрессия попытается предсказать логарифм отношения шансов на невозврат кредита (мы обозначим его y) при помощи следующего уравнения:



ЛОГАРИФМ ОТНОШЕНИЯ ШАНСОВ

Отношение шансов — это один из способов выражения вероятности. Вам, без сомнения, доводилось слышать такие выражения, как «шансы любимой команды на выигрыш 3 к 1». Отношением (odds) называется вероятность успеха (например, выигрыша), поделенная на вероятность неудачи (проигрыша). Математически это выражается следующим образом:

Odds(A) = P(A) / P(~A) = вероятность A, поделенная на вероятность не A

Соответственно, отношение 3 к 1 эквивалентно 0,75 / 0,25 = 3, а log(3) = 0,47712...

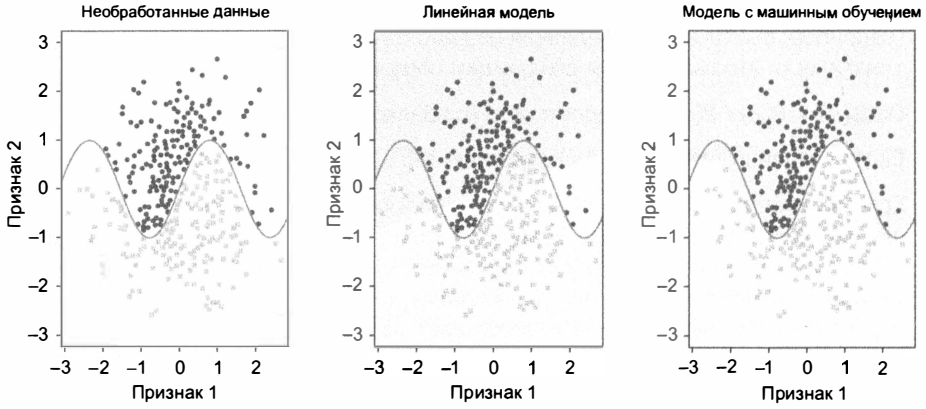
Если в качестве события A рассмотреть броски симметричной монеты, отношение шансов на выпадение «орла» составит 0,5 / 0,5 = 1. Log(1) = 0. Получается, выражение log(Odds) может иметь любое вещественное значение. Логарифм отношения шансов, стремящийся к $-\infty$, означает практически невероятное событие. Значение ∞ указывает на практически гарантированное наступление события, в то время как log(1) = 0 соответствует равной вероятности как одного, так и другого варианта. Применение логарифма отношения шансов вместо обычной вероятности — всего лишь математический прием, облегчающий некоторые вычисления, так как, в отличие от вероятности, логарифм не ограничен значениями от 0 до 1.

Оптимальное значение всех коэффициентов уравнения (в рассматриваемом случае это β_0, β_1, β_2 и β_3) определено на основе 1000 примеров обучающих данных.

Когда соотношение между входными данными и результатом удается выразить формулой, результат (y) можно легко предсказать по значениям признаков (кредитный лимит, уровень образования и возраст). Остается только определить, какие значения β_1, β_2 и β_3 дают наилучший результат, воспользовавшись данными за прошедшие периоды.

Но при более сложных зависимостях между входными данными и результатом такие алгоритмы, как логистическая регрессия, показывают свою ограниченность. В качестве примера рассмотрим набор данных из левой части илл. 1.6. Есть два исходных признака, и нужно отнести каждую точку на графике к одному из двух классов. В двумерном пространстве признаков эти классы разделены кривой, которую невозможно описать линейными уравнениями. Она называется *решающей границей* (decision boundary). Центральная часть рисунка демонстрирует результат, который дает для предоставленных данных модель на базе алгоритма логистиче-

ской регрессии. Мы получаем две области, разделенные прямой линией, что ведет к многочисленным ошибкам классификации (точки попадают не в ту область).



Илл. 1.6. В данной классификации отдельные элементы данных могут попасть в класс окружностей или в класс квадратов. Все представленные данные находятся в двумерном пространстве признаков с нелинейной решающей границей, обозначенной кривой линией. В то время как простая статистическая модель практически не в состоянии точно классифицировать данные (в центре), ML-модель (справа) дает возможность без особых усилий найти реальную границу между классами

В данном случае проблема в том, что на центральном рисунке мы попытались объяснить сложное нелинейное явление с помощью простой *параметрической* модели. Формальное определение параметрических и непараметрических моделей требует слишком сложных для данной книги математических формул. Суть же состоит в том, что параметрические модели прекрасно применимы только в случаях, когда заранее примерно известно соотношение между входными данными и прогнозируемым результатом. При наличии достаточного количества сведений о нелинейном соотношении иногда можно преобразовать входные данные или результат таким образом, что параметрическая модель будет работать. Скажем, если частота наблюдения некоего заболевания среди населения выше у пожилых людей, вы можете обнаружить линейное соотношение между вероятностью заражения и квадратом возраста. Но в реальности, как правило, приходится иметь дело с задачами, для которых предсказать подобные закономерности практически невозможно.

Нам требуются более гибкие модели, умеющие автоматически выявлять в данных сложные тенденции и структуры без предварительной инфор-

мации о виде будущего шаблона. Именно здесь на помощь приходят *непараметрические* алгоритмы машинного обучения. В правой части илл. 1.6 вы видите результат применения к задаче непараметрического обучающего алгоритма (в данном случае это *классификатор на базе алгоритма «случайный лес»*). Очевидно, что предсказанная решающая кривая намного лучше совпадает с настоящей границей раздела, а значит, точность классификации намного выше, чем у параметрической модели.

Непараметрические ML-модели дают высокий уровень точности при работе со сложными многомерными реальными наборами данных. Именно поэтому они выбираются для решения множества управляемых данными задач. В их число попадают такие широко используемые методы машинного обучения, как, к примеру, метод k-ближайших соседей, ядерное сглаживание, метод опорных векторов, деревья принятия решений и композиционное обучение. Все эти подходы будут подробно рассмотрены в следующих главах, кроме того, в приложении вы найдете обзор важнейших алгоритмов. Привлекательность же линейных алгоритмов связана с совсем другими свойствами. Их проще интерпретировать, они быстрее обсчитываются и проще масштабируются при увеличении набора данных.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

Учебник Г. Джеймса «An Introduction to Statistical Learning»[1] детально знакомит с наиболее распространенными подходами к машинному обучению на уровне, доступном читателям без специальной подготовки в области математики и статистики. Английский вариант книги с согласия издательства доступен для скачивания в формате PDF на сайте авторов (*http://www-bcf.usc.edu/~gareth/ISL/ISLR%20First%20Printing.pdf*).

Но вернемся к задаче масштабирования микрофинансовой организации. В этом случае больше всего подойдет непараметрическая ML-модель. Она может обнаружить правила, которые вы вывели вручную, хотя, возможно, они будут выглядеть немного иначе для оптимизации статистических преимуществ. Скорее всего, модель самостоятельно выведет другую, более глубокую корреляцию входных параметров и желаемого результата, которую вы не могли даже представить.

[1] *Джеймс Г.* Введение в статистическое обучение с примерами на языке R. — М.: ДМК Пресс, 2016.

Вы можете не только автоматизировать рабочий процесс, но и увеличить его точность, из чего непосредственно вытекает рост коммерческой стоимости. Предположим, непараметрическая ML-модель дает прогноз с точностью на 25% большей, чем у алгоритма логистической регрессии. Это означает, что при обработке новых заявок будет допущено меньше ошибок: вы одобрите меньше недобросовестных заемщиков и откажете меньшему числу добросовестных клиентов. В результате возрастет средняя доходность по кредитам, что позволит выдать больше кредитов и получить более высокий доход.

Надеемся, что вы уже поняли, насколько мощным инструментом является машинное обучение. Но перед определением основ рабочего процесса ML рассмотрим его преимущества, а заодно расскажем об ожидающих вас трудностях.

1.2.3. Пять преимуществ машинного обучения

Перед завершающей стадией нашего примера рассмотрим самые существенные преимущества систем с машинным обучением в сравнении с общепринятыми альтернативами, такими как ручной анализ, жестко запрограммированные бизнес-правила и простые статистические модели. Вот они:

- *Точность.* Машинное обучение использует данные для создания принимающей решение программы, оптимизированной под поставленную задачу. По мере накопления данных автоматически возрастает точность прогнозов.

- *Автоматизация.* По мере подтверждения и отбрасывания ответов ML-модель может автоматически обнаруживать новые шаблоны. Это позволяет встраивать машинное обучение непосредственно в автоматизированные рабочие процессы.

- *Скорость.* Машинное обучение дает ответы за доли секунды после поступления новой информации, позволяя системам реагировать в реальном времени.

- *Возможность настройки.* Многие задачи, управляемые данными, можно решить с помощью машинного обучения. Модели строятся на базе ваших собственных данных и допускают настройку под любую систему мер, принятую в вашем бизнесе.

- *Масштабируемость.* При росте бизнеса ML-модель легко приспосабливается к увеличивающимся объемам данных. Некоторые алгоритмы можно использовать для обработки множества данных на разных вычислительных машинах в облаке.

1.2.4. Сложности

Естественно, что для получения вышеперечисленных преимуществ требуется разобраться с рядом сложностей. Они зависят от параметров стоящей перед вами задачи и могут как преодолеваться элементарным образом, так и требовать колоссальных усилий.

Чаще всего требуется получить данные в годной к употреблению форме. Было подсчитано, что специалисты по работе с данными тратят на их подготовку 80% времени.[1] Без сомнения, вы слышали, что при коммерческой деятельности в настоящее время фиксируется намного больше данных, чем когда-либо раньше, и это действительно так. Возможно, вам доводилось слышать и то, что эти данные называют «выхлопом» бизнес-процессов. Другими словами, драгоценные сведения нельзя напрямую использовать в качестве входных данных для ML-систем. Для извлечения из этих массивов полезной информации требуется кропотливая и нудная работа.

Фактически наша задача — сформулировать проблему таким образом, чтобы к ней можно было применить методы машинного обучения и получить имеющие практическую ценность и измеримые результаты. В рассматриваемом примере цель очевидна — предсказать, кто выплатит кредит вовремя, а кто нет. Классификация легко применима, а результат легко измерим. К счастью, реальные проблемы иногда бывают настолько же простыми. Например, предсказать, исходя из всех имеющихся у нас сведений о потенциальных покупателях (а их очень много), приобретут ли они наш товар. Такие задачи решаются без труда.

Вот более сложный пример — найти оптимальное сочетание средств массовой информации и комбинацию рекламных блоков для повышения узнаваемости торговой марки новой линии продуктов. Сама постановка задачи требует найти способ измерения узнаваемости торговой марки, по-

[1] Из статьи Steve Lohr, "*For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights*" (New York Times, August 17, 2014), доступной на странице *http://mng.bz/7W8n*.

лучить информацию о вариантах выбора рекламных средств и подобрать данные, отражающие релевантный опыт с альтернативами и связанными с ними результатами.

Когда нужен сложный результат, выбор алгоритма и способа его применения сам по себе начинает требовать гигантских усилий. Исследователи, работающие в области кардиологии над прогнозами вероятности послеоперационных осложнений, обладают умопомрачительным количеством данных на каждого пациента, но для ML-алгоритмов бесполезны необработанные данные электрокардиографии (ЭКГ) и результаты секвенирования ДНК. *Проектирование признаков* (feature engineering) представляет собой процесс преобразования таких входных данных в подходящие для предсказывающей модели признаки.

Нельзя не упомянуть и проклятие всех, кто занимается созданием прогнозирующих моделей, — модель, которая идеально работает на обучающих данных, но демонстрирует полную неспособность к достоверным прогнозам на основании неизвестных ранее данных. Причиной в большинстве случаев становится *переобучение* (overfitting).

Вы убедитесь, что машинное обучение позволяет решить множество проблем, иногда кардинально упрощая процесс поиска решения. Хотя имеет смысл отметить, что полезность решения далеко не всегда соответствует усилиям, затраченным на его получение. Более того, машинное обучение не является панацеей от всех бед. Но после прочтения этой книги вы увидите, насколько хорошо оно справляется с множеством определяемых данными задач.

1.3. Рабочий процесс: от данных до внедрения

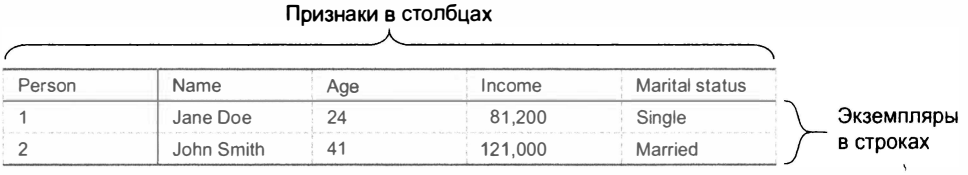
В этом разделе мы рассмотрим базовый процесс интеграции моделей машинного обучения в приложения или конвейеры данных. Его можно разбить на пять стадий: подготовка данных, построение модели, оценка, оптимизация и прогноз на новых данных. Этапы следуют друг за другом в определенном порядке, но большинство реальных приложений с машинным обучением требует множественного повторения каждого этапа в процессе последовательных приближений. Они подробно описываются в главах со 2-й по 4-ю, пока же мы дадим общее описание, чтобы показать, с чем вам предстоит иметь дело. Схема рабочего процесса представлена на

илл. 1.7, и в следующих разделах мы детально разберем все упомянутые концепции. Этот рисунок будет то и дело попадаться в книге при рассмотрении различных этапов рабочего процесса ML.

Илл. 1.7. Рабочий процесс реальных систем с машинным обучением. Данные за прошедшие периоды позволяют построить модель на базе ML-алгоритма. После этого нужно оценить производительность модели и оптимизировать ее точность и масштабируемость в соответствии с выдвинутыми требованиями. Готовая модель дает прогнозы для новых данных

1.3.1. Сбор и подготовка данных

Сбор и подготовка данных для систем с машинным обучением обычно влечет за собой их представление в виде таблицы, если изначально они имеют другую форму. Представьте, что данные распределены по строкам и столбцам, причем каждая строка соответствует изучаемому *экземпляру* (instance), а столбец — значению этого экземпляра. Существует несколько исключений, но можно смело утверждать, что большинство алгоритмов машинного обучения требует данных именно в таком формате. По поводу исключений мы пока беспокоиться не будем. Рассмотрим илл. 1.8, на которой простой набор данных представлен в виде таблицы.



Илл. 1.8. В наборе данных, представленном в виде таблицы, строки называются *экземплярами*, а столбцы представляют собой *признаки*

При взгляде на таблицу первым делом бросается в глаза, что столбцы, как правило, содержат данные одного типа, в то время как данные в строках принадлежат разным типам. На илл. 1.8 мы встречаемся с данными четырех типов: строковая переменная `Name`, целочисленная переменная `Age`, вещественная переменная `Income` и категориальная переменная `Marital status` (принимающая дискретное число значений). Такой набор данных называют гетерогенным (в отличие от гомогенного), и в главе 2 мы поговорим о том, как и зачем выполняется приведение некоторых типов данных к другим типам в зависимости от конкретного алгоритма машинного обучения.

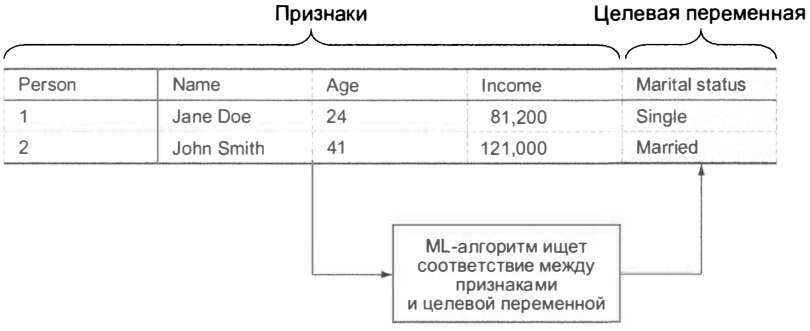
Реальные данные могут быть «запутаны» разными способами. Представьте, что на этапе сбора данных измерить какое-то значение не представляется возможным. Нельзя и вернутся назад, чтобы отыскать недостающий фрагмент информации. В подобных случаях некоторые ячейки таблицы *останутся незаполненными*, что усложнит как построение модели, так и последующее прогнозирование. Иногда сбор данных осуществляется вручную, а мы все знаем, как легко делаются ошибки при выполнении повторяющихся задач. В результате часть сведений оказывается некорректной. Вы должны уметь работать с подобными сценариями или, по крайней мере, знать, как конкретный алгоритм ведет себя при наличии недостоверных данных. Подробно методы работы с отсутствующими и недостоверными данными будут рассматриваться в главе 2.

1.3.2. Обучение модели на данных

Первый этап построения успешной системы с машинным обучением — это формулировка вопроса, ответ на который должны дать наши данные. Например, наша простая таблица с личными данными позволяет построить ML-модель, предсказывающую семейное положение заявителя. Такая

информация может пригодиться, например, при выборе демонстрируемой пользователю рекламы.

При этом переменная `Marital status` будет использоваться как *целевая* (target) или как *метка* (label), а все прочие переменные станут *признаками* (features). Наш ML-алгоритм должен понять, каким образом набор входных признаков позволяет успешно предсказывать значение целевой переменной. После этого вы сможете пользоваться построенной моделью для предсказания семейного положения заявителя, когда оно по каким-то причинам не указано. Иллюстрация 1.9 демонстрирует этот процесс на примере нашего небольшого набора данных.



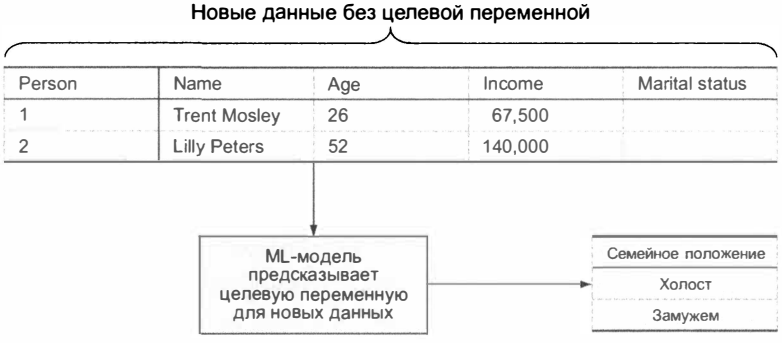
Илл. 1.9. Процесс моделирования с машинным обучением

На данном этапе ML-алгоритм можно представить в виде волшебной коробочки, которая выполняет отображение входных признаков на выходные данные. Но для построения работающей модели нужно более двух рядов. Одним из преимуществ алгоритмов машинного обучения в сравнении с другими распространенными методами является умение обрабатывать множество признаков. Иллюстрация 1.9 демонстрирует только четыре признака, из которых идентификатор заявителя и его имя никак не помогут предсказать его семейное положение. Некоторые алгоритмы в достаточной степени невосприимчивы к неинформативным признакам, в то время как другие дают более точные предсказания, когда такие признаки убираются из рассмотрения. Более подробно различные типы алгоритмов и их производительность в случае разных задач и наборов данных рассматриваются в главе 3.

Впрочем, ценные сведения порой могут быть извлечены и из неинформативных на первый взгляд признаков. Например, такой признак, как

местоположение, сам по себе кажется бесполезным, но он может дать информацию о плотности населения. Такой вариант усовершенствования данных, называемый *извлечением признаков* (feature extraction), крайне важен в реальных ML-проектах и будет рассматриваться в главах 5 и 7.

Построив ML-модель, вы можете делать прогнозы для новых данных с неизвестной целевой переменной. Этот процесс показан на илл. 1.10.



Илл. 1.10. Применение модели для предсказания новых данных

Предсказанная целевая переменная возвращается в той форме, в которой она фигурировала во взятых для обучения модели исходных данных. Прогнозирование с помощью модели, по сути, является заполнением пустого столбца новыми значениями. Некоторые ML-алгоритмы также включают в результат своей работы связанные с каждым классом вероятности. В рассматриваемом примере *вероятностная* ML-модель выдает для каждого нового заявителя два значения: вероятность, что человек состоит в браке, и вероятность того, что он свободен.

Мы опустили некоторые детали, но, в принципе, вы только что наблюдали за проектированием ML-системы. Любая система, связанная с машинным обучением, занимается созданием моделей и их применением для получения прогнозов. Представим базовый рабочий процесс ML в виде псевдокода, чтобы вы еще раз убедились, насколько он прост.

Листинг 1.1. Исходная структура программы с рабочим процессом ML

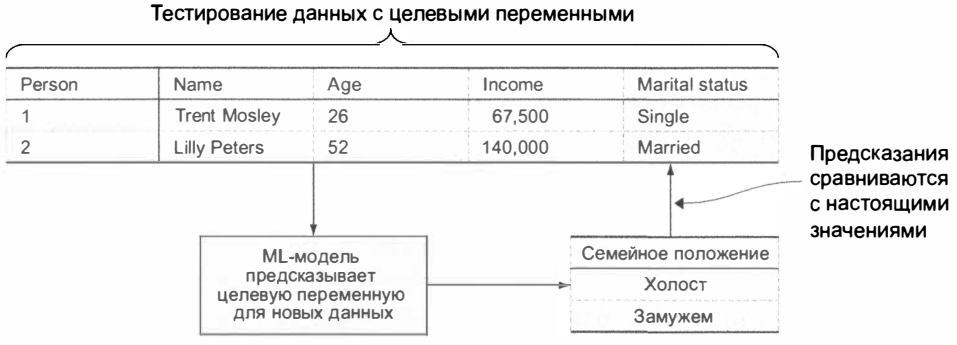
```
данные = загрузка_данных("данные/люди.csv")
модель = строим_модель(данные, цель="Семейное положение")
новые_данные = загрузка_данных("данные/новые_люди.csv")
прогнозы = модель.предсказывает(новые_данные)
```

Ни одна из этих функций пока не запрограммирована, мы показали только базовую структуру. К главе 3 вы поймете, что представляют собой все эти этапы. Остальная часть книги (главы с 4-й по 10-ю) учит строить модели, наилучшим образом подходящие для решения конкретных задач.

1.3.3. Оценка производительности модели

Системы с машинным обучением практически никогда не используются, пока не будет проверена их производительность. В этой главе мы для простоты опускаем многие детали, поэтому просто представим, что вы уже знаете, как строить модели и получать прогнозы с их помощью. И теперь нужен хитрый прием, который позволит понять, насколько готовая модель справляется со стоящими перед ней задачами.

Возьмем набор данных и представим, что целевая переменная неизвестна. Построим на их основе модель и используем их в качестве тестовых данных для нескольких прогнозов. Процесс тестирования схематично показан на илл. 1.11.



Илл. 1.11. Используя тестовый набор для оценки производительности модели, мы делаем вид, что целевые переменные неизвестны, и сравниваем предсказанные значения с настоящими

А вот так рабочий процесс можно представить с помощью псевдокода.

Листинг 1.2. Программа ML-процесса с оценкой модели

```
данные = загрузка_данных(...)
тренировочные_данные, тестовые_данные = разделение_данных(данные)
модель = построение_модели(тренировочные_данные, цель="Marital status")
истиные_значения = тренировочные_данные.извлечь_столбец("Marital status")
прогноз = модель.предсказать(тестовые_данные)
точность = сравнить_прогнозы(прогнозы, истинные_значения)
```

Теперь можно сравнить результаты прогнозов с известными «истинными» значениями и оценить точность модели. В псевдокоде этот алгоритм скрыт внутри функции `сравнить_прогнозы`. Большая часть главы 4 посвящена рассмотрению этой функции для различных типов задач.

1.3.4. Оптимизация производительности модели

В главе 4 будет рассмотрен последний фрагмент основной задачи машинного обучения. Вы увидите, как результаты оценки модели позволяют вернуться назад и сделать модель лучше. Увеличение точности достигается тремя способами:

- *Редактирование параметров модели.* Каждый ML-алгоритм обладает набором параметров, оптимальные значения которых зачастую зависят от типа и структуры данных. При этом на производительность модели может влиять значение как любого отдельного параметра, так и их произвольной комбинации. Мы рассмотрим различные способы поиска и подбора оптимальных значений и покажем, как выбрать наилучший алгоритм для имеющегося набора данных.

- *Выбор подмножества признаков.* Зачастую задачи, связанные с машинным обучением, включают множество признаков, и вносимые ими помехи порой мешают алгоритму обнаружить верную закономерность, даже если сами по себе эти признаки являются информативными. Во многих случаях наличие большого количества данных — благо, но иногда это становится проклятием. А так как заранее никогда не известно, как это скажется на эффективности модели, следует тщательно выбирать признаки, которые дадут наиболее универсальную и точную модель.

- *Предварительная обработка данных.* В Интернете можно найти наборы данных для машинного обучения, которые без проблем «скармливаются» многим ML-алгоритмам. Реальные же данные в большинстве случаев находятся далеко не в таком чистом виде и подлежат очистке и обработке. Этот процесс называют *выпасом данных* (data munging или data wrangling). Набор может включать имена, которые пишутся по-разному, хотя и относятся к одной и той же сущности, а также отсутствующие или недостоверные

значения. Все это негативным образом сказывается на эффективности модели. Может показаться, что речь идет о редких случаях, но вы удивитесь, когда увидите, насколько часто подобные вещи встречаются даже в тщательно разработанных управляемых данными структурах.

Итак, вы получили первые представления об основных элементах машинного обучения. В следующем разделе мы дадим обзор более сложных функциональных возможностей, а потом перейдем к их детальному рассмотрению.

1.4. Усовершенствованные способы повышения эффективности

В предыдущем разделе вы познакомились с основными этапами любого связанного с машинным обучением проекта, теперь же мы рассмотрим дополнительные техники, часто применяемые для дальнейшего повышения производительности моделей. Для некоторых наборов данных и задач описанные техники позволяют значительно увеличить точность предсказаний, хотя порой за это приходится платить скоростью работы на стадиях обучения и прогнозирования. Детально эти техники будут рассматриваться в главах с 5-й по 10-ю, пока же мы дадим вам общее представление.

1.4.1. Предварительная обработка данных и проектирование признаков

Различные типы данных и распространенные способы их упорядочивания будут рассматриваться в главе 2. Но можно не просто очистить данные, а сделать еще один шаг и извлечь из них дополнительное значение, положительно влияющее на производительность модели.

В любой предметной области требуются специальные знания, позволяющие решать, какие именно данные подлежат сбору. Именно эти знания применяются для извлечения из собранных данных ценной информации, которая добавляется к признакам строящейся модели. Мы называем этот процесс *проектированием признаков* (feature engineering), и после того,

как вы полностью освоитесь с основными этапами рабочего процесса ML, вы обнаружите, что практически все время уходит на эту часть процедуры оптимизации. Кроме всего прочего, это творческая часть машинного обучения, ведь приходится задействовать знания и воображение, чтобы найти способы усовершенствования модели путем исследования данных и извлечения из них скрытой информации. Вы будете широко использовать наши статистически обоснованные этапы оценки и оптимизации модели, отделяя привлекательные на первый взгляд идеи от по-настоящему полезных вещей. Вот несколько важных примеров проектирования признаков:

- *Дата и время.* Эти переменные часто встречаются в наборах данных, но для ML-алгоритмов, которым, как правило, требуются необработанные числа или категории, сами по себе они бесполезны. Впрочем, из них можно извлечь ценную информацию. Например, при выборе демонстрируемой рекламы, несомненно, важно знать время, день недели и время года. Благодаря проектированию признаков эти сведения извлекаются из переменных даты и времени и делаются доступными для модели.

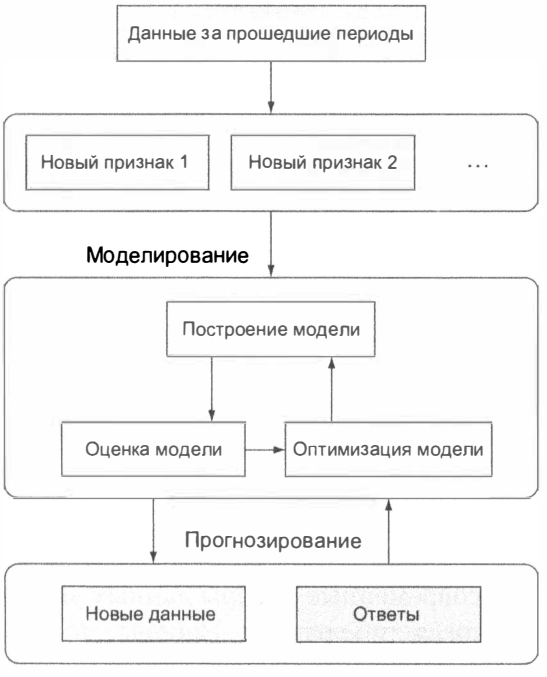
Возникают такие переменные и при наблюдении за повторяющейся активностью, например за повторными визитами посетителя на сайт в течение месяца или года. В этом случае они позволяют вычислить промежутки между визитами, что может дать информацию для прогнозирования. Скажем, сайты интернет-магазинов пользователи чаще посещают перед совершением покупки, чтобы осмотреть и сравнить товары и цены.

- *Местоположение.* В некоторых наборах данных присутствуют координаты в виде широты и долготы или названия места. Такая информация иногда оказывается полезной сама по себе, но из нее можно извлечь и дополнительные параметры, необходимые для решения более специфических задач. Скажем, для прогнозирования результатов выборов по стране потребуется извлечь информацию о плотности населения, среднем доходе и процентной доле бедных.

- *Цифровые средства коммуникации.* В эту группу попадают такие данные, как тексты, документы, изображения и видео. Проекти-

рование признаков, благодаря которому эти данные становятся доступными для работы, составляет сложную часть таких проектов, как конкурс «Собаки против кошек». Изображения первым делом дают информацию о контурах, формах и спектре цветов. С помощью математических преобразований они классифицируются, давая набор признаков, пригодных для алгоритмов классификации.

Надеемся, вы убедились в важности проектирования признаков для реальных ML-проектов. Подробно этот процесс будет рассматриваться в главах 5 и 7, где вы познакомитесь с конкретными техниками. Вы научитесь добавлять эти техники к рабочему процессу машинного обучения таким образом, чтобы модель не сделалась слишком сложной и не стала подверженной переобучению. Иллюстрация 1.12 показывает интеграцию проектирования признаков в рабочий процесс машинного обучения, описанный в разделе 1.3.



Илл. 1.12. Этап проектирования признаков, вставленный в исходный рабочий процесс машинного обучения

1.4.2. Непрерывное совершенствование моделей

В большинстве своем традиционные ML-модели статические и перестраиваются редко. Но во многих случаях данные и прогнозы будут возвращаться обратно в систему, и крайне желательно, чтобы модель постепенно совершенствовалась и адаптировалась к изменениям в этих данных. Существуют ML-алгоритмы, поддерживающие данный тип *динамического обучения* (online learning). Эти алгоритмы и их потенциальные недостатки будут рассматриваться в главе 8. Иллюстрация 1.13 показывает, каким образом непрерывное переучивание встраивается в рабочий процесс ML.

Илл. 1.13. На этой диаграмме динамической ML-системы прогнозы возвращаются в модель для ее непрерывного улучшения

1.4.3. Масштабирование моделей

Всем известно, что современные наборы данных увеличиваются в размерах быстрее, чем когда-либо раньше. Наборы для методов машинного обучения с учителем, в которых целевые ответы входят в обучающую выборку, были традиционно небольшими, так как для получения ответов

прибегали к помощи человека. В настоящее время множество данных (включая ответы) продуцируется непосредственно измерительными элементами, машинами или компьютерами, и для обработки таких объемов требуются уже масштабируемые ML-алгоритмы.

Глава 9 содержит описание методов машинного обучения, умеющих масштабироваться по мере роста набора данных. Вы найдете там сравнение таких алгоритмов друг с другом и с немасштабируемыми алгоритмами.

1.5. Заключение

Эта глава представила вам машинное обучение как более совершенный, более ориентированный на данные подход к принятию решений. Вот основные положения, которые вам следует усвоить:

- В отличие от систем на базе бизнес-правил, алгоритмы машинного обучения создают на основе данных свои собственные модели. Системы с обучением обобщают информацию, изучая ее на примерах с известным результатом.
- Зачастую машинное обучение является более точным, автоматизированным, быстрым, настраиваемым и масштабируемым, чем конструируемые вручную системы на базе бизнес-правил.
- К сложным аспектам машинного обучения относятся распознавание и формулировка задач, к которым оно применимо, получение данных и преобразование их в пригодную для использования форму, обнаружение корректных алгоритмов, проектирование признаков и переобучение.
- Базовый рабочий процесс машинного обучения состоит из подготовки данных, построения модели, оценки этой модели, ее оптимизации и предсказания на новых данных.
- Модели с динамическим обучением непрерывно переучиваются, используя результаты своих же прогнозов для собственного обновления.

1.6. Терминология

Термин	Определение
Экземпляр (instance) или пример (example)	Один объект, наблюдение, транзакция или запись
Цель (target) или метка (label)	Численный или категориальный (метка) атрибут, представляющий интерес. Это переменная, которая предсказывается для каждого нового экземпляра
Признаки (features)	Входные атрибуты, используемые для предсказания целевой переменной. Они могут быть как числовыми, так и категориальными
Модель (model)	Математический объект, описывающий соотношение между признаками и целевой переменной
Обучающая выборка (training data)	Набор экземпляров с известной целевой переменной, используемый для подгонки ML-модели
Воспроизведение (recall)	Использование модели для предсказания целевой переменной или метки
Машинное обучение с учителем (supervised machine learning)	Машинное обучение, в котором при наличии примеров с известным результатом тренировочный процесс выводит функцию, связывающую входные значения с результатом
Самообучение (unsupervised machine learning)	Техника машинного обучения, не использующая примеры с метками, а пытающаяся обнаружить скрытые закономерности в данных, не имеющих меток
Рабочий процесс ML (ML workflow)	Стадии процесса машинного обучения: подготовка данных, построение модели, оценка, оптимизация и предсказание
Динамическое машинное обучение (online machine learning)	Форма машинного обучения, в которой делаются предсказания и обновляется модель для каждого нового примера

В главе 2 вы займетесь практическими задачами сбора данных и их подготовки к применению в процессе машинного обучения, а также научитесь использовать визуализации для получения аналитической информации, позволяющей выбирать оптимальные инструменты и методы.

2 Реальные данные

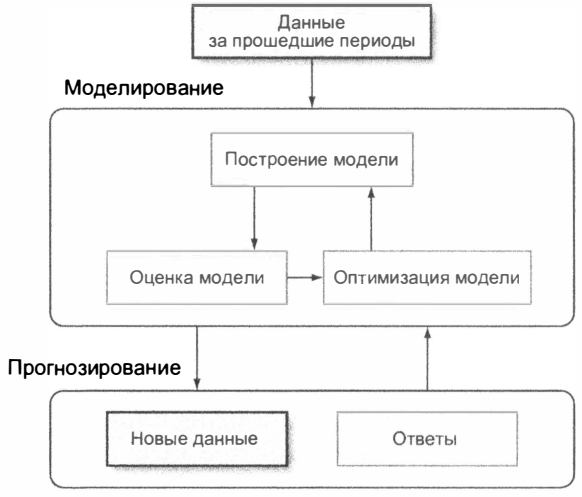
В этой главе мы:

- ✓ приступаем к машинному обучению;
- ✓ формируем обучающую выборку;
- ✓ используем техники визуализации данных;
- ✓ подготавливаем данные для машинного обучения.

При обучении с учителем мы используем данные, чтобы научить автоматизированные системы принимать безошибочные решения. Алгоритмы машинного обучения спроектированы таким образом, что умеют обнаруживать в сведениях за прошедшие периоды закономерности и ассоциации. Они учатся на этих данных и строят по результатам модель, которая будет предсказывать важные атрибуты для новых данных.

Соответственно, тренировочные данные являются фундаментом машинного обучения. Качественные данные позволяют точно выявить тонкие нюансы и корреляции и построить на их основе высокоточную прогнозирующую систему. В то же время плохое качество обучающей выборки может свести на нет работу даже лучших ML-алгоритмов.

Эта глава послужит руководством по сбору и компиляции тренировочных данных, предназначенных для обучения с учителем (илл. 2.1). Вы познакомитесь с общими принципами подготовки таких данных и получите информацию о часто встречающихся подводных камнях. Искусство машинного обучения по большей части сводится к изучению и визуализации обучающей выборки с целью оценки ее качества и руководства процессом обучения, поэтому мы подготовили обзор наиболее полезных техник визуализации данных. В конце мы обсудим, как подготовить обучающую выборку к процессу построения модели, о котором пойдет речь в главе 3.



Илл. 2.1. Базовый процесс машинного обучения. Так как в этой главе пойдет речь о данных, мы выделили фрагменты, соответствующие данным за прошедшие периоды и новым данным

В этой главе мы рассмотрим реальную задачу — *прогноз оттока клиентов* (churn prediction). В бизнесе термин *отток* (churn) означает акт отказа клиента от платной услуги. И крайне важно получить прогноз, сообщающий, какие клиенты, скорее всего, сделают это в ближайшем будущем. Это даст возможность вмешаться в процесс путем отправки сообщения или предложения скидки. Подобные вмешательства позволяют фирмам экономить миллионы долларов, так как на приобретение новых клиентов, как правило, приходится тратиться куда больше, чем на попытки удержать уже существующих. Именно поэтому такую ценность имеет прогноз с машинным обучением, на основании которого о намерении клиента уйти можно узнать за недели до фактического события.

Также в этой главе мы поработаем с данными, доступными в Интернете и широко используемыми в посвященных машинному обучению книгах и документации, — это списки пассажиров «Титаника» и сведения о расходе топлива.

2.1. Первый этап: сбор данных

Первым делом следует корректно сформулировать вопрос, который будет решаться с помощью машинного обучения. Большинство реальных задач сводится к предсказанию целевой переменной (или переменных). В этой книге в основном будут рассматриваться именно такие задачи для машинного обучения с учителем. Вот как могут выглядеть вопросы в рассматриваемой нами ситуации:

- кто из клиентов уйдет в этом месяце?
- среагирует ли конкретный пользователь на наши рекламные баннеры?
- сфальсифицирована ли конкретная учетная запись?
- какое настроение — негативное, позитивное или нейтральное — создает эта запись в микроблоге?
- каким будет спрос на продукцию в следующем месяце?

В этих вопросах можно заметить общие черты. Во-первых, все они требуют оценки одного или нескольких экземпляров. Роль экземпляров могут играть люди (вопрос об оттоке клиентов), события (вопрос о восприятии

записи в микроблоге) или даже временны́е отрезки (вопрос о спросе на продукцию).

Во-вторых, в каждом вопросе четко прописана желательная форма ответа. Иногда это бинарный ответ (уходит клиент или остается, сфальсифицирована учетная запись или нет), иногда принадлежит к набору классов (отрицательный, положительный или нейтральный) или даже к сотням и тысячам классов (выбор песни из огромной библиотеки), а иногда принимает численное значение (спрос на продукт). В статистике и теории вычислительных систем целевая переменная называется *откликом* (response), или *зависимой переменной* (dependent variable). Эти термины могут использоваться взаимозаменяемо.

В-третьих, для каждой из поставленных выше задач можно получить набор данных за прошедшие периоды с известной целевой переменной. Например, за недели или месяцы сбора данных вы узнали, кто из ваших подписчиков отписался, а кто переходит по вашим баннерам. Некоторые усилия позволят оценить и ощущение от различных записей в микроблоге. В дополнение к известным целевым значениям файлы с данными за прошедшие периоды будут содержать информацию для каждого экземпляра, который был доступен для изучения на момент создания прогноза. Это *входные признаки* (input features), также широко известные как *объясняющие переменные* (explanatory variables) или *независимые переменные* (independent variables). Например, для предсказания оттока пригодится история использования продукта каждым клиентом наряду с демографическими характеристиками и сведениями об учетной записи. Эти входные признаки вкупе с известными значениями целевой переменной составят *обучающую выборку* (training set).

Наконец, каждый из поставленных вопросов подразумевает некие *действия* в случае познаваемой целевой переменной. Например, знай вы заранее, что какой-то пользователь среагирует на вашу рекламу, вы рассчитывали бы на него и предоставили бы его вниманию баннер. А точные сведения о будущем спросе на продукт заставят организовать канал поставок для его удовлетворения. Во всех описываемых случаях ML-алгоритм должен по обучающей выборке определить, каким способом набор входных признаков позволяет наиболее точно предсказать целевую переменную. Результат этого «определения» кодируется в виде модели машинного обучения. Признаки новых образцов (с неизвестной целевой переменной) «скармливаются» ML-модели, которая генерирует для них прогнозы, давая конечному пользователю возможность предпринять

более точные (и быстрые) действия. Кроме того, ML-модель позволяет получить представление о том, как выглядит зависимость между входными признаками и целевой переменной.

Рассмотрим все вышесказанное в контексте предсказания оттока. Представьте, что вы сотрудник телекоммуникационной компании и хотите узнать, кто из текущих абонентов перестанет обслуживаться в следующем месяце. В данном случае экземпляром является абонент. Целевая переменная предполагает двоичный результат — перестанет абонент обслуживаться в этом месяце или нет. Входные признаки могут включать в себя информацию об абоненте, которую можно узнать на начало месяца, например срок существования учетной записи (Acct length), детали тарифного плана (Int'l plan), а также сведения о пользовании услугами, такие как общее количество звонков за прошлый месяц (Total calls) и сколько минут длились разговоры (Total mins). Иллюстрация 2.2 демонстрирует первые четыре ряда обучающей выборки для предсказания оттока.

Признаки | Целевая переменная

Cust. ID	State	Acct length	Area code	Int'l plan	Voicemail plan	Total messages	Total mins.	Total calls	Churned?
502	FL	124	561	No	Yes	28	251.4	104	False
1007	OR	48	503	No	No	0	190.4	92	False
1789	WI	63	608	No	Yes	34	152.2	119	False
2568	KY	58	606	No	No	0	247.2	116	True

Илл. 2.2. Обучающая выборка из четырех экземпляров для задачи на отток абонентов телекоммуникационной компании

В этом разделе мы хотим дать базовое преставление о сборе данных для машинного обучения. Сам процесс сбора может сильно варьироваться в зависимости от отрасли, но при накоплении сведений для обучающей выборки есть ряд типовых вопросов. Вот четыре основных вопроса, возникающие на этапе сбора данных, и ниже мы на практике рассмотрим общие принципы ответа на них:

- какие входные признаки следует включить?
- как получить известные значения целевой переменной?
- сколько обучающих данных требуется?
- как оценить качество обучающей выборки?

2.1.1. Определяем набор входных признаков

В задачах, связанных с машинным обучением, как правило, присутствуют десятки признаков, которые можно использовать для предсказания целевой переменной. В задаче с оттоком абонентов на эту роль подходят все виды связанных с клиентами атрибутов: демографические (возраст, пол, местоположение), тарифный план (статус, время до конца подписки, время с момента последнего обновления, привилегированный статус) и данные о пользовании услугой (история звонков, данные текстовых сообщений, трафик данных, история платежей). Существуют два реальных ограничения на использование сведений в качестве входного признака:

- значение признака должно быть известно на момент прогноза (скажем, в рассматриваемом примере с оттоком клиентов это начало месяца);
- признак должен быть численным или категориальным (в главе 5 вы узнаете, как преобразовать нечисловые данные в признаки).

Поток данных `Учет вызовов` можно превратить в набор числовых и/или категориальных признаков, подсчитав суммарную статистику, например, по всем использованным минутам, по соотношению минут, использованных днем и ночью, соотношению минут, использованных в будни и в выходные, а также по доле минут, использованных в Сети.

Какие именно признаки из столь обширного набора нам нужны? Простое эмпирическое правило гласит, что признак имеет смысл использовать, только если вы подозреваете, что он каким-то образом связан с целевой переменной. Поскольку целью машинного обучения с учителем является предсказание целевой переменной, признаки, очевидно не имеющие с ней ничего общего, следует исключить. Например, соответствующий каждому абоненту отличительный идентификационный номер не имеет смысла использовать в качестве входного признака для прогнозирования ухода. Такие бесполезные признаки мешают отличать истинную взаимосвязь (сигналы) от случайных помех в данных (шума). Чем больше неинформативных признаков, тем ниже соотношение сигнал–шум и, соответственно, менее точна (в среднем) ML-модель.

Но при этом точности модели может повредить исключение признака, о связи которого с целевой переменной изначально ничего не было известно. Ведь машинное обучение предназначено для обнаружения

в данных новых шаблонов и соотношений! Представьте, что из набора признаков исключили количество не открытых на текущий момент сообщений голосовой почты. Но человек мог перестать проверять голосовую почту именно потому, что собрался менять оператора. В данных этот сигнал проявится как слегка увеличившаяся условная вероятность ухода абонентов с большим количеством неоткрытых голосовых сообщений. Исключение этого входного признака лишает алгоритм важной информации и, следовательно, уменьшает точность прогнозов. Благодаря способности ML-алгоритмов обнаруживать тонкие, нелинейные связи признаки, выходящие за рамки известных эффектов первого порядка, могут существенно влиять на точность модели.

При отборе входных признаков приходится идти на компромисс. С одной стороны, добавляя в модель все, что приходит в голову, вы рискуете заглушить группу признаков, которая содержит хоть какой-то сигнал, подавляющим все шумом. От этого пострадает точность модели, так как она перестанет отличать реальные шаблоны от случайных флуктуаций. С другой стороны, тщательно выбирая только немногие признаки, гарантированно связанные с целевой переменной, можно оставить за бортом полезную информацию. И это снова негативно повлияет на точность ML-модели, так как о дополнительных признаках, которые могли помочь в прогнозировании целевых переменных, она просто не узнает.

Исходя из этого, практичнее всего поступать следующим образом:

1. Включите все признаки, которые кажутся хоть как-то связанными с целевой переменной. Выполните обучение ML-модели. Если точность прогноза вас устраивает, останавливайтесь.

2. В противном случае расширьте набор, добавив туда признаки, связь которых с целевой переменной менее очевидна. Снова выполните обучение модели и оцените ее точность. Если она вас устраивает, останавливайтесь.

3. Если точность все еще неудовлетворительна, запустите для расширенного набора признаков *алгоритм отбора* (feature selection algorithm), чтобы выбрать оптимальное, сильнее всего влияющее на процесс прогнозирования подмножество.

Алгоритмы отбора признаков мы рассмотрим в главе 5. Они ищут наиболее точную модель на основе подмножества из набора признаков,

сохраняя сигнал и отбрасывая шум. Они крайне ресурсоемки, но дают потрясающий прирост производительности модели.

В заключение важно заметить, что признак можно добавить в набор, даже если он присутствует не у каждого экземпляра. Например, если возраст клиента известен только в 75% случаев, он все равно может служить входным признаком. О способах обработки отсутствующих данных мы поговорим чуть ниже.

2.1.2. Наблюдаемое значение целевой переменной

Одна из самых больших трудностей на первом этапе машинного обучения с учителем — формирование обучающей выборки с известными значениями целевых переменных. Зачастую для этого требуется работа существующей, условно-оптимальной системы до тех пор, пока не будут собраны все данные. Например, при построении ML-решения, предсказывающего отток абонентов, первым делом нужно в течение нескольких недель или месяцев наблюдать за тем, как уходят одни абоненты и появляются другие. Получив достаточную для построения точной ML-модели обучающую выборку, можно задействовать собственно машинное обучение.

В каждом случае процесс получения и оценки наблюдаемых значений целевой переменной будет выглядеть по-разному. Вот как выглядит формирование обучающей выборки в следующих ситуациях:

- *Рассылка целевых рекламных объявлений.* В течение нескольких дней проводится кампания, чтобы определить, какие пользователи реагируют на рекламные баннеры, а какие нет, и какие будут конвертированы.
- *Распознавание мошенничества.* Тщательное исследование данных за прошлые периоды позволяет выявить недобросовестных пользователей.
- *Прогнозирование спроса.* Журналы с информацией об управлении поставками позволят узнать спрос за последние месяцы или годы.
- *Настроение записей в микроблоге.* Намного сложнее узнать, какое именно настроение на самом деле предполагалось. Можно вручную проанализировать набор записей, заставляя людей читать и высказывать мнение по поводу прочитанного.

Сбор экземпляров известных целевых переменных зачастую — тяжелый и кропотливый труд, но выгоды от перехода к ML-решению, скорее всего, многократно компенсируют временные и денежные затраты. Существуют и другие способы получения достоверных значений целевой переменной:

- нанять аналитика, который будет вручную просматривать текущие данные и данные за прошлые периоды с целью определения или оценки значений целевой переменной;
- привлечь к оценке значений целевой переменной широкий круг лиц (crowdsourcing), то есть задействовать «коллективный разум»;
- по результатам проведенных мероприятий опрашивать клиентов и проводить с ними другие практические эксперименты;
- запускать управляемые эксперименты (например, A/B-тестирование) и следить за ответами.

Все эти стратегии являются трудоемкими, но процесс обучения можно ускорить, а время на сбор обучающей выборки сократить, если рассматривать целевые переменные только для экземпляров, оказывающих максимальное влияние на ML-модель. В качестве примера можно вспомнить метод *активного обучения* (active learning). При наличии существующей (маленькой) обучающей выборки и (большого) набора данных с неизвестной переменной отклика активное обучение идентифицирует в этом наборе подмножество экземпляров, включение которых в обучающую выборку позволит получить максимально точную ML-модель. В этом смысле активное обучение ускоряет построение ML-модели, концентрируя собираемые вручную ресурсы. Более подробно об активном обучении и связанных с ним методах можно узнать в докладе Дасгупты и Лэнгфорда на ICML в 2009 году[1].

2.1.3. Достаточный объем обучающих данных

С учетом того, как трудно наблюдать и собирать значения переменной отклика для экземпляров, возникает вопрос: какой размер обучающей

[1] См.: *http://videolectures.net/icml09_dasgupta_langford_actl/*

выборки достаточен для построения и запуска ML-модели? К сожалению, ответ настолько зависит от конкретной задачи, что универсальных рекомендаций попросту не существует. Нет и общих правил.

Вот факторы, от которых зависит количество необходимых данных:

- *Сложность задачи*. Можно ли описать связь между входными признаками и целевой переменной простым шаблоном или же она запутанна и не имеет линейной зависимости?
- *Требования к точности*. Если достаточно всего 60% успешных результатов, вы обойдетесь меньшей обучающей выборкой, чем в случае, когда необходимо получить 95% успешных результатов.
- *Размерность пространства признаков*. Если доступны всего два входных признака, обучающих данных потребуется меньше, чем при наличии 2000 таких признаков.

Нужно запомнить следующий руководящий принцип: по мере увеличения обучающей выборки модели (в среднем) становятся более точными. (Предполагается, что мы получаем образцы продолжающегося процесса генерации данных, как будет подробно рассмотрено в следующем разделе.) Большая обучающая выборка обеспечивает более высокую точность из-за того, что ML-модели базируются именно на данных. Взаимосвязь между признаками и целевой переменной выводится целиком из обучающей выборки, соответственно чем больше размер этой выборки, тем выше способность модели распознавать и фиксировать менее выраженные шаблоны и соотношения.

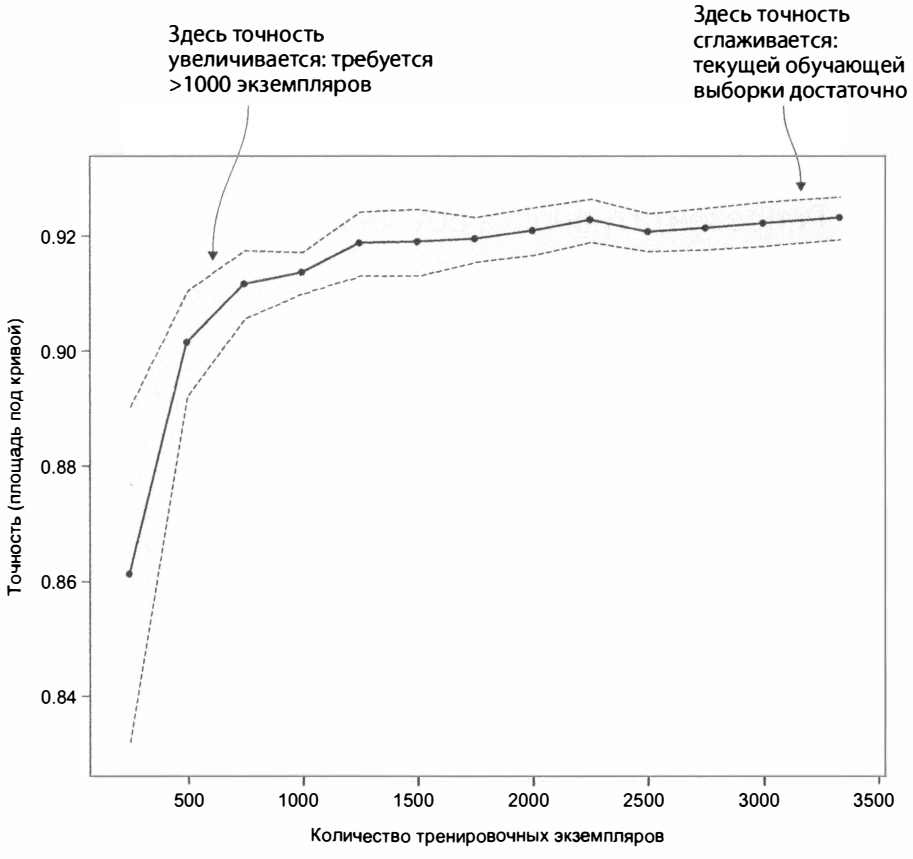
На примере приведенных выше данных телекоммуникационной компании можно показать, как совершенствуется ML-модель по мере роста обучающей выборки, а заодно предложить стратегию, позволяющую определить, требуется ли и дальше увеличивать эту выборку. Тренировочный набор данных телекоммуникационной компании состоит из 3333 экземпляров, каждый из которых содержит 19 признаков, а также двоичный результат в виде ушедших или продливших обслуживание абонентов. В данном случае оценить необходимость дополнительных данных очень просто:

1. В имеющихся тренировочных данных выберем подвыборки разных размеров для экспериментов. Например, если набор состоит из 3333 экземпляров, сетка размеров может выглядеть так: 500; 1000; 1500; 2000; 2500; 3000.

2. Для каждой подвыборки случайным образом нарисуем указанное количество экземпляров (без замещения) из обучающей выборки.

3. Для каждого подмножества тренировочных данных построим ML-модель и оценим ее точность (с единицами оценки точности вы познакомитесь в главе 4).

4. Рассмотрим изменение точности как функцию от размера выборки. Если при больших размерах эта функция выравнивается, существующего набора тренировочных данных, скорее всего, достаточно. Но если точность продолжает расти, скорее всего, не помешает увеличить количество тренировочных данных.



Илл. 2.3. Проверка того, хватит ли обучающей выборки из 3333 экземпляров для построения точной ML-модели оттока абонентов из телекоммуникационной компании. Черная линия показывает среднюю точность за 10 повторений оценочной процедуры, а затененные полосы отмечают области погрешности

При этом если нужный уровень точности явно указан, описанная стратегия позволит оценить, достигнут ли он в модели, построенной на основе существующей обучающей выборки (в случае положительного ответа на этот вопрос дальше накапливать данные не нужно).

На илл. 2.3 на примере набора данных из телекоммуникационной компании точность обученной ML-модели показана как функция от количества тренировочных экземпляров. Очевидно, что ML-модель совершенствуется по мере роста обучающей выборки: переход от 250 к 500, а затем и к 750 тренировочным экземплярам приводит к значительному повышению уровня точности. Но когда количество тренировочных экземпляров превышает 2000, кривая точности выравнивается. Это означает, что точность прогнозирования при дальнейшем увеличении выборки существенно меняться не будет. (Но это не означает, что невозможны значительные усовершенствования за счет увеличения количества признаков.)

2.1.4. Репрезентативность обучающей выборки

Кроме размера обучающей выборки фактором, влияющим на точность прогнозов ML-модели, является *репрезентативность* этой выборки. Насколько входящие в нее экземпляры похожи на те, которые будут накапливаться в будущем? Так как целью машинного обучения с учителем является генерация точных предсказаний для новых данных, крайне важно, чтобы обучающая выборка включала в себя экземпляры тех типов, для которых будет создаваться прогноз. Выборка с экземплярами, не имеющими отношения к будущим данным, называется *систематической ошибкой отбора* (sample-selection bias), или *ковариационным сдвигом* (covariate shift).

Отсутствие репрезентативности в обучающей выборке может быть обусловлено рядом факторов:

- Получить экспериментальные значения целевой переменной удалось только для определенного, содержащего ошибку подмножества данных. Скажем, если случаи мошенничества за прошедшие периоды фиксировались только при убытках более $1000, обученная на таких данных модель будет иметь трудности с распознаванием мошенничества, потери от которого составляют менее $1000.

- Свойства экземпляров со временем менялись. Скажем, если обучающий пример состоит из данных о мошенничестве в сфере медицинского страхования, а новые законы значительно изменили условия работы страховых компаний, предсказания, основанные на новых данных, могут оказаться некорректными.
- Набор входных признаков меняется со временем. Например, изменился набор атрибутов местоположения, сформированный для каждого клиента. Изначально в него входили индекс и название штата, а теперь вы собираете IP-адреса. Такая ситуация может потребовать модификации набора признаков, используемых для обучения модели, и, возможно, удаления старых данных из обучающей выборки.

В каждом из вышеуказанных случаев ML-модель, хорошо работающая с обучающей выборкой, не сможет хорошо выполнить экстраполяцию для новых данных. Другими словами, если модель обучали на данных о яблоках, вряд ли она сможет точно предсказывать урожаи апельсинов.

Чтобы избежать подобных проблем, обучающая выборка должна максимально репрезентативно представлять будущие данные. Значит, требуется процесс структурирования набора тренировочных данных, убирающий систематические ошибки. В следующем разделе мы поговорим о том, как визуализация помогает отслеживать репрезентативность данных.

Теперь, когда вы получили представление о формировании обучающей выборки, требуется структурировать и скомпоновать данные, подготовив их к построению ML-модели. В следующем разделе рассказывается о предварительной обработке тренировочных данных.

2.2. Подготовка данных к моделированию

Первый этап подготовки данных к моделированию — это их сбор, но иногда таких этапов может быть и несколько, в зависимости от структуры набора данных. Многие алгоритмы машинного обучения работают только с численными данными — целыми и вещественными. Самые простые наборы ML-данных имеют именно такой формат, но зачастую в набор входят и признаки других типов, например категориальные переменные. Кроме того, часть значений может попросту отсутствовать. Иногда при-

знаки приходится проектировать или рассчитывать. Бывают численные признаки, требующие нормализации, чтобы их можно было сравнить друг с другом или привести в соответствие с гистограммой (к таким случаям относится, например, сортировка по кривой нормального распределения). Мы рассмотрим типовые этапы предварительной обработки данных при решении реальных задач с машинным обучением.

2.2.1. Категориальные признаки

Вторым наиболее распространенным типом признаков является категориальный. Признак считается таковым, если его значения можно отнести к какой-то группе, при этом нам не важен их порядок. Иногда принадлежность к этому типу определяется достаточно легко (например, когда параметры принимают всего несколько строковых значений, скажем, «спам» и «обычная почта»). Но бывает и так, что разница между численным (целым) и категориальным признаком не очевидна. Любой из типов может оказаться правильным представлением, и при этом выбор влияет на производительность модели. Скажем, день недели допустимо представить как в виде числа (пронумеровав дни, начиная с воскресенья), так и в виде категории (взяв за основу названия). Построение моделей и оценка их производительности будут рассматриваться в главах 3 и 4, а пока мы просто познакомим вас с методикой обработки категориальных признаков. Иллюстрация 2.4 выделяет такие признаки в нескольких наборах данных.

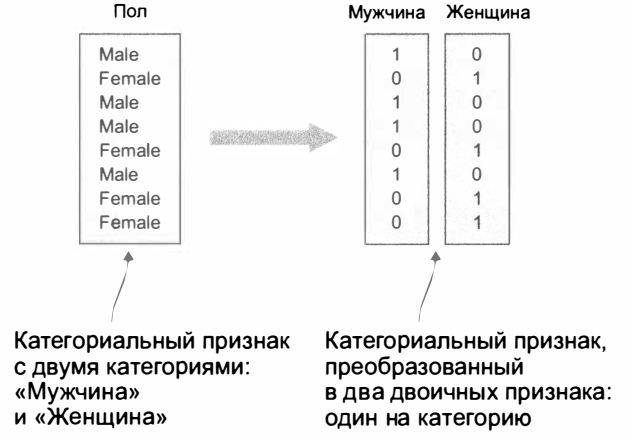
Некоторые алгоритмы машинного обучения используют категориальные признаки в исходном виде, но обычно данные требуется представить в численной форме. Можно присвоить каждой категории номер, но такие зашифрованные данные нельзя использовать в качестве истинных категориальных признаков, так как при этом добавляется случайным образом выбранный вами порядок категорий. А ведь одним из свойств категориальных признаков является их неупорядоченность. Куда лучше преобразовать каждую категорию в отдельный двоичный признак, имеющий значение 1 для экземпляров, попадающих в категорию, и 0 для не попадающих. В результате каждый категориальный признак преобразуется в набор двоичных признаков, по одному на категорию. Сконструированные таким способом признаки иногда называют *вспомогательными переменными* (dummy variables). Эту концепцию демонстрирует илл. 2.5.

Person	Name	Age	Income	Marital status
1	Jane Doe	24	81,200	Single
2	John Smith	41	121,000	Married

Категориальные признаки

PassengerId	Survived	Pclass	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Male	22	1	0	A/5 21171	7.25		S
2	1	1	Female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Female	26	0	0	STON/02. 3101282	7.925		S
4	1	1	Female	35	1	0	113803	53.1	C123	S
5	0	3	Male	35	0	0	373450	8.05		S
6	0	3	Male		0	0	330877	8.4583		Q

Илл. 2.4. Определение категориальных признаков. Вверху простой набор данных о заявителях, в котором категориальным является столбец со сведениями о семейном положении. Внизу набор сведений о пассажирах «Титаника». К категориальным здесь относятся столбцы с информацией о том, остался ли пассажир в живых, каким классом он путешествовал, женщина он или мужчина и в каком городе пассажир сел на «Титаник»



Илл. 2.5. Преобразование категориальных столбцов в численные

Ниже представлен псевдокод, иллюстрирующий преобразование категориальных признаков с илл. 2.5 в двоичные. Обратите внимание, что переменная `categories` принадлежит к особому типу NumPy (из библиотеки *www.numpy.org*), вследствие чего (`data == cat`) получает список булевских значений.

Листинг 2.1. Преобразование категориальных признаков в численные двоичные признаки

```
def cat_to_num(data):
    categories = unique(data)
    features = []
    for cat in categories:
        binary = (data == cat)
        features.append(binary.astype("int"))
    return features
```

ПРИМЕЧАНИЕ

Знакомые с языком программирования Python читатели, скорее всего, заметили, что листинг содержит не псевдокод, а полноценный код на языке Python. Вы не раз будете сталкиваться с подобной ситуацией по мере чтения этой книги: мы объявляем фрагмент кода псевдокодом, но если в явном виде не отмечено обратное, это работающий код. Для его упрощения мы импортировали несколько вспомогательных библиотек, например `numpy` и `scipy`. В общем случае приведенные примеры будут работать, если добавить к ним строки `from numpy import *` и `from scipy import *`. Это удобный подход для интерактивной проверки наших примеров, но в реальных приложениях применять его ни в каком случае не следует, так как оператор `import *` может привести к конфликту имен и непредвиденным результатам. Все примеры кода из книги доступны для просмотра и непосредственного исполнения в репозитории GitHub (https://github.com/brinkar/real-world-machine-learning).

Описанная техника преобразования категориальных признаков в численные работает для большинства ML-алгоритмов. Но есть и алгоритмы (например, с деревьями принятия решений, такие как «случайный лес»), которые используют категориальные признаки в исходном виде. Для исключительно категориальных наборов данных они часто дают очень хорошие результаты, но подробно мы будем разбирать их в следующей главе. Наш же простой набор данных о заявителях после преобразования показан на илл. 2.6.

Person	Name	Age	Income	Marital status: Single	Marital status: Married
1	Jane Doe	24	81,200	1	0
2	John Smith	41	121,000	0	1

Илл. 2.6. Простой набор данных после преобразования категориального признака «Семейное положение» в двоичные численные признаки (в исходном виде этот набор фигурирует на илл. 2.4.)

2.2.2. Отсутствующие данные

Вы уже видели несколько наборов с отсутствующими данными. В табличном представлении такие данные принимают вид пустых ячеек или ячеек со значением NaN (Not a Number), N/A или None. Как правило, это артефакты процесса сбора, возникающие, когда по какой-то причине конкретное значение для экземпляра данных измерить невозможно. Пример отсутствующих данных в сведениях о пассажирах «Титаника» показан на илл. 2.7.

PassengerId	Survived	Pclass	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Male	22	1	0	A/5 21171	7.25		S
2	1	1	Female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Female	26	0	0	STON/02. 3101282	7.925		S
4	1	1	Female	35	1	0	113803	53.1	C123	S
5	0	3	Male	35	0	0	373450	8.05		S
6	0	3	Male		0	0	330877	8.4583		Q

Отсутствующие значения

Илл. 2.7. Набор сведений о пассажирах «Титаника» имеет отсутствующие значения в столбцах «Возраст» и «Каюта». Информация о пассажирах извлекалась из доступных исторических источников, поэтому в данных случаях интересующие сведения просто не были обнаружены

Обработка двух основных типов отсутствующих данных происходит по-разному. Во-первых, иногда *сам факт отсутствия* данных может быть информацией, полезной для ML-алгоритма. Разумеется, если измерение значений было попросту невозможным, этот факт не несет никакого глубинного смысла. Например, в сведениях о пассажирах «Титаника» пропущенный номер каюты может означать принадлежность к более низкому социальному или экономическому классу, в то время как отсутствие данных в столбце «Возраст» не несет никакой полезной информации (просто не были обнаружены данные о возрасте конкретного пассажира в то время).

Рассмотрим сначала *информативные* отсутствующие данные. Если нам кажется, что в данных не хватает какой-то информации, значит, мы хотим, чтобы ML-алгоритм смог ею воспользоваться и теоретически увеличить точность предсказаний. Для этого отсутствующее значение

нужно привести к формату остальных значений в столбце. Для числовых столбцов это осуществляется путем присваивания значения –1 или –999, в зависимости от того, какие типичные значения имеют ненулевые параметры. Выберите для обозначения отсутствующих значений число с одного из концов числового спектра и не забывайте, что для численных столбцов важен порядок следования. При этом нельзя выбирать вариант из центральной части диапазона значений.

Для категориальных столбцов с потенциально информативными отсутствующими данными можно создать новую категорию, назвав ее Missing, None или другим подобным образом, после чего новый категориальный признак обрабатывается обычным способом (например, с помощью описанной в предыдущем разделе техники). Схематично работа с отсутствующими значимыми данными представлена на илл. 2.8.

Илл. 2.8. Действия при отсутствии значимых данных

Если же отсутствие значений отдельных экземпляров не несет информационной нагрузки, порядок действий будет другим. Добавлять специальное число или категорию уже нельзя, так как есть риск, что значение окажется некорректным. Скажем, вставив в пустые ячейки столбца «Возраст» значение –1, вы, скорее всего, навредите модели, с непонятной целью внеся путаницу в диапазон возрастов. Некоторые ML-алгоритмы просто игнорируют такие пропуски. Если же алгоритм этого не умеет, данные следует предварительно обработать, вообще убрав отсутствующие значения или заменив их неким оценочным значением. Такой подход к обработке отсутствующих данных называется *заполнением пропусков* (imputation).

В случае большого набора данных с небольшим количеством отсутствующих значений самое простое — отбросить их. Если же значения

отсутствуют в изрядной части наблюдений, потеря данных уменьшит прогностическую силу вашей модели. Более того, если распределение наблюдений с отсутствующими значениями по набору данных не является случайным, отбрасывание может внести неожиданную систематическую ошибку.

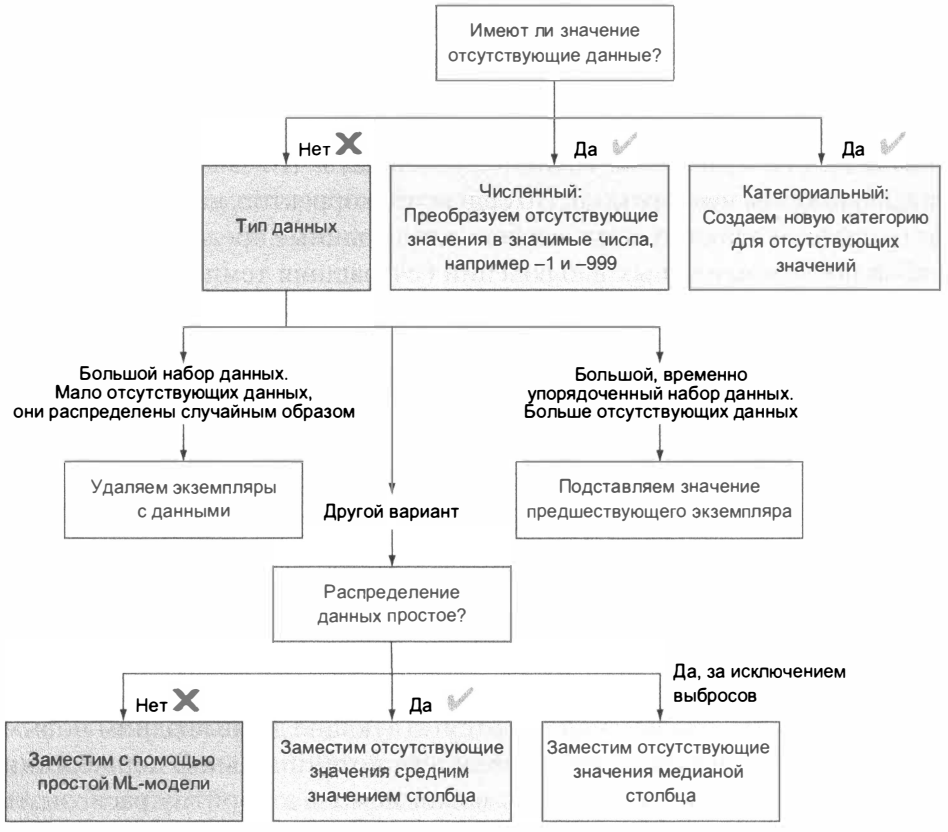
Есть и другой простой подход. Можно предположить, что экземпляры данных обладают некой временной упорядоченностью, и вставить на места пропусков значения из предшествующей строки этого же столбца. Нам неоткуда получить другую информацию, поэтому мы просто предполагаем, что при переходе от одного экземпляра к другому результат измерений не изменился. Излишне упоминать, что зачастую подобное предположение некорректно. Но еще менее корректно, например, заполнять пустые ячейки нулями, особенно если данные представляют собой набор последовательных наблюдений (вчерашняя температура вполне может применяться для оценки сегодняшней). Кроме того, с огромными наборами данных далеко не всегда получается использовать более сложные методы, и тогда на помощь придут эти простые приемы.

Когда это возможно, лучше использовать большие фрагменты существующих данных для приблизительной оценки отсутствующих значений. Их можно заменить средним значением или медианой столбца. При отсутствии другой информации предполагается, что среднее значение ближе всего к истине. Но в зависимости от распределения значений в столбце иногда лучше использовать медиану; среднее значение чувствительно к выбросам. Перечисленные техники широко применяются в современном машинном обучении и прекрасно работают во многих случаях. Но, заменяя все отсутствующие данные одним новым значением, вы снижаете наблюдаемость потенциальной корреляции с другими переменными, которая может помочь алгоритму распознать имеющиеся закономерности.

По возможности мы хотим, используя все доступные данные, предсказать значение отсутствующей переменной. Вам это ничего не напоминает? Это же основной принцип работы машинного обучения, то есть, по сути, мы думаем о построении ML-моделей для того, чтобы получить возможность строить ML-модели. На практике для заполнения пробелов обычно применяется простой алгоритм (например, линейная или логистическая регрессия, о которой вы подробнее узнаете в главе 3). Он вовсе не обязан совпадать с основным ML-алгоритмом. В любом случае, создается кон-

вейер ML-алгоритмов, дающий дополнительные способы оптимизации окончательной модели.

Но важно понимать, что универсального способа восстановления отсутствующих данных не существует. Перечисленные в этом разделе техники схематично представлены на илл. 2.9.



Илл. 2.9. Полная диаграмма решений для обработки отсутствующих значений при подготовке данных к ML-моделированию

2.2.3. Основы проектирования признаков

Предметно-ориентированные и усовершенствованные техники проектирования признаков будут рассматриваться в главе 5, а пока изучим базовые принципы предварительной подготовки данных, позволяющей улучшить модель.

Снова воспользуемся сведениями о пассажирах «Титаника». Иллюстрация 2.10 демонстрирует еще один вариант этих данных, и на этот раз нас интересует такой признак, как `Cabin` (каюта). Без обработки он вряд ли пригоден к употреблению. По всей видимости, некоторые значения включают в себя несколько кают, но даже всего одна каюта вряд ли подойдет в качестве хорошего категориального признака, так как для каждой каюты придется завести отдельную категорию. Невозможно провести корреляцию, к примеру, между фактом спасения пассажира и тем, что он занимал именно свою каюту, а не соседнюю.

PassengerId	Survived	Pclass	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Male	22	1	0	A/5 21171	7.25		S
2	1	1	Female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Female	26	0	0	STON/02. 3101282	7.925		S
4	1	1	Female	35	1	0	113803	53.1	C123	S
5	0	3	Male	35	0	0	373450	8.05		S
6	0	3	Male		0	0	330877	8.4583		Q

Илл. 2.10. В сведениях о пассажирах «Титаника» некоторые ячейки столбца «Cabin» содержат несколько значений, в то время как другие пусты. И сами по себе номера кают не являются хорошим категориальным признаком

Проживание же в определенной части корабля может оказаться важным фактором выживания. Из идентификатора каюты можно извлечь букву, как категориальный признак и номер, как числовой признак, предположив, что это соответствует различным частям корабля. Можно найти план «Титаника» и определить, в каком классе и с какой стороны располагалась каюта, имела она иллюминаторы или нет и проч. Набор кают этим способом обработать нельзя, но так как с большой вероятностью все каюты, занятые одним пассажиром, располагаются рядом, достаточно взять для анализа ID первой из них. Еще можно включить номера кают в новый признак, который также будет релевантным.

В итоге мы создали из признака `Cabin` три новых признака. Следующий листинг демонстрирует код для их извлечения.

Листинг 2.2. Простое извлечение признаков из данных о каютах на «Титанике»

```
def cabin_features(data):
    features = []
    for cabin in data:
        cabins = cabin.split(" ")
```

```
        n_cabins = len(cabins)
        # Первая литера из обозначения каюты cabin_char
        try:
            cabin_char = cabins[0][0]
        except IndexError:
            cabin_char = "X"
            n_cabins = 0
        # Остальные символы – номер каюты
        try:
            cabin_num = int(cabins[0][1:])
        except:
            cabin_num = -1
        # Добавляем 3 признака для каждого пассажира
        features.append( [cabin_char, cabin_num, n_cabins] )
    return features
```

Надеемся, вы уже поняли, что подразумевается под проектированием признаков. Это использование существующих признаков для создания новых, увеличивающих ценность исходных данных, путем применения наших знаний о данных или рассматриваемой предметной области. Как уже упоминалось, позднее вы познакомитесь с более сложными концепциями проектирования признаков и распространенными типами данных, которые без обработки не могут использоваться большинством алгоритмов. Сюда входят и признаки в виде текста свободного формата, например с веб-страниц или микроблогов. Другие важные признаки извлекаются из изображений, видео и временных рядов данных.

2.2.4. Нормализация данных

Некоторые ML-алгоритмы требуют, чтобы данные были *нормализованными*. Это означает, что каждый признак обрабатывается с целью его подгонки под единую числовую шкалу. Диапазон значений признака может влиять на его важность относительно прочих признаков. Если значения одного признака варьируются от 0 до 10, а второго — от 0 до 1, вес первого признака по отношению ко второму составит 10. Иногда вес конкретного признака регулируется вручную, но, как правило, лучше оставить определение относительных весов на откуп ML-алгоритму. Но чтобы гарантировать, что все признаки учитываются одинаково, данные требуется нормализовать. Зачастую нормализация сводится к переводу в диапазон от 0 до 1 или от –1 до 1.

Посмотрим, как реализуется данный процесс. Он показан в представленном ниже листинге. Мы хотим, чтобы для каждого признака значения данных распределялись от минимального (обычно –1) до максимального (обычно +1). Чтобы получить такой результат, разделим каждое значение на весь диапазон значений. В результате все переменные окажутся в диапазоне 0–1. Отталкиваясь от этой процедуры, их можно расширить до требуемого диапазона (2 в случае –1 и +1) путем умножения на преобразованное значение. В конце начальная точка смещается от 0 к минимальному необходимому значению (например, –1).

Листинг 2.3. Нормализация признака

```
def normalize_feature(data, f_min=-1.0, f_max=1.0):
    d_min, d_max = min(data), max(data)
    factor = (f_max - f_min) / (d_max - d_min)
    normalized = f_min + (data - d_min)*factor
    return normalized, factor
```

Обратите внимание, что возвращается не только нормализованное значение, но и коэффициент, с помощью которого выполнялась нормализация. Это делается потому, что для получения значимых результатов любые новые данные (например, для предсказаний) потребуют нормализации этим же способом. Также это означает, что человек, выполняющий ML-моделирование, должен запоминать способы нормализации конкретных признаков и сохранять соответствующие значения (коэффициент и минимальное значение).

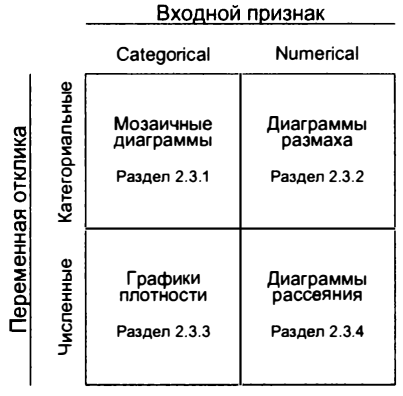
Реализацию функции, которая берет новые данные, коэффициент нормализации и нормализованное минимальное значение, а затем снова проводит нормализацию, мы оставляем вам в качестве упражнения.

По мере расширения доступного инструментария и исследования различных данных вы убедитесь, что каждый набор обладает качествами, которые делают его по-своему интересным и зачастую нетривиальным. Но большие наборы данных с множеством переменных сложно полностью понять по виду их табличного представления. Здесь вам на помощь придут графические инструменты визуализации, жизненно необходимые для получения представления о данных, из которых вы надеетесь извлечь скрытую информацию.

2.3. Визуализация данных

Между сбором/предварительной обработкой данных и построением ML-модели находится важный этап визуализации. Он предназначен для проверки работоспособности обучающих признаков и целевой переменной. Простые техники визуализации позволяют увидеть, как входные признаки связаны с целевой переменной. Эта информация служит ориентиром при построении модели и помогает лучше понять модель и генерируемые ею прогнозы. Более того, визуализация зачастую дает представление о степени репрезентативности обучающей выборки и о том, экземпляров каких типов там, возможно, не хватает.

Мы сосредоточимся на методах визуализации зависимости целевой переменной от входных признаков. Рассмотрим четыре техники визуализации: мозаичные диаграммы, диаграммы размаха, графики плотности и диаграммы рассеяния. Каждая техника предназначена для конкретной комбинации типов (численных или категориальных) входных признаков и целевой переменной, как показано на илл. 2.11.



Илл. 2.11. Четыре техники визуализации, систематизированные по типам входных признаков и переменной отклика, которые нужно представить в графическом виде

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

Статистической визуализации и отображению данных посвящено множество книг. Если вы хотите более глубоко ознакомиться с этой темой, рекомендуем следующие издания:

- Классический учебник Edward Tufte, «*The Visual Display of Quantitative Information*» (Graphics Press, 2001) детально рассматривает процесс визуализации данных для анализа и презентаций. На русском языке он доступен на странице: http://envisioninginformation.daiquiri.ru/
- Тем, кто программирует на языке R, рекомендуем книгу Winston Chang, «*R Graphics Cookbook*» (O'Reilly, 2013), рассказывающую на примерах о процессе визуализации данных на языке R.
- Пользователям Python рекомендуем книгу Igor Milovanović, Dimitry Foures, and Giuseppe Vettigli, «*Python Data Visualization Cookbook*» (Packt Publishing, 2015), которая содержит основные сведения, позволяющие приступить к работе с библиотекой Matplotlib.

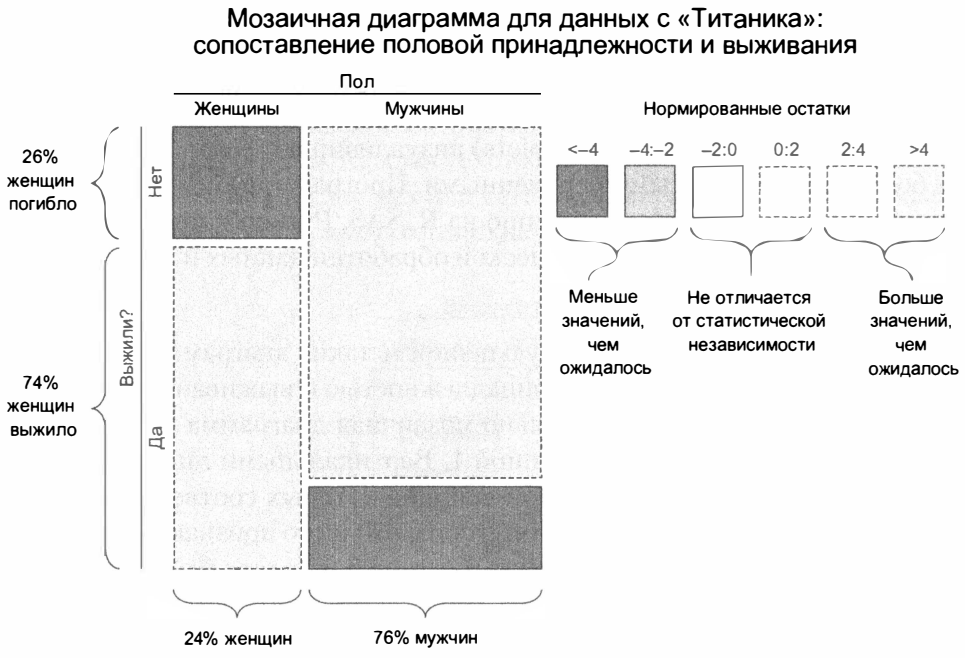
2.3.1. Мозаичные диаграммы

Мозаичные диаграммы (mosaic plots) визуализируют связи между двумя и более категориальными переменными. Программное обеспечение для создания таких диаграмм доступно на R, SAS, Python и прочих языках, предназначенных для статистической обработки данных и работы с графикой.

Продемонстрируем практическую ценность таких диаграмм, рассмотрев соотношение между половой принадлежностью и выживанием во время крушения «Титаника». Изначально мозаичная диаграмма представляет собой квадрат со сторонами длиной 1. Вертикальными линиями он делится на набор прямоугольников, ширина которых соответствует доле данных, попадающих в каждую категорию входного признака. Например, в данных о пассажирах «Титаника» женщины составляли 24%, поэтому разделим наш единичный квадрат вдоль оси x на два прямоугольника, соответствующих 24% / 76% ширины всей области.

Затем каждый вертикальный прямоугольник делится горизонтальными линиями на дополнительные прямоугольники, относительные площади которых пропорциональны проценту экземпляров в каждой категории переменной отклика. Например, из путешествовавших на «Титанике» женщин выжило 74% (это *условная вероятность* выживания в случае, если пассажир является женщиной). Следовательно, прямоугольник «Женщины» делится горизонтальной линией на две части с площадями 74% / 26%. Та же операция проделывается с прямоугольником «Мужчины» (для мужчин распределение составляет 19% / 81%).

Результат дает быструю визуализацию соотношения между половой принадлежностью и выживанием. При отсутствии связи горизонтальные линии будут располагаться близко друг к другу по оси *y*. В случае сильной взаимосвязи они будут далеко отстоять друг от друга. Эффект усиливается тем, что прямоугольники окрашены для оценки статистической значимости соотношения. Независимость входного признака и переменной отклика сравнивается с большими отрицательными остатками («меньше значений, чем ожидалось»), помеченными светло-серым цветом, и большими положительными остатками («больше значений, чем ожидалось»), помеченными темно-серым цветом (илл. 2.12).

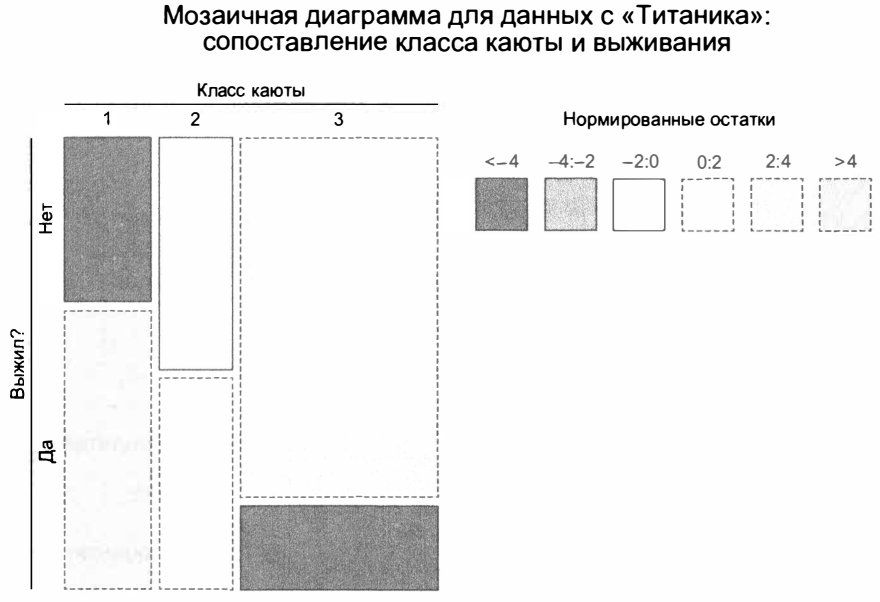


Илл. 2.12. Мозаичная диаграмма, демонстрирующая связь между полом пассажира и его шансами на спасение при крушении «Титаника». Визуализация показывает, что выжила большая доля женщин (и значительно меньшая доля мужчин), чем ожидалось бы при независимости шансов на выживание от половой принадлежности. Здесь действует принцип «сначала женщины и дети»

Диаграмма показывает, что при построении модели с машинным обучением, прогнозирующей шансы на выживание при крушении, половая принадлежность является важным фактором, который нужно учитывать. Кроме того, она позволяет проверить рассматриваемое соотношение с точки зрения здравого смысла — общеизвестно, что в катастрофах жен-

щины выживают чаще. Это дает дополнительную гарантию обоснованности ваших данных. Подобные визуализации помогают интерпретировать и обосновывать уже готовые модели с машинным обучением.

На илл. 2.13 представлена еще одна мозаичная диаграмма. На этот раз шансы на спасение рассматриваются в зависимости от класса занимаемой пассажиром каюты (первого, второго или третьего). Как и ожидалось, крушение пережила изрядная часть пассажиров первого класса (и меньшая часть пассажиров третьего класса). Очевидно, что класс каюты также является важным фактором в ML-модели, прогнозирующей шансы на спасение. Полученное соотношение полностью совпадает с нашими ожиданиями — чем выше класс каюты, тем больше для занимающего ее человека вероятность спастись.



Илл. 2.13. Мозаичная диаграмма, демонстрирующая соотношение между классом занимаемой пассажиром каюты и выживанием во время крушения «Титаника»

2.3.2. Диаграммы размаха

Диаграммы размаха (box plots), или «ящики с усами», — это стандартная статистическая техника визуализации распределения численных переменных. Для одной переменной такая диаграмма показывает квартили

ее распределения: минимум, 25-й процентиль, медиану, 75-й процентиль и максимум значения. Диаграмма размаха одной переменной дает представление о центре, разбросе и асимметрии в распределении значений, а также о наличии выбросов.

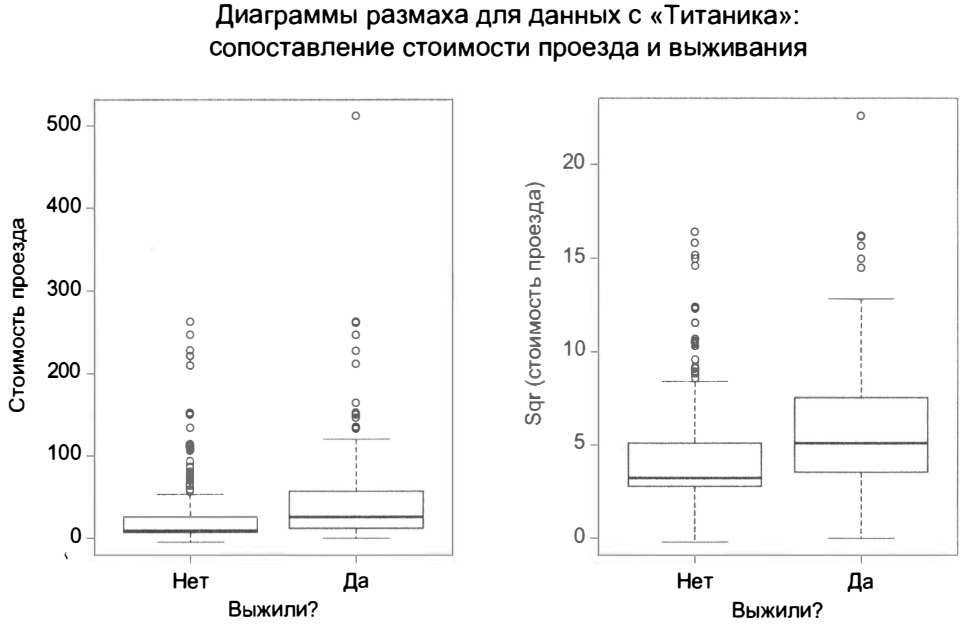
Кроме того, диаграммы размаха можно использовать для сравнения распределений параллельных графиков. В частности, они позволяют визуализировать разницу в распределении числового признака как функцию от различных категорий категориальной переменной отклика.

В случае с пассажирами «Титаника» можно визуализировать разницу в возрасте спасшихся и погибших с помощью параллельных диаграмм размаха, как показано на илл. 2.14. Мы видим, что особой разницы в распределении возрастов нет, ведь две диаграммы выглядят практически идентично по своей форме и местоположению.

Илл. 2.14. Диаграмма размаха показывает соотношение между возрастом пассажиров и выживанием во время крушения «Титаника». Распределения возрастов спасшихся и погибших не обнаруживают заметной разницы. (Но само по себе это не является причиной для исключения признака «Возраст» из ML-модели, так как он все равно может влиять на прогноз.)

Важно понимать ограниченность техник визуализации. Визуализация не заменяет ML-моделирования! Модели с машинным обучением могут обнаруживать и использовать тонкие, скрытые глубоко внутри данных взаимосвязи, которые не проявляются при простой визуализации. Не следует автоматически исключать из рассмотрения признаки только потому, что на диаграмме не видно их четкой связи с целевой переменной. На самом деле такая связь может обнаружиться при их рассмотрении в совокупности с другими входными признаками. К примеру, хотя однозначной связи возраста с выживанием не прослеживается, для пассажиров третьего класса этот фактор может оказаться важным (более молодые и сильные имеют больше шансов выбраться на палубу, чем пожилые). Хорошая ML-модель обнаружит и проявит такие взаимосвязи. Именно поэтому только на основании визуализации исключать возраст из числа признаков нельзя.

На илл. 2.15 показана диаграмма размаха, демонстрирующая соотношение между стоимостью проезда и спасением пассажиров. По левой



Илл. 2.15. Диаграммы размаха показывают связь между стоимостью проезда и выживанием на «Титанике». Преобразование путем извлечения квадратного корня четко проявляет закономерность: в среднем спасшиеся пассажиры платили за проезд больше

диаграмме понятно, что распределение цен на билеты имеет сильную асимметрию (много маленьких значений и несколько больших выбросов), что затрудняет визуализацию разницы. Тут нам придет на помощь простое преобразование — мы извлечем из цены квадратный корень (результат показан справа). После этого станет очевидной корреляция стоимости проезда с возможностью выживания — вполне логично, что пассажиры с более дорогими билетами имеют больше шансов на спасение. Соответственно, стоимость проезда следует добавить в модель, если вы хотите обнаружить и использовать эту положительную связь.

2.3.3. Графики плотности

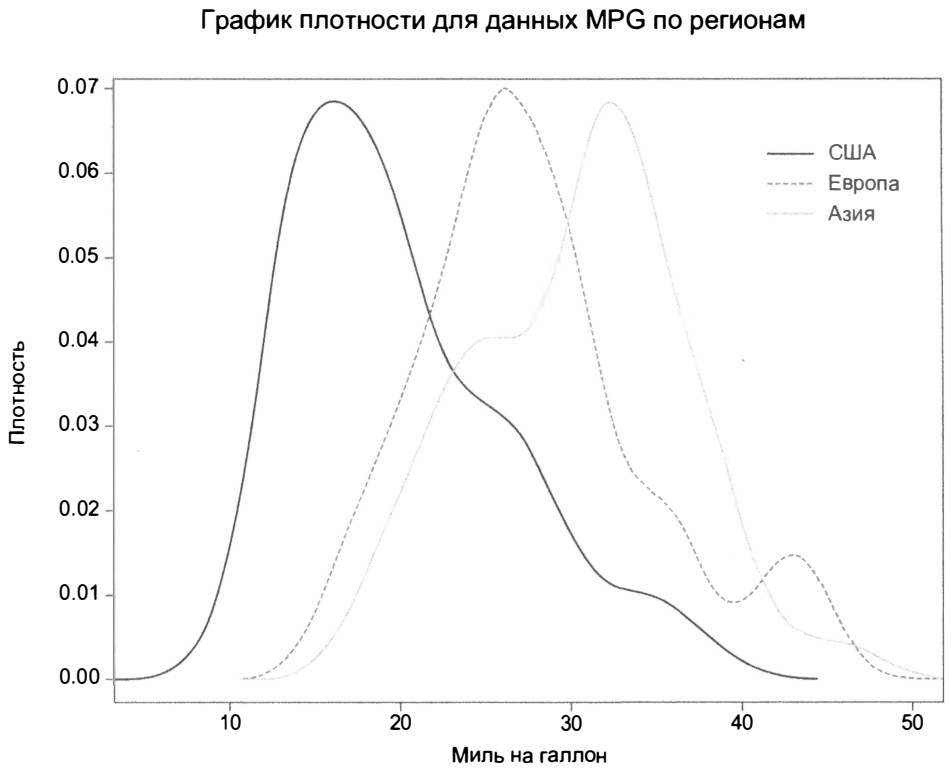
Теперь перейдем от категориальных переменных отклика к числовым. Соотношение между парой категориальных переменных можно увидеть на диаграмме размаха, как мы это делали в предыдущем разделе. Кроме того, к нашим услугам графики плотности.

График плотности (density plot) отображает распределение одной переменной куда детальнее, чем диаграмма размаха. Во-первых, рассчитывается сглаженная оценка распределения вероятности переменной (обычно для этого используется техника, называемая *ядерным сглаживанием* (kernel smoothing)). Во-вторых, распределение отображается в виде кривой, показывая значения, которые, скорее всего, будет иметь переменная. Создав график плотности для переменной отклика из каждой категории входного признака, вы сможете легко визуализировать любые расхождения значений для разных категорий. Графики плотности аналогичны гистограммам, но их сглаженная природа упрощает процесс визуализации набора распределений на одном рисунке.

В следующем примере мы рассмотрим расход топлива разных автомобилей[1]. Этот набор данных содержит информацию о том, сколько миль проезжают на одном галлоне топлива (MPG — miles per gallon) различные автомобили 1970–1982 годов выпуска, дополненную такими атрибутами, как лошадиные силы, вес, происхождение и год появления модели. Иллюстрация 2.16 демонстрирует графики плотности для параметра

[1] Этот набор данных доступен на странице *https://archive.ics.uci.edu/ml/datasets/Auto+MPG*. Он встроен в язык программирования R и открывается командой `data(mtcars)`.

MPG в зависимости от региона производства (США, Европа, Азия). Сразу же бросается в глаза, что азиатские машины, как правило, имеют более высокий показатель MPG, за ними следуют европейские и затем американские. Соответственно, местоположение в нашей модели будет важным прогностическим параметром. Кроме того, видны несколько вторичных «всплесков» плотности для каждой кривой, что может быть связано с различными типами автомобилей (например, грузовик в сравнении с седаном и в сравнении с гибридной моделью). Следовательно, оправданно дополнительное исследование этих вторичных всплесков с целью понять их природу и использовать как ориентир в дальнейшем проектировании признаков.

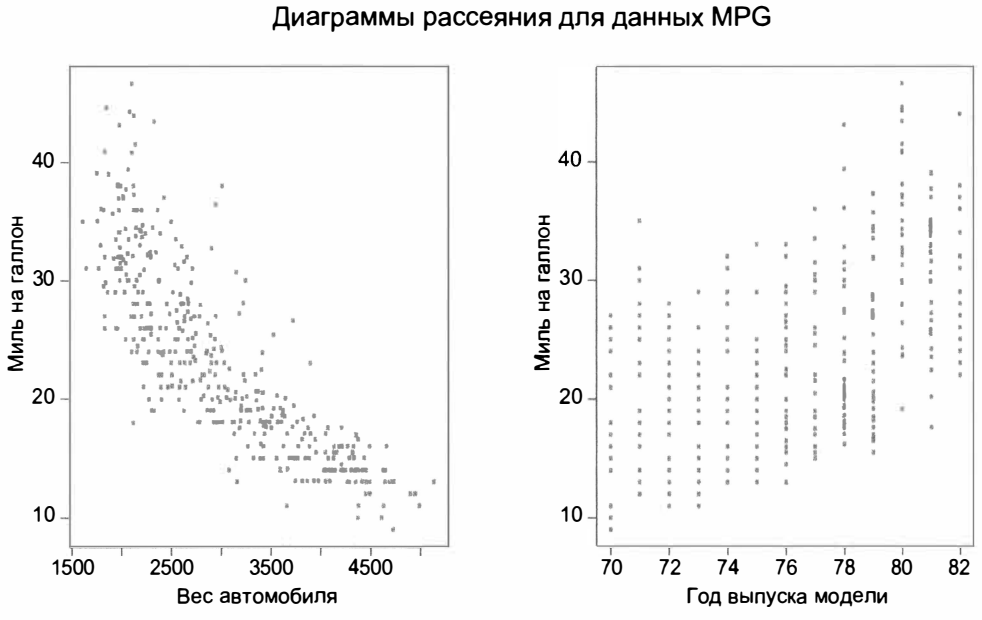


Илл. 2.16. График плотности для набора данных Auto MPG, показывающий распределение потребления топлива автомобилями для каждого региона производства. На графике видно, что азиатские автомобили, как правило, имеют самый высокий показатель MPG, а американские — самый низкий. То есть страна происхождения является устойчивым признаком для MPG

2.3.4. Диаграммы рассеяния

Диаграмма рассеяния (scatter plot) даст простую визуализацию соотношения между двумя численными переменными. Это один из самых популярных графических инструментов. Значение признака на такой диаграмме отмечается относительно значения переменной отклика, и каждый экземпляр представляется в виде точки. Несмотря на свою простоту, диаграммы рассеяния обнаруживают как линейные, так и нелинейные соотношения между входной переменной и переменной отклика.

Иллюстрация 2.17 демонстрирует две диаграммы рассеяния: MPG как функции от веса автомобиля и MPG как функции от года выпуска. В обоих случаях существует однозначная связь между входными признаками и параметром MPG, соответственно оба признака нужно задействовать при моделировании. На левой диаграмме данные имеют форму, отчетливо напоминающую банан, демонстрируя нелинейное уменьшение MPG по мере увеличения веса автомобиля. Правая же диаграмма показывает увеличивающееся линейное соотношение между MPG и годом выпуска



Илл. 2.17. Диаграммы рассеяния для соотношения расхода бензина в зависимости от веса автомобиля (слева) и года выпуска модели (справа)

модели. Оба графика однозначно дают понять, что указанные входные признаки имеет смысл использовать для предсказания MPG, и оба они демонстрируют ожидаемую зависимость.

2.4. Заключение

В этой главе мы рассмотрели важные аспекты данных в контексте прикладных задач машинного обучения.

- Этапы компиляции обучающей выборки:
  - решаем, какие входные признаки включить в работу;
  - находим способ получения непосредственных значений целевой переменной;
  - определяем, какое количество обучающих данных является достаточным;
  - убираем нерепрезентативные или имеющие систематическую ошибку обучающие данные.
- Этапы предварительной обработки обучающей выборки:
  - перекодируем категориальные признаки;
  - разбираемся с отсутствующими данными;
  - выполняем нормализацию признаков (для некоторых ML-алгоритмов);
  - выполняем проектирование признаков.
- Четыре полезные техники визуализации данных — мозаичные диаграммы, графики плотности, диаграммы размаха и диаграммы рассеяния:

		Входной признак	
		Категориальный	**Численный**
Переменная отклика	**Категориальная**	Мозаичные диаграммы	Диаграммы размаха
	Численная	Графики плотности	Диаграммы рассеяния

Теперь, когда мы подготовили свои данные к моделированию, приступим к созданию моделей!

2.5. Терминология

Термин	Определение
Вспомогательная переменная (dummy variable)	Двоичный признак, указывающий, принадлежит ли наблюдение к рассматриваемой категории
Наблюдаемое значение (ground truth)	Значение известной целевой переменной или метки для обучающего или тестового набора данных
Отсутствующие данные (missing data)	Признаки с неизвестными значениями для подмножества экземпляров
Заполнение пропусков (imputation)	Замещение отсутствующих данных численными или категориальными значениями

3 Моделирование и прогнозирование

В этой главе:

- ✓ выявление связей между данными посредством ML-моделирования;
- ✓ прогнозирование и выводы с помощью ML-моделей;
- ✓ модели для решения задач классификации;
- ✓ модели для решения задач регрессии.

Предыдущая глава дала вам представление о том, в каком порядке и по каким принципам осуществляется сбор, предварительная обработка и визуализация данных. Следующий этап рабочего процесса машинного обучения — применение данных для выявления соотношений между входными признаками и целевой переменной. В машинном обучении эта задача решается построением статистической модели на базе собранных данных. Эта глава содержит базовые сведения, позволяющие понять процесс ML-моделирования и построить собственные модели. Мы не собираемся, как в большинстве учебников по машинному обучению, много говорить о различных подходах к ML-моделированию, а сосредоточим внимание на концепциях, дающих представление о ситуации в целом. Это даст вам общее представление о процессе построения моделей и позволит быстро приступить к решению практических задач. Для тех, кому требуется дополнительная информация о конкретных техниках ML-моделирования, мы подготовили приложение.

Начнем мы с обобщенного обзора статистического моделирования. Именно это позволит сфокусироваться на глобальных концепциях ML-моделирования, таких как предназначение моделей, способы их практического применения и типы техник моделирования с их слабыми и сильными сторонами. Затем мы вплотную займемся двумя самыми распространенными моделями машинного обучения — решающей задачу классификации и решающей задачу регрессии. В соответствующих разделах будет более подробно рассмотрен процесс построения моделей на основе данных. Врезки «Важнейшие аспекты алгоритмов» дадут информацию о самых распространенных алгоритмах.

3.1. Основы моделирования с машинным обучением

Цель машинного обучения — обнаружение закономерностей и взаимосвязей в данных и практическое применение полученной информации. Процесс обнаружения реализуется методами, которые в течение 30 лет разрабатывались в статистике, компьютерной науке и прикладной математике. Существуют как простые, так и чрезвычайно сложные методы, но все они подчинены одной задаче — оценить функциональное соотношение между входными признаками и целевой переменной.

Кроме того, все техники следуют одному и тому же рабочему процессу, показанному на илл. 3.1. Данные за прошедшие периоды используются для построения и оптимизации модели, которая в свою очередь генерирует прогнозы на основе новых данных. Этот раздел готовит вас к практическим заданиям второй половины главы. Затем мы поговорим об общих целях моделирования с машинным обучением и перейдем к рассказу о том, каким образом может быть использован его конечный продукт, заодно перечислив несколько важных аспектов, отличающих ML-алгоритмы друг от друга.

Илл. 3.1. Основной рабочий процесс машинного обучения

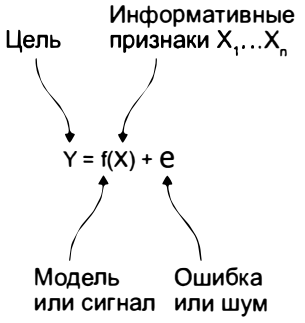
3.1.1. Поиск связи между входными данными и целевой переменной

Обсудим ML-моделирование на примере, для этого возьмем знакомый по главе 2 набор данных Auto MPG. Он содержит такие параметры автомобилей, как регион производства, год выпуска модели, вес машины, мощность в лошадиных силах и число цилиндров. Нам нужно обнаружить соотношение между входными признаками и расходом топлива MPG.

Входные признаки обычно обозначают символом X с нижним индексом. Например, предположим, что X_1 обозначает регион производства, X_2 —

год выпуска модели, X_3 — вес автомобиля и т. п. Набор всех входных признаков обозначим жирным символом **X**. Аналогичным образом целевая переменная, как правило, обозначается символом Y.

Соотношение между входными параметрами **X** и результатом Y можно представить простой формулой:



В этом уравнении f представляет собой неизвестную функцию, которая связывает входные переменные с целевой переменной Y. Цель ML-моделирования — точно оценить f, *используя данные*. Символ ε представляет случайный шум в данных, не связанный с функцией f. Функция f обычно называется *сигналом*, в то время как случайную переменную ε называют *шумом*. Сложность машинного обучения состоит в том, чтобы использовать данные для определения настоящего сигнала, игнорируя при этом шум.

В примере с данными Auto MPG функция f описывает истинный расход топлива каждого автомобиля как функцию от множества его входных признаков. Точно зная вид этой функции, вы имели бы информацию о расходе топлива любой машины, как реальной, так и вымышленной. Но у нас есть многочисленные источники шума ε, включая следующие (это неисчерпывающий список):

- несовершенные результаты измерений расхода MPG каждой машины, обусловленные небольшими неточностями в измерительных приборах, — *помехи при измерениях* (measurement noise);
- флуктуации производственного процесса, приводящие к тому, что каждый выпущенный автомобиль будет иметь немного различающиеся показатели MPG, — *шум от производственного процесса* (manufacturing process noise);

- шум при измерениях таких входных признаков, как вес и мощность в лошадиных силах;
- отсутствие доступа к более широкому набору признаков, который позволил бы точно определить MPG.

Так как работать приходится с зашумленными данными, полученными измерением сотен автомобилей, техники машинного обучения способны дать всего лишь хорошую оценку функции f. Результирующая оценка и называется *ML-моделью*.

В разделах 3.2 и 3.3 мы подробно опишем, как работают все эти техники моделирования. Более того, огромное количество научной литературы по машинному обучению посвящено теме наилучшей оценки f.

3.1.2. Зачем нужна хорошая модель

Предположим, что у нас уже есть хорошая предполагаемая функция f. Что же дальше? У машинного обучения две основные задачи — *прогнозы* и *выводы*.

Прогнозы

Готовую модель можно использовать для прогнозирования значений целевой переменной Y на новых данных $\mathbf{X}_{new}$ путем вставки этих новых признаков в модель. В математической записи, если f_{est} означает вашу оценку функции f, полученную путем машинного обучения (напоминаем, что f выражает истинное соотношение между признаками и целевой переменной), то прогноз значений для новых данных можно получить, вставив эти данные в следующую формулу:

$$Y_{pred} = f_{est}(\mathbf{X}_{new})$$

Затем эти прогнозы можно использовать для принятия решений, касающихся новых данных, или вернуть в автоматизированный процесс.

Вернемся к набору данных Auto MPG. Предположим, у нас есть ML-модель f_{est}, описывающая связь между MPG и входными характеристиками автомобиля. В этом случае мы можем задать вопрос: «Какой расход будет у конкретного автомобиля с известными характеристиками?» Подобное прогнозирование пригодилось бы при проектировании

автомобилей, ведь инженеры могли бы сразу получить данные по MPG для различных конструктивных решений и гарантировать соответствие каждого из них требованиям к расходу топлива.

Предсказание является одной из наиболее частых областей применения систем с машинным обучением. Оно реализуется во множестве случаев:

- расшифровка написанных от руки цифр или записей речи;
- прогнозирование состояния рынка ценных бумаг;
- перспективные оценки;
- предсказание, кто из пользователей с большей вероятностью среагирует на рекламный баннер, подвергнется конвертации или приобретет товар;
- предсказание, кому из пользователей потребуется техническая поддержка, а кто, скорее всего, откажется от услуг;
- выявление мошеннических транзакций;
- подбор рекомендаций.

Из-за высокой точности и скорости генерации прогнозов ML-модели каждый день используются тысячами фирм.

Выводы

Модели с машинным обучением можно использовать не только для прогнозов на основе новых данных, но и для лучшего понимания соотношения между входными признаками и результирующей целью. Хорошо подобранная функция f позволяет ответить на серьезные вопросы о связи между имеющимися переменными. Например:

- какие из входных признаков сильнее всего связаны с целевой переменной?
- это позитивные или негативные связи?
- является ли f простым соотношением или это более детализированная и нелинейная функция?

Такие выводы могут много рассказать о процессе генерации данных и дать ключ к факторам, управляющим взаимосвязями внутри этих данных. Скажем, в случае набора Auto MPG выводы позволят ответить на такие вопросы, как: «Влияет ли регион производства на MPG?», «Какие из признаков сильнее всего связаны с MPG?», «Это позитивная или негативная связь?». В результате мы получим представление о том, какие факторы влияют на расход топлива в автомобиле и как спроектировать машину с более высоким значением MPG.

3.1.3. Типы методов моделирования

Пришло время освежить ваши знания статистики и погрузиться в математические детали ML-моделирования. Не стоит волноваться, это будет обсуждение на достаточно широком уровне, доступное для понимания даже при отсутствии соответствующей подготовки!

Статистическое моделирование — это поиск компромисса между точностью прогнозов и интерпретируемостью модели. Простые модели легко понимаются, но не дают точных предсказаний (особенно в случае сложных взаимосвязей). Комплексные модели могут давать верные прогнозы, но представлять собой сложно интерпретируемый черный ящик.

Модели с машинным обучением делятся на два основных типа — параметрические и непараметрические. Их основное отличие состоит в том, что в первом случае f принимает вид конкретной функции, в то время как непараметрические модели не имеют столь строгих ограничений. Соответственно, параметрические подходы, как правило, просты и интерпретируемы, но менее точны. Верно и обратное: непараметрические подходы обычно менее интерпретируемы, но дают более точные решения для широкого спектра задач. Рассмотрим подробнее оба типа моделей.

Параметрические методы

Простейшим примером параметрического подхода является линейная регрессия. В этом случае предполагается, что f представляет собой линейную комбинацию числовых значений входных параметров.

Стандартная модель линейной регрессии выглядит следующим образом:

$$f(\mathbf{X}) = \beta_0 + X_1 \times \beta_1 + X_2 \times \beta_2 + \ldots$$

В этом уравнении неизвестные параметры β_0, β_1, ... можно интерпретировать как точку пересечения с осью и угловой коэффициент (по отношению к каждому из входных признаков). Настраивая параметрическую модель по выборке данных, вы подбираете наилучшие значения всех неизвестных параметров. После этого можно вставить результаты этого подбора в формулу $f(\mathbf{X})$ и воспользоваться новыми данными для генерации предсказания.

Из распространенных параметрических моделей можно вспомнить также логистическую регрессию, полиномиальную регрессию, линейный дискриминантный анализ, квадратичный дискриминантный анализ, (параметрические) модели смесей и наивный байесовский классификатор (когда используется параметрическая оценка функции плотности). Для выбора параметрической модели часто используются такие подходы, как метод регуляризации Тихонова, лассо и регрессия на главные компоненты. Более подробно некоторые из этих методов будут рассматриваться ниже, а их описания вы найдете в приложении.

Недостатком параметрических подходов являются слишком сильные допущения о виде функции f. В большинстве реальных задач f не имеет настолько простой формы, особенно при наличии множества входных переменных (X). В этих случаях параметрические модели плохо подходят к данным, что ведет к некорректным прогнозам. Поэтому большинство реальных подходов к машинному обучению рассчитывает на непараметрические методы.

Непараметрические методы

В *непараметрических* моделях f не имеет простой фиксированной формы. Форма и сложность этой функции регулируются сложностью данных. Например, если соотношение между X и Y напоминает волну, непараметрический метод выберет вид f, совпадающий с искривленными шаблонами. Но если существует непрерывное соотношение между входной переменной и переменной отклика, будет выбрана простая функция f.

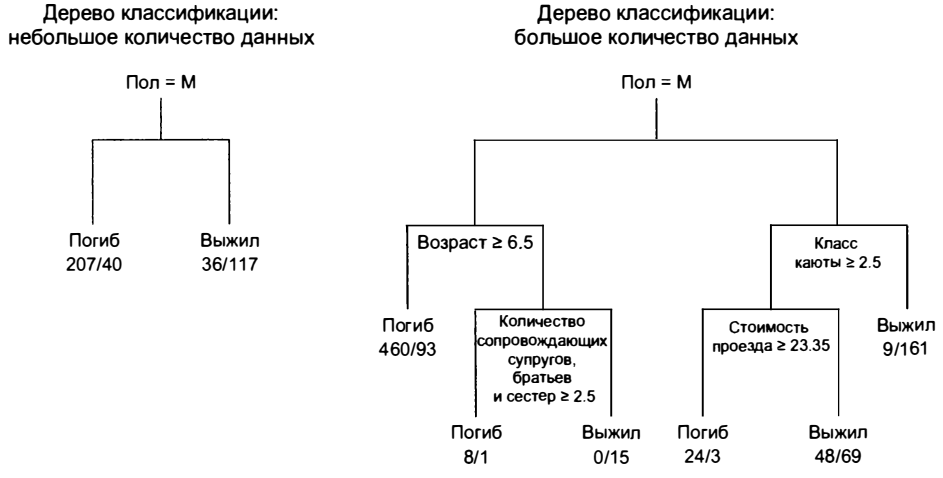
Простым примером непараметрической модели является *дерево классификации* (classification tree). Это набор рекурсивных двоичных решений на базе входных признаков. Алгоритм обучения в этом случае использует целевую переменную для того, чтобы узнать оптимальный набор рас-

щеплений, при котором в конечных листьях появляются экземпляры со сходным значением целевой переменной.

В качестве примера рассмотрим сведения о пассажирах «Титаника». Сначала алгоритм с деревом классификации ищет признак, лучше всего обеспечивающий разбиение, дающее листья с информацией о пассажирах, которые по большей части погибли или же по большей части спаслись. В рассматриваемом случае разбиение лучше всего выполнять по половой принадлежности пассажиров. В новых узлах алгоритм продолжит разбиение по другим входным признакам, пока возможность разбиения не будет исчерпана.

Деревья классификации — непараметрический алгоритм, так как глубина и сложность дерева изначально неизвестны, они определяются самими данными. В случае сложной взаимосвязи между целевой переменной и входными признаками и достаточного количества данных дерево будет расти вглубь, обнаруживая более детализированные шаблоны.

Иллюстрация 3.2 демонстрирует два дерева классификации, полученные из различных подмножеств набора данных о пассажирах «Титаника». Слева дерево, полученное для всего 400 пассажиров: в результате получилась простая модель со всего одним разбиением. Справа дерево, для



Илл. 3.2. Дерево решений — это пример непараметрического ML-алгоритма, так как оно не обладает фиксированной формой. Сложность модели растет с увеличением количества данных, фиксируя более подробные шаблоны. В каждом конечном узле дерева показано соотношение между погибшими и спасшимися в рассматриваемой выборке

создания которого была взята информация о 891 пассажире: большее количество данных позволило увеличить сложность модели и обнаружить более подробные шаблоны.

Другие примеры непараметрических подходов к машинному обучению включают в себя метод k-ближайших соседей, сплайны, основные методы разложения, ядерное сглаживание, обобщенные аддитивные модели, нейронные сети, бэггинг, бустинг, «случайные леса» и метод опорных векторов. Еще раз напомним, что в данной главе будут подробно разбираться некоторые из этих подходов, а их описания вы найдете в приложении.

3.1.4. Обучение с учителем и без

Задачи с машинным обучением делятся на два типа — обучение с учителем и без учителя. В случае *обучения с учителем* (supervised problems) доступна целевая переменная в обучающей выборке, в то время как *обучение без учителя* (unsupervised problems) проводится без идентифицированной целевой переменной.

Все рассматривавшиеся до этого примеры попадали в первую категорию. В каждой задаче мы задавали вопрос: «Выжил ли пассажир "Титаника"?», «Ушел ли клиент?», «Каким был расход топлива?», — а обучающая выборка содержала уже известные ответы. В действительности большинство задач с машинным обучением относятся к первому типу, и большинство ML-техник спроектированы под обучение с учителем. И именно этой теме посвящена большая часть нашей книги.

При обучении без учителя доступны только входные признаки, а связанная с ними целевая переменная отсутствует. Какой анализ можно выполнить в подобном случае? Существуют два основных подхода к обучению без учителя:

- *Кластеризация* (clustering). Входные признаки используются для обнаружения в данных сходных черт и последующего разделения этих данных на группы. Сюда относятся метод k-средних, модели гауссовых смесей и иерархическая кластеризация.
- *Сокращение размерности* (dimensionality reduction). Входные признаки преобразуются в небольшое количество координат с со-

хранением существующей информации об объектах. В эту группу попадают: метод главных компонент, многомерное шкалирование, моделирование многообразий.

Как кластеризация, так и сокращение размерности крайне популярны (особенно метод k-средних и метод главных компонент), поэтому зачастую ими злоупотребляют и используют даже в случаях, когда целесообразнее воспользоваться обучением с учителем.

Впрочем, методы обучения без учителя играют значительную роль, часто применяясь как вспомогательные при обучении с учителем для компиляции обучающей выборки или для извлечения новых признаков. Обучение без учителя будет рассматриваться в главе 8.

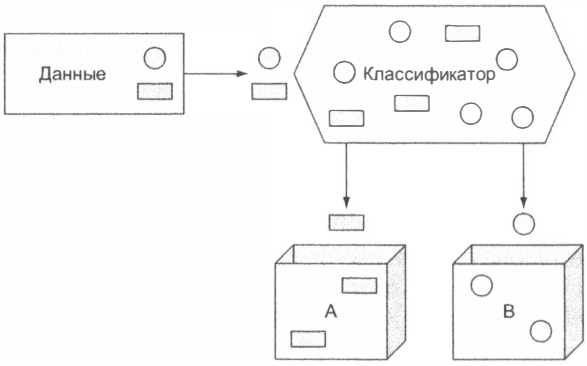
Перейдем к более практическим аспектам ML-моделирования. В следующем разделе мы опишем этапы построения моделей на основе данных, а также практические соображения по выбору алгоритма. Остаток главы разбит на две части, которые посвящены двум наиболее распространенным задачам машинного обучения — классификации и регрессии. Начнем с решения задачи классификации.

3.2. Классификация: распределение по классам

В машинном обучении задача *классификации* (classification) сводится к распределению новых данных по классам с помощью встроенных в алгоритм инструментов. Фильтры спама распределяют почту по папкам «Spam» и «No Spam», а средства, распознающие рукописные цифры, помещают картинки в категории с номерами от 0 до 9. В этом разделе вы научитесь строить классификаторы для своих данных. Схематично решение задачи классификации представлено на илл. 3.3.

Для иллюстрации снова воспользуемся примером. В главе 2 мы использовали сведения о пассажирах «Титаника» для предсказания шансов на выживание на борту злополучного корабля. Иллюстрация 3.4 демонстрирует подмножество этих данных.

Как правило, ML-проект проще всего начать с визуализации данных, чтобы получить о них более полное представление. К примеру, счита-



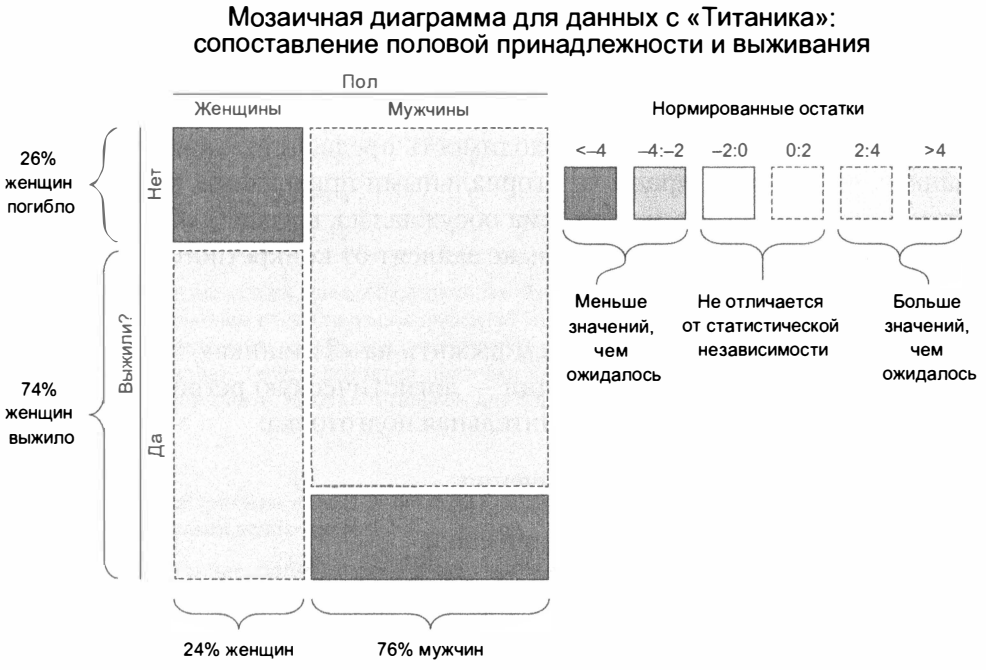
Илл. 3.3. Процесс классификации. Прямоугольники и круги распределяются по классам A и B, это случай бинарной классификации

PassengerId	Survived	Pclass	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Male	22	1	0	A/5 21171	7.25		S
2	1	1	Female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Female	26	0	0	STON/02. 3101282	7.925		S
4	1	1	Female	35	1	0	113803	53.1	C123	S
5	0	3	Male	35	0	0	373450	8.05		S
6	0	3	Male		0	0	330877	8.4583		Q

Илл. 3.4. Подмножество данных о пассажирах «Титаника»

ется общеизвестным, что в катастрофах выживает больше женщин, чем мужчин. Вы можете увидеть подтверждение этого на мозаичной диаграмме с илл. 3.5 (если вы забыли, что такое мозаичная диаграмма, перечитайте раздел 2.3.1).

Описанные в разделе 2.3 техники визуализации дают представление об эффективности каждого признака в наборе данных. Но важно понимать, что привлекательность или бесполезность единичного признака ничего не говорит о его эффективности в комбинации с другим или другими признаками. Может оказаться, что совокупность возраста, половой принадлежности и социального статуса делит пассажиров куда лучше, чем любой из признаков по отдельности. По большому счету, именно для этого и нужны алгоритмы с машинным обучением — чтобы обнаруживать сигналы в большем числе измерений, чем может представить человек.



Илл. 3.5. Мозаичная диаграмма однозначно подтверждает представление о том, что в катастрофах выживает больше женщин, чем мужчин

Ниже мы рассмотрим методологию построения моделей классификации и получения прогнозов с их помощью. Вы на примерах увидите разницу между линейными и нелинейными алгоритмами.

3.2.1. Построение классификатора и получение предсказаний

Первым делом необходимо выбрать алгоритм, который ляжет в основу классификатора. В данном случае у нас широкий выбор, и каждый алгоритм имеет свои сильные и слабые места с точки зрения различных требований к данным и внедрению модели. В приложении вы найдете таблицу со списком алгоритмов и сравнением их свойств. Именно ею мы будем пользоваться для выбора алгоритмов при решении представленных в книге задач. Для случая, который мы собираемся рассматривать, в принципе подходят несколько алгоритмов, поэтому процесс выбора особой роли не играет. В следующей же главе мы поговорим о том, как

корректно измерить эффективность алгоритмов и выбрать наиболее подходящий для решения поставленной задачи.

Затем нужно подготовить данные к моделированию. Исследование признаков может показать необходимость предварительной обработки данных — решить вопрос с категориальными признаками, отсутствующими значениями и т. п. (эта тема обсуждалась в главе 2). Требования к предварительной обработке также зависят от конкретного алгоритма и перечислены в приложении.

Для модели, оценивающей шансы выжить на «Титанике», мы выберем простой алгоритм классификации — логистическую регрессию[1]. Нам потребуется следующая предварительная подготовка:

1. Вставить отсутствующие значения.
2. Расширить категориальные признаки.

В главе 2 вы узнали, что признак `Стоимость проезда` имеет сильную асимметрию. В этой ситуации (для некоторых ML-моделей) выгодно преобразовать переменную, сделав распределение признака более симметричным. Это уменьшит потенциально вредное влияние выбросов. В рассматриваемом случае преобразование сведется к извлечению квадратного корня.

Окончательный вид набора данных для моделирования показан на илл. 3.6.

Теперь можно построить модель, пропустив наши данные через алгоритм логистической регрессии. Алгоритм реализован на базе библиотеки scikit-learn для языка Python, а код модели и генерируемого ею прогноза показан в листинге 3.1.

После построения модели можно прогнозировать выживание ранее не рассматривавшихся пассажиров на базе их признаков. Новые признаки должны быть в формате, представленном на илл. 3.6, поэтому новые

[1] Слово *регрессия* в названии не означает, что перед нами алгоритм регрессии. Логистическая регрессия расширяет линейную с помощью логистической кривой, что и дает нам в итоге линейный классификатор.

Pclass	Age	SibSp	Parch	sqrt_Fare	Gender = female	Gender = male	Embarked = C	Embarked = Q	Embarked = S
3	22	1	0	2.692582	0	1	0	0	1
1	38	1	0	8.442944	1	0	1	0	0
3	26	0	0	2.815138	1	0	0	0	1
1	35	1	0	7.286975	1	0	0	0	1
3	35	0	0	2.837252	0	1	0	0	1

Илл. 3.6. Первые пять строк данных о пассажирах «Титаника» после обработки категориальных признаков, заполнения отсутствующих значений и преобразования переменной Стоимость проезда путем извлечения из нее квадратного корня (см. функцию prepare_data в репозитории с фрагментами кода). Теперь все наши признаки являются численными, то есть находятся в формате, предпочтительном для большинства ML-алгоритмов

Листинг 3.1. Построение классификатора методом логистической регрессии на базе библиотеки scikit-learn

```
from sklearn.linear_model import LogisticRegression as Model

def train(features, target):
    model = Model()
    model.fit(features, target)
    return model

def predict(model, new_features):
    preds = model.predict(new_features)
    return preds

# Данные с "Титаника" загружены в titanic_feats,
# titanic_target и titanic_test
model = train(titanic_feats, titanic_target)
predictions = predict(model, titanic_test)
```

Импортирует алгоритм логистической регрессии

Обучает алгоритм логистической регрессии на признаках и целевых данных

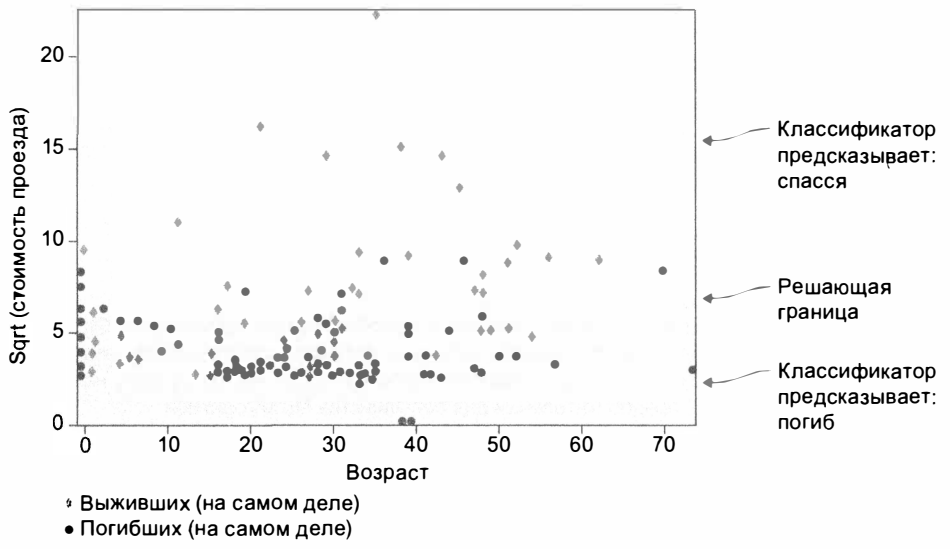
Дает прогноз для нового набора признаков, используя построенную модель

Возвращает модель, построенную алгоритмом

Возвращает предсказание (0 или 1)

сведения о пассажирах следует подвергнуть той же самой процедуре, что и обучающую выборку. Функция предсказания вернет 1, если пассажир, с ее точки зрения, спасется, и 0 в противном случае.

Полезно визуализировать работу классификатора, представив решающую границу в графической форме. При наличии в наборе данных двух признаков можно отобразить границу, которая, согласно модели, отделяет спасшихся пассажиров от погибших. Иллюстрация 3.7 демонстрирует ее для таких признаков, как **Возраст** и **Квадратный корень из стоимости проезда**.



Илл. 3.7. Решающая граница для признаков Возраст и Квадратный корень из стоимости проезда. Ромбы означают спасшихся пассажиров, а круги — погибших. Светлый фон отмечает комбинацию признаков Возраст и Стоимость проезда, которая используется для предсказания шансов на спасение. Обратите внимание на экземпляры, выходящие за границу. Построенный классификатор не идеален, но мы рассматриваем только два измерения. На полном наборе данных алгоритм обнаруживает решающую границу в 10 измерениях, но визуализировать такое очень сложно

ВАЖНЕЙШИЕ АСПЕКТЫ АЛГОРИТМОВ: ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ

Во врезках «Важнейшие аспекты алгоритмов» мы будем подробно рассматривать основные принципы работы используемых в книге алгоритмов. Благодаря этому любознательные читатели после некоторых дополнительных изысканий смогут попробовать себя в написании рабочих версий алгоритмов. Хотя в книге основное внимание уделяется уже готовым пакетам, понимание основ функционирования алгоритмов порой помогает полностью осознать их прогностический потенциал.

Первым мы рассмотрим *алгоритм логистической регрессии*, считающийся простейшим ML-алгоритмом для задач классификации. Задачу проще рассмотреть, представив, что у нас есть всего два признака, а набор данных делится на два класса. Иллюстрация 3.7 демонстрирует именно такой пример с признаками `Возраст` и `Квадратный корень из стоимости проезда`; целевая переменная принимает два значения — спасся или погиб. Для построения классификатора нужно найти границу, наилучшим способом разбивающую данные по целевым классам. В двух измерениях такая линия описывается двумя параметрами. Это и есть параметры модели, значение которых нужно определить.

Затем алгоритм разбивается на следующие этапы:

- Изначально параметры выбираются случайным образом, то есть на плоскость помещается случайная линия.
- Измерьте, насколько хорошо эта линия разделяет два класса. В логистической регрессии для оценки точности измерений используется статистическое *отклонение*.
- Подберите новые значения параметров и измерьте разделительную способность алгоритма.
- Повторяйте, пока не перестанут наблюдаться улучшения. Эта процедура *оптимизации* осуществляется различными специализированными алгоритмами. Зачастую для этой цели выбирается такой несложный алгоритм, как градиентный спуск.

Описанный подход можно расширить на произвольное количество измерений, то есть в модели вы не ограничены всего двумя признаками. Если вас интересуют подробности, советуем провести дополнительные изыскания и попытаться реализовать этот алгоритм на любом известном вам языке программирования. Затем сравните свой код с реализацией из широко используемых ML-пакетов. Большую часть деталей мы опустили, но описанная выше процедура дает представление об основах алгоритма.

Вот некоторые свойства логистической регрессии:

- Алгоритм относительно прост для понимания по сравнению с другими. Кроме того, он легко обсчитывается, что позволяет масштабировать его для больших наборов данных.
- Производительность алгоритма сильно падает, когда разделяющая два класса решающая граница оказывается сильно нелинейной. См. раздел 3.2.2.
- Иногда алгоритм логистической регрессии может переобучать данные. Чтобы уменьшить этот эффект, приходится прибегать к технике, называемой *регуляризацией*. Пример переобучения приведен в разделе 3.2.2.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

Если вы хотите узнать больше о логистической регрессии и ее применении к реальным задачам, рекомендуем книгу: David Hosmer. *Applied Logistic Regression* (Wiley, 2013).

3.2.2. Классификация сложных нелинейных данных

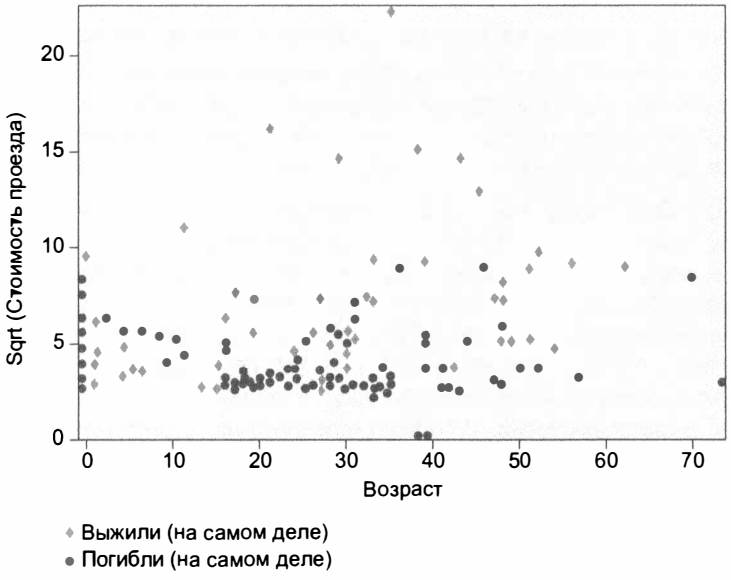
По илл. 3.7 можно понять, почему логистическую регрессию причисляют к линейным алгоритмам, — решающая граница представляет собой прямую линию. Но бывает и так, что корректно разделить данные прямой линией невозможно. Для таких случаев нужен нелинейный алгоритм. Но такие алгоритмы, как правило, требуют много вычислительных ресурсов и не очень хорошо масштабируются к большим наборам данных. Масштабируемость различных типов алгоритмов будет рассматриваться в главе 8.

В приложении можно выбрать нелинейный алгоритм для модели, прогнозирующей шансы на выживание во время крушения «Титаника». Например, такой популярный способ решения нелинейных задач, как метод опорных векторов с нелинейной функцией ядра. По своей природе метод опорных векторов линеен, но благодаря ядру модель превращается в мощный нелинейный метод. Чтобы воспользоваться этим новым алгоритмом, достаточно поменять одну строчку в листинге 3.1. Вид решающей границы для обновленного кода показан на илл. 3.8:

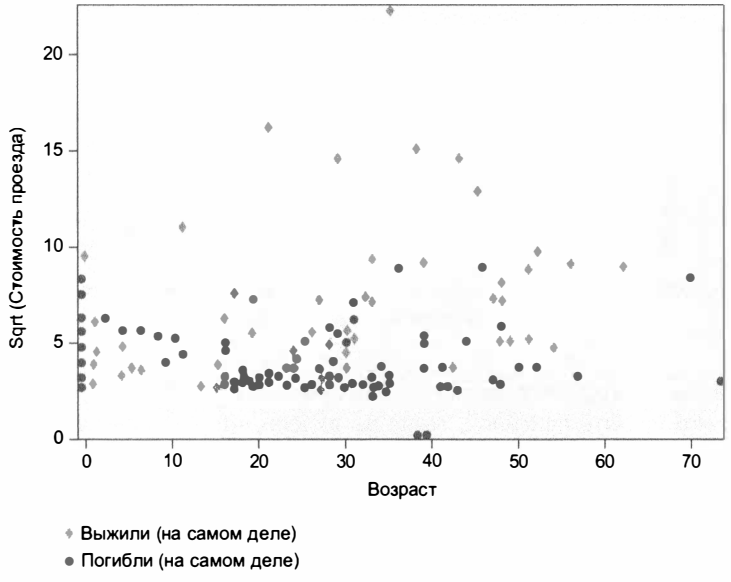
```
from sklearn.svm import SVC as Model
```

Как видите, решающая граница на илл. 3.8 отличается от прямой линии с илл. 3.7. В данном случае мы наблюдаем пример важной в машинном обучении концепции — переобучения. Алгоритм может обучиться так хорошо, что идеально аппроксимирует все записи до единой, но при этом он лишается способности генерировать прогнозы для новых, не входящих в обучающую выборку данных. Чем сложнее модель, тем выше риск переобучения.

Обычно, чтобы избежать переобучения, используют встроенные в алгоритм параметры модели. Посредством тонкой настройки параметров при неизменных данных можно получить более отвечающую ситуации решающую границу. Обратите внимание, что сейчас мы определяем возникновение переобучения интуитивно. В главе 4 вы научитесь с помощью данных и статистики выражать эту интуицию численно. Пока же воспользуйтесь нашим опытом и отредактируйте так называемый *параметр регуляризации*, обозначаемый буквой γ. Мы не будем сейчас детально вникать в его суть, просто запомните, что он снижает риск переобучения. В главе 5 вы узнаете, как оптимизировать параметры модели, не полагаясь исключительно на собственные догадки. Присвоив этому параметру значение 0,1, вы значительно улучшите решающую границу, как показано на илл. 3.9.



Илл. 3.8. Нелинейная решающая граница шансов выжить на «Титанике», построенная классификатором на базе метода опорных векторов с нелинейной функцией ядра. Светлый фон отмечает комбинацию таких признаков, как Возраст и Стоимость проезда, которые предсказываются для получения информации о выживании пассажиров

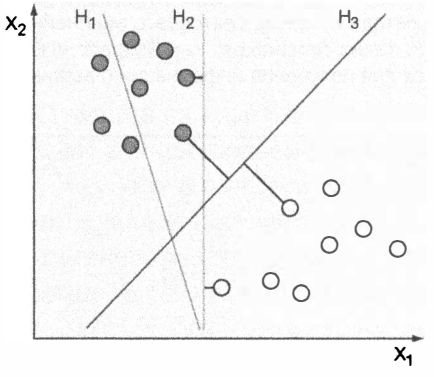


Илл. 3.9. Решающая граница для метода опорных векторов с нелинейной функцией ядра при $\gamma = 0{,}1$

ВАЖНЕЙШИЕ АСПЕКТЫ АЛГОРИТМОВ: МЕТОД ОПОРНЫХ ВЕКТОРОВ

Алгоритм, который называется методом опорных векторов (SVM — support vector machine), часто выбирают для решения как линейных, так и нелинейных задач. Он обладает рядом интересных теоретических и практических свойств, которые делают его полезным во многих сценариях.

Основная идея алгоритма, как и у рассмотренной выше логистической регрессии, состоит в поиске линии (или гиперплоскости в случае большего числа измерений), оптимально разделяющей два класса. Но алгоритм не измеряет расстояние до всех точек, а пытается найти максимальный *зазор* исключительно между точками по обе стороны от линии решений. Основная идея такого подхода состоит в том, что нам не приходится беспокоиться о точках, корректно расположенных внутри границ, нас интересуют только близкие к границам точки. Из представленного ниже рисунка понятно, что линии H_1 и H_2 — плохие разделительные границы, так как расстояние до ближайшей точки от обеих сторон линии далеко не максимальное. Оптимальна линия H_3.



Найденная методом опорных векторов решающая граница (H_3) зачастую куда лучше, чем границы, проводимые другими ML-алгоритмами

Хотя данный алгоритм и является линейным, ведь разделительная граница вполне может быть линейной, выше вы видели, что он также умеет обучать нелинейные данные. В этом случае SVM использует прием с ядрами. *Ядром* (kernel) называется математическая конструкция, которая может «включать» в себя пространство с данными. В этом дополнительном пространстве алгоритм в состоянии найти линейную границу, которая после возвращения в обычное пространство оказывается нелинейной.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

Об алгоритмах машинного обучения написаны сотни книг, содержащих сведения обо всем — от теоретических обоснований и эффективных реализаций до вариантов практического применения. Если вы хотите найти более строгое описание, рекомендуем два классических издания, посвященных ML-алгоритмам:

- Тревор Хасти «Введение в статистическое обучение с примерами на языке R» (ДМК Пресс, 2016).
- Christopher Bishop. *Pattern Recognition and Machine Learning* (Springer, 2007).

3.2.3. Классификация в случае множества классов

До этого момента мы рассматривали задачу распределения всего по двум классам. Но на практике дело этим не ограничивается. Хороший пример многоклассовой классификации — задача на распознавание написанных от руки цифр. Каждый раз при отправке старой доброй почты робот считывает с конверта индекс и определяет, куда должно уйти письмо. Для этого крайне важна способность корректно распознавать цифры. Общедоступная база данных MNIST[1] хорошо подходит для решения задач такого типа. Она содержит 60 000 изображений рукописных цифр, например таких, как показано на илл. 3.10.

Илл. 3.10. Четыре цифры, случайным образом выбранные из базы данных MNIST

Все изображения имеют размер 28 × 28 пикселей, но мы будем преобразовывать каждое такое изображение в $28^2 = 784$ признаков, по одному признаку на каждый пиксель. В итоге задача становится не только многоклассовой, но и многомерной. Закономерность, которую должен обнаружить алгоритм, представляет собой сложную комбинацию целого набора признаков. Кроме того, задача нелинейна по своей природе.

[1] Адрес базы данных MNIST: *http://yann.lecun.com/exdb/mnist/*

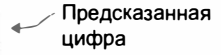
Задачу построения классификатора мы начнем с выбора в приложении подходящего алгоритма. Первым в списке нелинейных алгоритмов со встроенной поддержкой многоклассовых задач является метод k-ближайших соседей. Это еще один простой, но мощный алгоритм нелинейного ML-моделирования. Чтобы им воспользоваться, заменим одну строчку в листинге 3.1 и добавим функцию для получения всех прогнозируемых вероятностей, а не только конечного прогноза:

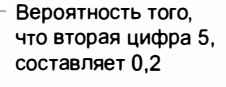
```
from sklearn.neighbors import KNeighborsClassifier as Model

def predict_probabilities(model, new_features):
    preds = model.predict_proba(new_features)
    return preds
```

Построив классификатор на основе метода k-ближайших соседей и сгенерировав прогноз для четырех цифр с рис 3.10, мы получаем таблицу вероятностей, показанную на илл. 3.11.

Как видите, прогноз для первой и третьей цифр попал в точку, а для четвертой цифры неопределенность составляет всего 10%. Что же касается второй цифры (3), нет ничего удивительного в том, что ее сложно как следует классифицировать. Именно поэтому мы рассматриваем полный набор вероятностей — это дает возможность что-то сделать, если у нас нет полной уверенности. Необходимость такого подхода легко обосновать на примере с почтой. Когда робот сильно сомневается в некоторых цифрах, письмо должен осмотреть человек, чтобы исключить вероятность отправки в неверном направлении.

	Actual value	0	1	2	3	4	5	6	7	8	9
Digit 1	7	0	0	0	0.0	0	0.0	0	1	0	0.0
Digit 2	3	0	0	0	0.7	0	0.2	0	0	0	0.1
Digit 3	9	0	0	0	0.0	0	0.0	0	0	0	1.0
Digit 4	5	0	0	0	0.0	0	0.9	0	0	0	0.1





Илл. 3.11. Таблица предсказанных вероятностей от классификатора по методу k-ближайших соседей, примененного к базе данных MNIST

ВАЖНЕЙШИЕ АСПЕКТЫ АЛГОРИТМОВ: МЕТОД k-БЛИЖАЙШИХ СОСЕДЕЙ

Метод k-ближайших соседей — простое, но мощное нелинейное средство машинного обучения. Его часто используют, когда требуется быстро обучить модель, а само прогнозирование, как правило, происходит медленнее. Скоро вы увидите, почему дело обстоит именно так.

Основная идея метода состоит в классификации новых данных путем их сравнения с аналогичными записями из обучающей выборки. Если записи набора данных состоят из ряда чисел n_i, *расстояние* между записями можно найти по известной формуле:

$$d = \sqrt{n_1^2} + n_2^2 + ... + n_n^2.$$

Чтобы получить предсказание для новых записей, мы ищем ближайшую известную запись и предполагаем, что новая запись принадлежит к этому же классу. Это классификатор по одному ближайшему соседу, так как рассматривается всего одна, ближайшая запись. Обычно используется 3, 5 или 9 соседей и выбирается чаще других встречающийся класс. В задачах с двумя классами число соседей берут нечётным, чтобы не возникало неоднозначной ситуации, когда одинаковое число соседей принадлежит разным классам.

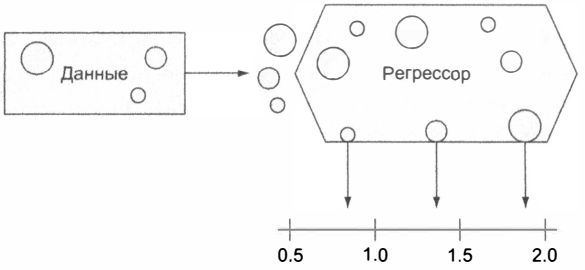
Фаза обучения проходит относительно быстро, так как мы индексируем известные записи для быстрого вычисления расстояний до новых данных. На фазе прогнозирования, когда большая часть работы уже проделана, ищется ближайший сосед из всего набора данных.

В предыдущем простом примере расстояние вычислялось по евклидовой метрике. Но можно воспользоваться и более продвинутыми вариантами расчета метрики, в зависимости от имеющегося набора данных.

Метод k-ближайших соседей применяется для решения не только задач классификации, но и задач регрессии. Во втором случае вместо наиболее общего класса соседей рассматривается среднее значение или медиана целевых значений соседей. Регрессия будет рассматриваться в разделе 3.3.

3.3. Регрессия: предсказание численных значений

Не каждая задача машинного моделирования сводится к распределению записей по классам. Иногда целевая переменная принимает численные значения, например при прогнозировании стоимости доллара в финансовой модели. Предсказание численных значений мы называем регрессией, а саму модель *регрессором*. Концепция регрессии показана на илл. 3.12.



Илл. 3.12. В процессе регрессии модель предсказывает численное значение записи

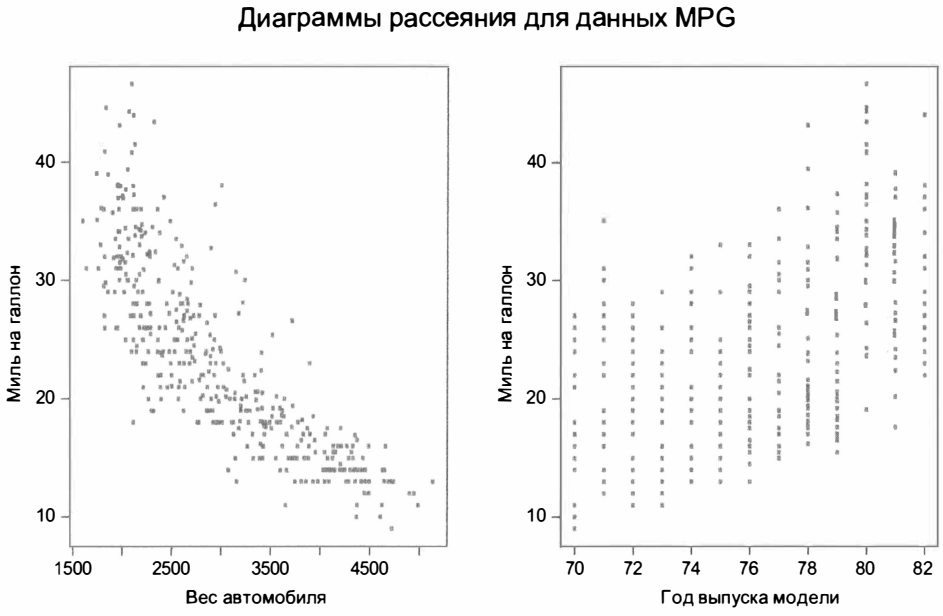
Рассмотрим пример регрессионного анализа, взяв знакомый нам по главе 2 набор данных Auto MPG. Построим модель, которая сможет предсказывать среднее число миль на галлон топлива при условии, что нам известны характеристики автомобиля, например его мощность в лошадиных силах, вес, место и год выпуска модели. Небольшая выдержка из этого набора данных показана на илл. 3.13.

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model/year	Origin
0	18	8	307	130	3504	12.0	70	1
1	15	8	350	165	3693	11.5	70	1
2	18	8	318	150	3436	11.0	70	1
3	16	8	304	150	3433	12.0	70	1
4	17	8	302	140	3449	10.5	70	1

Илл. 3.13. Небольшая выдержка из данных Auto MPG

В главе 2 мы обнаружили полезные соотношения между уровнем расхода топлива, весом автомобиля и годом появления модели. Они показаны на илл. 3.14.

В следующем разделе вы узнаете, как построить базовую линейную регрессионную модель, которая будет предсказывать количество миль, проезжаемое на одном галлоне топлива по набору данных для различных автомобилей. После этого мы рассмотрим более усовершенствованные алгоритмы моделирования нелинейных данных.



Илл. 3.14. Диаграммы рассеяния показывают, что такие признаки, как Вес автомобиля и Год выпуска модели, полезны для предсказания расхода топлива. Более подробно см. в главе 2

3.3.1. Построение регрессора и генерация прогнозов

И снова мы начнем с выбора рабочего алгоритма и приведения данных к подходящему формату. Можно утверждать, что линейная регрессия является простейшим методом регрессионного анализа. Как легко понять по названию, это линейный алгоритм, и в приложении указано, какой именно предварительной обработке должны подвергаться данные. Нужно: 1) вставить отсутствующие значения и 2) расширить категориальные признаки. Наш набор данных Auto MPG является полным, поэтому первый пункт мы пропускаем, но в нем присутствует один категориальный столбец — `Origin` (Происхождение). После того как мы его расширим (как описано в разделе 2.2.1), данные примут форму, показанную на илл. 3.15.

Теперь можно воспользоваться алгоритмом и построить модель. Для этого опять же достаточно заменить в листинге 3.1 эту строчку:

```
from sklearn.linear_model import LinearRegression as Model
```

	MPG	Cylin-ders	Displace-ment	Horse-power	Weight	Accele-ration	Model/ year	Origin = 1	Origin = 2	Origin = 3
387	27	4	140	86	2790	15.6	82	1	0	0
388	44	4	97	52	2130	24.6	82	0	1	0
389	32	4	135	84	2295	11.6	82	1	0	0
390	28	4	120	79	2625	18.6	82	1	0	0
391	31	4	119	82	2720	19.4	82	1	0	0

Илл. 3.15. Данные Auto MPG после расширения категориального столбца Происхождение

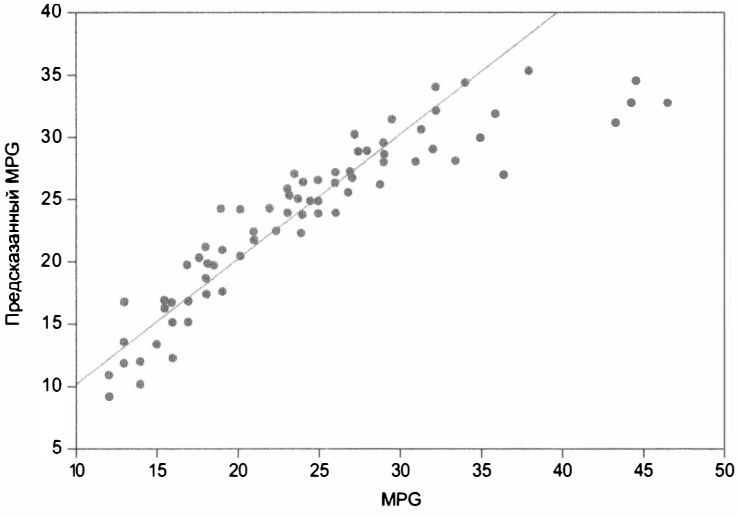
С готовой моделью на руках можно приступать к предсказаниям. Но в рассматриваемом примере мы перед построением модели разобьем данные на обучающую выборку и тестовый набор. В главе 4 процесс оценки моделей будет рассмотрен детально, а пока просто воспользуемся несложной техникой. Обучая модель только на части данных и оставив остальные для тестирования, можно затем генерировать прогнозы для тестового набора и посмотреть, насколько полученные значения совпали с реальными. Обучение модели на всем объеме данных и последующее тестирование на небольшой выборке является подтасовкой, так как уже знакомая с данными модель с большей вероятностью даст корректный прогноз.

Результаты прогнозирования для тестового набора данных и их сравнение с реальными значениями показаны на илл. 3.16. В нашем случае обучение модели происходило на 80% данных, а остальные 20% были оставлены для тестирования.

Origin = 1	Origin = 3	Origin = 2	MPG	Predicted MPG
0	0	1	26.0	27.172795
1	0	0	23.8	24.985776
1	0	0	13.0	13.601050
1	0	0	17.0	15.181120
1	0	0	16.9	16.809079

Илл. 3.16. Сравнение результатов предсказания MPG для тестовых данных с реальными значениями

Хорошим способом сравнения в случае, когда предсказанных строк много, является наша старая добрая диаграмма рассеяния. В регрессионных задачах как реальные, так и предсказанные значения целевых переменных — численные. Сопоставив на диаграмме рассеяния предсказанные значения с реальными, вы сможете оценить корректность прогноза. Такая диаграмма для тестового набора данных Auto MPG показана на илл. 3.17.



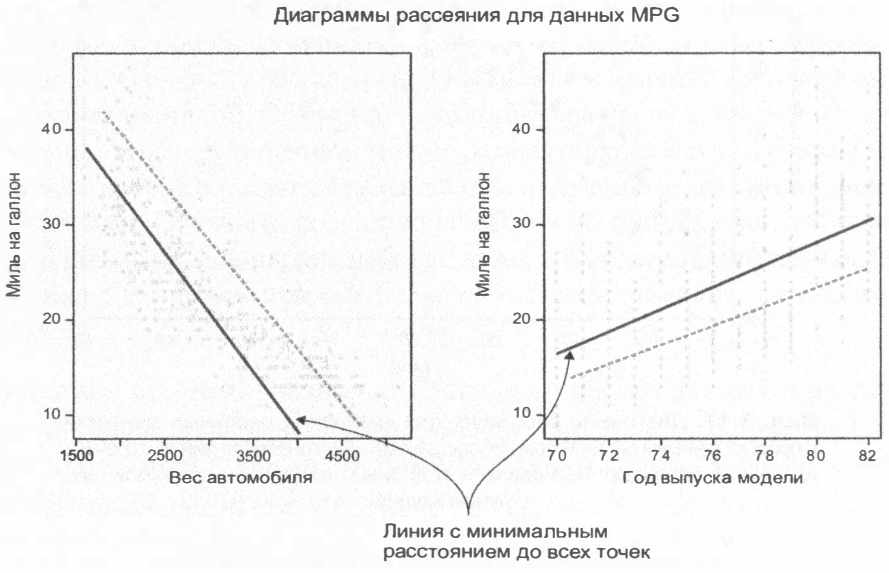
Илл. 3.17. Диаграмма рассеяния для сравнения реальных значений с предсказанными на тестовом наборе данных. Диагональная линия отмечает идеальный регрессор. Чем ближе к этой линии результаты прогнозов, тем лучше модель

Мы видим, что эффективность модели достаточно высока, так как все предсказанные значения лежат близко к оптимальной диагональной линии. По виду этой диаграммы можно понять, как ML-модель поведет себя с новыми данными. В рассматриваемом случае мы видим, что некоторые прогнозы для более высоких значений MPG занижены, что дает информацию к размышлению. Для получения более корректных оценок при высоких значениях MPG могут потребоваться дополнительные примеры автомобилей с низким расходом топлива или же нужно будет получить более качественные данные для этого режима.

ВАЖНЕЙШИЕ АСПЕКТЫ АЛГОРИТМОВ: ЛИНЕЙНАЯ РЕГРЕССИЯ

Аналогично логистической регрессии для классификации *линейная регрессия* считается самым простым и широко используемым алгоритмом для построения регрессионных моделей. Основными его достоинствами являются линейная масштабируемость и высокий уровень интерпретируемости.

Алгоритм выводит записи набора данных как точки, отмечая значения целевой переменной по оси y и располагая их в виде прямой линии (или гиперплоскости в случае двух и более признаков). Следующий рисунок иллюстрирует процесс оптимизации расстояния между этими точками и прямой линией, которую формирует модель.



Демонстрация того, как линейная регрессия определяет наилучшую эмпирическую линию. Непрерывная линия здесь показывает оптимальный результат линейной регрессии, дающий минимальное среднеквадратичное отклонение от данных до любой другой возможной линии (например, обозначенной на рисунке пунктиром)

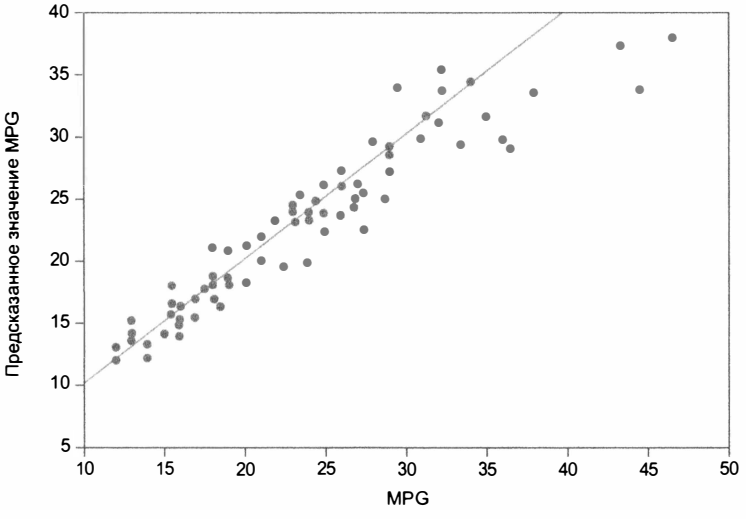
Прямую линию можно описать двумя параметрами, по мере роста размерности число параметров тоже растет. Вам знакомо уравнение $y = a \times x + b$ из базового курса математики. Параметры a и b подгоняются к данным и после оптимизации полностью описывают модель и могут использоваться для получения прогнозов на основе новых данных.

3.3.2. Регрессия для сложных нелинейных данных

В некоторых наборах данных соотношение между признаками невозможно описать линейной моделью, и алгоритмы, подобные линейной регрессии, не дадут нужной точности предсказаний. Но такие свойства этих алгоритмов, как масштабируемость, могут сместить чашу весов в их пользу, примирив с более низкой точностью. Тем более что нет никаких гарантий, что нелинейный алгоритм даст более точный прогноз, в то время как риск переобучения возрастает. Нелинейную регрессионную модель мы рассмотрим на примере алгоритма «случайный лес». Это

Origin = 1	Origin = 3	Origin = 2	MPG	Predicted MPG
0	0	1	26.0	27.1684
1	0	0	23.8	23.4603
1	0	0	13.0	13.6590
1	0	0	17.0	16.8940
1	0	0	16.9	15.5060

Илл. 3.18. Таблица, сравнивающая реальные и предсказанные значения MPG для нелинейной модели на базе алгоритма «случайный лес»



Илл. 3.19. Сравнение данных MPG со значениями, предсказанными нелинейной моделью на базе алгоритма «случайный лес»

популярный метод решения задач с высокой нелинейностью, для которых важна точность. Как следует из представленной в приложении информации, пользоваться им легко, так как требуется минимальная предварительная обработка данных. На илл. 3.18 и 3.19 показаны результаты предсказаний для тестовых данных Auto MPG, сделанные моделью на базе алгоритма «случайный лес».

Эта модель не сильно отличается от линейного алгоритма, по крайней мере визуально. Непонятно, какой из алгоритмов более эффективен с точки зрения точности. В следующей главе мы покажем, как количественно оценить производительность, и вы сможете измерять точность предсказаний.

ВАЖНЕЙШИЕ АСПЕКТЫ АЛГОРИТМОВ: «СЛУЧАЙНЫЙ ЛЕС»

Последний алгоритм в этой главе — «*случайный лес*» (RF — random forest). Это высокоточный нелинейный алгоритм, широко применяемый для решения задач классификации и регрессии.

В его основе лежит дерево решений. Представим, что нужно принять решение, например выбрать, над чем работать дальше. Существуют переменные, которые могут помочь в принятии этого решения, причем некоторые из них весят больше остальных. В рассматриваемом случае можно начать с вопроса: «Сколько денег это принесет?». Если меньше $10, то не стоит приниматься за это дело. Если же больше $10, то можно задать следующий вопрос из нашего дерева: «Работа над этим доставляет удовольствие?» с ответами «Да»/«Нет». Построение дерева продолжается, пока вы не придете к решению и не выберете себе задачу.

Алгоритм дерева решений позволяет компьютеру определить по обучающей выборке, какие переменные являются самыми важными. Именно их он помещает на верх дерева, постепенно переходя затем к менее важным переменным. Такой подход дает возможность скомбинировать переменные и сказать: «Если будет заработано больше $10, работа принесет мне удовольствие и может быть закончена менее чем за час, тогда "Да"».

К сожалению, верхние уровни дерева решений сильно влияют на ответ, и если новые данные не следуют тому же распределению, что и обучающая выборка, может пострадать способность модели к обобщениям. Именно здесь на помощь приходит модель на базе алгоритма «случайный лес». Построение набора деревьев снижает этот риск. При поиске ответа в случае классификации выбирается большинство голосов, а в случае регрессии — среднее. Благодаря использованию голосов или средних появляется возможность получить полные вероятности, которую дают не все алгоритмы.

«Случайные леса» имеют и ряд других преимуществ, например нечувствительность к несущественным признакам, зашумленным наборам данных с пропущенными значениями и неверно помеченным записям.

3.4. Заключение

В этой главе мы познакомили вас с моделированием на базе машинного обучения. Вот список основных положений, которые следует запомнить:

- Цель моделирования — описать взаимосвязь между входными признаками и целевой переменной.
- Модели позволяют как генерировать предсказания для новых данных (с неизвестной целевой переменной), так и делать выводы о связях (или их отсутствии) в представленных данных.
- Существуют сотни методов ML-моделирования. Некоторые из них параметрические, в них форма математической функции, связывающей признаки с целью, изначально фиксирована. Параметрические модели, как правило, проще интерпретируются, но дают менее точные прогнозы, чем более гибкие непараметрические подходы, умеющие адаптироваться к реально сложным соотношениям между признаками и целевой переменной. Благодаря высокой точности прогнозов и гибкости непараметрические модели предпочитаются большинством практикующих машинное обучение.
- Методы машинного обучения делятся на обучение с учителем и без учителя. Для обучения с учителем требуется обучающая выборка с известной целевой переменной, в то время как методы обучения без учителя обходятся без нее. Большая часть этой книги посвящена обучению с учителем.
- Две основные задачи в случае обучения с учителем — это задача классификации с категориальной целевой переменной и задача регрессии, в которой эта переменная численная. В этой главе вы научились строить модели для решения задач классификации и регрессии и узнали, как с помощью этих моделей получить прогноз для новых данных.
- Кроме того, мы достаточно подробно рассмотрели задачу классификации. Линейные алгоритмы определяют линейную решающую границу между классами. Если же данные невозможно разделить линейно, требуются нелинейные методы. Применение последних, как правило, требует больше вычислительных ресурсов.

- В отличие от задачи классификации, в которой предсказывается категориальная целевая переменная, в регрессионных моделях ищутся числовые значения. Вы рассмотрели примеры линейных и нелинейных методов и узнали, как визуализировать прогнозы этих моделей.

3.5. Терминология

Термин	Определение
Модель (model)	Основной продукт применения ML-алгоритма на обучающей выборке
Предсказание (prediction)	Предсказания выполняются путем прогона через модель новых данных
Вывод (inference)	Процесс получения представления о данных путем построения модели
(Не)параметрическая ((non)parametric)	Параметрические модели делают предположение о структуре данных, непараметрические — нет
(Un)supervised	Модели, обучающиеся с учителем, например для решения задач классификации и регрессии, ищут соответствие между входными признаками и целевой переменной. Модели, обучающиеся без учителя, ищут шаблоны в данных, не прибегая к известной целевой переменной
Кластеризация (clustering)	Форма обучения без учителя, распределяющая данные по кластерам на основе информации об их сходстве
Уменьшение размерности (dimensionality reduction)	Еще одна форма обучения без учителя, позволяющая сопоставить многомерному набору данных представление с более низкой размерностью, обычно для создания двух- или трехмерного графика
Классификация (classification)	Метод обучения с учителем, распределяющий данные по классам
Регрессия (regression)	Метод обучения с учителем, предсказывающий численные значения целевой переменной

В следующей главе мы рассмотрим процесс создания и тестирования моделей — восхитительную часть машинного обучения. Вы увидите, будут ли выбранные алгоритмы работать над решением поставленной им задачи. Также вы узнаете, как гарантированно убедиться в корректности модели и понять, насколько хорошие прогнозы она будет генерировать для новых данных. Вы получите информацию о методах проверки, метрических показателях и некоторых полезных визуализациях, позволяющих оценить производительность моделей.

4 Оценка и оптимизация модели

В этой главе:

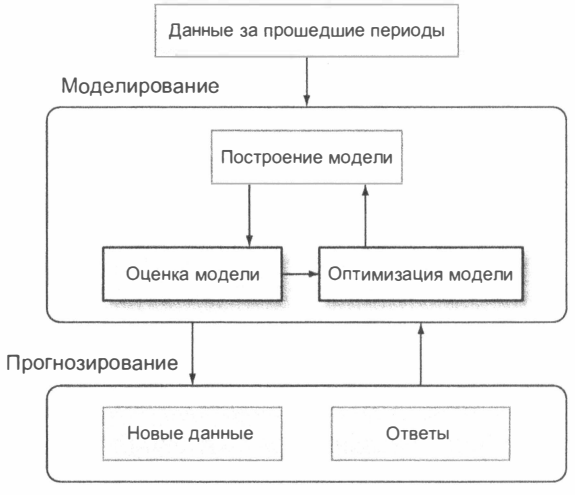
- ✓ техника скользящего контроля для корректной оценки прогностической производительности моделей;
- ✓ переобучение и способы его предотвращения;
- ✓ стандартные оценочные показатели и визуализация двоичной и многоклассовой классификации;
- ✓ стандартные оценочные показатели и визуализация регрессионных моделей;
- ✓ оптимизация модели путем подбора параметров.

После обучения модели требуется оценить ее точность. Модель готова к использованию только после того, как станет понятно, насколько хорошо она генерирует предсказания для новых данных. Только убедившись в хорошей прогностической производительности модели, можно применять ее для анализа. Недостаточная эффективность предсказаний, в свою очередь, является поводом для пересмотра данных и самой модели, а также попыток улучшить и оптимизировать ее. Последний раздел этой главы познакомит вас с простой техникой оптимизации. Более сложные методы увеличения точности будут рассматриваться в главах 5, 7 и 9.

Корректная оценка прогностической производительности модели — задача нетривиальная. Мы покажем статистически строгие методики оценки этого параметра, демонстрируя принцип их работы как графически, так и с помощью псевдокода.

Затем мы подробно поговорим об оценке моделей классификации, сделав особый упор на оценочные метрики и графические инструменты, используемые при решении практических задач. После этого аналогичные аспекты будут рассмотрены для моделей регрессии. Напоследок мы опишем простой способ оптимизации прогностической производительности путем подбора параметров модели.

К концу главы вы будете владеть техниками и знаниями, позволяющими оценить точность предсказаний построенных в главе 3 ML-моделей и выполнить их оптимизацию (илл. 4.1). Подобный анализ моделей даст



Илл. 4.1. Этапы оценки и оптимизации в рабочем процессе машинного обучения

информацию о том, достаточно ли хороша готовая модель, или же с ней нужно еще поработать.

4.1. Оценка прогностической точности на новых данных

Основная цель машинного обучения с учителем — получение точных предсказаний. Наши ML-модели должны как можно точнее прогнозировать значения целевой переменной при работе с новыми данными (в которых эта переменная неизвестна). Другими словами, нужно, чтобы построенная на базе обучающей выборки модель умела работать с произвольными данными. В этом случае, внедряя модель в производство, можно быть уверенным, что она генерирует прогнозы высокой точности.

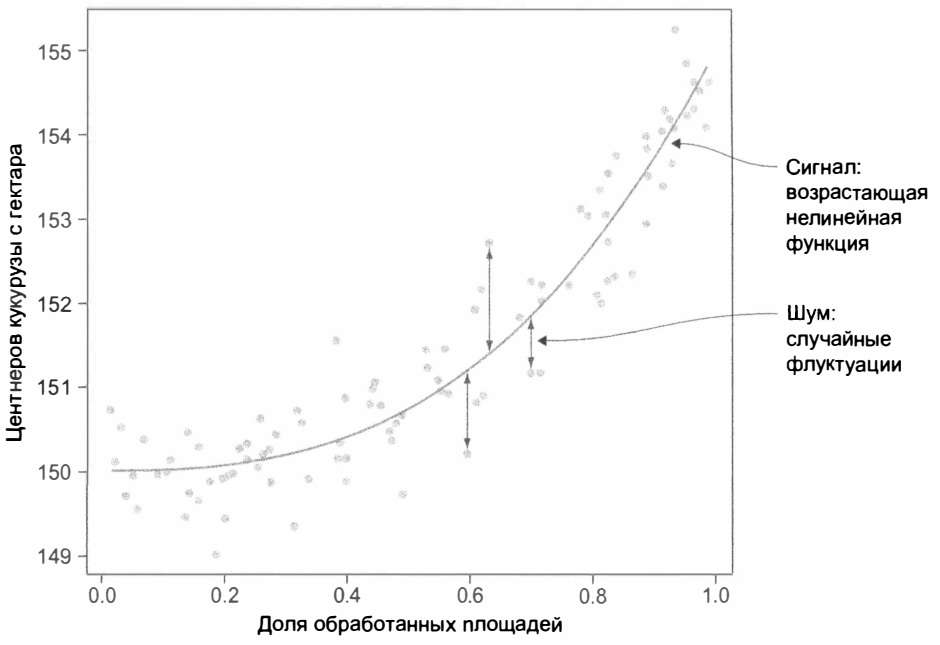
Таким образом, производительность модели оценивается, чтобы узнать, *насколько хорошо она работает с новыми данными*. Эта, казалось бы, несложная задача связана с таким количеством тонких моментов и подводных камней, что способна сбить с толку даже опытного человека.

Опишем трудности, возникающие при оценке ML-моделей, и рассмотрим простой рабочий процесс для преодоления возникающих проблем и получения непредвзятой оценки эффективности.

4.1.1. Проблема: переобучение и чрезмерно оптимистическая оценка модели

Сложности, связанные с оценкой прогностической точности модели, проще всего описать на примере.

Предположим, требуется предсказать урожаи кукурузы в центнерах на гектар как функцию от доли посевных площадей, обработанных новым пестицидом. Для решения этой задачи регрессии есть данные по 100 фермам. Графически отобразив целевую переменную (центнеры кукурузы на гектар) как функцию от признака (процент обработанных площадей), мы сразу увидим наличие возрастающего нелинейного соотношения, а также случайных флуктуаций в данных (илл. 4.2).

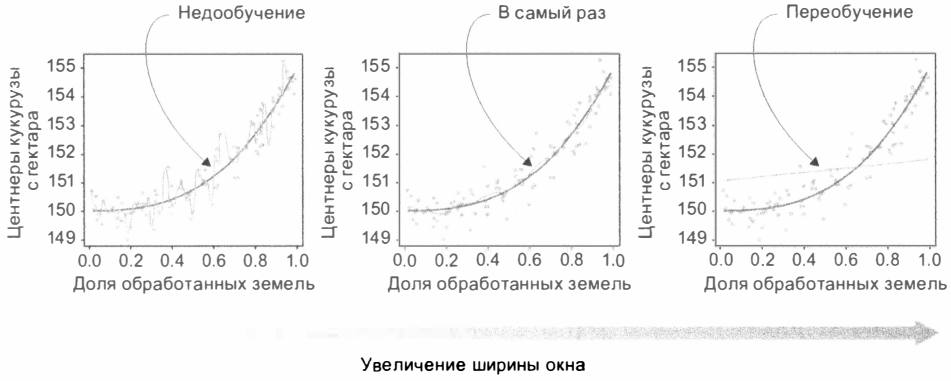


Илл. 4.2. Обучающая выборка для задачи регрессии по урожайности кукурузы содержит четкий сигнал и шум

Возьмем для построения модели, предсказывающей урожайность кукурузы как функцию от доли обработанных площадей, простой непараметрический алгоритм регрессии. Одной из простейших подходящих под это описание техник является *ядерное сглаживание* (kernel smoothing). Этот алгоритм рассматривает локальные среднис: для каждой точки данных значение целевой переменной моделируется как среднее соответствующей переменной по тем данным из обучающей выборки, у которых значение признака близко к значению признака рассматриваемой точки. Окно локального усреднения имеет единственный параметр — *ширину* (bandwidth parameter).

Иллюстрация 4.3 показывает, что происходит во время ядерного сглаживания для различных значений ширины окна. Для больших значений практически все данные из обучающей выборки усредняются, предсказывая целевую переменную на каждом значении входного признака. В результате модель выглядит как обычная прямая и не замечает очевидных тенденций в обучающей выборке. В свою очередь, при маленьких размерах окна результат применения модели к каждому признаку обучающей

выборки определяется всего по одному или двум экземплярам. Как следствие, модель точно отслеживает все выбросы и отклонения в данных. Подобная чувствительность модели к шумам вместо настоящего сигнала называется *переобучением* (overfitting). Нас же интересует промежуточная ситуация, то есть не недообученная и не переобученная модель.



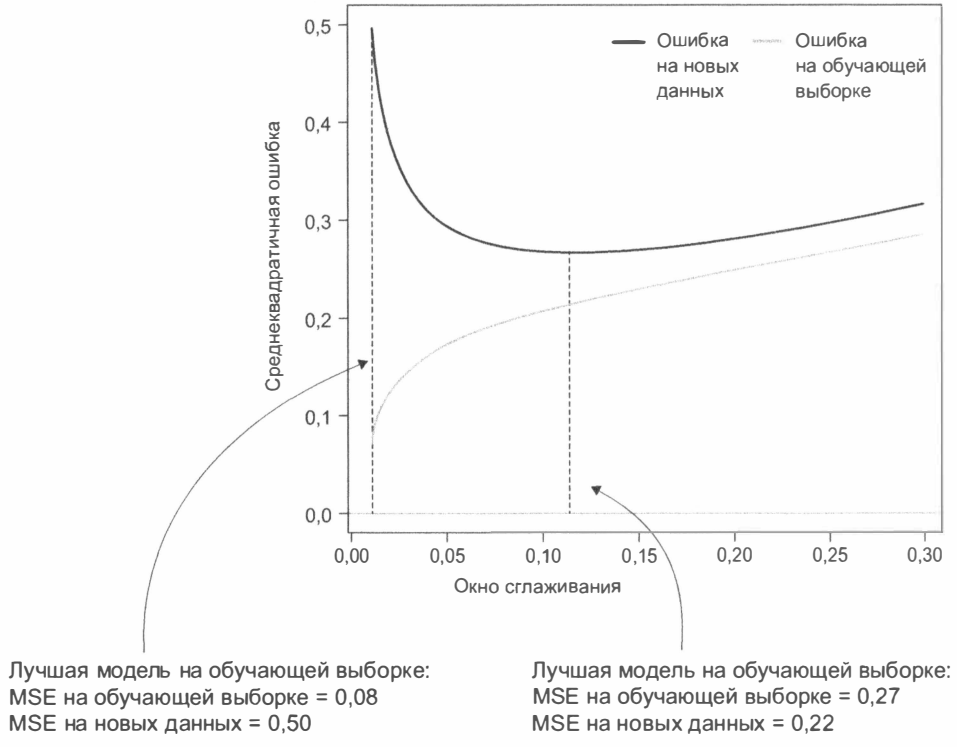
Илл. 4.3. Три случая обучения регрессионной модели с ядерным сглаживанием для обучающей выборки с данными об урожаях кукурузы. При маленьком окне мы получаем переобученную модель, что делает ее слишком бугристой. При большой ширине окна модель оказывается недообученной и выглядит как обычная прямая. Корректный выбор данного параметра дает правильно обученную модель

Вернемся к нашей задаче — определим, насколько хорошо наша ML-модель способна к обобщениям, чтобы предсказать будущие урожаи зерна по данным с различных ферм. Для этого первым делом нужно выбрать оценочную метрику, которая будет определять качество предсказаний. Стандартная метрика оценки для регрессии — *среднеквадратичная ошибка* (MSE — mean squared error). Это сумма квадратов разницы между истинными значениями целевой переменной и значениями, предсказанными моделью (чуть ниже вы познакомитесь с другими оценочными метриками для задач регрессии и классификации).

В этом месте возникает щекотливая ситуация. Оцененная на примере обучающей выборки ошибка (в виде MSE) в прогнозах модели уменьшается с уменьшением размеров окна. Это вполне ожидаемо: чем большая гибкость закладывается в модель, тем лучше она отслеживает закономерности (как в сигнале, так и в шуме) в тренировочных данных. Но модели с минимальным размером окна изрядно переобучены и отслеживают малейшую флуктуацию в данных обучающей выборки. С неизвестными

данными точность предсказаний такой модели будет крайне низкой, так как шум в новых данных будет выглядеть не так, как шум в обучающей выборке.

Соответственно, возникает расхождение между ошибкой обучающей выборки и ошибкой обобщения ML-модели. Это расхождение демонстрирует илл. 4.4. При малых размерах окна среднеквадратичная ошибка на обучающей выборке крайне мала, в то время как на новых данных (в рассматриваемом случае это 10 000 новых экземпляров) она намного больше. Попросту говоря, эффективность прогнозирования модели, оцененная на обучающей выборке, вовсе не характеризует эффективность прогнозирования на новых данных. Поэтому не стоит оценивать производительность модели на тех данных, на которых она обучалась.



Илл. 4.4. Сравнение ошибки на обучающей выборке и на новых данных в задаче регрессии, предсказывающей урожаи кукурузы. Ошибка на обучающей выборке является чрезмерно оптимистичной характеристикой эффективности модели для новых данных, особенно при маленьком размере окна. Использовать ее для оценки прогностической эффективности модели нельзя

НЕ ИСПОЛЬЗУЙТЕ ОБУЧАЮЩУЮ ВЫБОРКУ ДВАЖДЫ

Использование тренировочного набора данных как для обучения модели, так и для ее оценки дает излишне оптимистичные представления об эффективности прогнозирования. В результате зачастую выбирается неоптимальная модель, плохо работающая с новыми данными.

Как видно из данных по урожаям кукурузы, если модель выбирается по минимальному значению MSE на обучающей выборке, мы получаем модель с минимальным размером окна. На тренировочных данных ошибка прогнозирования такой модели составляет 0,08. Но при работе с новыми данными она демонстрирует уже MSE, равное 0,50, что намного хуже оптимального случая (когда ширина окна = 0,12, а MSE = 0,27).

Нам нужна оценочная метрика, лучше аппроксимирующая эффективность работы с новыми данными. Только в этом случае можно гарантировать точность прогнозирования. Но это уже тема следующего раздела.

4.1.2. Решение: скользящий контроль

Итак, мы столкнулись с трудностью при оценке модели: ошибка прогнозирования на обучающей выборке не дает представления о точности работы с ранее неизвестными данными. Оценить вероятность ошибки в общем случае поможет более сложная методика, которая называется *скользящим контролем*, или *перекрестной проверкой* (CV — cross-validation). Она позволяет строго оценить на обучающей выборке точность работы с новыми данными.

Существуют два способа скользящего контроля: метод отложенных данных и контроль по k-блокам.

Метод отложенных данных

Обучение и оценка точности модели на одних и тех же данных дает чрезмерно оптимистичные показатели. Проще всего этого избежать, разделив тренировочные данные на два подмножества. Первое используется для обучения модели, а на втором проверяется точность ее работы.

Такой подход называют *методом отложенных данных* (holdout method), так как из обучающей выборки случайным образом изымается некоторая

часть. Обычно для тестирования оставляется 20–40% данных. Иллюстрация 4.5 демонстрирует базовый алгоритм работы метода, а листинг 4.1 содержит псевдокод на языке Python.

Листинг 4.1. Скользящий контроль методом отложенных данных

```
# у нас есть два варианта входных переменных:
# features - матрица входных признаков
# target - массив соответствующих этим признакам целевых переменных
features = rand(100,5)
target = rand(100) > 0.5

N = features.shape[0] # Общее число экземпляров
N_train = floor(0.7 * N) # Общее число обучающих экземпляров

idx = random.permutation(N) <-- Случайным образом выбирает индекс

idx_train = idx[:N_train]
idx_test = idx[N_train:]    | Разбивает индекс

features_train = features[idx_train,:]
target_train = target[idx_train]
features_test = features[idx_test,:]
target_test = target[idx_test]    | Разбивает данные на обучающее и тестирующее подмножества

# Строим, предсказываем, оцениваем (для заполнения)
# model = train(features_train, target_train)
# preds_test = predict(model, features_test)
# accuracy = evaluate_acc(preds_test, target_test)
```

Применим метод отложенных данных к сведениям об урожаях кукурузы. Для каждого значения ширины окна выполним моделирование (используя разбиение 70/30) и вычислим MSE предсказаний для отложенных 30% данных. На илл. 4.6 оценка ошибки методом отложенных данных показана поверх ошибки модели на новых данных. Бросаются в глаза два основных момента:

- Ошибка, вычисленная методом отложенных данных, близка к ошибке модели на новых данных. Она намного ближе, чем ошибка для обучающей выборки (илл. 4.4), особенно при маленьких размерах окна.
- Оценка ошибки методом отложенных данных сильно зашумлена. Мы видим большой разброс по сравнению с гладкой кривой ошибки на новых данных.

Илл. 4.5. Диаграмма для метода отложенных данных

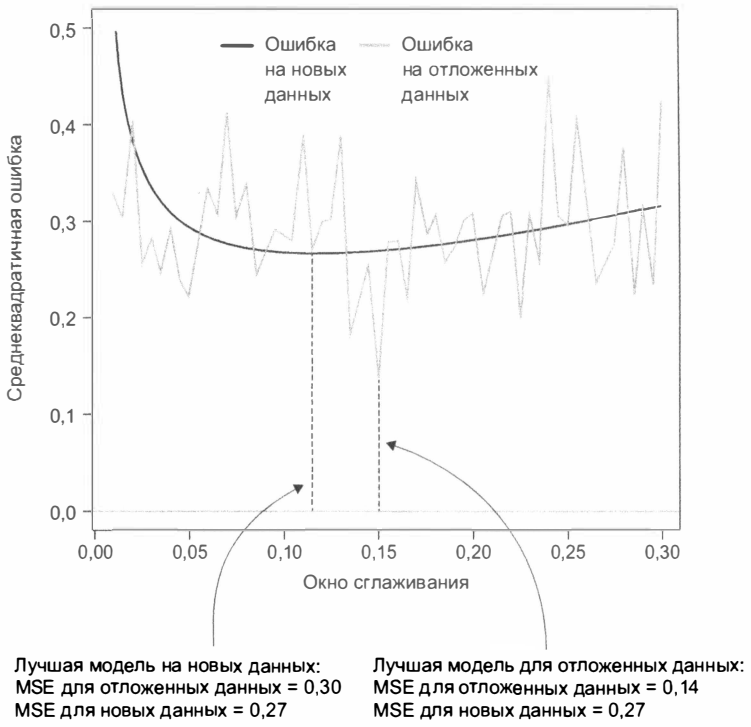
Для борьбы с шумом имеет смысл многократно случайным образом разбить данные на обучающую и тестовую выборки и усреднить результат. Но в процессе итераций каждая точка данных будет попадать в тестовое подмножество различное число раз, что может привести к смещению конечной кривой.

Куда лучшие результаты дает второй подход — контроль по k-блокам.

Контроль по k-блокам

Намного лучшие результаты дает второй подход, к сожалению, очень ресурсоемкий, — *контроль по k-блокам* (k-fold cross-validation).

Как и метод отложенных данных, контроль по k-блокам в процессе обучения модели реализуется путем выделения некоего подмножества тестовых данных. Основное отличие состоит в том, что на этот раз данные случайным образом делятся на k непересекающихся подмножеств (как правило, k равно 5, 10 или 20). Затем модель раз за разом обучается на всем наборе, за исключением данных очередного подмножества, которые в свою очередь используются для генерации последующих предсказаний.



Илл. 4.6. Сравнение ошибки MSE, вычисленной методом отложенных данных, с ошибкой на новых данных для сведений об урожаях кукурузы. Этот метод дает несмещенную оценку ошибки для каждой итерации модели с новыми данными. Но полученная оценка имеет большой разброс, который при ширине окна, дающей практически оптимальную модель, колеблется от 0,14 до 0,40 (ширина окна = 0,12)

После циклического перебора всех k-блоков полученные для каждого из них предсказания усредняются и сравниваются со значением целевой переменной для оценки точности. Процедуру скользящего контроля по k-блокам иллюстрирует илл. 4.7, а листинг 4.2 содержит соответствующий псевдокод.

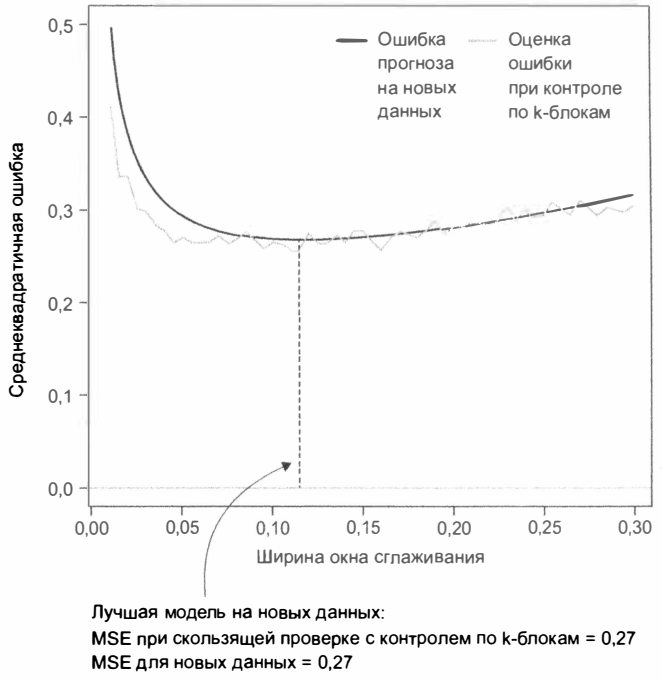
Напоследок применим метод контроля по k-блокам к данным об урожаях кукурузы. Для каждой ширины окна мы применим этот метод с k = 10 и рассчитаем MSE предсказанных значений. Иллюстрация 4.8 демонстрирует оценку MSE для результатов, полученных путем контроля по k-блокам, в сравнении с оценкой MSE прогноза, данного моделью на новых данных. Две кривые очень близки друг к другу.

Илл. 4.7. Диаграмма контроля по k-блокам

Листинг 4.2. Перекрестная проверка с контролем по k-блокам

```
N = features.shape[0]
K = 10 # количество блоков

preds_kfold = np.empty(N)
folds = np.random.randint(0, K, size=N)

for idx in np.arange(K):          <-- Циклический перебор блоков

    features_train = features[folds != idx,:]
    target_train = target[folds != idx]
    features_test = features[folds == idx,:]

    # Строим модель и предсказываем для СК-блока (для заполнения)
    # model = train(features_train, target_train)
    # preds_kfold[folds == idx] = predict(model, features_test)

# accuracy = evaluate_acc(preds_kfold, target)
```

Разбиение данных на обучающее и тестовое подмножества



Илл. 4.8. Сравнение ошибки MSE, вычисленной методом контроля по k-блокам, с ошибкой на новых данных для сведений об урожаях кукурузы. Метод контроля по k-блокам дает хорошее представление о поведении модели с новыми данными, позволяя адекватно оценить будущую ошибку прогнозов и выбрать наилучшую модель

4.1.3. На что следует обращать внимание при перекрестной проверке

Скользящий контроль оценивает точность прогнозов ML-модели при работе с произвольными данными. Это крайне мощная техника, так как она позволяет выбрать модель, наиболее подходящую для решения поставленной задачи.

Но, применяя перекрестную проверку на практике, следует обращать внимание на несколько моментов:

- Методы перекрестной проверки (в том числе и рассмотренные выше метод отложенных данных и метод контроля по k-блокам) предполагают, что обучающая выборка является репрезентативным примером интересующих нас данных. Если вы собираетесь

использовать модель для прогнозов на новых данных, эти данные должны быть хорошо представлены в обучающей выборке. В противном случае оценка вероятности ошибок методами скользящего контроля может оказаться чрезмерно оптимистичной. Решение: убедитесь, что любые потенциальные перекосы в обучающей выборке изучены и сведены к минимуму.

- В некоторых наборах данных присутствуют временные признаки — например, информация о доходах за предыдущий месяц для предсказания прибыли в течение текущего месяца. В подобных случаях нужно следить за тем, чтобы признаки, доступные в будущем, не использовались для предсказания прошлого. Решение: структурировать сведения для метода отложенных данных и для контроля по k-блокам таким образом, чтобы обучающие данные собирались раньше данных для тестирования.
- Чем выше количество блоков при перекрестной проверке по k-блокам, тем точнее оценка ошибки, но тем дольше работает программа. Решение: используйте по крайней мере 10 блоков (или больше), если это возможно. Для моделей, которые быстро обучаются и быстро генерируют прогноз, можно использовать *контроль по отдельным объектам* (leave-one-out) (в этом случае k = количеству экземпляров данных).

Теперь попрактикуемся в применении методов перекрестной проверки и более подробно рассмотрим, как происходит строгая оценка моделей классификации.

4.2. Оценка моделей классификации

Начнем обсуждение, посвященное оценке моделей классификации, с задач так называемой *двоичной классификации* (binary classification). В главе 3 вы впервые познакомились с этим мощным методом прогнозирования положительного/отрицательного результата на базе набора факторов или переменных. Хороший пример двоичной классификации — задачи на выявление заболеваний или предсказание шансов на выживание.

Предположим, что по личным, социальным и экономическим характеристикам произвольного пассажира «Титаника» требуется узнать,

выживет ли он при крушении. Соберем все доступные сведения о пассажирах и обучим классификатор сопоставлять определенные признаки с шансами на выживание. С этим набором данных вы уже встречались в главе 2, но его первые пять строк еще раз демонстрируются на илл. 4.9.

Целевой столбец

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	Male	22	1	0	A/5 21171	7.25	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	Female	38	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Helkkinen, Miss Laina	Female	26	0	0	STON/02. 3101282	7.925	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	Female	35	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	Male	35	0	0	373450	8.05	NaN	S

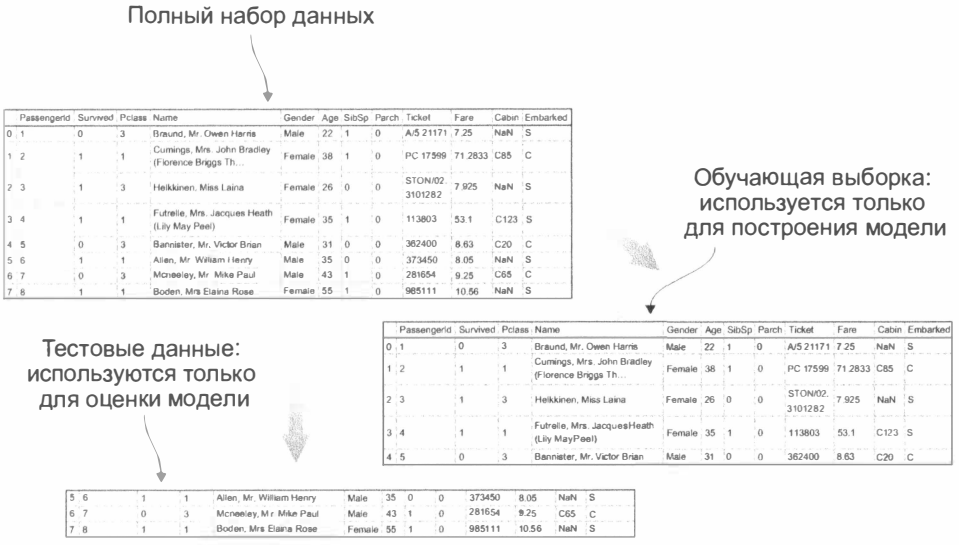
Илл. 4.9. Первые пять строк набора данных о пассажирах «Титаника». В целевом столбце указано, спасся пассажир во время крушения или погиб

Для построения классификатора нужно ввести данные из набора в соответствующий алгоритм. Наш набор содержит данные различных типов, и первым делом нужно убедиться, что выбранный алгоритм умеет работать со всеми этими типами. В предыдущих главах мы говорили о том, что иногда перед обучением модели требуется предварительная обработка данных, но сейчас мы будем рассматривать классификатор как черный ящик, сопоставляющий входные переменные целевой. Ведь целью данного раздела является оценка модели для оптимизации точности ее прогнозов и сравнения с другими моделями.

Поэтому мы сразу переходим к следующей задаче — перекрестной проверке. Полный набор нужно разделить на два подмножества: данные для обучения и данные для тестирования, — а затем воспользоваться методом отложенных данных. Для построения модели возьмем обучающий набор, в то время как ее оценка будет выполнена на отложенных данных. Еще раз подчеркнем, что важно добиться максимальной точности прогнозов модели не на обучающей выборке, а на новых, еще незнакомых ей данных. Но на этапе моделирования таких данных по определению быть

не может, поэтому мы делаем вид, что часть обучающих данных скрыта от алгоритма.

Иллюстрация 4.10 показывает этап разбиения данных в рассматриваемом примере.

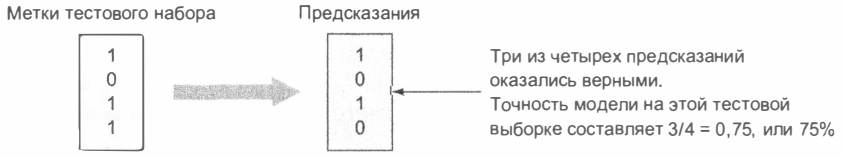


	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	Male	22	1	0	A/5 21171	7.25	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	Female	38	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss Laina	Female	26	0	0	STON/02. 3101282	7.925	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	Female	35	1	0	113803	53.1	C123	S
4	5	0	3	Bannister, Mr. Victor Brian	Male	31	0	0	362400	8.63	C20	C
5	6	1	1	Allen, Mr. William Henry	Male	35	0	0	373450	8.05	NaN	S
6	7	0	3	Mcneeley, Mr. Mike Paul	Male	43	1	0	281654	9.25	C65	C
7	8	1	1	Boden, Mrs Elaina Rose	Female	55	1	0	985111	10.56	NaN	S

Илл. 4.10. Разбиение полного набора данных на обучающий и тестовый для оценки точности работы модели

Теперь, когда у нас есть обучающая выборка, можно построить классификатор и сгенерировать предсказание на базе тестового набора данных. Воспользовавшись методом отложенных данных, схема которого показана на илл. 4.5, мы получим для всех строк тестового набора список предсказанных значений — 0 (погиб) или 1 (спасся). После этого, возвращаясь к этапу 3 процесса оценки, мы сравним эти значения с реальными данными о выживших в катастрофе и получим метрику производительности, доступную для дальнейшей оптимизации.

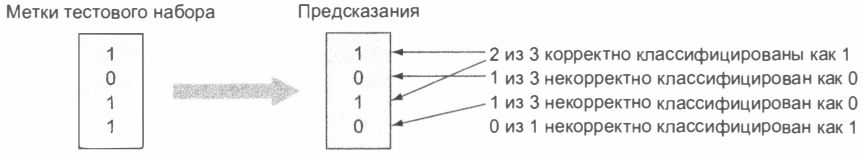
Простейший способ оценки моделей классификации — вычисление доли *правильных* ответов. Если правильно предсказанными оказались три из четырех строк, то можно сказать, что точность модели на конкретной контрольной выборке составляет 3/4 = 0,75, или 75%. На илл. 4.11 показан этот результат. В следующих разделах вы познакомитесь с более сложными способами оценки.



Илл. 4.11. Сравнение предсказаний для тестовой выборки с реальными значениями показывает точность модели

4.2.1. Точность для отдельных классов и таблица сопряженности

Нам доступна не только информация о корректности предсказаний, но и другие связанные с ними сведения. Например, можно проанализировать точность для каждого класса (сколько спасшихся согласно предсказанию на самом деле спаслись или погибли). В случае двоичной классификации можно ошибиться двумя способами: предсказать 0, когда корректное значение 1, или предсказать 1, когда корректное значение 0. Соответственно, угадать тоже можно двумя способами. Этот принцип демонстрирует илл. 4.12.



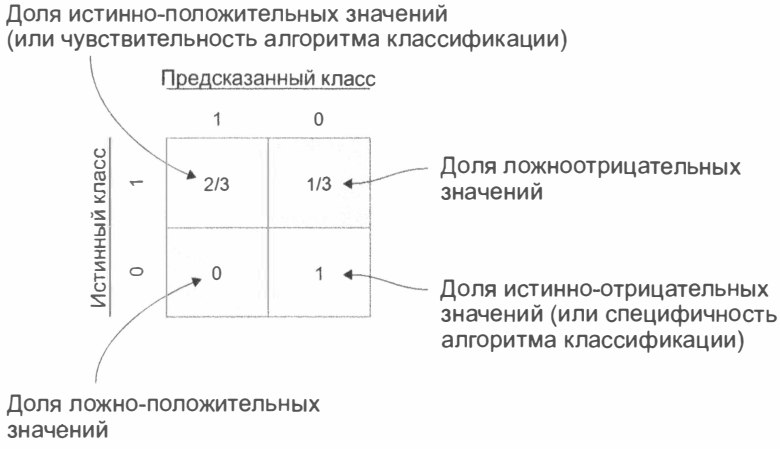
Илл. 4.12. Подсчет точности и вероятности ошибок для отдельных классов дает дополнительную информацию о производительности модели

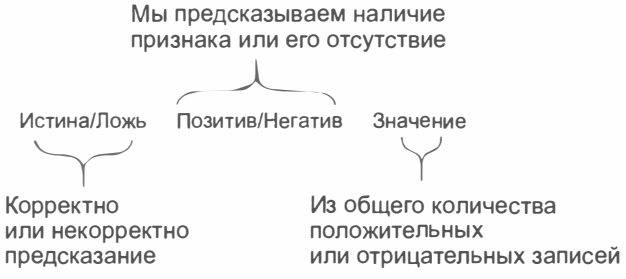
Во многих задачах классификации имеет смысл выход за пределы обычной оценки точности, когда рассматривается точность распределения по классам. Полученные при этом значения для наглядности представляют в виде показанной на илл. 4.13 диаграммы, которая называется *таблицей сопряженности* (confusion matrix).

Каждый элемент таблицы показывает правильность и ошибочность классификации несущих признак и не несущих такового объектов. Иллюстрация 4.14 сопоставляет таблице сопряженности с илл. 4.13 универсальную концепцию *рабочей характеристики приемника* (ROC –

Илл. 4.13. Точность распределения по классам в виде таблицы сопряженности

receiver operating characteristics), которой мы, начиная с этого момента, будем активно пользоваться. На первый взгляд эта новая терминология несколько сбивает с толку, но в будущем она пригодится вам при обсуждении производительности моделей с другими людьми.





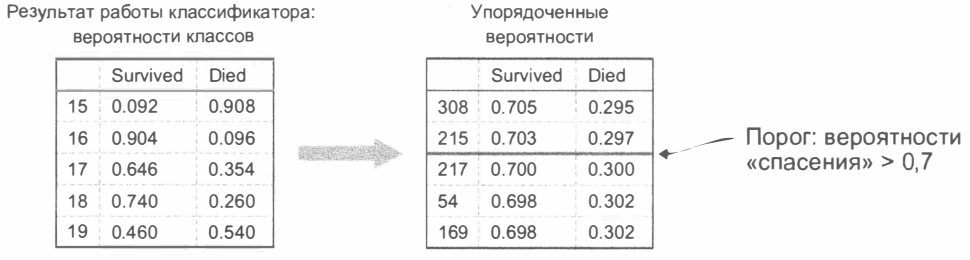
Илл. 4.14. Таблица сопряженности для двоичного классификатора, протестированная всего на четырех строках. Фигурирующая на этом рисунке ROC-кривая будет подробно разбираться ниже

4.2.2. Компромиссы при оценке точности и ROC-кривые

До этого момента мы рассматривали только прогнозы, в результате которых каждый объект попадал в конкретный класс. В примере с «Титаником» это было значение 1 при спасении и 0 в противном случае. Прогнозы, генерируемые моделями с машинным обучением, обычно содержат некую степень неопределенности, и многие алгоритмы классификации дают в результате не только нули и единицы, но и полное распределение *вероятностей*. Скажем, человек, которому наша модель предсказала *спасение* во время крушения «Титаника», на самом деле мог иметь шансы на спасение, равные 0,8, 0,99 или 0,5. Очевидно, что указанные ответы имеют разную степень достоверности, поэтому сейчас мы займемся более детальной оценкой наших моделей.

Результат работы *вероятностного классификатора* (probabilistic classifier) — это так называемые *вектора вероятностей* (probability vectors), или *вероятности классов*. Для каждой строки тестового набора мы получаем вещественное число в диапазоне от 0 до 1, характеризующее каждый класс нашего классификатора (в сумме эти числа дадут 1). До этого момента мы считали, что вероятности выше 0,5 определяют прогноз класса. Именно так вычислялись все оценки производительности в предыдущем разделе. То есть *порог* (threshold), определяющий класс, был равен 0,5. Очевидно, что можно выбрать любое другое значение, получив иные показатели для всех наших переменных.

Иллюстрация 4.15 демонстрирует процесс сортировки векторов вероятности и задания порогового значения 0,7. Все строки, оказавшиеся выше



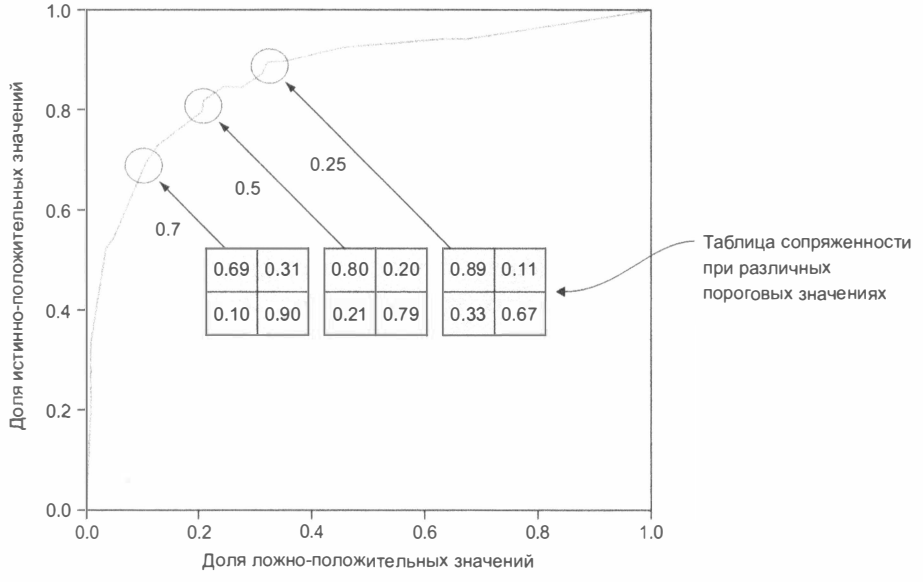
	Survived	Died
15	0.092	0.908
16	0.904	0.096
17	0.646	0.354
18	0.740	0.260
19	0.460	0.540

	Survived	Died
308	0.705	0.295
215	0.703	0.297
217	0.700	0.300
54	0.698	0.302
169	0.698	0.302

Илл. 4.15. Подмножество вероятностных предсказаний из тестового набора данных о пассажирах «Титаника». После сортировки таблицы по уменьшению вероятности выживания можно задать порог и считать, что все находящиеся выше строки относятся к выжившим. Обратите внимание, что индексы сохранены, так что мы легко можем определить, к какой исходной строке относится каждый экземпляр

пороговой линии, рассматриваются как выжившие, и их можно сравнить с реальными данными, чтобы получить таблицу сопряженности и ROC-показатели для указанного порогового значения. Повторив этот процесс для всех пороговых значений от 0 до 1, мы определим *ROC-кривую*, показанную на илл. 4.16.

Из илл. 4.16 выводятся таблицы сопряженности для всех пороговых значений, что делает ROC-кривую мощным инструментом визуализации при оценке производительности классификатора. Расчет ROC-кривой при наличии истинных и предсказанных любым методом скользящего контроля значений представлен в листинге 4.3.



Илл. 4.16. ROC-кривая, определенная путем расчета таблицы сопряженности и ROC-метрик при 100 пороговых значениях от 0 до 1. Долю ложно-положительных значений принято откладывать по оси x, в то время как доля истинно-положительных значений откладывается по оси y

По виду ROC-кривой понятно, что увеличение доли ложно-положительных значений означает уменьшение доли истинно-положительных. Как видите, в машинном обучении приходится идти на компромиссы. Можно пожертвовать долей экземпляров, классифицированных как истинные, ради большей уверенности в корректности предсказанных значений, и наоборот, в зависимости от того, каким образом вы выберете порог вероятности.

Листинг 4.3. ROC-кривая

```
import numpy as np

def roc_curve(true_labels, predicted_probs, n_points=100, pos_class=1):

    thr = np.linspace(0,1,n_points)
    tpr = np.zeros(n_points)
    fpr = np.zeros(n_points)

    pos = true_labels == pos_class
    neg = np.logical_not(pos)
    n_pos = np.count_nonzero(pos)
    n_neg = np.count_nonzero(neg)

    for i,t in enumerate(thr):
        tpr[i] = np.count_nonzero(np.logical_and(
                predicted_probs >= t, pos)) / n_pos
        fpr[i] = np.count_nonzero(np.logical_and(
                predicted_probs >= t, neg)) / n_neg
    return fpr, tpr, thr
```

Возвращает доли ложно-положительных и истинно-положительных значений при пороговом значении n_points для рассматриваемой пары истинное значение/предсказанное значение

Выделяет списки для пороговых значений и точек ROC-кривой

Вычисляет значения для положительного и отрицательного случаев, которые затем используются циклом

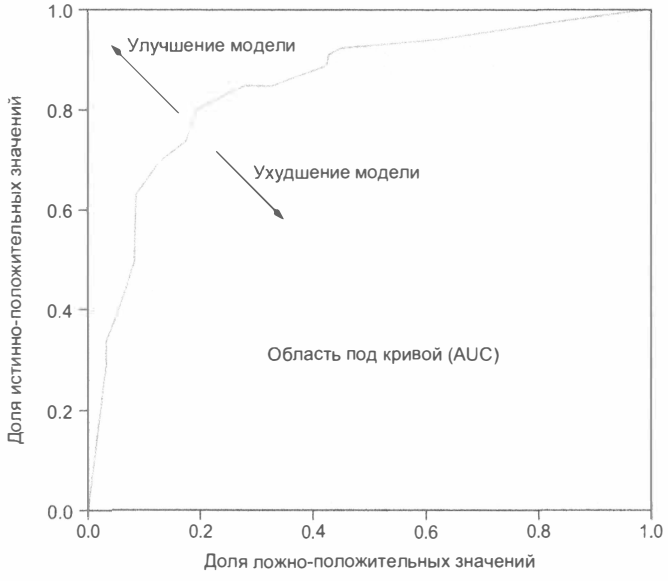
Для каждого порогового значения вычисляет долю истинных и ложных положительных значений

При решении реальных задач описанный выше выбор имеет большое значение. Прогнозируя рак у пациентов, лучше классифицировать несколько здоровых пациентов как больных, чтобы полностью избежать классификации больных пациентов как здоровых. Поэтому следует выбрать пороговое значение, которое минимизирует долю ложно-отрицательных значений и, соответственно, дает максимальную долю истинно-положительных значений, помещая нас как можно выше на ROC-кривой. При этом мы жертвуем долей ложно-положительных значений.

Еще один хороший пример — спам-фильтры. Здесь приходится выбирать между появлением нежелательной почты в почтовом ящике и отправкой нужной почты в папку со спамом. Или компании, занимающиеся обслуживанием кредитных карт. Что лучше — частые сообщения о ложных атаках, получаемые клиентами, или риск пропустить потенциально вредоносную транзакцию?

Кроме информации о компромиссных вариантах ROC-кривая дает представление об общей производительности классификатора. Совершенный классификатор не дает ложных положительных оценок и не имеет пропущенных обнаружений, то есть кривая сдвигается к верхнему левому углу, как показано на илл. 4.17. Это естественным образом наводит на мысль

об еще одной оценочной метрике — *площади под ROC-кривой* (AUC — area under curve). Чем больше площадь, тем выше производительность классификатора. Параметр AUC широко применяется для оценки и сравнения моделей, хотя в большинстве случаев важно рассмотреть и ROC-кривую, чтобы получить представление о компромиссах производительности. В дальнейшем для проверки моделей классификации мы будем использовать и ROC-кривую, и показатель AUC.



Илл. 4.17. ROC-кривая иллюстрирует общую производительность модели. Количественно ее можно выразить через показатель AUC — площадь области под ROC-кривой

Листинг 4.4. Область под ROC-кривой

```
from numpy import trapz

def auc(true_labels, predicted_labels, pos_class=1):
    fpr, tpr, thr = roc_curve(true_labels, predicted_labels,
                              pos_class=pos_class)
    area = -trapz(tpr, x=fpr)
    return area
```

Возвращает площадь области под ROC-кривой для реальных значений и соответствующих им значений, предсказанных моделью классификации

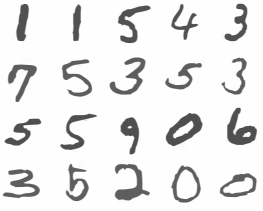
Доли ложно-положительных и истинно-положительных значений ROC-кривой из листинга 4.3

Интеграл, дающий площадь поверхности, вычисляется методом трапеций из библиотеки numpy

4.2.3. Многоклассовая классификация

Выше мы имели дело только с задачами двоичной классификации, но, к счастью, большинство описанных инструментов применимы и к задачам на распределение по более чем двум классам. Популярная задача многоклассовой классификации — распознавание написанных от руки цифр. Все мы время от времени отправляем обычные письма, и, скорее всего, в процессе определения конечного адреса участвует алгоритм машинного обучения. Если учесть, как коряво пишут некоторые люди, задача кажется непомерно сложной. Тем не менее подобные автоматизированные системы используются почтовыми работниками уже много десятилетий.

Успехи машинного обучения в деле распознавания рукописных цифр привели к тому, что оно описывается практически во всей литературе на данную тему как эталон производительности для многоклассовой классификации. Основная идея состоит в сканировании рукописных цифр и разделении их на изображения, каждое из которых содержит один символ. Затем в ход идут алгоритмы обработки изображений или встроенные многоклассовые классификаторы для необработанных пикселей различных оттенков серого, умеющие предсказывать цифры. Иллюстрация 4.18 демонстрирует примеры рукописных цифр из базы данных MNIST.

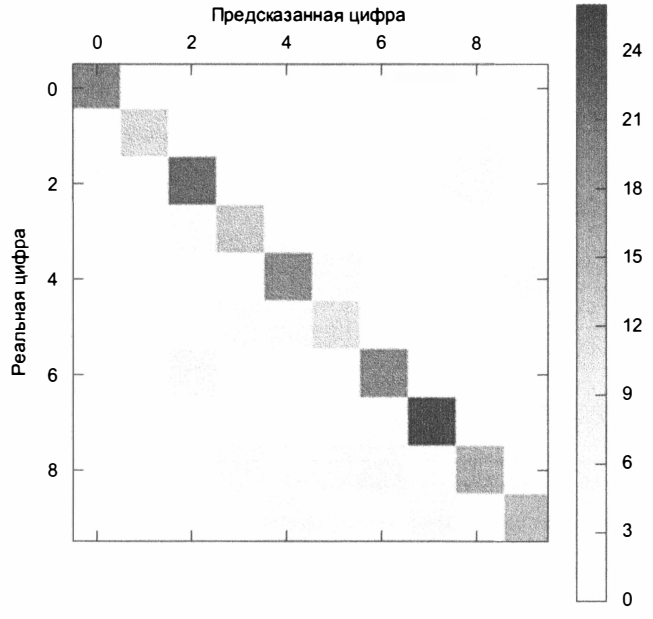


Илл. 4.18. Рукописные цифры в наборе данных MNIST. Этот набор состоит из 80 000 подобных изображений, каждое из которых имеет размер 28 × 28 пикселей. Без какой-либо обработки изображений каждая строка нашего набора данных состоит из известных меток (от 0 до 9) и 784 признаков (по одному на каждый из 28 × 28 пикселей)

Воспользуемся алгоритмом «случайный лес» (с ним вы познакомились в главе 3) и построим на базе обучающей выборки классификатор, после чего сгенерируем таблицу сопряженности из отложенных тестовых данных. До этого момента вы встречали эту таблицу только для двоичного классификатора. Но ничто не мешает определить ее для произвольного набора классов, так как строки таблицы сопряженности соответствуют

значениям одной переменной, а столбцы — значениям другой. Для классификатора, работающего с изображениями из базы MNIST, илл. 4.19 показывает, что максимальная производительность занимает диагональ таблицы. Это вполне логично, так как данная характеристика показывает количество экземпляров, *корректно* классифицированных для каждой цифры. Самые большие недиагональные элементы — места, в которых классификатор допустил максимальную ошибку. Изучив рисунок, вы заметите, что самая большая путаница возникает между цифрами 4 и 9, 3 и 5, а также 7 и 9, что вполне разумно, если посмотреть на форму цифр.

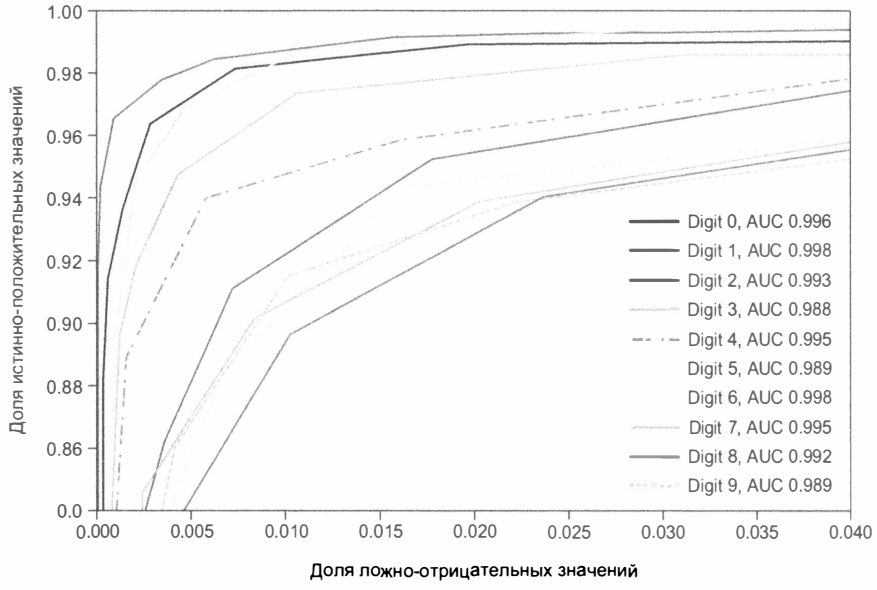
Мы представляем точность распределения по классам в виде таблицы, чтобы воспользоваться преимуществом визуальной обработки информации. Иллюстрация 4.19 хорошо показывает, как добавление в таблицу сопряженности контрастных элементов помогает задействовать это преимущество.



Илл. 4.19. Таблица сопряженности для 10-классовой задачи классификации рукописных цифр из базы данных MNIST

Как сгенерировать ROC-кривую для многоклассовых классификаторов? В своей основе эта кривая применима только к задачам двоичной классификации, ведь мы делим предсказания на *положительные* и *отрицатель-*

ные, чтобы получить такие параметры, как доля истинно-положительных и ложно-положительных значений, откладываемых вдоль координатных осей. Для имитации такой классификации в случае многоклассовой задачи воспользуемся приемом *один против всех*. Для каждого класса какой-то конкретный класс будет браться как *положительный*, в то время как все остальные станут *отрицательными*, после этого мы сможем нарисовать ROC-кривую обычным образом. Иллюстрация 4.20 показывает 10 ROC-кривых, полученных таким способом для классификатора цифр из базы MNIST. Точнее всего классифицированы цифры 0 и 1, что согласуется с информацией из показанной на илл. 4.19 таблицы сопряженности. Но эта таблица генерируется из наиболее вероятных предсказаний классов, в то время как ROC-кривая показывает производительность класса при всех пороговых значениях вероятности.



Илл. 4.20. ROC-кривые для всех классов 10-классового классификатора, полученные сопоставлением одного класса всем остальным для имитации двоичной задачи классификации. Так как наш классификатор дает хорошие прогнозы, мы сильно увеличили верхний угол ROC-кривой, чтобы посмотреть разницу в производительности модели для отдельных классов. Показатель AUC, вычисленный для каждого из классов, также показывает, что в целом модель функционирует хорошо

Но имейте в виду, что многоклассовые ROC-кривые не показывают всю таблицу сопряженности. В принципе каждой точке кривой соответствует

полная таблица 10 × 10, но простого способа ее визуализации не существует. Именно поэтому в случае множества классов важно рассматривать как таблицу сопряженности, так и ROC-кривую.

4.3. Оценка моделей регрессии

В предыдущих главах вы уже сталкивались с регрессионными моделями. В общем случае термин *регрессия* используется для моделей, результатом работы которых становятся целые или вещественные значения. Производительность моделей регрессии оценивается по другим показателям, о которых мы сейчас и поговорим.

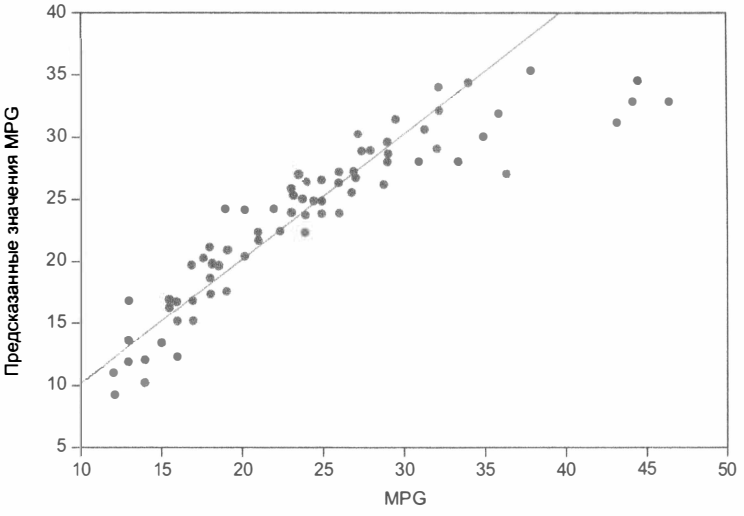
В качестве рабочего примера рассмотрим уже знакомый по предыдущим главам набор данных Auto MPG. Фрагмент данных из этого набора показан на илл. 4.21. Требуется произвести все необходимые преобразования (они рассматривались в разделе 2.2) и выбрать подходящую модель, как мы это делали в главах 2 и 3. Но сейчас нас больше всего интересует измерение эффективности этой модели.

Целевая переменная

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model/year	Origin
0	18	8	307	130	3504	12.0	70	1
1	15	8	350	165	3693	11.5	70	1
2	18	8	318	150	3436	11.0	70	1
3	16	8	304	150	3433	12.0	70	1
4	17	8	302	140	3449	10.5	70	1

Илл. 4.21. Подмножество набора данных Auto MPG

Воспользуемся описанным в начале этой главы базовым процессом оценки модели и построим модель регрессии с перекрестной проверкой. Показатели потенциальной производительности модели будут рассматриваться в следующем разделе, а базовая визуализация производительности регрессии, на которой основаны эти показатели, показана на илл. 4.22. Мы видим диаграмму рассеяния, сравнивающую предсказанные значения с истинными.



Илл. 4.22. Диаграмма рассеяния, сопоставляющая предсказанные на обучающей выборке значения MPG с реальными. Диагональная линия обозначает оптимальную модель

4.3.1. Простые показатели эффективности регрессионных моделей

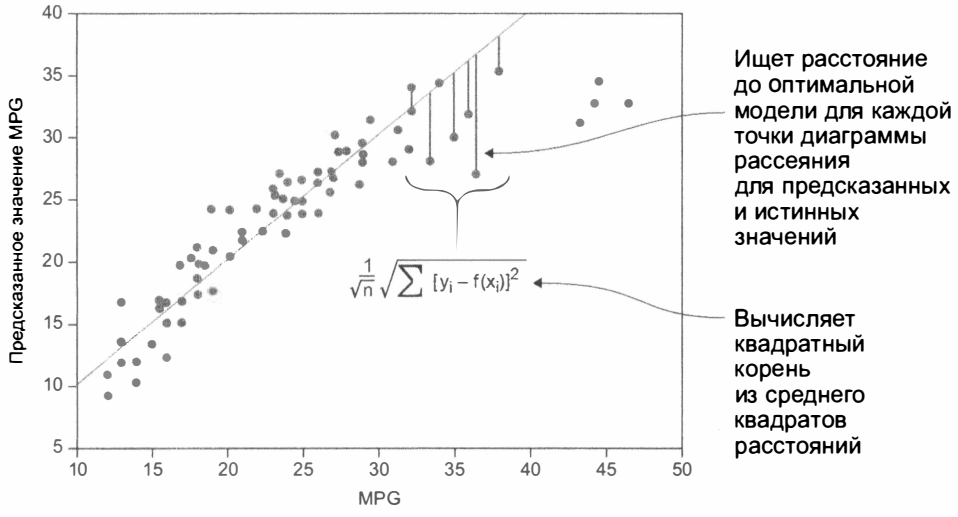
В отличие от моделей классификации, для моделей регрессии не существует понятия *корректного* предсказания. Вообще, численный прогноз практически никогда точно не совпадает с истинным значением, но может быть *близким* к нему или *далеким* от него. Это следует и из природы того, что мы подразумеваем под корректным значением, так как обычно численные измерения мы получаем из распределения с некоторой степенью неопределенности — мы называем ее *ошибкой*. Есть два простых показателя для измерения производительности регрессии: квадратный корень из среднеквадратичной ошибки и коэффициент детерминации.

Простейшей формой измерения производительности модели регрессии является *квадратный корень из среднеквадратичной ошибки* (RMSE — root-mean-square error). Этот алгоритм оценки рассматривает разницу всех предсказанных и соответствующих им известных значений и вычисляет среднее способом, который позволяет пренебречь тем фактом, что каждое из предсказанных значений может быть как больше, так и меньше реального. Процесс вычисления RMSE показан на илл. 4.23.

Чтобы представить процесс расчета RMSE более наглядно, приведем следующий фрагмент кода.

Листинг 4.5. Квадратный корень из среднеквадратичной ошибки

```
def rmse(true_values, predicted_values):
    n = len(true_values)
    residuals = 0
    for i in range(n):
        residuals += (true_values[i] - predicted_values[i])**2.
    return np.sqrt(residuals/n)
```



Илл. 4.23. Вычисление RMSE: в уравнении y_i и x_i это i-й вектора целевых переменных и признаков соответственно. Функция f(x) обозначает применение модели к вектору признаков, результатом чего становится предсказанное целевое значение

Преимущество RMSE состоит в том, что результат мы получаем в тех же единицах, что и сами значения. Одновременно это является и недостатком в том смысле, что полученное значение зависит от масштаба параметров конкретной задачи, соответственно уменьшается сопоставимость различных наборов данных. Если предсказанные или реальные значения велики, выше окажется и RMSE. Это не проблема, когда модели сравниваются в рамках одного проекта, а вот оценить общую производительность модели и сопоставить ее с другими моделями становится сложнее.

Чтобы выйти из положения, зачастую целесообразно вычислить также так называемый показатель R-квадрат, или коэффициент детерминации, ко-

торый всегда относителен и всегда лежит в диапазоне от 0 до 1. Чем лучше модель умеет предсказывать данные, тем ближе значение этого коэффициента к 1. Подробно процесс его вычисления показан в следующем листинге.

Листинг 4.6. Вычисление коэффициента детерминации

```
def r2(true_values, predicted_values):
    n = len(true_values)
    mean = np.mean(true_values)
    residuals = 0
    total = 0
    for i in range(n):
        residuals += (true_values[i] - predicted_values[i])**2.
        total += (true_values[i] - mean)**2.
    return 1.0 - residuals/total
```

Но какой бы параметр оценки вы ни использовали — MSE, RMSE или R^2, — надо иметь в виду следующее:

- Всегда пользуйтесь перекрестной проверкой для оценки модели. В противном случае вышеупомянутые показатели всегда будут улучшаться по мере усложнения модели, приводя к переобучению.
- Параметры оценки следует по возможности приводить в соответствие решаемым задачам. Например, при предсказании MPG по набору характеристик автомобиля RMSE, равное 5, означает, что ожидаемое отличие среднего предсказания от истинного значения составляет 5 миль на галлон.

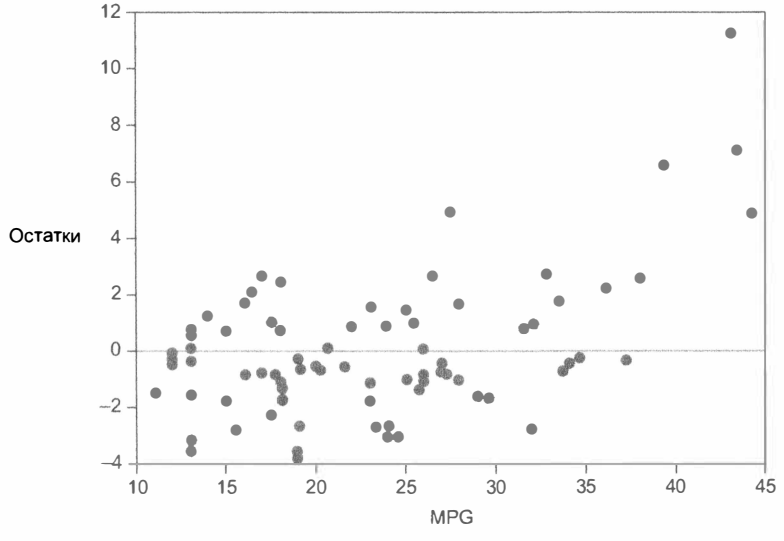
Кроме того, для задач регрессии применяется множество других параметров оценки, многие из которых имеют встроенные средства снижения переобучения (и, соответственно, не требуют перекрестной проверки). В качестве примеров можно привести информационный критерий Акаике (AIC — Akaikie information criterion) и Байесовский информационный критерий (BIC — Bayesian information criterion). Информация об этих и других усовершенствованных методах есть в большинстве учебников по регрессионному анализу.

4.3.2. Исследование остатков

В предыдущем разделе вы видели, как остатки — расстояние между предсказанным и реальным значениями — использовались в двух простых

параметрах оценки. Не менее интересные результаты дает их визуальный анализ.

Иллюстрация 4.24 демонстрирует графическое представление остатков для нашего набора данных по расходу топлива. Фактически это та же самая информация, что и на диаграмме рассеяния с илл. 4.23, но увеличенная до масштаба остатков. В идеальном случае ожидается, что они будут случайным образом распределены вокруг линии с нулевой координатой. В нижней части графика для значений MPG от 10 до 35 действительно создается впечатление такого распределения, возможно, с небольшим смещением в сторону завышенных значений. Но в диапазоне 35–45 мы видим четкое смещение в сторону заниженных значений, что даст большие величины остатков. Эта информация может быть использована для улучшения модели путем подбора ее параметров, обработкой или дополнением данных. Если существует возможность получить дополнительные данные, можно попробовать узнать значения для более низких показателей расхода. В рассматриваемом случае можно найти еще несколько автомобилей с высоким значением MPG и добавить сведения о них в наш набор, чтобы улучшить предсказания в этой части шкалы.



Илл. 4.24. Остатки для предсказаний на основе данных о MPG. Горизонтальная линия с координатой 0 соответствует нулевому значению остатков

Итак, вы узнали, как протестировать модель методами скользящего контроля и познакомились с некоторыми метриками оценки производительности. Для простейших моделей все сводится к обучению, тестированию и вычислению подходящих показателей производительности. Более сложные алгоритмы обладают параметрами настройки — ручками, которые может покрутить пользователь, — влияющими на процессы обучения и применения. Каждая комбинация параметров дает свой режим. В следующем разделе вы увидите, как иногда небольшая регулировка коренным образом меняет результаты.

4.4. Оптимизация модели путем подбора параметров

Большинство моделей с машинным обучением оснащены одним или несколькими *параметрами настройки* (tuning parameters), которые контролируют внутреннюю работу обучающего алгоритма. Как правило, эти параметры отвечают за сложность взаимосвязи между входными признаками и целевой переменной. Поэтому они сильно влияют на обученную модель и точность прогнозирования на новых данных.

К примеру, в разделе 4.1 вы видели, как всего один параметр настройки (ширина окна в алгоритме регрессии с ядерным сглаживанием) дает колоссальную разницу в обучении модели, предсказывающей урожаи кукурузы. При малой ширине окна функция регрессии оказалась чрезмерно бугристой и слишком точно аппроксимировала данные. А для широкого окна мы получили слишком сглаженную функцию, которая не замечала закономерностей, присутствующих в данных.

В этом разделе мы опишем строгую методологию оптимизации ML-моделей путем настройки параметров алгоритма машинного обучения.

4.4.1. Параметры настройки ML-алгоритмов

Каждый алгоритм машинного обучения имеет набор параметров, определяющих, как именно он строит модель на обучающей выборке. По мере усложнения алгоритмов умножаются и становятся более сложными и их параметры настройки. Вот упорядоченный по возрастанию сложности

список параметров для некоторых популярных алгоритмов классификации, с которыми вы познакомились в главе 3:

- *логистическая регрессия* — отсутствует;
- *метод k-ближайших соседей* — количество ближайших к среднему соседей;
- *деревья решений* — критерий разбиения, максимальная глубина дерева, минимальное количество экземпляров, необходимое для разбиения;
- *метод опорных векторов с ядерным сглаживанием* — тип ядра, коэффициент ядра, штрафной параметр;
- *«случайный лес»* — количество деревьев, количество признаков для разбиения в каждом узле, критерий разбиения, минимальное количество экземпляров, необходимое для разбиения;
- *бустинг* — количество деревьев, скорость обучения, максимальная глубина дерева, критерий разбиения, минимальное количество экземпляров, необходимое для разбиения.

В качестве примера вспомним главу 3, где мы применяли метод опорных векторов с ядерным сглаживанием к набору сведений о пассажирах «Титаника». На илл. 3.8 и 3.9 показаны два варианта обучения модели с разными значениями коэффициента ядра (он обозначался γ). Обратите внимание на разницу в результатах: значение $\gamma = 0{,}01$ давало сложную, сегментированную решающую границу между двумя классами, в то время как при $\gamma = 0{,}1$ получалась более гладкая модель. В данном случае обученная модель крайне чувствительна к выбору значения параметра настройки γ.

Трудность состоит в том, что корректный выбор каждого параметра полностью зависит от данных и поставленной задачи. То, что прекрасно работает в одном случае, может оказаться совершенно некорректным в другом. Эвристические, подобранные эмпирическим путем исходные значения параметров алгоритма с большой вероятностью дадут низкую прогностическую эффективность. Именно строгий выбор параметров настройки гарантирует, что модель обеспечит максимально возможную для используемых данных точность.

4.4.2. Сеточный поиск

Стандартный способ оптимизации параметров настройки в ML-моделях сводится к *сеточному поиску* (grid search) путем перебора. Когда будете использовать показанный ниже базовый алгоритм сеточного поиска, обратите внимание, что описанная стратегия связывает воедино материал по методам скользящего контроля и оценке моделей, рассмотренный в предыдущих разделах:

1. Выберите параметр оценки, который будет использоваться как основной (например, AUC для задачи классификации, R^2 для задачи регрессии).

2. Выберите ML-алгоритм (например, «случайный лес»).

3. Выберите параметры, которые собираетесь оптимизировать (например, количество деревьев и количество признаков на одно разбиение), и массив значений для тестирования каждого параметра.

4. Определите сетку как прямое произведение массивов всех параметров настройки. К примеру, если для количества деревьев у нас есть массив [50, 100, 1000], а для количества признаков на одно разбиение — массив [10, 15], сетка будет выглядеть так [(50,10), (50,15), (100,10), (100,15), (1000,10), (1000,15)].

5. Для каждой комбинации настроечных параметров в сетке используйте обучающую выборку для проведения перекрестной проверки (или методом отложенных данных, или методом контроля по k-блокам) и рассчитайте оценочную метрику для полученных предсказаний.

6. Наконец, выберите набор параметров настройки, соответствующий максимальному значению оценочного показателя. Это и есть оптимизированная модель.

Почему это работает? Мы выполняем широкий поиск по возможным комбинациям значений каждого из параметров настройки. Для каждой комбинации оценивается производительность модели на новых данных путем сравнения методом перекрестной проверки предсказанных и ре-

альных значений переменной. Затем выбирается модель с самой высокой оценкой точности (на новых данных). Именно она с максимальной вероятностью даст самый верный прогноз.

Применим сеточный поиск к набору сведений о пассажирах «Титаника». В качестве показателя оптимизации возьмем AUC, а в качестве алгоритма классификации — метод опорных векторов с ядром в виде радиальной базисной функции (RBF — radial basis function). В принципе сеточным поиском можно воспользоваться и для выбора самого лучшего ядра. Более того, он позволяет выбирать и между различными алгоритмами!

Теперь выберем, какие параметры будут оптимизироваться. Для алгоритма SVM с радиальной базисной функцией в роли ядра существует два стандартных параметра настройки: коэффициент ядра γ и штрафной параметр C. Представленный ниже листинг проводит сеточный поиск для указанных параметров.

Листинг 4.7. Сеточный поиск для метода опорных векторов с ядерным сглаживанием

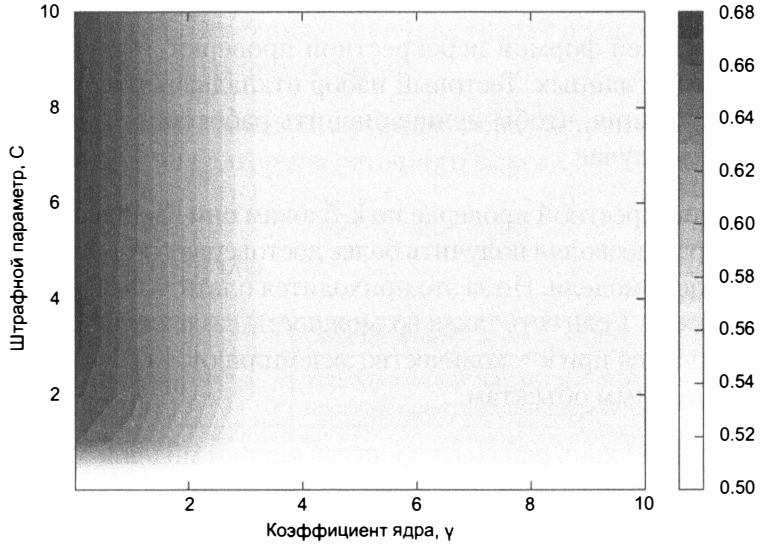
```
# Входные данные: X - признаки, y - цель

import numpy as np
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

# сетка (gamma, C) значений для проверки
gam_vec, cost_vec = np.meshgrid(np.linspace(0.01, 10., 11),
                    np.linspace(1., 10., 11))

AUC_all = []      <-- Инициализирует пустой массив для сохранения результатов AUC

# задает блоки перекрестной проверки
N = len(y)
K = 10     <-- Количество блоков для перекрестной проверки
folds = np.random.randint(0, K, size=N)

# поиск по каждому значению сетки
for param_ind in np.arange(len(gam_vec.ravel())):

    # инициализация перекрестной проверки предсказаний
```

```
    y_cv_pred = np.empty(N)

    # цикл для блоков перекрестной проверки
    for ii in np.arange(K):
        # разбиение данных на обучающую и тестовую выборки
        X_train = X.ix[folds != ii,:]
        y_train = y.ix[folds != ii]
        X_test = X.ix[folds == ii,:]

        # построение модели на обучающей выборке
        model = SVC(gamma=gam_vec.ravel()[param_ind],
    C=cost_vec.ravel()[param_ind])
        model.fit(X_train, y_train)

        # генерация и сохранение предсказаний на тестовых данных
        y_cv_pred[folds == ii] = model.predict(X_test)

    # оцениваем AUC предсказаний
    AUC_all.append(roc_auc_score(y, y_cv_pred))

indmax = np.argmax(AUC_all)
print "Maximum = %.3f" % (np.max(AUC_all))
print "Tuning Parameters: (gamma = %f, C = %f)" % (gam_vec.ravel()[indmax],
cost_vec.ravel()[indmax])
```

Оказывается, для набора данных о пассажирах «Титаника» максимальное значение AUC, полученное методом перекрестной проверки, составляет 0,670. И появляется оно при векторе параметров настройки ($\gamma = 0{,}01$, $C = 6$). Представленная на илл. 4.25 контурная диаграмма с оцененными по всей сетке значениями AUC дает следующую информацию:

- максимум возникает на границе сетки ($\gamma = 0{,}01$), что означает желательность повторного поиска уже на расширенной сетке;
- численное предсказание параметра γ выполнено с высокой точностью. Это значит, что нужно увеличить детальность выборки для данного параметра;
- максимальное значение появляется рядом с $\gamma = 0$, поэтому имеет смысл выразить сетку в логарифмическом масштабе (например, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1});
- показатель AUC обнаруживает не очень сильную зависимость от C, поэтому можно использовать грубую выборку этого параметра.

Повторно запустив поиск на модифицированной сетке, мы обнаружим, что максимальное значение AUC составляет 0,690 и появляется при (γ = 0,08, C = 20). Смысл оптимизации путем регулировки параметров показан достаточно наглядно: модель со случайно выбранными параметрами настройки позволяет получить неэффективный результат с AUC = 0,5 (по сути, это случайное гадание); точность оптимизированной путем сеточного поиска модели возрастает до AUC = 0,69.



Илл. 4.25. Контурная диаграмма для полученного методом перекрестной проверки значения AUC как функции от двух параметров настройки — γ и C. Максимум возникает близко к верхнему левому углу, что указывает на необходимость расширить поиск и сфокусироваться на этой области

Следует заметить, что сеточный поиск не гарантирует, что мы получим наилучший набор параметров настройки. Существуют ограничения, связанные с конечностью сетки возможных значений. И фактическое наилучшее значение может оказаться между узлами сетки. Читатели, имеющие представление о процессе оптимизации, могут спросить, почему для выбора параметров настройки не применяются более сложные процедуры.

К сожалению, в настоящее время эти методы только начинают добавлять в стратегии оптимизации параметров.

4.5. Заключение

В этой главе вы изучили основные принципы оценки производительности ML-моделей, вот список основных положений:

- Использовать обучающую выборку два раза — сначала для обучения, потом для оценки модели — недопустимо.
- Более надежным методом оценки является перекрестная проверка.
- Простейшей формой перекрестной проверки является метод отложенных данных. Тестовый набор откладывается для этапа прогнозирования, чтобы лучше оценить работоспособность модели в общем случае.
- При перекрестной проверке по k-блокам они выделяются по одному за раз, позволяя получить более достоверную оценку производительности модели. Но за это приходится платить вычислительными ресурсами. Если есть такая возможность, самая лучшая оценка рассчитывается при k = количество экземпляров, то есть при контроле по отдельным объектам.
- Вот общая схема рабочего процесса оценки модели:

    1. Получение и предварительная обработка данных для моделирования (глава 2) и выбор подходящего ML-метода и алгоритма (глава 3).
    2. Построение модели и генерация предсказаний или методом отложенных данных, или путем скользящего контроля по k-блокам, в зависимости от доступных вычислительных ресурсов.
    3. Оценка точности предсказаний по выбранному вами критерию в зависимости от того, какой ML-метод используется — классификации или регрессии.
    4. Регулировка данных и модели до достижения нужной производительности. В главах 5–8 вы познакомитесь с методами увеличения эффективности для распространенных практических сценариев.

- Для моделей классификации мы рассмотрели несколько показателей производительности, которые могут использоваться на этапе 3 рабочего процесса. В эту категорию попадает как простой расчет точности, так и таблица сопряженности, ROC-кривая и область под этой кривой.
- Для моделей регрессии были описаны такие способы оценки, как квадратный корень из среднеквадратичной ошибки и коэффициент детерминации. Эффективны также простые визуализации, такие как диаграмма рассеяния, сопоставляющая предсказанные значения реальным, и график остатков.
- Для оптимизации моделей путем подбора настраиваемых параметров применяется алгоритм сеточного поиска.

4.6. Терминология

Термин	Определение
Недообучение/переобучение (underfitting/overfitting)	Использование для решения задачи слишком простой или слишком сложной модели
Метрика оценки (evaluation metric)	Число, характеризующее производительность модели
Среднеквадратичная ошибка (mean squared error)	Особая метрика оценки, используемая для моделей регрессии
Скользящий контроль (cross-validation)	Метод разбиения обучающей выборки на два или более обучающих/тестовых подмножества для лучшей оценки точности
Метод отложенных данных (holdout method)	Форма скользящего контроля, в которой выделяется одно тестовое подмножество данных, не принимающее участия в обучении модели
Контроль по k-блокам (k-fold cross-validation)	Вид скользящего контроля, при котором данные разбиваются на k случайных, не связанных друг с другом наборов (блоки). Блоки откладываются по одному за раз и применяются для скользящего контроля моделей, построенных на остальных данных
Таблица сопряженности (confusion matrix)	Матрица, показывающая для каждого класса количество предсказанных значений, которые были корректно или некорректно классифицированы

Термин	Определение
Рабочая характеристика приемника (receiver operating characteristic — ROC)	Число, представляющее истинно-положительные, ложно-положительные, истинно-отрицательные или ложно-отрицательные значения
Площадь под ROC-кривой (area under the ROC curve — AUC)	Оценочная метрика для задач классификации, определенная как площадь, ограниченная ROC-кривой и осью доли ложных положительных классификаций
Параметр настройки (tuning parameter)	Внутренний параметр алгоритма машинного обучения, такой как, к примеру, ширина окна сглаживания для регрессии с ядрами сглаживания
Сеточный поиск (grid search)	Стратегия перебора, позволяющая выбрать наилучшие параметры настройки для оптимизации ML-модели

В следующей главе мы поговорим о том, как улучшить модель с помощью входных признаков. Кроме базовых техник проектирования признаков будут рассмотрены усовершенствованные методы извлечения информации из текста, изображений и временных рядов данных. Также вы узнаете, как выбирать признаки, позволяющие оптимизировать производительность модели и избежать переобучения.

5 Основы проектирования признаков

В этой главе:

- ✓ зачем нужно проектирование признаков;
- ✓ приемы базового проектирования признаков, к которым в числе прочего относится обработка дат, времени и простого текста;
- ✓ выбор оптимальных признаков и уменьшение статистической и вычислительной сложности моделей;
- ✓ проектирование признаков на стадии построения модели и во время генерации прогнозов.

В первых четырех главах рассказывалось о том, как обучить, оценить и оптимизировать алгоритм машинного обучения с учителем при наличии набора входных признаков и целевой переменной. Но откуда берутся входные признаки? Как их определить и рассчитать? И как узнать, правильный ли набор признаков вы выбрали для решения поставленной задачи?

5.1. Мотивация: в чем польза проектирования признаков?

В этой главе мы научим вас создавать признаки из необработанных данных — этот процесс называют *проектированием признаков* (feature engineering) — и покажем несколько простых практических примеров. Это подготовит почву для более сложных алгоритмов проектирования, которые будут рассматриваться в главе 7.

5.1.1. Что такое проектирование признаков?

Проектированием признаков называется применение математических преобразований к необработанным входным данным с целью получения новых признаков для ML-модели. Вот примеры таких преобразований:

- деление общей суммы на количество платежей для получения доли, приходящейся на один платеж;
- подсчет числа появлений конкретного слова в тексте;
- расчет статистических сводок (таких, как среднее, медиана, стандартное отклонение и асимметрия) распределения времени пользовательских запросов с целью оценки состояния Сети;
- объединение двух таблиц (например, платежей и поддержки) по идентификатору пользователя;
- применение сложных инструментов обработки сигналов к изображениям и обобщение полученных результатов (например, гистограммы градиентов).

Но перед тем как углубиться в примеры, на практике демонстрирующие процесс проектирования признаков, ответим на простой вопрос: зачем это вообще нужно?

5.1.2. Пять причин проектирования признаков

Разберем несколько примеров, демонстрирующих важность проектирования признаков для приложений с машинным обучением. Этот список не является исчерпывающим, а скорее позволяет увидеть, как проектирование признаков позитивно влияет на точность и вычислительную эффективность ML-моделей.

Соотнесение исходных данных с целевой переменной

Иногда имеет смысл преобразовать исходные данные, добиваясь более близкой связи с целевой переменной. Возьмем, к примеру, данные о личных счетах клиентов, а именно текущий баланс банковского счета и долг по кредитной карте. При построении модели, предсказывающей, просрочит ли конкретный клиент платежи в ближайшие три месяца, новый признак

```
Соотношение долга к балансу = сумма долга / сумма на счету
```

скорее всего, позволит с более высокой уверенностью спрогнозировать целевую переменную.

В данном случае необработанные входные данные присутствуют в исходном наборе, но ML-модели будет проще найти взаимосвязь между соотношением сумм долга и баланса и будущими задержками платежей, если в качестве входного будет использоваться именно спроектированный признак. Это увеличит точность прогнозирования.

Добавление внешних источников данных

Проектирование признаков позволяет добавлять к ML-моделям внешние источники данных. Представьте, что вы руководите службой подписки и хотите при первой авторизации каждого нового клиента спрогнозировать время его пользования вашими услугами. В такой ситуации доступен всего один параметр — географическое положение клиента. Эта информация может использоваться непосредственно как категориальный признак (например, в виде IP-адреса или почтового индекса), но, скорее всего, модели будет сложно распознать значимые сигналы (а в качестве таковых в рассматриваемом случае может выступать средний доход для каждого местоположения или противопоставление города и сельской местности).

Добиться лучших результатов помогут сторонние демографические данные. Например, вы сможете рассчитать средний доход и плотность населения для каждого местоположения пользователей и вставить эти факторы непосредственно в обучающую выборку. То есть вместо того чтобы ждать, пока модель сама заметит достаточно слабую зависимость в необработанных данных о месте жительства клиентов, мы предоставляем ей сведения, упрощающие вывод. Кроме того, преобразовывая информацию о местоположении в данные о доходах и населении, вы получаете возможность оценить, какая из этих производных исходного признака оказывает большее влияние.

Использование источников неструктурированных данных

Проектирование признаков позволяет использовать в ML-моделях источники неструктурированных данных. Большинство источников по своей природе не в состоянии генерировать вектора признаков, сразу доступные для вставки в ML-фреймворки. В основном приходится иметь дело с такими данными, как текст, временны́е ряды, изображения, видео, сведения из журналов событий и истории посещения сайтов. И именно проектирование признаков позволяет преобразовывать все эти потоки данных в доступные для ML-алгоритмов вектора.

Сейчас мы коснемся только относительно простых примеров проектирования признаков из текстовых данных. Более сложные типы проектирования для текста, изображений и временных рядов будут рассматриваться в следующих главах.

Создание проще интерпретируемых признаков

Проектирование позволяет получать более интерпретируемые и действенные признаки. Часто применение машинного обучения для поиска шаблонов в данных позволяет делать предсказания, но ограничения в интерпретируемости модели уменьшают ее итоговую полезность и подвигают к изменениям. В таких случаях имеет смысл проектирование новых признаков, более точно характеризующих как процессы генерации данных, так и связь между необработанными данными и целевой переменной.

В качестве простого примера рассмотрим станки, производящие аппаратное обеспечение для компьютеров. Можно взять необработанные данные с этих станков, такие как результаты измерений сигнала и отклика и дру-

гих сигналов, связанных с процессом производства, и построить ML-модели, прогнозирующие отказ компонентов. Но только такие признаки, как прошедшее с момента последней регулировки станка время и объем производимой продукции, дадут истинное представление о динамических аспектах процесса производства.

Повышение потенциала информации с помощью больших наборов признаков

Проектирование дает возможность формировать большие наборы признаков и смотреть, какие из них лучше всего подходят к модели. Можно создать произвольное количество признаков, обучить на них модель и отобрать обладающие максимальной прогностической эффективностью. Это позволяет избежать косности мышления на этапе создания и тестирования признаков, а также может показать новые тенденции и шаблоны в данных.

При обучении ML-моделей на десятках или даже сотнях признаков возникает опасность переобучения, но строгие алгоритмы отбора позволяют уменьшить их количество. Например, вы можете автоматически определить, что предсказания по верхним 10 признакам настолько же хороши или даже лучше предсказаний по всей тысяче признаков. Эти алгоритмы описаны в разделе 5.3.

5.1.3. Проектирование признаков и знание предметной области

Проектирование признаков можно также представить как механизм, преобразующий опыт в конкретной области в модель с машинным обучением. Все очень просто: для каждой задачи сведения о данных и изучаемых системах накапливаются в течение некоторого времени. Иногда шаблоны достаточно очевидны и легко выявляются ML-моделью. Но в более сложных случаях эффективность моделей значительно возрастает, если закодировать вышеуказанный опыт в наборе признаков. Вот примеры обобщающих опыт формулировок, которые можно легко представить в виде ML-признаков:

- по вторникам уровень конверсии на сайте всегда выше (включает булевский признак «Сегодня вторник?»);

- потребляемая в бытовых условиях мощность увеличивается при более высокой температуре (включает температуру как признак);
- спам, как правило, приходит с бесплатных почтовых сервисов (проектируется булевский признак «Письмо с бесплатного сервиса?»);
- владельцы недавно открытых кредитных карт чаще отказываются от долговых обязательств (используется признак «Количество дней с открытия последней кредитной карты»);
- клиенты часто меняют поставщика услуг сотовой связи после того, как другие пользователи того же оператора переходят к другому провайдеру (проектируется признак, подсчитывающий количество абонентов сети, которые недавно ушли к конкурентам).

Понятно, что этот список можно легко продолжить. И в работе многих компаний длинные списки таких ситуативных правил применяются для принятия решений и прогнозирования. Подобные бизнес-правила являются идеальным набором спроектированных признаков, которые могут стать основой для построения ML-моделей!

С другой стороны, проектирование признаков может применяться и для *проверки* стереотипов, возникающих на базе рабочего опыта. Чтобы узнать, заслуживает ли гипотеза рассмотрения, достаточно использовать ее как признак в ML-модели. После чего точность работы модели проверяется с этим признаком и без него, чтобы оценить степень его влияния на прогнозирование целевой переменной. Если сильного приращения точности не происходит, очевидно, что гипотезу можно отбросить.

Перейдем к примерам проектирования признаков, демонстрирующим, как все это выглядит на практике. Мы опишем, как проектирование вписывается в общий рабочий процесс машинного обучения и как оно влияет на прогностическую точность ML-моделей.

5.2. Основные этапы проектирования признаков

Перед рассмотрением примеров еще раз вспомним диаграмму рабочего процесса машинного обучения и посмотрим, как ее расширяет проектирование признаков. Диаграмма представлена на илл. 5.1.

Илл. 5.1. Проектирование признаков как часть базового рабочего процесса ML. Перед построением модели мы добавляем в обучающую выборку новые признаки. На стадии прогнозирования нужно пропустить новые данные через тот же самый процесс проектирования признаков, чтобы убедиться в осмысленности ответов

Расширение рабочего процесса за счет проектирования признаков позволяет увеличить обучающую выборку и повысить точность работы ML-алгоритма. Чтобы убедиться в корректности новых признаков, предсказанные данные следует прогнать через процедуру проектирования, которая применялась к обучающей выборке. Этим вы гарантируете, что для генерации предсказаний будет применен тот же самый процесс, который был получен путем обучения.

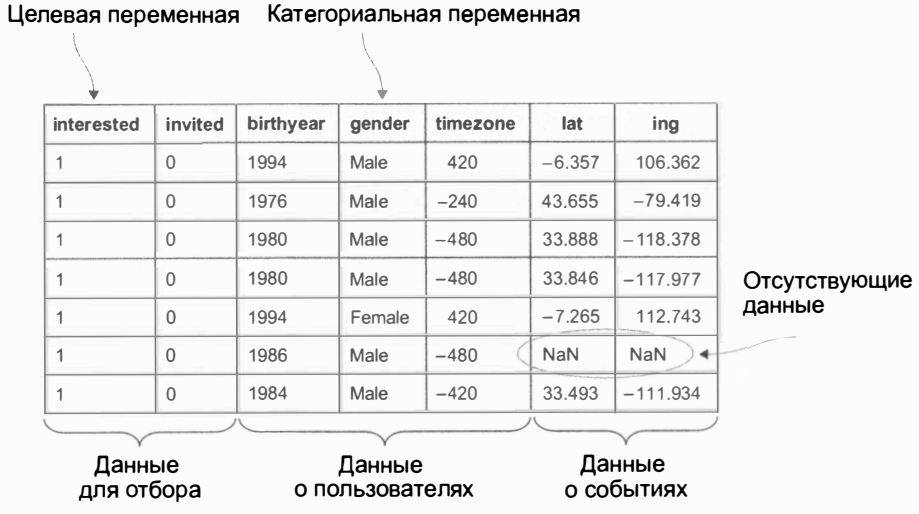
5.2.1. Пример: рекомендация события

Чтобы проиллюстрировать концепцию проектирования признаков, рассмотрим конкурсное задание с сайта Kaggle (*www.kaggle.com*).

Представьте, что вы поддерживаете сайт, рекомендующий различные события, и хотите заранее предсказывать, насколько некое событие

(встреча, неформальное общение или лекция) заинтересует конкретного пользователя. Есть набор данных, описывающий, к каким событиям пользователи проявляли интерес раньше, а также некоторые сведения о пользователях и о событиях. Нужно построить ML-модель, предсказывающую, будет ли конкретное событие интересно произвольно взятому пользователю, — это двоичный классификатор.

Все эти данные вместе с информацией о конкурсе доступны на странице *www.kaggle.com/c/event-recommendation-engine-challenge* после авторизации. Основные наборы данных находятся в файлах train.csv, events.csv и users.csv, которые можно объединить по идентификаторам пользователей или событий. Имеет смысл ограничиться событиями, к которым был проявлен открытый интерес или полное отсутствие интереса, а также базовыми численными и категориальными признаками. Иллюстрация 5.2 демонстрирует результат отбора исходной обучающей выборки.



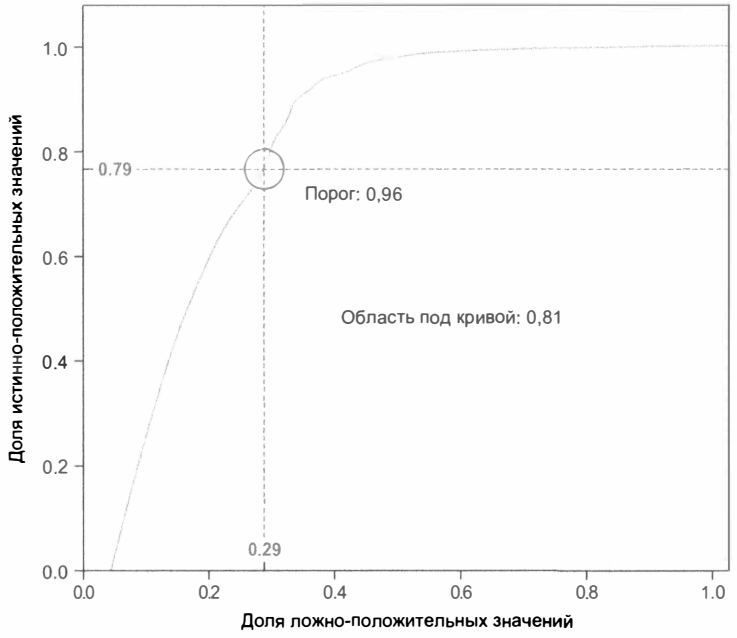
interested	invited	birthyear	gender	timezone	lat	ing
1	0	1994	Male	420	–6.357	106.362
1	0	1976	Male	–240	43.655	–79.419
1	0	1980	Male	–480	33.888	–118.378
1	0	1980	Male	–480	33.846	–117.977
1	0	1994	Female	420	–7.265	112.743
1	0	1986	Male	–480	NaN	NaN
1	0	1984	Male	–420	33.493	–111.934

Илл. 5.2. Пример набора данных, предназначенного для обучения модели, рекомендующей события

Пример обучающей выборки содержит следующие признаки:

- `invited` — булевское значение, указывающее, приглашался ли пользователь на мероприятие;
- `birthyear` — год рождения пользователя;
- `gender` — половая принадлежность пользователя;

- `timezone` — временная зона, в которой в настоящее время находится пользователь;
- `lat/lng` — широта/долгота, на которой происходит событие.



Илл. 5.3. Кривая ошибок после оценки методом скользящего контроля и показатель AUC для простой модели, рекомендующей события

Начнем с построения и оценки модели на базе только этих шести признаков. Очевидно, что количество шаблонов, определяющих, будет ли пользователь заинтересован в событии, в данном случае ограниченно. Чуть ниже мы используем несколько простых приемов проектирования признаков, которые позволят расширить базовый набор и вывести на поверхность новую информацию.

Начальную модель двоичной классификации для предсказания целевой переменной построим на шести входных признаках. В соответствии со схемой, описанной в главах 1–4, выполним следующие действия.

1. Произведем предварительную обработку данных (преобразуем категориальные столбцы в численные и добавим отсутствующие значения).

2. Выполним обучение модели (воспользовавшись алгоритмом «случайный лес»).

3. Оценим модель (используя скользящий контроль по 10 блокам и ROC-кривые). Итоговая ROC-кривая показана на илл. 5.3. Ее AUC достигает 0,81.

5.2.2. Обработка даты и времени

Попытаемся улучшить результат работы модели с помощью проектирования признаков. В дополнение к показанным на илл. 5.2 данным с каждым событием связана переменная `start_time`. Это элемент данных, представляющий собой строку, отформатированную в соответствии со стандартом ISO-8601 UTC. Она указывает время начала события. Форматирование поля данных вида `2012-10-02 15:53:05.754000+00:00` записывается как yyyy-mm-dd hh:mm:ss.mmmmmm_HH:MM.

Описанные в главе 3 типы ML-моделей поддерживают численные и категориальные входные признаки, но строка `datetime` не является ни тем ни другим. Поэтому нельзя просто взять и вставить в модель столбец с такими данными. Но можно преобразовать элементы `datetime` в численные признаки, взяв для этого информацию из строки `datetime`. То есть простая, но мощная концепция проектирования позволяет преобразовать каждую строку `datetime` в набор следующих признаков:

- hour of the day (час дня);
- day of the week (день недели);
- month of the year (месяц года);
- minute of the hour (минута часа);
- quarter of the year (квартал года).

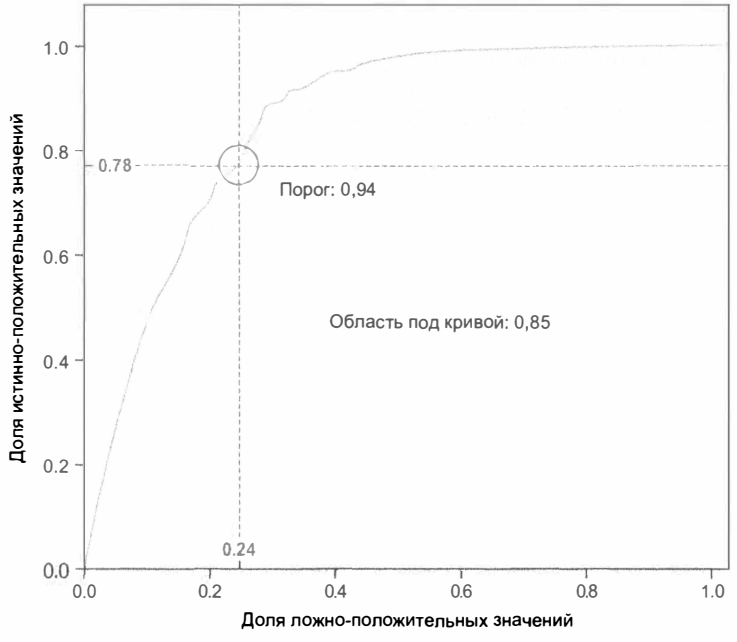
Иллюстрация 5.4 демонстрирует первые пять строк данных, полученных в результате преобразования одного признака `start_time` в 10 признаков `datetime`.

Теперь воспользуемся алгоритмом «случайный лес» и на базе этого нового, состоящего из 16 признаков набора данных построим еще одну модель. Вид ROC-кривой после перекрестной проверки показан на илл. 5.5.

datetime_ hour_of_day	datetime_day_ of_week	datetime_day_ of_month	datetime_day_ of_year	datetime_month_ of_year
13	4	26	300	10
13	4	26	300	10
13	4	26	300	10
13	4	26	300	10
13	4	26	300	10

datetime_ minute_of_hour	datetime_second_ of_minute	datetime_year	datetime_quarter_ of_year	datetime_week_ of_year
30	0	2012	4	43
30	0	2012	4	43
30	0	2012	4	43
30	0	2012	4	43
30	0	2012	4	43

Илл. 5.4. Дополнительные столбцы даты-времени, извлеченные из столбца с временной меткой набора данных по рекомендации событий



Илл. 5.5. Кривая ошибок после процедуры скользящего контроля для модели, включающей признаки даты-времени

Для этой модели AUC увеличился с 0,81 до 0,85. Очевидно, что в информации из переменной `start_time` было скрыто некое значение, которое после вставки в ML-модель помогло увеличить ее точность. Скорее всего, дело в том, что мероприятия, происходящие в определенные дни недели и в определенные часы, популярнее остальных.

5.2.3. Извлечение признаков из обычного текста

Кроме времени начала события, данные включают в себя текстовые признаки, сгенерированные простыми процедурами обработки лингвистической информации. Аналогично тому, как признаки `datetime` не могут напрямую использоваться моделью, так как не являются ни численными, ни категориальными, нельзя напрямую скормить ML-алгоритму и произвольный текст. Требуется предварительная обработка, приводящая его к одному из двух вышеуказанных типов. Для превращения текста в ML-признаки воспользуемся методом, который называется «*мешок слов*» (bag of words). В его основе лежит простая идея: мы считаем количество вхождений каждого слова в текст и вставляем в набор данных столбец с соответствующим числом. При этом, как обычно, мы сталкиваемся с усложняющими ситуацию факторами.

Признаки для ML-алгоритма должны быть гомогенными — у всех экземпляров набора данных должно быть одно и то же количество признаков, соответствующих одной и той же основной идее. Скажем, если один экземпляр содержит пять вхождений слова *family*, а в следующем за ним экземпляре это слово отсутствует, нужно или добавить столбец `Family` и указать во втором случае значение 0, или отбросить оба экземпляра. Обычно рассматриваются все слова из текстового фрагмента и решается, какие из них имеет смысл превратить в столбцы, а какие — нет. В большинстве случаев «мешок слов» строится для всего набора данных, а затем слова, появляющиеся в тексте чаще всего, превращаются в столбцы. Для остальных слов создается обобщающий столбец, позволяющий оценить полную длину текста.

Предположим, мы начали работать с текстом из ста слов. При этом появится множество столбцов с распространенными, но не несущими информации словами, такими как предлоги, частицы, артикли. В теории поиска информации они называются *шумовыми*, или *стоп-словами* (stop words), и обычно удаляются из текста перед подсчетом для «мешка слов».

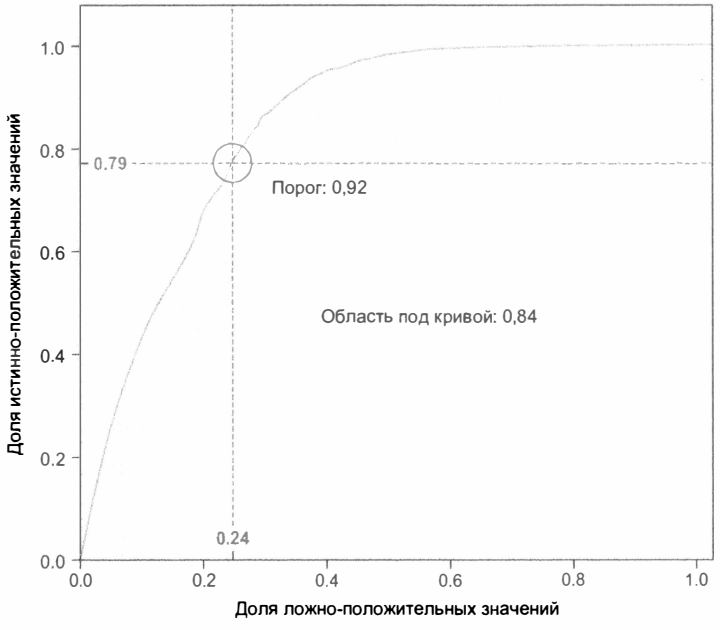
С более сложными концепциями текстовых признаков мы познакомим вас в следующей главе, пока же упомянем всего один осложняющий фактор — «мешок слов» быстро становится большим и разреженным. Появляется множество признаков, по большей части заполненных нулями, так как вероятность появления конкретных слов в произвольном фрагменте текста стремится к нулю. Английский словарь состоит из более чем 200 тысяч слов, и в большинстве текстов фигурирует только малая их доля. В некоторых ML-задачах пространство, в котором класс слов представлен больше, чем в общем случае, еще более узко. Скажем, на илл. 5.6 показаны экземпляры признаков для слов, чаще всего встречающихся в примере с рекомендацией событий; разреженность данных сразу бросается в глаза. Существуют такие ML-алгоритмы, как наивный байесовский классификатор, которые хорошо обрабатывают разреженные данные (не требуя дополнительной памяти для нулей), но большинство алгоритмов этого делать не умеет.

2	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	2	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
1	1	0	0	0	0	0	2	0	0	0	0	0	0	1	0	0	0	1	2	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2	0	0	33	0	3	1	0	0	0	1	1	1	1	0	0	0	0	0	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Илл. 5.6. Срез «мешка слов» для примера с рекомендацией событий. Числа показывают количество вхождений слов, чаще всего встречающихся в описании событий. Большая часть ячеек содержит значение 0, поэтому мы называем такой набор данных разреженным

В файле events.csv 100 признаков представляют «мешок» из 100 чаще всего встречающихся слов. Мы используем их в качестве признаков для нашей модели, так как некоторые события могут оказаться популярнее остальных. Иллюстрация 5.7 показывает результирующую ROC-кривую после добавления этих признаков в модель.

Показатель AUC на илл. 5.7 практически такой же, как и в предыдущем варианте модели, включающем в себя только базовые признаки и признаки, связанные с датой-временем. Это означает, что описания событий не влияют на внимание пользователей. Интересы конкретных пользователей в модели не учитываются, оценивается только пользовательская база в целом. В случае настоящего движка для выдачи рекомендаций мы бы строили модель для каждого пользователя или для каждого класса пользователей. Другие популярные методы выдачи рекомендаций для поиска оптимальных вариантов рассматривают связи между событиями, пользователями и друзьями пользователей.



Илл. 5.7. Кривая ошибок, построенная после перекрестной проверки для полной модели, включающей в себя и такие признаки, как дата-время и текст

5.3. Выбор признаков

Преимущество алгоритмов машинного обучения перед базовыми статистическими методами и человеческой способностью к распознаванию закономерностей — умение обрабатывать большие наборы признаков. Большинство ML-алгоритмов в состоянии обработать тысячи, а то и миллионы признаков. И зачастую целесообразно добавить в модель допол-

нительные признаки для увеличения точности ее работы. Но, как и во многих других случаях, в машинном обучении «больше» далеко не всегда означает «лучше».

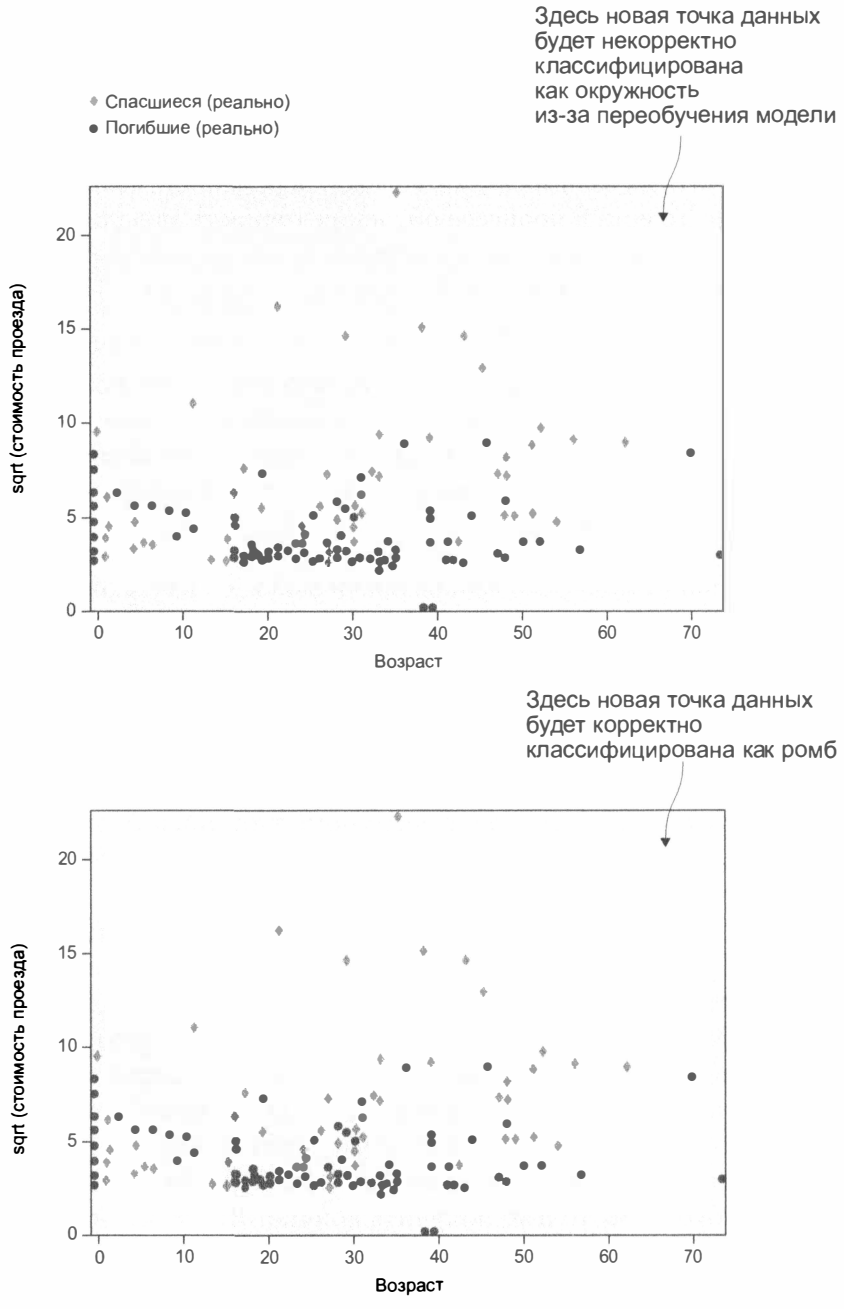
Так как с ростом числа признаков модель получает возможность более детально изучить их взаимосвязи с целевой переменной, возникает риск переобучения. То есть в процессе обучения точность модели возрастет, но пострадает прогностическая производительность на новых, ранее неизвестных данных. Пример переобучения показан на илл. 5.8 (впервые мы обсуждали эту тему в главе 3).

В этом разделе мы рассмотрим методы выбора признаков, позволяющие избежать переобучения и, соответственно, увеличить точность моделей на новых данных. Есть алгоритмы более и менее восприимчивые к переобучению, но когда важна точность модели, имеет смысл выполнить описанную выше оптимизацию.

Другой выгодой от меньшего набора признаков и, соответственно, меньших моделей является то, что вычислительные затраты на обучение и прогнозирование, как правило, связаны с количеством признаков. Потратив время на этапе внедрения модели, вы сможете сэкономить его при повторном обучении и генерации предсказаний.

Наконец, выбор признаков и связанная с этим процессом концепция *важности признака* (feature importance) помогают увидеть взаимосвязи внутри модели и в использовавшихся для ее построения данных. В некоторых случаях модели создаются не для прогнозов, а для получения информации о важных признаках. Сведения о наиболее значительных из них позволяют обнаружить такие шаблоны, как, к примеру, корреляция кредитной истории с определенными демографическими и социальными факторами. Вычислительные затраты могут быть связаны с получением данных для конкретных признаков, и нет никакой нужды страдать от потерь, если такой признак не важен для рассматриваемой модели. Важность конкретных признаков также может дать ценную информацию о генерируемых моделью предсказаниях. При решении реальных задач зачастую требуется не только получить конкретный ответ, но и понять, *зачем* было сделано определенное предсказание.

Надеемся, что у вас появилась мотивация более детально подойти к вопросу выбора признаков. Простейший способ выбрать оптимальное подмножество признаков — перебор всех комбинаций. Например, вы строите



Илл. 5.8. Решающая граница двух моделей, обученных на одних и тех же данных. В верхней модели граница слишком детализирована, что влияет на производительность классификации на новых данных

модель для всех подмножеств и, пользуясь сведениями из главы 4, измеряете ее производительность. К сожалению, такой подход становится неосуществимым уже при небольшом количестве признаков. Поэтому нужны техники, позволяющие приблизительно определить оптимальное подмножество. Ниже мы рассмотрим некоторые из них. Одним из самых распространенных классов методов является прямой отбор/обратное исключение, с которого мы и начнем. После чего поговорим и об остальных эвристических методах.

АЛГОРИТМЫ СО ВСТРОЕННЫМИ ФУНКЦИЯМИ ОТБОРА ПРИЗНАКОВ

Методы, которые будут разбираться ниже, применимы ко всем алгоритмам машинного обучения, но некоторые из них обладают преимуществом с точки зрения выбора признаков, так как обладают сходным встроенным поведением. Разумеется, они далеко не всегда дают результаты, сравнимые с универсальными методами, но могут оказаться более эффективными с точки зрения вычислительных затрат. И возможно, имеет смысл начинать с таких встроенных методов и только потом переходить к более ресурсоемким универсальным алгоритмам. Кроме того, методы со встроенными функциями отбора можно использовать для получения начальных данных, экономя тем самым время.

Примеры встроенных методов отбора признаков: веса, присваиваемые признакам в алгоритмах линейной и логистической регрессии, значения важности признаков в деревьях решений и вариантах с ансамблями, таких как «случайные леса». Последние определяют — эффективным с точки зрения вычислений способом, — как уменьшится прогностическая точность при замене признака случайным шумом. Проверим самые важные признаки в рекомендующей события модели, построенной на базе алгоритма «случайный лес».

Признак	Важность
birthyear	
timezone	
datetime_week_of_year	
datetime_day_of_year	
lat	
datetime_hour_of_day	
lng	

Семь наиболее важных с точки зрения алгоритма «случайный лес» признаков в модели, рекомендующей события. Эта оценка показывает, что двумя наиболее важными факторами, определяющими интересность события для произвольного пользователя, являются год рождения и временная зона события

5.3.1. Прямой отбор и обратное исключение

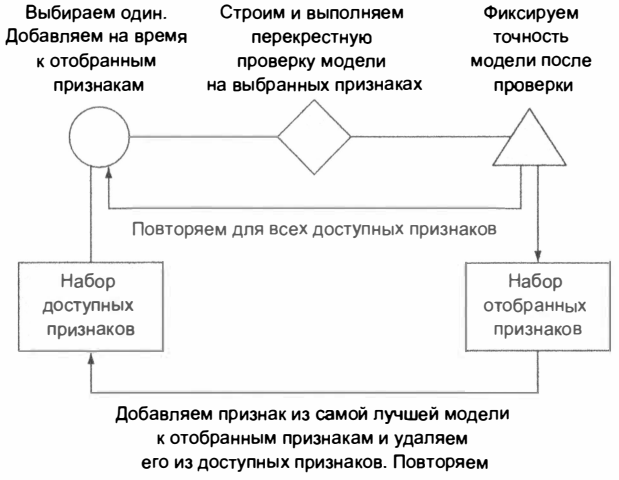
Чаще всего для примерного определения оптимального подмножества признаков применяются итерационные методы отбора, которые мы сейчас рассмотрим. Они базируются на одном из двух принципов: начать с отсутствия признаков и постепенно найти самые лучшие, которые будут добавлены в подмножество, или же начать со всех доступных признаков и последовательными итерациями исключить самые худшие. Поиск останавливается после того, как добавление или исключение новых признаков перестает влиять на уровень точности, или при достижении заранее указанного размера подмножества.

Эти методы называются *прямым отбором* (forward selection) и *обратным исключением* (backward elimination) соответственно. Они вовсе не гарантируют того, что выбранный набор признаков окажется самым лучшим, поэтому мы называем их *аппроксимацией*. Какой-то из оставленных вне подмножества признаков мог давать большую прогностическую точность в паре с конкретным набором других признаков, еще не собранных вместе на момент его рассмотрения. Напоминаем, что эффективность машинного обучения связана с возможностью обнаружения шаблонов путем объединения множества признаков. Или, говоря другими словами, слабый признак может оказаться сильным в присутствии правильно подобранного набора других признаков.

Впрочем, на практике прямой отбор и обратное исключение прекрасно справляются с задачей отбора признаков в подмножество, требуя при этом намного меньшего количества вычислительных ресурсов, чем исчерпывающий поиск. Хотя при очень большом количестве признаков с вычислениями может не справиться даже этот подход. В таких случаях имеет смысл положиться на встроенную оценку важности признаков или другие эвристические алгоритмы поиска, которые мы рассмотрим в следующем разделе.

Процесс прямого отбора признаков показан на илл. 5.9. Количество моделей, которые нужно будет построить, зависит от количества признаков. Если алгоритм работает до конца, потребуется $N + (N - 1) + (N - 2)... (N - N + 2) + (N - N + 1)$ или $\sum_{i=0}^{N-1}(N-i)$ моделей. Для 20, 100, 500 или 1000 признаков это будет 210, 5050, 125 250 и 500 500 моделей соответственно. Кроме того, каждая перекрестная проверка требует еще k моделей, поэтому если построение модели требует значительного времени,

процесс становится неподъемным. Но для небольших наборов признаков или при меньшем количестве итераций (например, потому что точность быстро перестает расти) такой подход вполне эффективен.



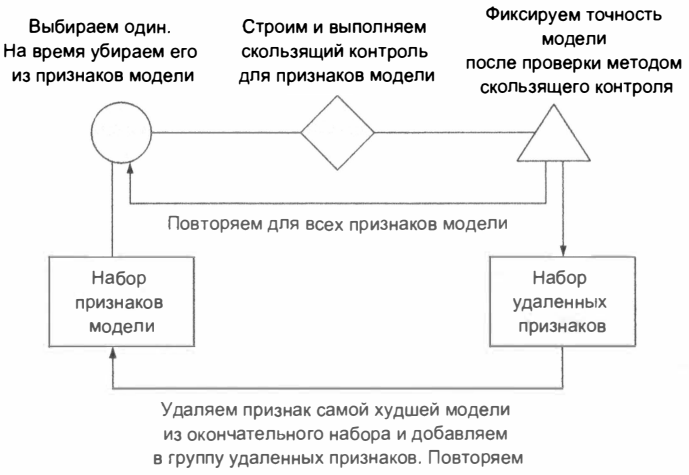
Илл. 5.9. Процесс прямого отбора признаков. Начиная с крайнего левого блока, признаки добавляются по одному, пока не будет отобран набор, самый лучший с точки зрения оценки методами скользящего контроля

Иллюстрация 5.10 демонстрирует аналогичную схему обратного исключения. Вычислительные требования остаются теми же, поэтому выбор между прямым отбором и обратным исключением обычно определяется особенностями поставленной задачи и взятого для ее решения алгоритма. Скажем, есть алгоритмы, которые с очень маленькими наборами признаков работают хуже, соответственно для них имеет смысл применять обратное исключение.

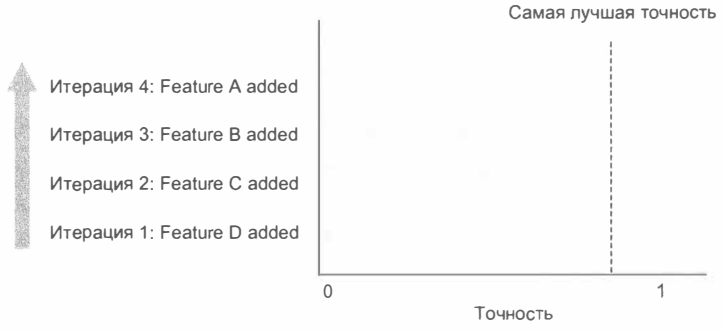
Наглядным способом визуализации процедуры отбора признаков является столбчатая вертикальная диаграмма, пример которой показан на илл. 5.11.

Как уже упоминалось, в некоторые ML-алгоритмы встроены методы отбора признаков. Ими можно воспользоваться для гибридного подхода, в котором определенная встроенным методом важность признака применяется при поиске наилучшего или наихудшего признака на каждой итерации процесса прямого отбора или обратного исключения. Это значительно уменьшает время вычислений при большом количестве при-

знаков, хотя и дает в итоге менее точную аппроксимацию оптимального подмножества.



Илл. 5.10. Процесс обратного исключения



Илл. 5.11. Столбчатая диаграмма циклического отбора признаков, иллюстрирующая оценку точности в процедуре отбора, — в рассматриваемом случае это прямой отбор. Здесь можно использовать любые подходящие методы измерения точности (см. главу 4)

5.3.2. Отбор признаков для исследования данных

Отбор признаков производят не только для того, чтобы избежать переобучения или сделать модель более компактной. Эффективный отбор дает глубокое понимание модели и обучающих данных. В некоторых случаях классификаторы создаются только для запуска алгоритма отбора и не делают никаких предсказаний.

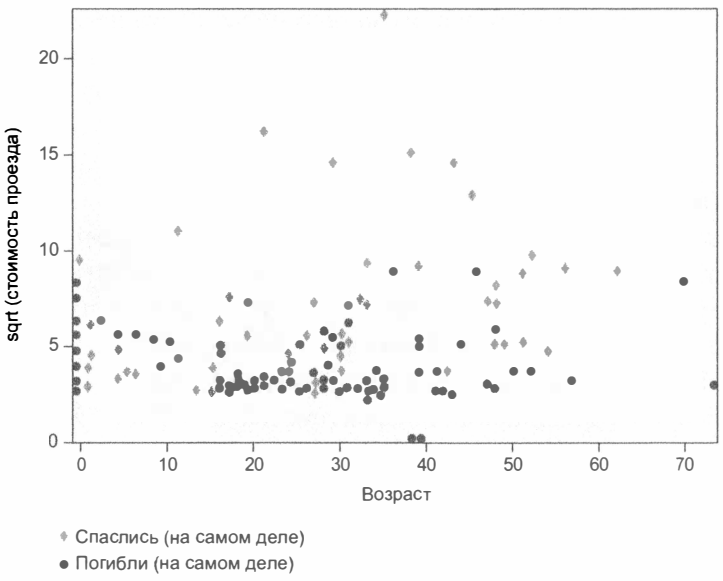
С помощью алгоритма отбора вы можете выполнить поисковый анализ данных, использованных для построения модели. Этот алгоритм дает наиболее важные признаки — максимально информативное подмножество для предсказания целевой переменной, извлеченное из всего набора признаков. Полученный список сообщает информацию о самих данных, что иногда ценно само по себе. Представьте, что требуется предсказать, есть ли у пациента рак. Поскольку уверенности в причинах конкретных форм рака нет, вы добавляете все признаки, которые удается получить, и применяете процедуру отбора, чтобы найти самые важные из них. Этим не только повышается точность модели, но и перечисляются факторы, которые с самой большой вероятностью вызывают рак или, по меньшей мере, находятся в определенном соотношении с вероятностью диагноза. Обсуждавшиеся выше методы не показывают, в положительном или отрицательном направлении влияет конкретный признак (в случае бинарной классификации), но это можно легко понять, визуализировав связь признака с целевой переменной (например, с помощью методов, обсуждавшихся в разделе 2.3).

При обучении без учителя отбор признаков используется еще и для *уменьшения размерности*. Наборы данных с более чем тремя переменными очень сложно визуализировать. Человеческий мозг настроен на трехмерный мир, и мы сталкиваемся с трудностями при попытках его расширения. Но реальные данные, ограничивающиеся всего тремя признаками, встречаются крайне редко, поэтому для визуализации многомерных наборов данных приходится прибегать к различным техникам. Отбор признаков позволяет показать, как ML-алгоритмы делят данные на классы, например, построив график зависимости двух или трех лучших признаков от целевой переменной. Иллюстрация 5.12 показывает пример границы решений в двух измерениях, хотя для построения модели использовалось намного большее число признаков.

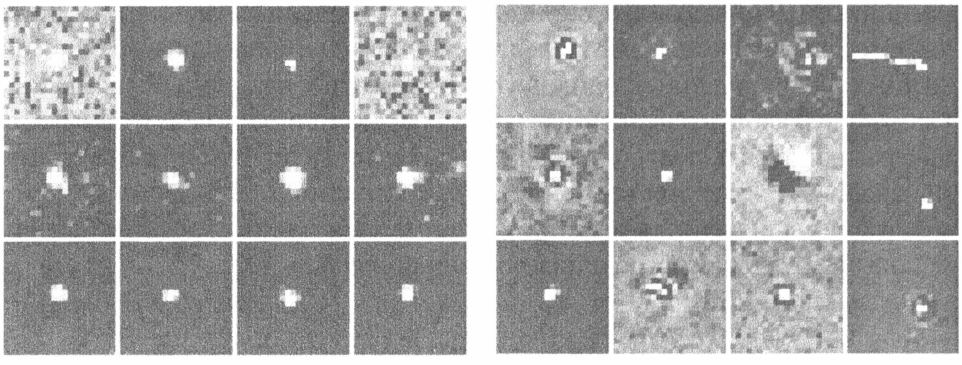
5.3.3. Практический пример отбора признаков

В качестве практического примера проектирования и отбора признаков рассмотрим следующую научную задачу.

Нужно найти реальные снимки вспышки сверхновой среди большого количества фиктивных изображений (фотографий, которые выглядят как снимки реальных событий, но таковыми не являются). Примеры реальных и фиктивных изображений представлены на илл. 5.13.

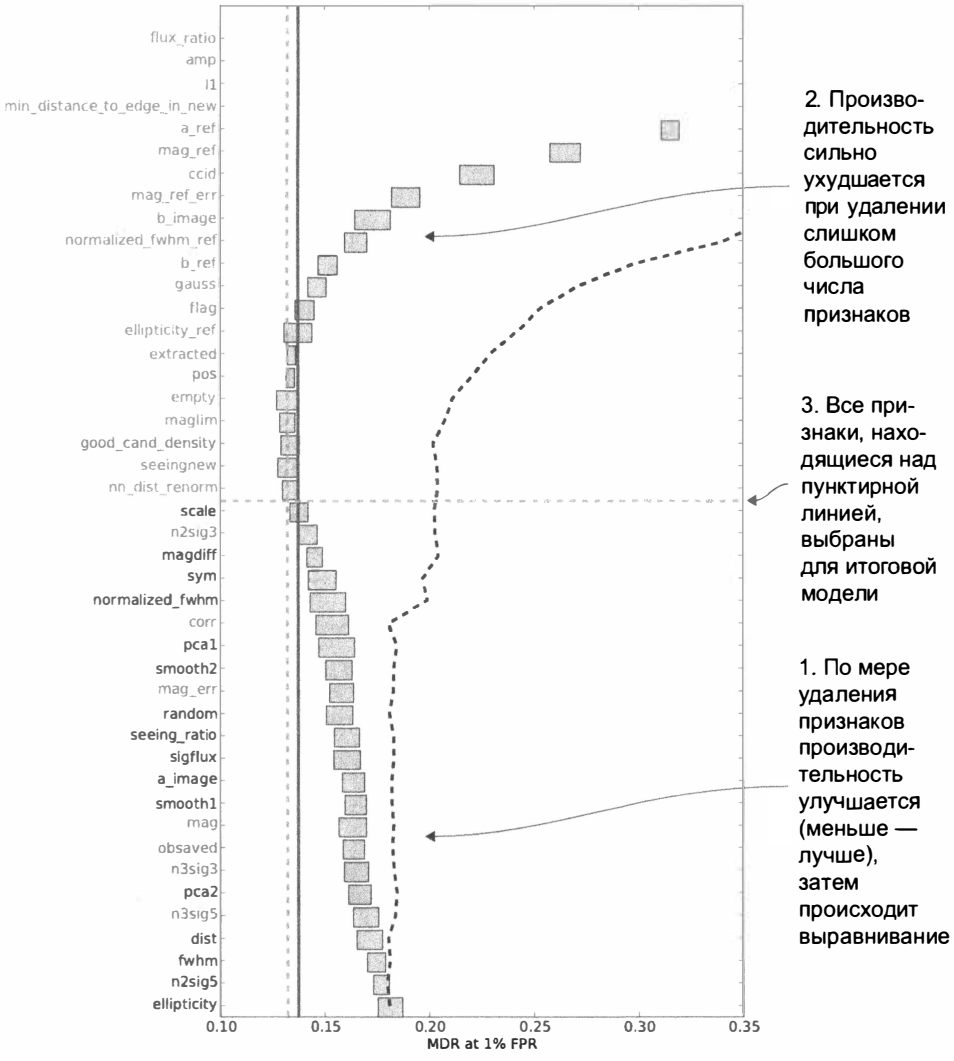


Илл. 5.12. Эта решающая граница для разделения на круги и ромбы обладает всего двумя признаками. Модель построена на множестве признаков, но алгоритм отбора показал, что для рассматриваемой проблемы (шанс выживания на «Титанике») важнее всего такие признаки, как sqrt (стоимость проезда) и Возраст. Впервые этот график появился в главе 3



Илл. 5.13. Реальные фотографии вспышек сверхновых показаны слева. Фиктивные изображения собраны справа[1]. Классификатору нужно понять разницу между двумя типами фотографий, воспользовавшись извлеченными из снимков признаками. (Здесь собраны очевидные примеры; зачастую произвести классификацию сложно даже специально обученным людям.)

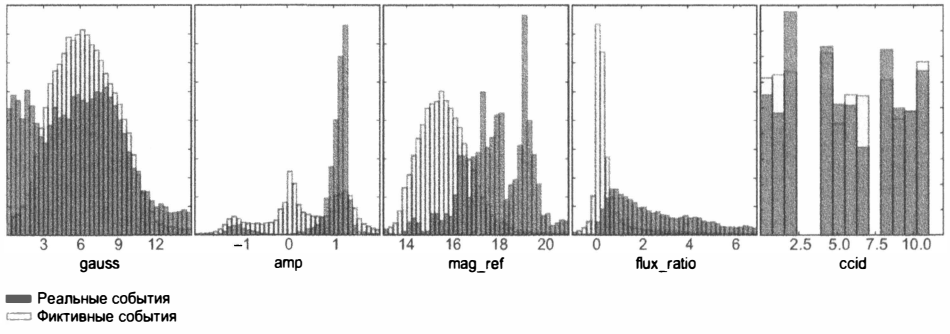
[1] Фигурирующие в этом разделе фотографии вспышек сверхновых и связанные с ними графики впервые были опубликованы в 2013 году в журнале *Monthly Notices of the Royal Astronomical Society*, том 435, номер 2, страницы 1047–1060 (*http://mnras.oxfordjournals.org/content/435/2/1047*).



Илл. 5.14. Процесс обратного исключения. По мере работы алгоритма каждый признак, начиная снизу, выбирается для удаления, и на каждом шаге производится индивидуальная оценка доли пропущенных обнаружений (MDR — missed detection rate) при 1% ложно-положительных значений. Полоски показывают оценку производительности на каждом шаге (в этом случае чем меньше, тем лучше) после удаления признака (со стандартным отклонением от результатов перекрестной проверки). После удаления 23 признаков (из 44) полученная при перекрестной проверке производительность выравнивается и, в конечном итоге, после удаления слишком большого числа признаков, ухудшается. В конце этот параметр удается подрастить на 5 %, удалив зашумленные признаки

Для построения классификатора реальных/фиктивных фотографий первым делом было выполнено преобразование необработанных изображений в набор признаков. Обсуждение некоторых из этих преобразований вы найдете в следующей главе. Подвергшиеся преобразованию данные были пропущены через алгоритм «случайный лес» с целью построить классификатор и выполнить различные процедуры по оптимизации модели, часть из которых описана в главах 3 и 4. Перед тем как отправить в модель поток данных с телескопа, нужно выбрать самые лучшие признаки, избежать переобучения и обеспечить минимальный размер модели для работы в режиме реального времени. График отбора признаков, представляющий собой слегка усовершенствованную версию илл. 5.11, показан на илл. 5.14.

Теперь, когда мы знаем, какие признаки важнее всего для модели, можно построить график их зависимости от реальных и фиктивных событий и увидеть, как каждый конкретный признак помогает в решении задачи. Иллюстрация 5.15 демонстрирует производительность четырех лучших признаков.



Илл. 5.15. Визуализация эффективности четырех признаков, указанных алгоритмом отбора как лучшие. Гистограммы показывают количество реальных и фиктивных событий, определяемых при конкретном значении признака. Как видите, распределение соотношений реальных и фиктивных событий отличается для признаков amp и flux_ratio, которые, согласно алгоритму отбора, обеспечивают максимальную производительность модели

5.4. Заключение

Эта глава познакомила вас с процессом проектирования признаков, то есть преобразования необработанных данных с целью улучшения

точности ML-моделей. Вот основные положения, которые следует запомнить.

- Проектирование признаков — это процесс применения математических преобразований к необработанным данным с целью создания новых входных признаков для ML-моделирования. Эти преобразования могут быть как совсем простыми, так и чрезвычайно сложными.
- Ценность проектирования признаков состоит в следующем:
  - возможность получить признаки, сильнее связанные с целевой переменной;
  - возможность использовать внешние источники данных;
  - возможность задействовать неструктурированные данные;
  - возможность создавать более интерпретируемые признаки;
  - свобода создавать множество признаков и использовать только самые лучшие из них, определенные процедурой отбора.
- Существует внутренняя связь между проектированием признаков и знанием предметной области.
- Проектирование признаков можно вставить в рабочий процесс ML в двух местах:
  - в обучающую выборку до обучения модели;
  - в данные, используемые для предсказаний, до момента их генерации.
- В задаче на рекомендацию событий можно использовать два простых типа проектирования признаков:
  - извлечение признаков из информации о дате и времени;
  - проектирование признаков на основе обычного текста.
- Отбор признаков — это надежный способ получения наиболее перспективного в прогностическом отношении подмножества признаков из всего набора данных.

5.5. Терминология

Термин	Определение
Проектирование признаков (feature engineering)	Преобразование входных данных для извлечения дополнительных значений и увеличения прогностической точности ML-моделей
Отбор признаков (feature selection)	Процесс выбора наиболее перспективного в прогностическом отношении подмножества признаков из полного набора
Прямой отбор (forward selection)	Версия отбора признаков, в которой по одному итеративно добавляются признаки, сильнее всего увеличивающие точность модели, зависящие от активного в настоящий момент набора признаков
Обратное исключение (backward elimination)	Версия отбора признаков, в которой по одному удаляются признаки, сильнее всего уменьшающие точность модели, зависящие от активного в настоящий момент набора признаков
«Мешок слов» (bag of words)	Метод превращения произвольного текста в численные признаки, доступные для ML-алгоритма

В главе 7 мы расширим представленные выше простые подходы к проектированию признаков, и вы научитесь проектировать их для таких данных, как текст, изображения и временные ряды. А следующая глава посвящена практическому применению полученных вами знаний.

ЧАСТЬ II
Практическое применение

Во второй части мы выйдем за пределы базового рабочего процесса ML и научим вас извлекать признаки из текста, изображений и временных рядов, еще больше увеличивать точность прогнозирования и масштабировать ML-системы для больших объемов данных. Кроме того, три главы будут посвящены практическим примерам, и вы своими глазами увидите процесс моделирования.

Глава 6 полностью посвящена рассмотрению примера. Мы попробуем предсказать шансы нью-йоркских таксистов на получение чаевых.

В главе 7 мы рассмотрим усовершенствованные варианты проектирования признаков, позволяющие извлекать данные из обычного текста, изображений и временных рядов. Эти техники лежат в основе многих современных приложений с машинным обучением и искусственным интеллектом.

В главе 8 мы воспользуемся полученными знаниями и рассмотрим вариант более сложного проектирования признаков на еще одном большом примере — будем предсказывать тональность рецензий к фильмам.

В главе 9 вы прочитаете про техники масштабирования ML-систем для увеличения объемов данных, получения более высокой пропускной способности прогнозирования и уменьшения задержки при генерации предсказаний. Это важные аспекты развертки многих современных ML-приложений.

Глава 10 снова целиком посвящена примеру построения модели. На базе большого набора данных мы создаем модель, предсказывающую вероятность перехода пользователя по рекламному баннеру.

6 Пример: чаевые для таксистов

В этой главе:

- ✓ представление, визуализация и подготовка сведений о суммах чаевых, которые получают нью-йоркские таксисты;
- ✓ построение классификатора, предсказывающего отношение пассажира к чаевым;
- ✓ оптимизация ML-модели путем подбора ее параметров и проектирования признаков;
- ✓ построение и оптимизация модели регрессии, предсказывающей размер чаевых;
- ✓ применение моделей для более полного понимания описываемых ими данных и поведения.

Первые пять глав рассказали вам, как перейти от необработанных, хаотичных данных к построению, проверке работоспособности и оптимизации модели путем подбора параметров и проектирования признаков, отражающих связанную с задачей область знаний. Эти главы изобиловали небольшими примерами, иллюстрирующими различные аспекты машинного обучения, теперь пришло время применить полученные знания и рассмотреть целиком процесс решения реальной задачи. Это первая из трех глав (еще будут главы 8 и 10), полностью посвященная практическому применению машинного обучения.

В первом разделе мы внимательно рассмотрим различные варианты визуализации имеющихся у нас данных, чтобы понять, что именно можно сделать в данной ситуации. Вы увидите, каким образом выполняется изначальная обработка данных и их подготовка к моделированию. Во втором разделе будет определена задача классификации, а затем мы улучшим производительность модели путем настройки ее параметров и проектирования новых признаков.

6.1. Данные: сведения о чаевых и плате за проезд

В последние годы благодаря работающим с информацией компаниям и организациям появился доступ к полным и интересным наборам данных. Вдобавок ко всему, некоторые из этих организаций используют концепцию *открытых данных* (open data), разрешающую открытое распространенные данных и их использование любой заинтересованной стороной.

Недавно благодаря закону о свободе распространения информации штата Нью-Йорк появился доступ к детализированным сведениям обо всех поездках в такси в городе Нью-Йорк за 2013 год[1]. Можно узнать место посадки и высадки пассажира, время и продолжительность поездки, расстояние и стоимость. Эти данные квалифицируются как реальные не только по способу их генерации, но и по неоднородности: набор содержит отсутствующие данные, фальшивые записи, не имеющие значения столбцы, интегрированные смещения и т. п.

[1] Изначально появился в блоге Криса Вонга: *http://chriswhong.com/open-data/foil_nyc_taxi/*.

Кроме того, объем данных очень велик. Весь набор в формате CSV весит свыше 19 Гбайт, что неподъемно для многих приложений с машинным обучением в большинстве систем. Для простоты мы возьмем из него небольшое подмножество. О методах, позволяющих масштабировать модели до таких и даже больших размеров, расскажут главы 9 и 10, так что к концу книги вы узнаете, как проанализировать все 19 Гбайт.

Данные доступны для скачивания на сайте *www.andresmh.com/nyctaxitrips/*. Это 12 пар сжатых CSV-файлов поездка/оплата. В каждом файле около 14 миллионов записей, при этом файлы поездка/оплата совмещаются построчно.

Начнем с базового рабочего процесса: анализ данных; извлечение признаков; построение, оценка и оптимизация моделей; прогнозы на новых данных. Давайте рассмотрим данные, пользуясь методами визуализации из главы 2.

6.1.1. Визуализация данных

Приступая к решению новой задачи, первым делом следует понять, из чего состоит набор данных. Поэтому мы рекомендуем загрузить его и посмотреть в табличной форме. Объединим строчки поездка/стоимость друг с другом. Иллюстрация 6.1 показывает первые шесть строк.

Столбцы `medallion` (идентификатор машины) и `hack_license` (лицензия на занятие извозом), имеющие отношение к лицензии на профессиональную деятельность, выглядят как простые идентификаторы, которые могут пригодиться для бухгалтерии, но не очень интересны с точки зрения машинного обучения. Судя по названиям столбцов, некоторые из них содержат категориальные данные, скажем `vendor_id` (идентификатор фирмы), `rate_code` (код тарифа), `store_and_fwd_flag` (флаг, указывающий, что информация сохранена и передана) и `payment_type` (тип платежа). Распределение отдельных категориальных переменных имеет смысл визуализировать в табличной форме или в виде столбчатых диаграмм. Иллюстрация 6.2 демонстрирует эти диаграммы для вышеперечисленных столбцов.

Теперь рассмотрим численные столбцы. Интересно проверить, например, наличие корреляции между такими параметрами, как продолжительность поездки (`trip_time_in_secs`), расстояние (`trip_distance`) и полная стоимость поездки. Иллюстрация 6.3 демонстрирует диаграммы рассеяния для некоторых из этих факторов.

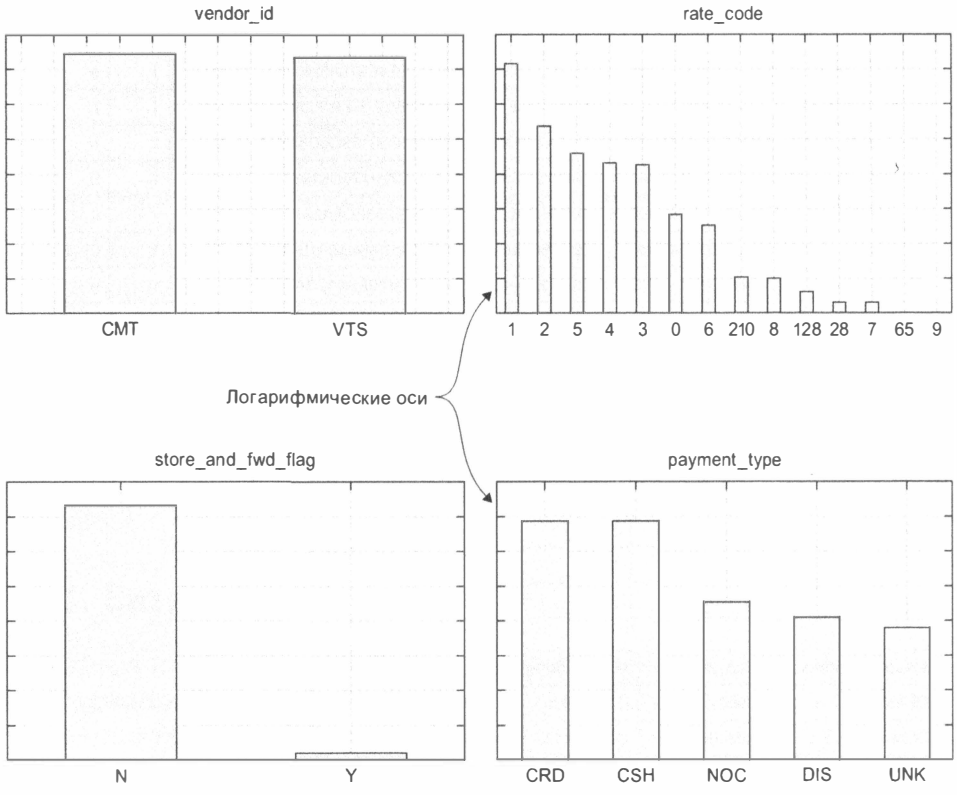
medallion	hack_license	vendor_id	rate_code	store_and_fwd_flag
CD847FE5884F10A28217E9FBA11B275B	5FEFD00D9773268B72EE4E879852F190	CMT	1	N
20D9ECB2CA0767CF7A01564DF2844A3E	598CCE5B9C1918568DEE71F43CF26CD2	CMT	1	N
A954A71B6D44265AE756BF807E069396	D5CA7D478A14BA3BBFC20153C5C88B1A	CMT	1	N
F6F7D02179BE915B23EF2DB57836442D	088879B44B80CC9ED43724776C539370	VTS	1	0
BE386D8524FCD16B3727DCF0A32D9B25	4EB96EC9F3A42794DEE233EC8A2616CE	VTS	1	0
E9FF471F36A91031FE5B6D6228674089	72E0B04464AD6513F6A613AABB04E701	VTS	1	0
A5D125F5550BE7822FC6EE156E37733A	08DB3F9FCF01530D6F7E70EB88C3AE5B	VTS	1	0

pickup_datetime	dropoff_datetime	passenger_count	trip_time_in_secs	trip_distance
1/8/2013 10:44	1/8/2013 10:46	1	123	0.30
1/8/2013 7:51	1/8/2013 7:51	1	4	0.00
1/7/2013 10:05	1/7/2013 10:13	1	446	1.10
1/13/2013 4:36	1/13/2013 4:46	5	600	3.12
1/13/2013 4:37	1/13/2013 4:48	2	660	3.39
1/13/2013 4:41	1/13/2013 4:45	1	240	1.16
1/13/2013 4:37	1/13/2013 4:47	5	600	2.91

pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	payment_type
-73.989296	40.756313	-73.987885	40.751122	DIS
-73.945396	40.802090	-73.945412	40.802025	NOC
-73.989090	40.748367	-73.974983	40.756035	DIS
-73.996933	40.720055	-73.993546	40.693043	CRD
-74.000313	40.730068	-73.987373	40.768406	CRD
-73.997292	40.720982	-74.000443	40.732376	CRD
-73.966843	40.756741	-73.987885	40.722713	CRD

fare_amount	surcharge	mta_tax	tip_amount	tolls_amount	tolls_amount	tipped
3.50	0.00	0.50	0.00	0.00	4.00	0
2.50	0.00	0.50	0.00	0.00	3.00	0
7.00	0.00	0.50	0.00	0.00	7.50	0
12.00	0.50	0.50	1.75	0.00	14.75	1
12.00	0.50	0.50	3.12	0.00	16.12	1
5.50	0.50	0.50	1.20	0.00	7.70	1
11.00	0.50	0.50	2.00	0.00	14.00	1

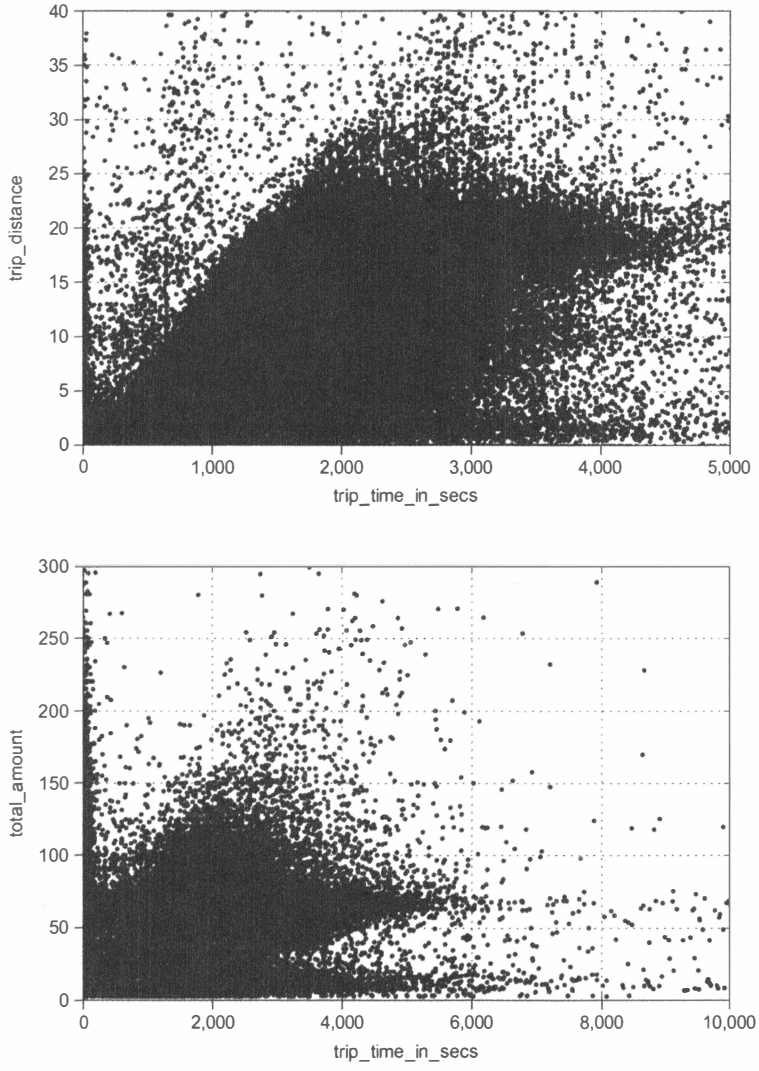
Илл. 6.1. Первые шесть строк данных о чаевых и стоимости проезда в нью-йоркском такси. Смысл большинства столбцов интуитивно понятен, но некоторые из них мы детально рассмотрим ниже



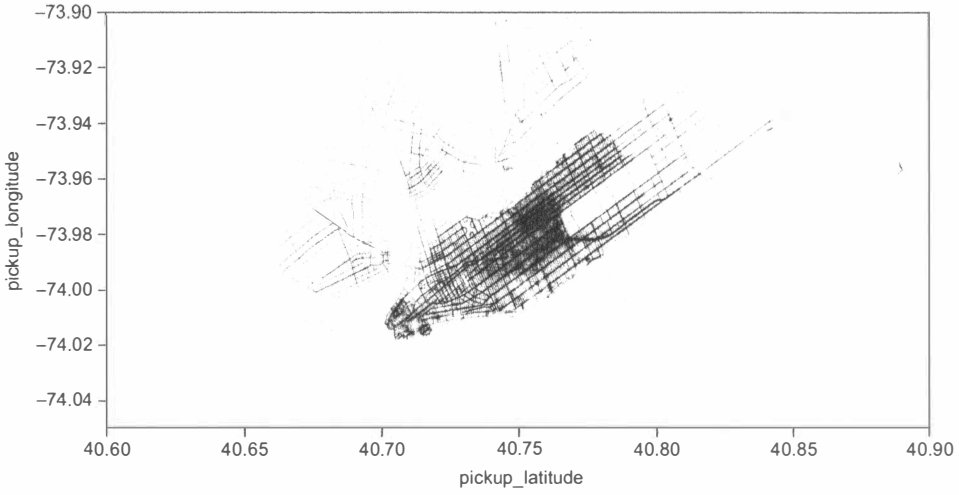
Илл. 6.2. Распределение значений в потенциально категориальных столбцах нашего набора данных

Наконец, илл. 6.4 визуализирует координаты начальной точки маршрута (столбцы `pickup_longitude` и `pickup_latitude`), определяющие карту перемещений такси по Нью-Йорку. Распределение выглядит вполне осмысленно: большинство начальных точек мы видим в центре Манхэттена, многие маршруты начинаются в других районах, а несколько точек вообще почему-то оказались на территории пролива Ист-Ривер!

Теперь, когда мы посмотрели на данные с другой точки зрения, давайте придумаем практическую задачу, которую можно решить, используя машинное обучение.



Илл. 6.3. Диаграммы рассеяния для времени поездки в секундах в зависимости от пройденного расстояния и от стоимости (в долларах США) соответственно. Как и ожидалось, определенная корреляция существует, но разброс остается относительно высоким. Видны и менее логичные кластеры, например множество поездок, иногда даже дорогих, с нулевым временем, что может указывать на поврежденные данные



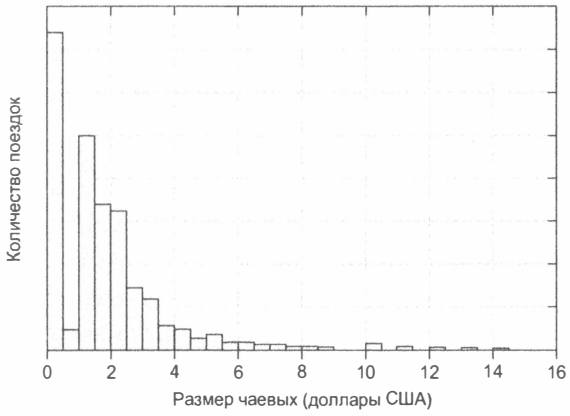
Илл. 6.4. Широта/долгота точек начала маршрутов. Обратите внимание, что ось x перевернута в сравнении с обычной картой. Видно, что многие пассажиры садились в такси на Манхэттене. По мере удаления от центра города этот показатель уменьшается

6.1.2. Формулировка задачи и подготовка данных

При первом же взгляде на данные внимание привлекает столбец `tip_amount`. Это информация о количестве чаевых (в долларах США), оставленных после каждой поездки. Интересно понять, какие факторы сильнее всего влияют на этот показатель.

Для этого можно построить классификатор, который будет на основе уже имеющейся информации о поездках предсказывать, оставит ли пассажир водителю чаевые. При наличии такой модели на мобильном устройстве водитель сможет в начале поездки получать предупреждения о потенциальном отсутствии чаевых и заранее принимать меры. Еще модель позволяет составить представление о том, какие параметры важнее всего или имеют наибольший прогностический потенциал и как по максимуму увеличить количество получаемых чаевых. Иллюстрация 6.5 демонстрирует гистограмму распределения размера чаевых для всех поездок.

Итак, модель должна предсказывать, какие поездки будут завершаться с чаевыми, а какие без них. Это работа для бинарного классификатора. Он даст возможность:



Илл. 6.5. Распределение размеров чаевых. Примерно в половине случаев сумма составляет $0, что превышает интуитивные ожидания

- помочь водителю такси, заранее сообщая о ситуации, когда чаевых не будет;
- понять, как и почему вообще возникают такие ситуации.

Случай из практики

Перед тем как приступить к построению модели, расскажем, как провалилась наша первая, внешне вполне успешная попытка решения данной задачи и как мы вышли из положения. Подобные ситуации часто возникают при работе с реальными данными, поэтому полезно поделиться полученным нами уроком. В машинном обучении важно избегать двух ловушек: *сценариев, которые слишком хороши, чтобы быть правдой*, и *преждевременных, не обусловленных данными предположений*.

В машинном обучении существует универсальное правило: если точность после скользящего контроля оказывается выше, чем вы ожидали, то, скорее всего, модель где-то вас обманывает. Реальность на удивление изобретательна в своих попытках осложнить жизнь работающим с данными людям. Наши первые классификаторы чаевые / отсутствие чаевых быстро показывали очень высокую прогностическую точность. Мы радовались высокой производительности при работе с новыми данными и на время забыли о том, что модель может попросту нас обманывать. Но имевшийся опыт заставил внимательнее присмотреться к чрезмерно оптимистичным результатам.

Мы обратили пристальное внимание на такой фактор, как важность входных признаков (в следующих разделах вы узнаете, как с ним работать). В нашем случае в модели однозначно доминировал признак *Тип платежа*.

Опыт поездок в такси показывает, что эта зависимость имеет смысл. При оплате картой меньше возможностей для чаевых. Человек, который платит наличными, практически всегда округляет сумму. Поэтому сегментирование мы начали с клиентов, которые платят картой. Но оказалось, что чаевые оставляет подавляющее большинство (более 95%) из миллионов расплачивающихся картой пассажиров. Слишком много для нашей теории.

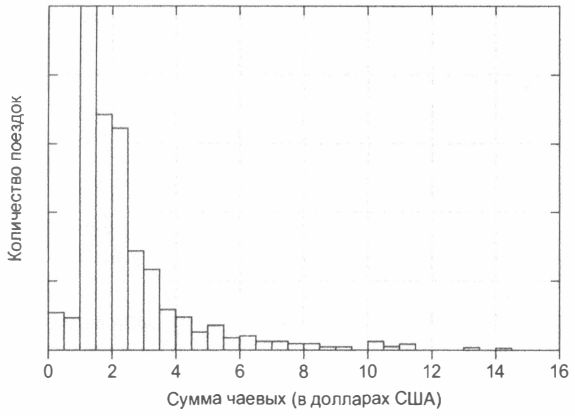
А сколько человек из расплачивающихся наличными оставляет чаевые? *Все?*

На самом деле *никто*! Мы быстро это поняли. При оплате наличными водитель не регистрирует чаевые должным образом, и они просто не попадают в данные. Рассмотрев ситуацию с точки зрения здравого смысла, мы обнаружили миллионы потенциальных злоупотреблений в системе нью-йоркского такси!

Вернемся к выводам для нашей ML-модели: если существует проблема с генерацией данных, строить на них модель нельзя. Если ответы некорректны по причине злоупотреблений, ML-модель не сможет показать реальное положение дел.

В конечном счете, для обхода этой проблемы из набора данных была попросту удалена вся информация об оплаченных наличными поездках. Изначально поставленная задача изменилась — теперь мы прогнозировали частоту чаевых только для случаев оплаты картой. Отказываться от части информации никогда не хочется. Но предположение о недостаточной достоверности сведений при оплате наличными нашло подтверждение в данных, соответственно мы поняли, что лучше всего использовать только проверяемые сведения и немного поменять формулировку задачи. Разумеется, гарантия корректности остальных записей о чаевых тоже отсутствует, но можно по крайней мере проверить новое распределение сумм. Иллюстрация 6.6 показывает гистограмму после удаления данных о чаевых при оплате наличными.

После удаления ложных данных распределение стало выглядеть намного лучше — отсутствием чаевых завершалось примерно 5% поездок. В следующем разделе мы попробуем выяснить, почему так получилось.



Илл. 6.6. Распределение сумм чаевых после удаления данных об оплате наличными (после того, как мы обнаружили, что в этом случае чаевые не фиксируются системой)

6.2. Моделирование

Итак, у нас есть готовые к работе данные. Информация из главы 3 поможет быстро создать и оценить модель. Мы построим несколько вариантов модели и попытаемся улучшить их производительность на каждой итерации.

6.2.1. Базовая линейная модель

Начнем моделирование с самого простого — с алгоритма логистической регрессии. Пока мы ограничимся только численными данными, так как указанный алгоритм умеет с ними обращаться и их не требуется дополнительно обрабатывать.

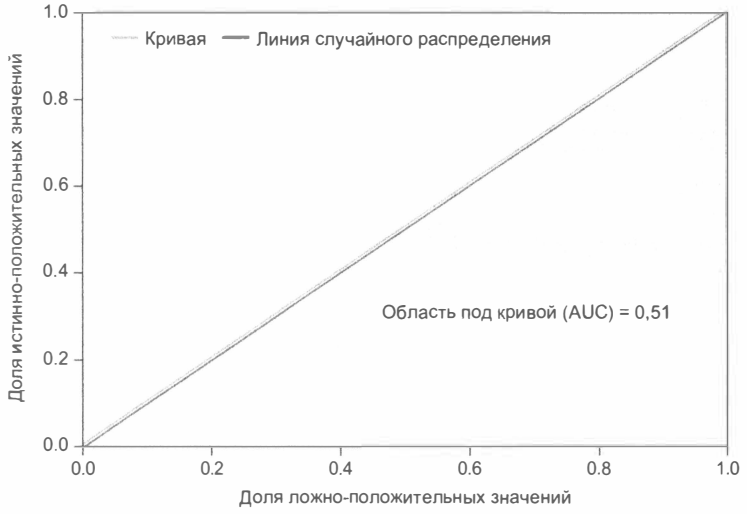
Воспользуемся библиотеками scikit-learn и pandas для языка Python. Первым делом случайным образом перемешаем экземпляры и разобьем их на две части: 80% сформируют обучающую выборку, а остальные 20% оставим для тестирования. Также нужно добиться того, чтобы ни один столбец не считался более важным, чем другие. Если загрузить данные в структуру для их хранения DataFrame из пакета pandas, код построения и проверки работоспособности модели будет примерно таким, как в следующем листинге.

Листинг 6.1. Модель на базе алгоритма логистической регрессии, предсказывающая получение чаевых

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, roc_auc_score
from pylab import *

sc = StandardScaler()                                    # Масштабирует данные, помещая их в диапазон от –1 до 1
data_scaled = sc.fit_transform(data[feats])

sgd = SGDClassifier(loss="modified_huber")               # Использует функцию потерь, которая хорошо обрабатывает выбросы

sgd.fit(                                                 # Обучает классификатор на тренировочных признаках и целевых данных
    data.ix[train_idx,feats],
    data['tipped'].ix[train_idx]
)

preds = sgd.predict_proba(                               # Создает прогноз на отложенных данных
    data.ix[test_idx,feats]
)

fpr, tpr, thr = roc_curve(                               # Вычисляет ROC-кривую и статистику AUC
    data['tipped'].ix[test_idx],
    preds[:,1]
)
auc = roc_auc_score(data['tipped'].ix[test_idx], preds[:,1])

plot(fpr,tpr)                                            # Выводит ROC-кривую
plot(fpr,fpr)
xlabel("False positive rate")
ylabel("True positive rate")
```

Последняя часть листинга 6.1 строит ROC-кривую для нашего первого простого классификатора. Кривая для отложенных данных показана на илл. 6.7.

Ничего не поделаешь, производительность этого классификатора никуда не годится! Значение AUC на отложенных данных, равное 0,51, показывает, что модель работает не лучше, чем случайное гадание (подбрасывание монетки, у которой сторона «чаевые» имеет вес 95%, сторона «отсутствие чаевых» — вес 5%, для предсказания результатов каждой поездки). К счастью, мы начали с самого простого варианта и модель легко усовершенствовать.



Илл. 6.7. Кривая рабочей характеристики приемника для классификатора чаевые / отсутствие чаевых на базе алгоритма логистической регрессии. Параметр AUC, равный 0,5, показывает, что наша модель работает не лучше, чем случайное гадание

6.2.2. Нелинейный классификатор

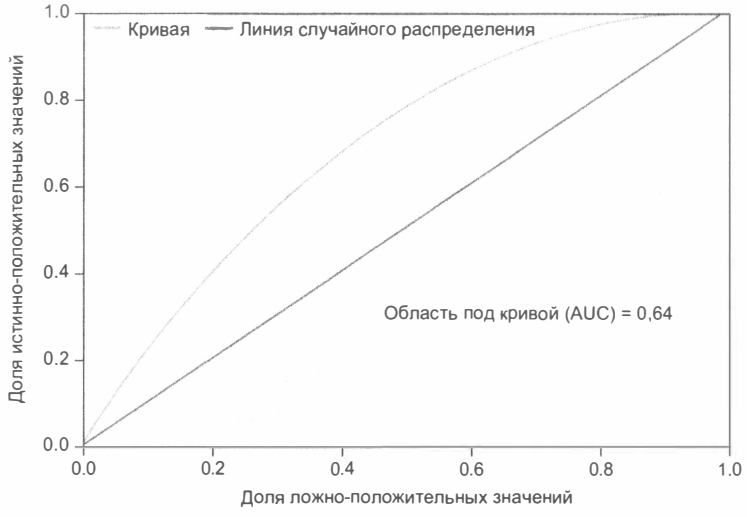
Первым делом возьмем за основу другой алгоритм, на этот раз нелинейный. Непродуктивность первой попытки наводит на мысли, что линейная модель тут просто не подходит; расчет вероятности чаевых — это сложный процесс! Поэтому воспользуемся нелинейным алгоритмом «случайный лес», который известен точностью прогнозирования на реальных данных. Вы можете выбрать любой другой алгоритм (см. приложение) и даже построить несколько моделей на базе различных алгоритмов и сравнить их друг с другом. Это задание для самостоятельной работы. Ниже показан код построения модели (связанный с предыдущей моделью).

Листинг 6.2. Модель на базе алгоритма «случайный лес»

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, roc_auc_score
from pylab import *

rf = RandomForestClassifier(n_estimators=100)
```

```
rf.fit(data.ix[train_idx,feats], data['tipped'].ix[train_idx])
preds = rf.predict_proba(data.ix[test_idx,feats])

fpr, tpr, thr = roc_curve(data['tipped'].ix[test_idx], preds[:,1])
auc = roc_auc_score(data['tipped'].ix[test_idx], preds[:,1])

plot(fpr,tpr)                                    | Выводит ROC-кривую
plot(fpr,fpr)
xlabel("False positive rate")
ylabel("True positive rate")

fi = zip(feats, rf.feature_importances_)         | Важность
fi.sort(key=lambda x: -x[1])                     | признаков
fi = pandas.DataFrame(fi, columns=["Feature","Importance"])
```

Результаты кода из листинга 6.2 показаны на илл. 6.8. Мы видим значительное увеличение точности на отложенных данных — теперь параметр AUC имеет значение 0,64. Очевидно, что в наборе данных присутствует доступный для прогнозирования сигнал. Некоторые комбинации входных признаков позволяют заранее узнать, получит ли водитель такси чаевые от пассажира. Если повезет, проектирование признаков и оптимизация модели обеспечат еще более высокую точность прогнозирования.



Илл. 6.8. Кривая рабочей характеристики приемника для модели на базе нелинейного алгоритма «случайный лес». Параметр AUC значительно увеличился: значение 0,64 показывает, что, скорее всего, в наборе данных присутствует реальный сигнал

Еще в этой умеренно прогностической модели можно определить важность различных признаков. Это требуется по следующим причинам:

- Можно идентифицировать вводящие в заблуждение признаки (например, проблему с оплатой по безналичному расчету) и внести коррективы в модель.
- Появляется отправная точка для проектирования признаков. Скажем, если самыми важными признаками окажутся широта и долгота, ими можно воспользоваться для получения других признаков, например расстояния от площади Таймс-сквер. Если признак, который вы считали важным, *не* появился в верхней части списка, имеет смысл проанализировать, визуализировать и, возможно, очистить или преобразовать его.

Иллюстрация 6.9 (также сгенерированная кодом из листинга 6.2) демонстрирует список признаков и их относительную важность для модели на базе алгоритма «случайный лес». Как видите, самыми важными оказались такие признаки, как координаты посадки и высадки, а также время поездки, расстояние и ее стоимость. Может оказаться, что пассажиры в некоторых частях города менее снисходительны к медленным, дорогим поездкам. Более подробно полученные результаты будут рассматриваться в разделе 6.2.5.

	Feature	Importance
0	dropoff_latitude	0.165411
1	dropoff_longitude	0.163337
2	pickup_latitude	0.163068
3	pickup_longitude	0.160285
4	trip_time_in_secs	0.122214
5	trip_distance	0.112020
6	fare_amount	0.067795
7	passenger_count	0.017850
8	surcharge	0.014259
9	rate_code	0.006974
10	tolls_amount	0.004067
11	mta_tax	0.002720

Илл. 6.9. Важные признаки модели на базе алгоритма «случайный лес». Похоже, что основными признаками являются координаты места посадки и высадки пассажира

Теперь, когда алгоритм выбран, нужно задействовать все необработанные признаки, то есть не только численные, но и категориальные столбцы.

6.2.3. Добавление категориальных признаков

Еще до проектирования признаков можно увеличить точность прогнозирования путем несложной обработки данных.

В главе 2 были показаны приемы работы с категориальными признаками. Некоторые ML-алгоритмы умеют обрабатывать их в исходном виде, но в данном случае мы воспользуемся популярным приемом «буленизации» — для каждой возможной категории рассматриваемого признака создадим столбец из значений 0 и 1. Это позволит обработать категориальные данные любому ML-алгоритму.

Код преобразования всех категориальных признаков показан в следующем листинге.

Листинг 6.3. Преобразование категориальных признаков в численные

```
def cat_to_num(data):
    categories = unique(data)
    features = {}
    for cat in categories:
        binary = (data == cat)
        features["%s:%s"%(data.name, cat)] = binary.astype("int")
    return pandas.DataFrame(features)

payment_type_cats = cat_to_num(data['payment_type'])
vendor_id_cats = cat_to_num(data['vendor_id'])
store_and_fwd_flag_cats = cat_to_num(data['store_and_fwd_flag'])
rate_code_cats = cat_to_num(data['rate_code'])

data = data.join(payment_type_cats)
data = data.join(vendor_id_cats)
data = data.join(store_and_fwd_flag_cats)
data = data.join(rate_code_cats)
```

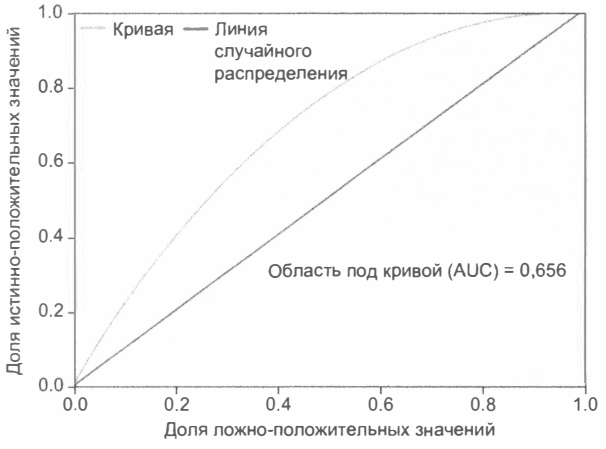
Функция преобразования категориального столбца в набор численных столбцов

Преобразует четыре категориальных признака из набора данных в численные

Добавляет преобразованные данные в полный набор, используемый для обучения и тестирования

После создания «буленизированных» столбцов снова пропустим наши данные через код из листинга 6.2. Полученная ROC-кривая и список важности признаков показаны на илл. 6.10. Обратите внимание,

что значение параметра AUC на отложенных данных слегка возросло с 0,64 до 0,656.



	Feature	Importance
0	dropoff_latitude	0.163023
1	pickup_latitude	0.161114
2	dropoff_longitude	0.160988
3	pickup_longitude	0.158672
4	trip_time_in_secs	0.111172
5	trip_distance	0.106693
6	fare_amount	0.067567
7	passenger_count	0.019286
8	surcharge	0.010330
9	payment_type:NOC	0.008361
10	payment_type:CRD	0.008247

Новые признаки

Илл. 6.10. Список важности признаков и ROC-кривая для модели на базе алгоритма «случайный лес» после преобразования всех категориальных признаков в булевские столбцы (0/1), по одному на категорию на каждый признак. Новые признаки добавляют в таблицу полезную информацию, так как значение параметра AUC возросло по сравнению с предыдущей, не содержавшей категориальных признаков моделью

Так как производительность модели возрастает, можно рассмотреть дополнительные факторы. Мы пока еще не дошли до проектирования признаков, так как показанные выше преобразования данных считаются обычной предварительной обработкой.

6.2.4. Добавление временных признаков

Пришло время поработать с данными для получения новых признаков, то есть заняться их *проектированием*. В предыдущей главе вы видели, как даты и временные метки преобразовывались в численные столбцы. Можно предположить, что на количество оставляемых пассажирами чаевых влияет время дня или день недели.

Следующий листинг демонстрирует код получения этих признаков.

Листинг 6.4. Признаки, связанные с датой и временем

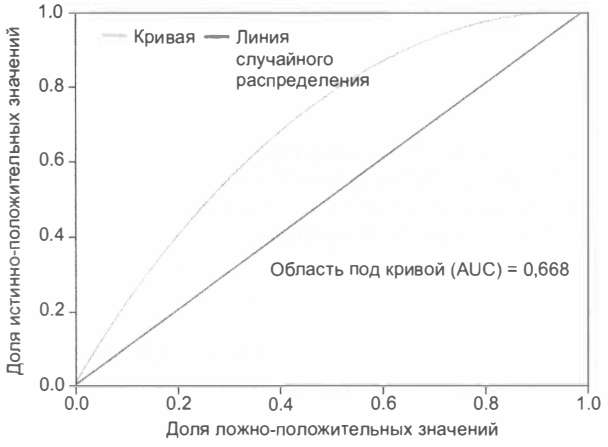
```
# Datetime features (hour of day, day of week, week of year)

pickup = pandas.to_datetime(data['pickup_datetime'])
dropoff = pandas.to_datetime(data['dropoff_datetime'])
data['pickup_hour'] = pickup.apply(lambda e: e.hour)
data['pickup_day'] = pickup.apply(lambda e: e.dayofweek)
data['pickup_week'] = pickup.apply(lambda e: e.week)
data['dropoff_hour'] = dropoff.apply(lambda e: e.hour)
data['dropoff_day'] = dropoff.apply(lambda e: e.dayofweek)
data['dropoff_week'] = dropoff.apply(lambda e: e.week)
```

Преобразует текстовые столбцы дата-время в реальные даты и реальное время

Добавляет к каждому времени посадки такие признаки, как час, день и неделя

Добавляет к каждому времени высадки такие признаки, как час, день и неделя

Добавим новые признаки в набор и построим еще одну модель. Еще раз пропустим данные через код из листинга 6.2, чтобы получить ROC-кривую и значения важности признаков, показанные на илл. 6.11.



	Feature	Importance
0	dropoff_latitude	1.342966e-01
1	dropoff_longitude	1.332890e-01
2	pickup_latitude	1.326898e-01
3	pickup_longitude	1.314082e-01
4	trip_time_in_secs	1.000610e-01
5	trip_distance	9.924300e-02
6	fare_amount	7.099682e-02
7	dropoff_hour	3.942205e-02
8	pickup_hour	3.924598e-02
9	pickup_day	2.417833e-02
10	dropoff_day	2.413835e-02

Новые признаки

Илл. 6.11. Список важности признаков и ROC-кривая для модели на базе алгоритма «случайный лес», включающей в себя все категориальные признаки и дополнительные признаки, связанные с датой и временем

Видно, как по мере дополнительной обработки данных и проектирования признаков повышается точность модели. Сейчас уже можно предсказать, даст ли пассажир чаевые водителю, с точностью, значительно превышающей случайное угадывание. До этого момента мы совершенствовали модели путем улучшения данных. Но есть и другие способы:

- менять параметры модели, так как их значения по умолчанию далеко не всегда дают оптимальный результат;
- увеличивать размер набора данных.

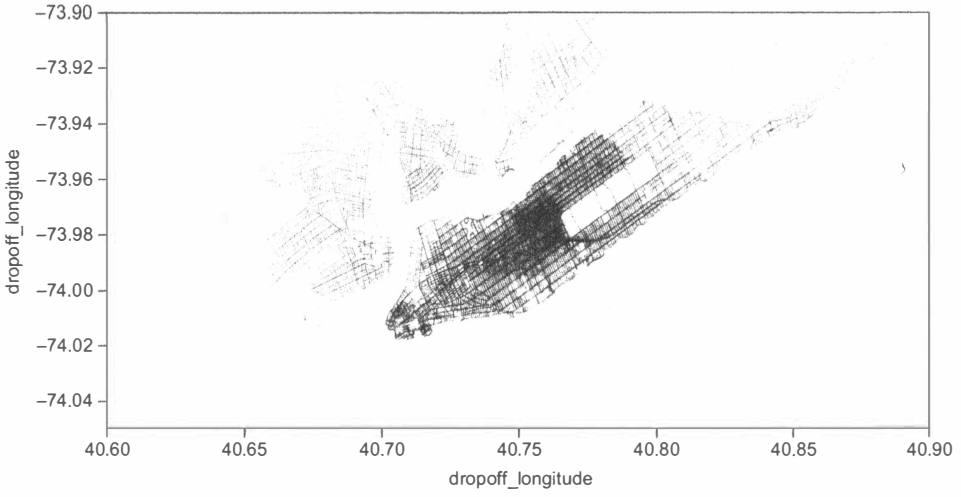
В этой главе в качестве примеров фигурировали небольшие выборки из набора данных. Это было сделано для того, чтобы алгоритмы могли их обработать даже на компьютерах со всего 16 Гбайт оперативной памяти. Масштабируемость методов будет обсуждаться в главах 9 и 10, а пока вы можете самостоятельно поработать с данными и еще сильнее увеличить точность модели после перекрестной проверки!

6.2.5. Аналитическая оценка модели

Интересно посмотреть на данные в процессе построения модели, предсказывающей определенный результат. Список важности показывает, какие признаки обладают максимальным прогностическим потенциалом. Воспользуемся ими и посмотрим на данные с новой стороны. Во время первой, неудачной попытки именно изучение списка важности признаков помогло обнаружить проблему в данных. Сейчас, когда у нас есть работающая модель, применим этот список для получения новых визуализаций.

На всех этапах рассмотрения модели самыми важными оказывались такие признаки, как координаты мест посадки и высадки. Иллюстрация 6.12 показывает географическое распределение всех мест высадки — как тех, где водитель получал чаевые, так и тех, где этого не происходило.

Иллюстрация 6.12 демонстрирует интересную тенденцию — отсутствие чаевых при высадке ближе к центру. Почему так получается? Возможно, из-за множества машин в центре такси едет медленно, и пассажирам это не нравится. А может, всё дело в том, что эта часть города наводнена туристами. Много тут и работников финансовой сферы, но деловая часть все-таки лежит в южной части Манхэттена. Так что еще одной причиной сдвига в данных вполне может оказаться отношение к чаевым, отличающееся от принятого в США. В ряде азиатских стран практически не принято давать на чай, во многих европейских странах на чай оставляют куда меньше и намного реже дают на чай таксистам. Изучая наш набор данных, можно прийти и к другим интересным выводам. То есть реальные данные позволяют многое понять о мире и генерирующих эти данные людях.



Илл. 6.12. Географическое распределение мест высадки

6.3. Заключение

В этой главе мы предложили вам реальный набор данных и сформулировали задачу, позволяющую применить полученные в предыдущих пяти главах знания. Был рассмотрен весь рабочий процесс ML с предварительной подготовкой данных, проектированием признаков и множественными итерациями модели на стадиях построения, оценки, оптимизации и предсказания. Вот основные положения, которые следует запомнить:

- Так как все больше организаций продуцирует огромное количество данных, все больше этих данных появляется в открытом доступе.
- В открытом доступе появились записи обо всех поездках в такси в Нью-Йорке за 2013 год.
- Реальные данные зачастую запутанны. Разобраться в них помогают различные техники визуализации и знание предметной области. Не попадайтесь на сценарии, которые слишком хороши, чтобы быть правдой, и не делайте преждевременных выводов.
- Начинайте итерации с простейшей модели. Не тратьте время на преждевременную оптимизацию. Увеличивайте сложность постепенно.

- Делайте выбор и двигайтесь вперед; например, алгоритм нужно выбирать на ранней стадии. В идеальном мире на каждом шаге итеративного процесса перебирались бы все возможные комбинации, но при этом пришлось бы многое исправлять, чтобы двигаться дальше.
- Производите аналитическую оценку модели и данных, чтобы больше узнать об исследуемой области и, возможно, еще больше усовершенствовать модель.

6.4. Терминология

Термин	Определение
Открытые данные (open data)	Данные, которые учреждения и организации выкладывают в общий доступ
FOIL	Свобода распространения информации
too-good-to-be-true scenario	Если точность модели превосходит ваши ожидания, возможно, некоторые признаки или особенности данных заставляют ее «жульничать»
Преждевременные предположения (premature assumptions)	Неподтвержденные гипотезы по поводу данных, которые могут повлиять на ваше восприятие результатов

7 Усовершенствованное проектирование признаков

В этой главе:

- ✓ применение усовершенствованных концепций проектирования признаков для увеличения точности работы систем с машинным обучением;
- ✓ извлечение из текста значимых признаков с помощью техник обработки обычных текстов;
- ✓ извлечение значений из изображений и применение их в качестве признаков в проектах с машинным обучением.

С базовыми представлениями о проектировании признаков вы познакомились в главе 5, а материал главы 6 показал процесс применения полученных знаний к реальным данным. А сейчас мы познакомим вас с усовершенствованными техниками, применимыми к распространенным типам данных. Самые важные из этих типов — тексты и изображения. В этой главе вы узнаете, как извлечь признаки из текста и картинок, чтобы добавить их в проекты с машинным обучением.

7.1. Более сложные текстовые признаки

С простым проектированием признаков для текстовых данных вы сталкивались в главе 5. Пришло время более подробно обсудить стоящие за техниками проектирования идеи и представить усовершенствованные концепции, которые позволят еще больше увеличить точность работы моделей.

Напоминаем, что работа с текстом сводится к извлечению из фрагментов произвольной длины общего набора признаков. В главе 5 впервые появилась концепция «мешка слов», состоящая в подсчете числа вхождений в текст каждого слова, после чего N чаще всего встречающихся слов превращается в N новых признаков. Такое преобразование текста в машинные данные называется *обработкой естественного языка* (NLP — natural language processing).

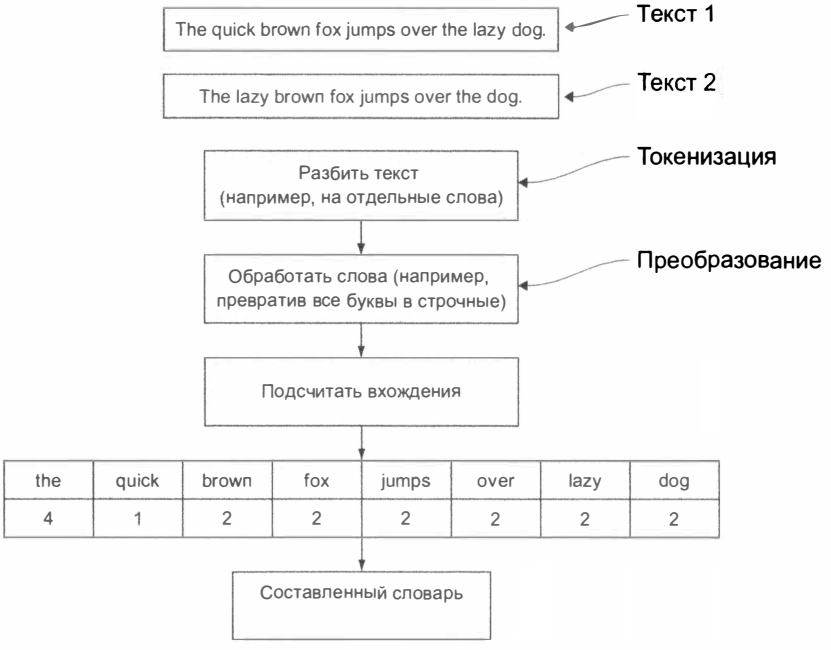
7.1.1. Модель «мешок слов»

«*Мешок слов*» (bag of words) — одна из простейших и вместе с тем наиболее широко используемых техник NLP. Именно с нее имеет смысл начинать решение любой связанной с текстом задачи. Кроме того, она служит основой множества других, более совершенных методов, с которыми вы познакомитесь чуть позднее. Информацию об этой модели мы разобьем на две части: сначала рассмотрим токенизацию и преобразование, а затем векторизацию.

Токенизация и преобразование

Разбиение текста на фрагменты называется *токенизацией*. Чаще всего разбиение происходит по отдельным словам, но в некоторых случаях (например, в языках с символьной письменностью) возможно разбиение

по символам. Также можно осуществлять разбиение по парам или группам слов или даже по какому-то более сложному принципу. Группы из n слов называются *n-граммами*. Комбинации из двух и трех слов будут биграммами и триграммами соответственно (это самые распространенные варианты разбиений после разбиения по словам). К биграммам на илл. 7.1 относятся сочетания «the lazy», «brown fox» и т. п. Триграммы состоят из словосочетаний «brown fox jumps» и «jumps over the»[1].



Илл. 7.1. Начальные этапы работы алгоритма «мешок слов»

Иногда моделям идет на пользу расширение до набора слов, так как это дает возможность лучше понять контекст. Но, как правило, такой подход сопровождается существенным увеличением количества признаков. Поэтому на практике обычно начинают с разбиения по отдельным словам. Если вы хотите рассмотреть n-граммы более высокого порядка, первым

[1] В примере рассматривается стандартный текст для визуализации шрифтов The quick brown fox jumps over the lazy dog (Шустрая бурая лиса прыгает через ленивую собаку) и его перифраз The lazy brown fox jumps over the dog (Ленивая бурая лиса прыгает через собаку). Это панграмма, то есть фраза, содержащая все буквы английского алфавита. — *Примеч. пер.*

делом убедитесь, что выбранный алгоритм умеет работать с разреженными данными. В следующем разделе мы поговорим об этом более подробно.

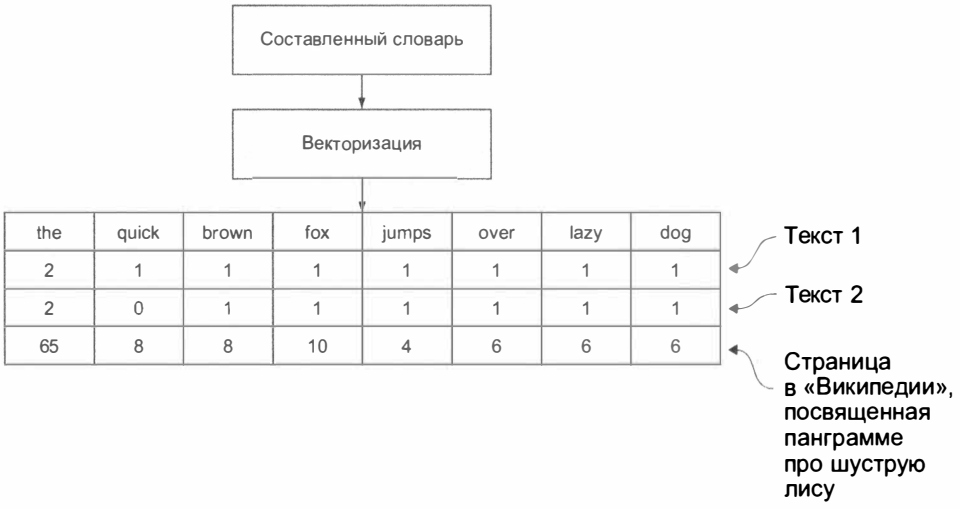
Следующий шаг алгоритма «мешок слов» — преобразования всех извлеченных из текста токенов, если таковые требуются. Например, все слова можно привести к нижнему регистру, чтобы не приходилось создавать отдельные признаки для слов «fox» и «Fox», добавляя в модель шум. Но бывают ситуации, когда регистр имеет смысл сохранить (например, если имена собственные повторяются и имеют большую прогностическую ценность или если ВСЕ, НАБРАННОЕ ПРОПИСНЫМИ БУКВАМИ, имеет значение). *Процесс нахождения основы слова* (stemming), например, путем отбрасывания аффиксов также относится к мощным преобразованиям, позволяющим извлечь дополнительные сигналы из слов с одинаковыми значениями. Благодаря этому процессу слова «jump», «jumping», «jumps» и «jumped» будут отображены в итоговом словаре как токен «jump». Бывают тексты, для которых требуются и другие преобразования, например особая обработка цифр, знаков препинания и специальных символов.

Когда преобразования будут закончены, можно определить словарь, на базе которого будут генерироваться признаки. Для проектов с машинным обучением принято устанавливать ограничение на количество признаков, то есть на количество слов в словаре. Поэтому выполняется сортировка по количеству вхождений каждого слова и используются только верхние N слов.

Векторизация

После составления словаря мы получаем набор чисел, показывающий, сколько раз каждое словарное слово появляется в тексте. Этот процесс, называемый *векторизацией*, показан на илл. 7.2.

Но есть один момент, которого мы пока не касались. Большая часть слов в произвольном тексте никак не влияет на способность человека понять основную тему этого текста. Это «заполнители», к ним относятся артикли, предлоги и глаголы-связки. В NLP-исследованиях они называются *шумовыми*, или *стоп-словами*, и обычно удаляются из словаря, так как не несут прогностической ценности и отвлекают внимание от более значимых слов, важных с точки зрения машинного обучения. После



Илл. 7.2. Словарь позволяет превратить любой текст в список чисел. Строки содержат число вхождений слов в две фразы с рис 7.1 и в статью в «Википедии», посвященную панграмме про лисицу

сортировки по числу вхождений от шумовых слов обычно избавляются, отбрасывая все слова, частота появления которых превышает некий заданный порог. Пример демонстрируется на илл. 7.2; в более длинном тексте (третья строчка) артикль «the» попадается чаще, чем все остальные слова. Сложность состоит в том, чтобы выбрать пороговое значение, после которого слово становится шумовым. Большинство NLP-библиотек, такие как NLTK для языка Python, содержат встроенные списки стоп-слов для различных языков, что сильно упрощает жизнь. Но бывают ситуации, когда список стоп-слов отличается от универсального, и тогда приходится выбирать порог вхождения (как правило, исключаются все слова, которые появляются в более чем 90% всех документов).

По илл. 7.2 этого не видно, но реальные словари состоят из множества слов, причем в тексте, для которого проектируются признаки, присутствует лишь небольшая их часть. Поэтому текстовые признаки обычно состоят из множества нулей, а признаки, построенные на базе «мешка слов», оказываются *разреженными*. Когда таких признаков много (у вас может быть более 1000 признаков с небольшим процентом отличных от нуля элементов), имеет смысл выбирать алгоритм, изначально умеющий их обрабатывать, или алгоритм, который в состоянии без потери точности работать с множеством признаков низкой значимости.

Например, работать с разреженными данными умеет наивный байесовский классификатор из библиотеки scikit-learn для языка Python. Еще известны своей способностью хорошо обрабатывать признаки низкой значимости такие алгоритмы, как «случайный лес», хотя по поводу него существуют разные мнения. Эффективность различных методов всегда следует проверять с помощью техник оценки и оптимизации, описанных в главе 4.

7.1.2. Тематическое моделирование

Метод «мешка слов» прост для понимания и реализации. Но существуют и более сложные методы, которые могут значительно повысить точность ML-модели. Три таких метода мы сейчас рассмотрим.

Есть один аспект, связанный с процессом подсчета слов. Если какое-то слово (нешумовое) является общим для коллекции документов — например, слово «данные» в коллекции статей по машинному обучению, — информация о том, что в новом тексте оно тоже появляется, нам ничего не даст. Целесообразнее фокусироваться на относительно редких словах, обладающих большим прогностическим потенциалом. Поэтому принято масштабировать число вхождений слова путем деления на число вхождений этого слова во все документы коллекции. Так как мы хотим описать текст наилучшим образом с помощью одних только чисел, а слово, редко встречающееся в обучающем наборе, но частое в новом документе, скорее всего, сильнее отображает значение этого документа, предпочтительно уделить внимание именно этому слову.

Частота слова — обратная частота документа

Широко используемый алгоритм, который пытается решить вышеописанную задачу, называется *частота слова – обратная частота документа* (tf-idf — term frequency – inverse document frequency). Статистическая мера рассчитывается как произведение частоты слова (tf) и обратной частоты документа (idf).

Параметр tf можно получить разными способами, но проще всего взять число вхождений слова в конкретный документ. Распространены и другие версии множителя tf, например двоичная (1, если слово присутствует в документе, и 0 в противном случае) и логарифмическая (1 + log[tf]).

Обратная частота документа вычисляется как логарифм общего числа документов, поделенный на число документов, содержащих слово. В результате высокие значения получают относительно редкие слова. В простой форме уравнение tf-idf выглядит так:

$$\text{tf} - \text{idf}(\text{слово, документ, документы}) =$$
$$= \text{число (слов в документе)} \frac{\text{число (документов)}}{\text{число (документов со словом)}}.$$

Алгоритм tf-idf генерирует хорошие признаки из любого набора документов. Применяется он и для других целей, например для поиска. Так как для любого документа генерируется вектор чисел, можно подсчитать «расстояние» между документами, взяв за основу разницу между их векторными tf-idf-представлениями. Если поисковый запрос пользователя касается документа, этот способ позволяет определить расстояние между любыми двумя документами в рассматриваемом наборе данных и дать ответ на запрос в виде ранжированного списка. В следующем разделе листинг 7.1 показывает, как средствами библиотеки scikit-learn для языка Python из документов генерируются tf-idf-векторы. Там же находится описание более продвинутой техники, которая называется латентно-семантическим индексированием.

Латентно-семантический анализ

Латентно-семантический анализ (LSA — latent semantic analysis), который также называют *латентно-семантическим индексированием* (LSI — latent semantic indexing), представляет собой более продвинутый вариант тематического моделирования. Он более совершенен как концептуально, так и в плане вычислений. В основе метода лежит идея построения на базе «мешка слов» терм-документной матрицы, в которой каждому слову соответствует строка, а каждому документу — столбец. Затем элементы матрицы нормируются с помощью tf-idf-процесса, чтобы нивелировать влияние часто встречающихся слов.

В основе алгоритма LSA лежат так называемые понятия. *Понятием* (concept) называется подмножество сходных слов в корпусе документа. Например, с понятием «собака» могут быть связаны термы (в данном случае слова) «лай», «поводок» и «конура». Алгоритм не маркирует понятие «собака», он выявляет слова, появляющиеся в разных документах, и уста-

навливает, что они связаны с одним и тем же абстрактным понятием. При этом само слово «собака» может быть важным термином, связанным с понятием «собака». Такие темы считаются *скрытыми*, или *латентными*. Отсюда и возникло называние *латентно*-семантический анализ.

Алгоритм LSA использует для разбиения терм-документной матрицы (A) на три матрицы (T, S, D) такой известный математический инструмент, как *сингулярное разложение* (SVD — uses singular value decomposition)[1]. Здесь T — матрица, связывающая термины (например, «лай» и «конура») с понятиями (например, «собака»). D — матрица, связывающая отдельные документы с понятиями, которые впоследствии будут использоваться для извлечения признаков из LSA-модели. Матрица S состоит из сингулярных значений. В латентно-семантическом анализе они указывают на относительную важность терма в документе. Аналогично тому, как мы ограничивали количество признаков в алгоритмах «мешка слов» и tf-idf, сейчас мы можем выбрать верхние сингулярные значения, сделав пространство признаков более управляемым; напомним, что терм-документные матрицы (A) бывают крайне большими и разреженными.

Из верхних N компонентов сингулярного разложения генерируется N признаков для ML-модели. Для этого мы просто берем соответствующие строки матрицы D. При появлении в предсказании новых документов из ранее освоенной LSA-модели генерируется новый набор признаков путем умножения матриц: $D = A^{T}TS^{-1}$. Здесь A^{T} — число слов (или метрика tf-idf), подсчитанное из определенного для нового документа словаря, а T и S — полученные при сингулярном разложении матрицы.

Понять принцип латентно-семантического анализа позволяет линейная алгебра, но не все владеют ею в достаточной степени, чтобы проделать полагающиеся вычисления. К счастью, в рамках ML-проекта можно воспользоваться готовыми реализациями. Библиотека scikit-learn для языка Python содержит всю необходимую функциональность для (1) генерации терм-документной матрицы на базе метрики tf-idf, (2) разложения этой

[1] Для читателей, знакомых с методом главных компонент (который будет рассматриваться чуть ниже), заметим, что сингулярное разложение — это та же самая техника, которая используется для извлечения из набора данных координат для метода главных компонент. Латентно-семантический анализ можно представить как «метод главных компонент для “мешка слов”».

матрицы и (3) преобразования документов в вектора. Все это показано в следующем листинге.

Листинг 7.1. Латентно-семантический анализ с помощью библиотеки scikit-learn

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD

def latent_semantic_analysis(docs):
    tfidf = TfidfVectorizer()
    tfidf.fit(docs)
    vecs = tfidf.transform(docs)
    svd = TruncatedSVD(n_components=100)
    svd.fit(vecs)
    return svd.transform(vecs)
```

Инициализирует объект tf-idf с параметрами по умолчанию

Создает из документов tf-idf словарь

Генерирует матрицу признаков tf-idf на базе этого словаря

Инициализирует LSA-объект по 100 координатам

Выполняет сингулярное разложение

Вычисляет LSA-признаки для всех документов

Теперь рассмотрим расширения латентно-семантического анализа, которые в последнее время набирают популярность в области тематического моделирования.

Вероятностные методы

В основе латентно-семантического анализа лежит линейная алгебра (математические операции с векторами и матрицами), но аналогичные операции можно выполнить и вероятностными методами, которые моделируют каждый документ как статистическую комбинацию тематических распределений. Это достаточно сложные принципы, и углубляться в математические детали мы не будем, но для некоторых наборов данных вероятностный подход обеспечивает более высокую точность моделирования.

Речь в данном случае идет о вероятностном латентно-семантическом анализе (pLSA – probabilistic latent semantic analysis). Наиболее распространена версия, называемая *латентным размещением Дирихле* (LDA – latent Dirichlet analysis), в которой сделаны определенные допущения в отношении распределения тем. Предполагается, что документ описывается небольшим набором тем и любой терм (слово) можно сопоставить одной из них. На практике LDA дает хорошие результаты для различных наборов данных. Следующий листинг показывает реализацию LDA на языке Python с использованием библиотеки Gensim.

Листинг 7.2. Латентное размещение Дирихле на языке Python с использованием библиотеки Gensim

```
import gensim.models.ldamodel.LdaModel

def lda_model(docs):
    return LdaModel(docs, num_topics=20)

def lda_vector(lda_model, doc):
    return lda_model[doc]
```

Сначала нужно установить библиотеку Gensim командой pip install gensim

Строит LDA-модель, задавая количество подлежащих извлечению тем

Генерирует признаки для нового документа

Количество тем, которые будут использоваться в модели LDA, — это параметр, значение которого выбирается в зависимости от данных и поставленной задачи. Рекомендуем выбирать метрику производительности и оптимизировать свои модели, используя техники из главы 4. Отметим, что библиотека Gensim при появлении нового документа позволяет на лету обновлять модель LDA, если новые обучающие данные добавляются непрерывно. Рекомендуем ознакомиться и с другими интересными алгоритмами обработки естественного языка и тематического моделирования из этой библиотеки. В главе 10 некоторые из них будут применяться для решения реальной задачи. А сейчас рассмотрим другой метод извлечения признаков — расширение содержимого.

7.1.3. Расширение содержимого

Извлекать признаки из текста можно и другими способами. В этом разделе мы рассмотрим методы, которые увеличивают текст за счет другого текста (допускающего превращение в признаки) или предлагают другую полезную для решения поставленной ML-задачи информацию. Вот самые распространенные варианты.

Переходы по ссылкам

При построении ML-классификатора путем извлечения признаков из записей в социальной сети Twitter (например, для анализа тональности, классифицирующего запись как позитивную или негативную) часто становится проблемой ограничение в 140 символов. Для достижения желаемой точности просто не хватает информации.

Но многие записи в этой социальной сети содержат ссылки на внешние страницы с куда большим количеством текста. Этот текст можно ис-

пользовать для улучшения качества данных. Можно даже пройти по ссылкам, встречающимся уже на этой странице, еще больше увеличив корпус текста.

Расширение базы знаний

Более продвинутый метод расширения текста состоит в обнаружении именованных сущностей, после чего в текст добавляется информация об этих сущностях, взятая из имеющихся в Интернете баз знаний, таких как «Википедия». При этом в именованную сущность превращается любое слово или словосочетание, которому соответствует статья в «Википедии». К статье затем применяется любой из описанных в разделе 7.1.2 алгоритмов извлечения признаков из текста.

Эта нетривиальная задача была темой работы нескольких исследовательских групп. Например, проблему создают корни неоднозначных слов. Если слово имеет несколько значений, возникает риск расширить набор признаков за счет нерелевантной информации. В качестве решения предлагается, например, устранение неоднозначности с помощью все той же базы знаний. Прежде всего, имеет смысл предположить, что любое другое слово из записи в микроблоге будет встречаться в статье из «Википедии». Можно воспользоваться графом ссылок и посмотреть, насколько близко две именованные сущности располагаются в базе знаний. В качестве примера рассмотрим записи, содержащие слово «Tesla». Некоторые из них посвящены компании по производству электромобилей, в то время как другие — изобретателю Николе Тесле. Если запись заодно содержит слова «автомобиль» или «модель», скорее всего, она рассказывает о фирме. Если же там встречается имя «Эдисон», это запись о человеке (Тесла и Эдисон вместе работали в Нью-Йорке в 1884 году).

Метатекстовые признаки

Еще одна техника получения дополнительных ценных данных — анализ текста на наличие *метапризнаков.* В отличие от ранее обсуждавшихся техник, эти типы признаков зависят от конкретной задачи.

В качестве примера снова рассмотрим записи в социальной сети Twitter. Они содержат разные типы специфических данных, которые могут оказаться полезными, например тематические метки (хештеги) и упоминания, а также метаданные, такие как число ретвитов и добавлений

в избранное. Возьмем другой пример связанного с Интернетом текста — гиперссылки. Из них можно извлекать сведения о домене верхнего уровня. Обычный же текст дает нам информацию о количестве слов, знаков или специальных символов в разных языках. Для решения всех этих задач применяется ML-классификатор, результат работы которого другой классификатор берет в качестве признаков.

Правильно выбрать метатекстовые признаки помогают воображение и знание связанной с поставленной задачей предметной области. Как вы помните, рабочий процесс машинного обучения цикличен. После появления нового признака цикл повторяют и проверяют, как изменилась точность прогнозирования.

Из текста извлекают данные и других типов. Текст может содержать сведения о дате и времени, которые могут пригодиться в ML-модели. Попадается такая информация и в текстовых метаданных. Извлечение таких признаков описано в главе 5.

При анализе веб-страниц периодически приходится сталкиваться с изображениями или видеороликами, помогающими понять контекст текста. В этих случаях для извлечения признаков требуются еще более продвинутые техники, о которых мы и поговорим в следующем разделе.

7.2. Признаки, извлекаемые из изображений

В числе прочего человеческий интеллект базируется на таких вещах, как зрительное и пространственное восприятие, а также способность распознавать шаблоны и образы в изображениях и трехмерных сценах, с которыми мы сталкиваемся каждый день. Именно эти умения во многом обусловливают способ нашего мышления. Компьютеры же воспринимают информацию как биты и их визуальные аналоги — пиксели. Раньше этот факт серьезно ограничивал возможности компьютеров, как только дело доходило до распознавания зрительных образов. Лишь с появлением сложных алгоритмов компьютерного зрения и искусственного интеллекта, которые, возможно, и дали толчок к возникновению машинного обучения, теоретики и практики смогли приблизиться к уровню человека, пусть зачастую исключительно в узкоспециализированных областях. С другой стороны, приблизившись к человеческому уровню точности распознавания образов с техниками компьютерного зрения и машинного

обучения, можно воспользоваться некоторыми преимуществами большинства компьютерных систем — масштабируемостью, доступностью и воспроизводимостью.

В этом разделе мы рассмотрим способы извлечения признаков из изображений. Начнем с рассмотрения простых признаков, к которым относятся пиксели, цвета и метаданные изображения.

7.2.1. Простые признаки

Самый простой способ работы с изображениями заслуживает рассмотрения не только потому, что в некоторых случаях им можно и ограничиться, но и потому, что он демонстрирует реальные преимущества машинного обучения перед обработкой вручную или обычными статистическими методами. Числовые значения пикселей изображения используются как признаки в ML-модели.

На практике создается один ряд из пикселей, то есть двумерное изображение превращается в одномерное. В случае цветной картинки у нас, по сути, получается три изображения в одном (красный, синий и зеленый каналы). Числовые значения пикселей обычно лежат в диапазоне от 0,0 до 1,0, или от 0 до 255 (для 8-битных изображений). Наверное, вы уже догадались, что для любой картинки такой подход приводит к появлению тысяч или даже миллионов признаков, что сильно увеличивает требования к вычислительным мощностям и потенциально ведет к переобучению, что, в свою очередь, влияет на точность. Именно поэтому на практике этот подход применяется нечасто. Но вы, наверное, удивитесь, насколько хорошо он работает для некоторых задач даже без сложного проектирования признаков, например для классификации изображений, полученных в помещении и на улице.

В принципе, в пикселях закодирована вся информация. Но если по причинам, связанным с производительностью, такой подход неприемлем, изображение нужно представить с помощью меньшего количества признаков, достаточного для точного решения конкретной задачи. Со сходными проблемами мы уже сталкивались в предыдущем разделе, посвященном текстовым признакам, а также при рассмотрении многих других техник проектирования признаков. К концу раздела 7.2.2 вы познакомитесь с новыми методами автоматического извлечения признаков,

но в большинстве связанных с графикой ML-проектов применяются уже знакомые вам техники.

Признаки, связанные с цветом

Предположим, нужно распределить картинки по категориям в соответствии с изображенным на них пейзажем. В качестве категорий возьмем небо, горы и траву. Для решения такой задачи имеет смысл представить изображения в виде составляющих их цветов. Для каждого канала цвета можно рассчитать простые статистические параметры, например *среднее*, *медиану*, *моду*, *среднеквадратичное отклонение*, *коэффициент асимметрии* и *коэффициент эксцесса*. Для обычного RGB-изображения это даст 6 × 3 = 18 признаков.

Другой набор признаков, связанный с цветом, — цветовой диапазон изображения. Таблица 7.1 содержит список возможных цветовых диапазонов, покрывающих большую часть цветового пространства.

Таблица 7.1. Примеры признаков, связанных с цветовыми диапазонами. К делителю добавляется 1, чтобы избежать исчезновения значений при делении на 0

Цветовой диапазон	Определение
Красный	Разность максимального и минимального значений в красном канале
Красно-синий	Красный диапазон / (разность максимального и минимального значений в синем канале плюс 1)
Сине-зеленый	(Разность минимального и максимального значений в синем канале) / (разность минимального и максимального значений в зеленом канале плюс 1)
Красно-зеленый	Красный диапазон / (разность максимального и минимального значений в зеленом канале плюс 1)

Признаки, связанные с метаданными изображений

Кроме цветовой информации изображение может содержать метаданные, которые тоже помогают в решении задач. К примеру, большинство фотографий включает в себя записываемые в момент фотографирования данные EXIF. Для построения модели, предсказывающей, сочтет ли зри-

тель снимок интересным или красивым, алгоритму может пригодиться информация о модели камеры и объектива, величине диафрагмы и коэффициенте масштабирования. Таблица 7.2 содержит список извлекаемых из метаданных признаков, которые могут оказаться полезными.

Таблица 7.2. Признаки, получаемые из метаданных изображений

Признак	Определение
Производитель	Компания, которая выпустила камеру
Ориентация	Расположение камеры (альбомное или портретное)
Дата-время	Время создания снимка (используйте признаки даты-времени, описанные в главе 5)
Сжатие	Способ сжатия изображения (обычно JPEG или RAW)
Разрешение	Количество пикселей на единицу площади изображения
Соотношение геометрических размеров	Параметр, получаемый делением ширины изображения на его высоту
Выдержка	Время экспозиции в долях секунды
Диафрагма	Величина f-числа (например, 2,8 или 4,0)
Вспышка	Была ли включена вспышка
Фокусное расстояние	Расстояние от объектива до точки фокусировки

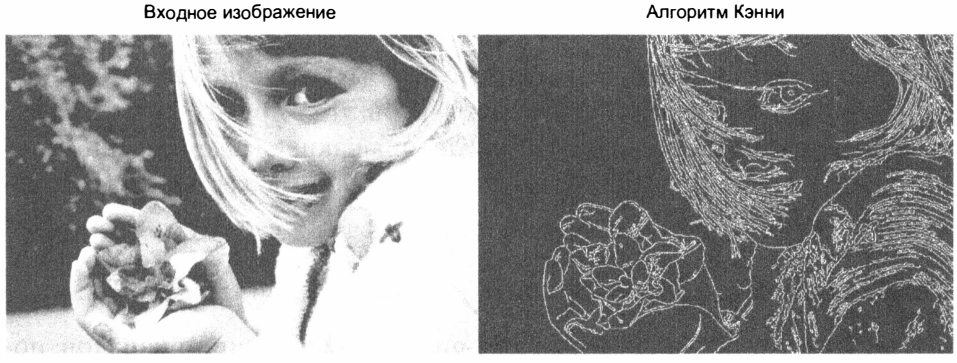
Эти простые признаки позволяют решать задачи с машинным обучением, в которых часть данных представлена в виде картинок. Разумеется, при этом не учитываются ни формы, ни объекты, которые очевидно важны в задачах на классификацию изображений! Более продвинутые техники компьютерного зрения, широко используемые для представления объектов и форм, мы рассмотрим в следующем разделе.

7.2.2. Извлечение объектов и форм

До этого момента при извлечении информации из изображений мы не касались таких вещей, как объекты или формы. Пришло время поговорить о способах представления форм в виде численных признаков, допускающих автоматическое извлечение статистическими и вычислительными методами.

Распознавание контуров

Наверное, самый простой способ представления содержащихся в изображении форм — обнаружение их границ с последующей генерацией признаков из этих данных. Пример *распознавания контуров* показан на илл. 7.3.



Илл. 7.3. Применение алгоритма Кэнни к фотографии девочки (входное изображение слева) дает новое бинарное изображение (справа), состоящее только из обнаруженных контуров. (Фото Джона Маклуна со страницы https://commons.wikimedia.org/w/index.php?curid=44894482 лицензия CC BY-SA 3.0)

Существует несколько алгоритмов распознавания контуров. Чаще всего применяются алгоритмы *Собеля* и *Кэнни*. Иллюстрация 7.3 демонстрирует результат применения алгоритма Кэнни.

ОБРАБОТКА ИЗОБРАЖЕНИЙ С ПОМОЩЬЮ БИБЛИОТЕКИ SCIKIT-IMAGE ДЛЯ ЯЗЫКА PYTHON

Мы уже несколько раз упоминали библиотеку scikit-learn как средство быстрого доступа ко многим алгоритмам машинного обучения. Ее аналог для компьютерного зрения и обработки изображений — библиотека scikit-image. Она позволяет быстро и легко воспользоваться обсуждавшимися в данном разделе алгоритмами.

Если вы пользуетесь системой управления пакетами Pip, библиотека scikit-image устанавливается командой:

```
$ pip install scikit-image
```

Вот простой пример применения этой библиотеки для распознавания контуров:

```
>>> import skimage
>>> image = skimage.data.camera()
>>> edges = skimage.filter.sobel(image)
```

Теперь, когда у нас есть извлеченные из изображения контуры, можно сгенерировать из них признаки. Проще всего рассчитать общий вес контуров на картинке. Если `edges` — изображения с контурами, а `res` — разрешение картинки, уравнение будет выглядеть так:

$$\text{edge\_score} = \frac{\sum \text{edges}}{\text{res}_x \times \text{res}_y}.$$

Вместе с другими признаками это может быть полезно, например при определении интересующих нас объектов. Для других задач на базе контуров можно сгенерировать другие признаки. Скажем, можно было вычислить предшествующий вес контуров для множественных частей изображения в сетке.

Более сложные признаки форм

Существуют и более сложные алгоритмы извлечения признаков, позволяющие распознавать конкретные формы и объекты. Один из них — *гистограмма направленных градиентов* (HOG — histogram of oriented gradients). В машинном обучении такие алгоритмы применяются для распознавания человеческих лиц или конкретных животных.

Алгоритм HOG представляет собой многоступенчатое применение различных техник обработки изображения. Его целью является описание форм и объектов в разных частях изображения, не слишком чувствительных к небольшим изменениям масштаба и ориентации. Это достигается следующим способом:

1. Вычисляется поле градиентов изображения (то есть определяется, в каком направлении «движутся» контуры изображения).
2. Изображение делится на маленькие блоки, называемые *ячейками*.
3. Вычисляется ориентация градиентов в ячейках.
4. Вычисляется гистограмма этих ориентаций в отдельных ячейках.

Обычно ячейки определяют блоки большего размера, которые используются для нормализации значений градиента в ячейках. Это позволяет избежать слишком большой чувствительности к изменениям освещения или теней. После этого каждую ячейку можно свести к списку признаков, описывающих имеющиеся формы.

Разумеется, вас интересует полезность рассмотренных алгоритмов с практической точки зрения, поэтому давайте рассмотрим уже готовое решение. Библиотека scikit-image для языка Python оснащена легкой в применении версией алгоритма HOG. Следующий листинг демонстрирует вычисление HOG-признаков для изображения. Иллюстрация 7.4 показывает результат применения HOG-преобразования к фотографии Айлин Коллинз — первой женщины-командира многоразового космического корабля.

Листинг 7.3. Гистограмма направленных градиентов с помощью библиотеки scikit-image

```
import skimage

image = skimage.color.rgb2gray(skimage.data.astronaut())
hog = skimage.feature.hog(image, orientations=9, pixels_per_cell=(8,8),
        cells_per_block=(3,3), normalise=True, visualise=True)
```

Илл. 7.4. Применение преобразования HOG. Изображение взято со страницы http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html#sphx-glr-auto-examples-features-detection-plot-hog-py

Как видите, с помощью алгоритма HOG легко получить признаки, указав количество рассматриваемых ориентаций, размер ячеек в пикселях и количество ячеек в блоке, а также отметив, следует ли нормализовать и визуализировать результат.

Такие признаки позволяют распознавать объекты на картинках. Как и в остальных случаях, алгоритм HOG далеко не всегда справляется с задачей — проблемы могут возникнуть, скажем, при значительном изменении ориентации объекта. Как обычно, следует сначала корректно протестировать ML-систему, определяя целесообразность применения алгоритма к поставленной задаче.

Уменьшение размерности

Практически всегда при извлечении признаков возникает задача уменьшения размерности. Исключением являются разве что описанные в предыдущих разделах методы расширения содержимого. Для решения такой задачи существует несколько техник, из которых наиболее широко применяется *метод главных компонент* (PCA — principal component analysis).

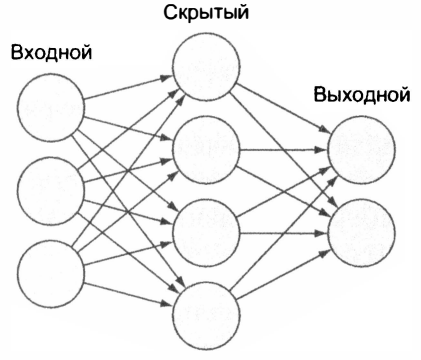
Этот метод позволяет выбирать из набора «типичные» изображения, которые можно использовать как строительные блоки для представления исходных изображений. Комбинация первой пары главных компонент восстанавливает большую часть тренировочных изображений, в то время как последующие компоненты охватывают реже встречающиеся структуры. Признаки для нового изображения генерируются определением «расстояния» до главной картинки, что позволяет представить новое изображение по отношению к главному при помощи всего одного числа. Количество главных компонент может быть произвольным и зависеть только от конкретной задачи.

Метод главных компонент является линейным алгоритмом, он не может представлять изначально нелинейные данные. Но существует ряд расширений для PCA или других вариантов уменьшения нелинейной размерности. В качестве примера можно называть *диффузные отображения* (diffusion maps).

Автоматическое извлечение признаков

В мире искусственных нейронных сетей началось возрождение. Изобретенные в 80-х годах прошлого века при попытках смоделировать протекающие в мозге процессы, эти сети были центральным компонентом

такой области, как искусственный интеллект, породившей машинное обучение в том виде, как мы его сегодня знаем. Несколько десятилетий они считались полезными методами для решения некоторых ML-задач. Но сложность конфигурации и интерпретации, проблемы с переобучением и меньшая вычислительная масштабируемость таких сетей постепенно превратили их в средство, к которому прибегали только в исключительных обстоятельствах. Благодаря прорывам в области машинного обучения эти сложности ушли в прошлое. *Глубокие нейронные сети* (DNN – deep neural net) в настоящее время считаются инструментом для решения множества ML-задач, особенно связанных с изображениями, видео или записями голоса. Иллюстрация 7.5 показывает компоновку нейронной сети.



Илл. 7.5. Простая искусственная нейронная сеть. Глубокие нейронные сети состоят из многих слоев таких простых сетей. (Изображение из «Википедии».)

Каждый слой глубокой нейронной сети способен дать набор новых признаков. Веса между узлами определяют важность этих признаков для следующего слоя и т. д. Раньше такой подход создавал предпосылки к переобучению, но недавно разработанные техники позволяют удалять связи между узлами, сохраняя уровень точности и одновременно уменьшая риск переобучения.

Варианты применения глубоких нейронных сетей, также известные как *глубокие сети доверия* (deep belief networks), или *глубокое обучение* (deep learning), все еще остаются относительно новой областью. Рекомендуем вам самостоятельно познакомиться с этими разработками.

7.3. Признаки временных рядов

Иногда наборы данных, накапливаемые современными системами сбора, имеют вид *временны́х рядов* (time series), то есть результатов измерения процесса или процессов с течением времени. Такие данные дают представление о зависящих от времени характеристиках рассматриваемого объекта и позволяют получать ML-прогнозы не только на базе статических снимков происходящего. Но полное извлечение информации из временных рядов зачастую затруднено. В этом разделе мы рассмотрим два типа данных: классические временные ряды и точечные процессы (данные событий), а также широко используемые признаки для этих типов.

7.3.1. Типы временных рядов

Существует два типа временных рядов: классические временные ряды и точечные процессы. Классические временные ряды состоят из численных измерений за некий период. Обычно они равномерно распределены по времени (раз в час, раз в день, раз в неделю и т. п.), но могут состоять и из нерегулярных выборочных данных. Вот примеры классических временных рядов данных:

- объем фондового рынка в миллиардах долларов (например, измеряемый ежечасно, ежедневно или еженедельно);
- ежедневное потребление энергии в промышленном здании или жилом доме;
- средства на банковском счету в течение некоторого времени;
- наборы диагностических сообщений на промышленном предприятии (например, физические измерения производительности в разных частях или измерения выхода продукции).

Точечные процессы (point processes), в свою очередь, представляют коллекцию событий, возникавших в течение некоторого времени. В отличие от растянутых во времени численных измерений, точечные процессы состоят из временной метки каждого события и (в ряде случаев) других метаданных, например категории или значения. По этой причине их еще называют *потоками событий* (event streams). Вот примеры точечных процессов:

- активность пользователя в Интернете, измеренная в течение некоторого времени, и так называемые *данные о посещении сайта* (clickstream data);
- случаи землетрясений, ураганов, вспышек заболеваний и т. п. по всему миру;
- покупки клиента за время существования его учетной записи;
- журнал событий на предприятии, куда записываются все входы сотрудников в систему и все завершенные этапы производственного процесса.

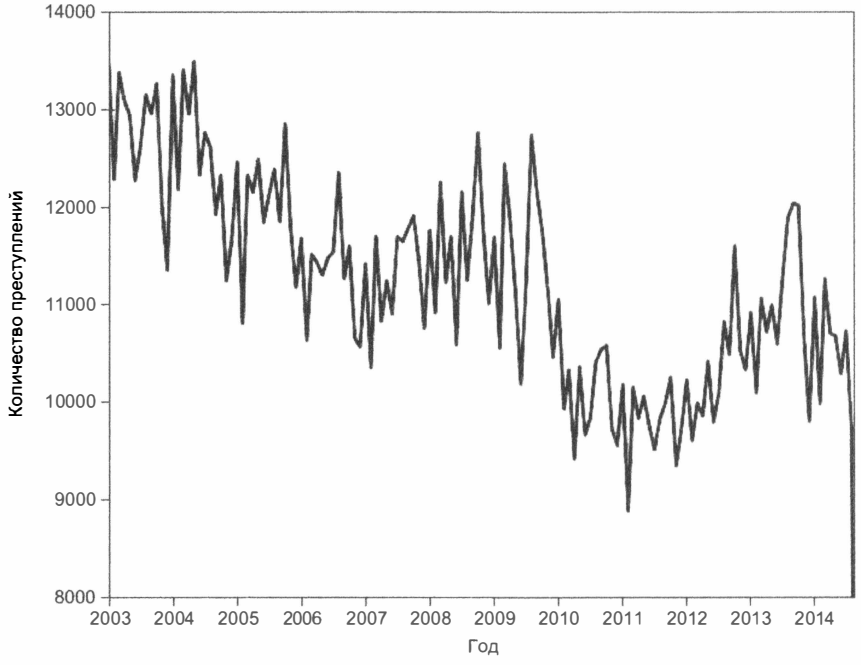
Внимательный читатель может заметить, что иногда существует однозначное соответствие между классическим представлением в виде временного ряда и точечным процессом. К примеру, банковский счет можно представить как в виде его меняющейся со временем величины (классический временной ряд), так и в виде списка отдельных транзакций (точечный процесс). Это соответствие позволяет получать из одного набора данных различные признаки. Но преобразование не всегда осуществимо. (Скажем, сложно представить связь классического временного ряда с переходами по ссылкам при посещении вебстраницы.)

Для прояснения ситуации рассмотрим временной ряд, который можно легко преобразовывать из одной формы в другую. Таблица 7.3 показывает несколько строк криминальной сводки из Сан-Франциско с 2003 по 2014 год (полный набор данных доступен по адресу *https://data.sfgov.org*). Целиком набор включает в себя более полутора миллионов записей о совершавшихся в городе преступлениях. Для каждого случая указаны точная дата и время совершения преступления, его тип и место совершения.

Объединить исходные данные в классический временной ряд можно разными способами — по году, месяцу, дню недели и т. п., потенциально с различными временными рядами для каждого района или категории. Листинг 7.4 демонстрирует получение временного ряда ежемесячного количества преступлений в Сан-Франциско. Полученный ряд целых чисел по месяцам представлен в виде графика на илл. 7.6. Мы видим значительное снижение от показателя за 2003 год (13 000 преступлений) и недавний всплеск криминальной деятельности.

Таблица 7.3. Криминальная сводка по Сан-Франциско в необработанном виде, представленная в виде последовательности событий

Номер	Дата	Время	Район	Категория
80384498	04/13/2008	00:54	Северный	Пьянство
80384147	04/13/2008	00:55	Центральный	Некриминальное
80384169	04/13/2008	00:56	Бейвью	Насилие
80384169	04/13/2008	00:56	Бейвью	Наркотики
80384153	04/13/2008	00:57	Бейвью	Другое
80384175	04/13/2008	01:00	Центральный	Насилие
80384943	04/13/2008	01:00	Центральный	Воровство
80392532	04/13/2008	01:00	Инглсайд	Воровство
80384943	04/13/2008	01:00	Центральный	Мошенничество
80384012	04/13/2008	01:15	Северный	Подозрительные действия



Илл. 7.6. Классический временной ряд данных для количества ежемесячно совершаемых в Сан-Франциско преступлений. Данные получены из сведений о соответствующих событиях. Получить признаки для ML-моделирования можно из данных о событиях, из классического временного ряда или из и того и другого

Листинг 7.4. Преобразование сведений о преступлениях в Сан-Франциско в классический временной ряд

```
import pandas as pd
from datetime import datetime
from matplotlib import pyplot as plt

df = pd.read_csv("sfpd_incident_all.csv")

df['Month'] = map(lambda x: datetime.strptime("/".join(x.split("/")[0::2]),
        "%m/%Y"),df['Date'])

df_ts = df.groupby('Month').aggregate(len)["IncidntNum"]   <- Создает классический временной ряд

plt.plot(df_ts.index,df_ts.values,'-k',lw=2)   <- Создает график для временного ряда
plt.xlabel("Month")
plt.ylabel("Number of Crimes")
```

7.3.2. Предсказания на основе временных рядов

Существуют два типа задач, для решения которых используются временные ряды. Во-первых, задача прогнозирования, то есть попытка предсказать будущие значения временного ряда (или время наступления будущих событий) на базе уже сделанных измерений. В эту категорию попадают:

- предсказание цены акций на завтра;
- предсказание на завтра температуры в определенном месте;
- предсказание потребления энергии по стране на следующий год;
- предсказание даты следующего сильного урагана в Северной Америке.

Первые три задачи сводятся к предсказанию будущих значений временных рядов, в то время как последняя строит прогноз по набору данных о точечных процессах. Во всех случаях мы анализируем значения одного временного ряда, чтобы получить прогноз на будущее. К сожалению, существующая литература в основном посвящена анализу временных рядов, а практическим вопросам машинного обучения уделяется относительно мало внимания (хотя ситуация постепенно меняется). Дополнительную информацию можно получить, воспользовавшись поиском в Google или Amazon, который выдаст вам множество результатов.

Второй тип задач на базе временных рядов — задачи классификации или регрессии. В данном случае вместо предсказания будущих значений нужно выполнить классификацию (или предсказать реальный результат) для сотен или даже тысяч временных рядов. В эту категорию попадают следующие задачи:

- используя данные о посещениях сайта пользователем, предсказать, среагирует ли он на конкретное объявление;
- используя данные о качестве продукции, определить, какой из производимых товаров (например, лампочки), скорее всего, не оправдает ожиданий в следующем месяце;
- предсказать, сколько времени каждый пользователь будет работать с онлайн-приложением, по его активным действиям в этом приложении в первую неделю после регистрации;
- предсказать, какие из пациентов, скорее всего, получат послеоперационные осложнения, на основе данных из их медицинских карт.

В отличие от прогнозов на базе временных рядов, на связанные с этими рядами задачи классификации и регрессии машинное обучение влияет довольно сильно. Поэтому в следующих разделах мы сосредоточимся в основном на создании признаков для этих типов задач. Впрочем, многие из описываемых методов применимы и к задачам прогнозирования.

7.3.3. Признаки классических временных рядов

В этом разделе мы опишем наиболее распространенные подходы к проектированию признаков на базе классических временных рядов. Начнем с простейших метрик, постепенно переходя к рассмотрению более сложных вариантов.

Простые признаки временных рядов

Может показаться абсурдным, но простейшая метрика для временных рядов связана с полным игнорированием временной оси. Анализ распределения результатов измерений без учета временных меток зачастую дает полезную информацию для решения задач классификации, регрессии или прогнозирования.

Для ознакомления приведем четыре простые (но мощные) характеристики, связанные только с маргинальным распределением значений временных рядов:

- *Средние.* Среднее или медиана показывают тенденции в среднем значении временного ряда.
- *Разброс.* Такие виды разброса, как стандартное отклонение, абсолютное отклонение с медианой в качестве точки отсчета или интерквантильный размах, могут показать тенденции в общей вариабельности измерений.
- *Выбросы.* Большую прогностическую ценность во многих случаях, скажем, при предсказании сбоев в технологической линии, имеет частота появления во временном ряду значений, выходящих за пределы обычного распределения (например, превышающих два, три или четыре стандартных отклонения от среднего).
- *Распределение.* В некоторых сценариях прогнозирования полезна оценка более высоких характеристик маргинального распределения значений временного ряда (скажем, коэффициента асимметрии или коэффициента эксцесса) или проведение статических тестов для именованного распределения (например, нормального или равномерного).

Ситуацию можно еще сильнее усложнить, рассматривая вышеперечисленные характеристики *в пределах выбранного временного окна.* Скажем, полезным в прогностическом отношении может оказаться рассмотрение среднего или стандартного отклонения для измерений за последнюю неделю. Иногда имеет смысл рассмотреть разницу характеристик в двух временных окнах. Следующий листинг демонстрирует пример рассчитывающего такие вещи кода.

Листинг 7.5. Статистика во временном окне и разница оконных характеристик

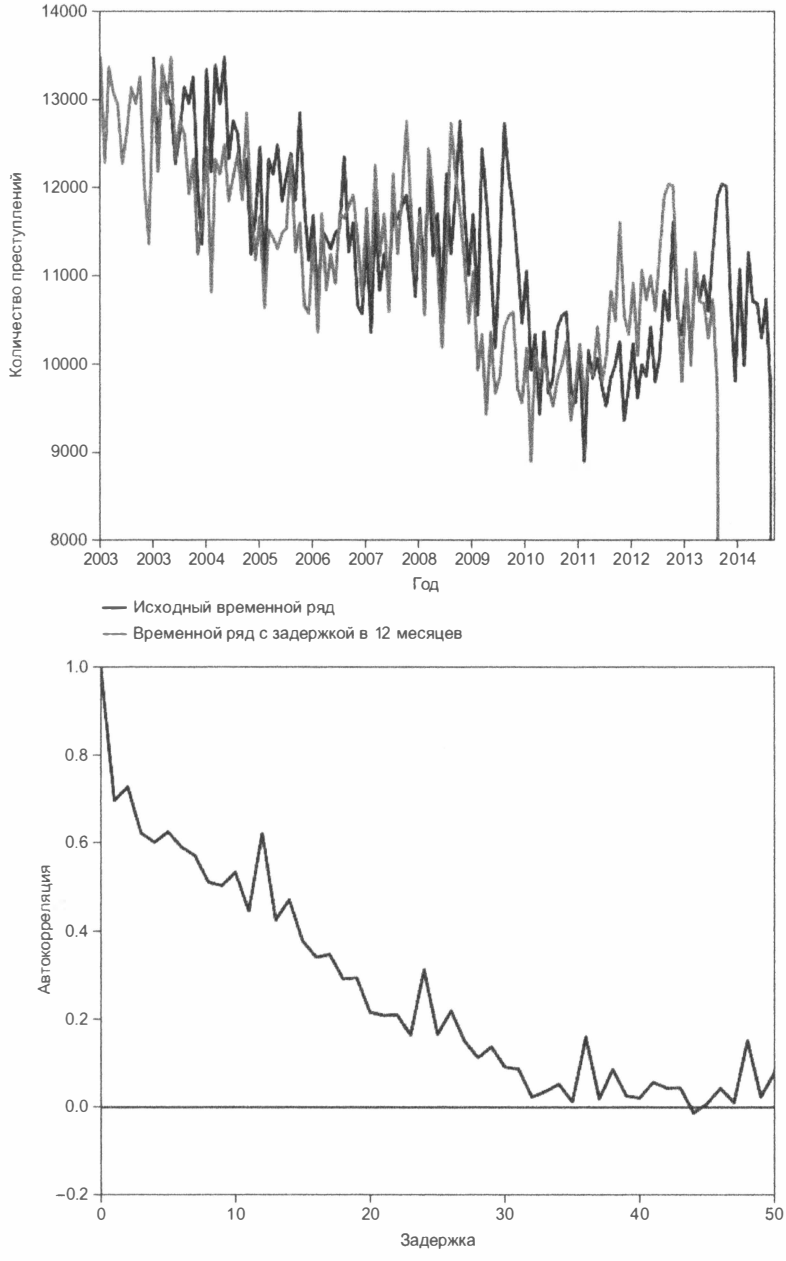
```
import pandas as pd
from datetime import datetime
import numpy as np

window1 = (datetime(2014,3,22),datetime(2014,6,21))  <-- окно = весна 2014

idx_window = np.where(map(lambda x: x>=window1[0] and x<=window1[1],
```

```
    df_ts.index))[0]  <-- Ищет, какие точки данных попадают внутри окна

mean_window = np.mean(df_ts.values[idx_window])  | Вычисляет среднее и стандартное
std_window = np.std(df_ts.values[idx_window])    | отклонение внутри окна

window2 = (datetime(2013,3,22),datetime(2013,6,21))  <-- Вычисляет разницу
                                                         с показателями
                                                         за весну 2013 года
idx_window2 = np.where(map(lambda x: x>=window2[0] and x<=window2[1],
        df_ts.index))[0]
    mean_wdiff = mean_window - np.mean(df_ts.values[idx_window2])
std_wdiff = std_window - np.std(df_ts.values[idx_window2])
                                                  Вычисляет разницу средних
                                                  и стандартных отклонений
                                                  для двух окон
```

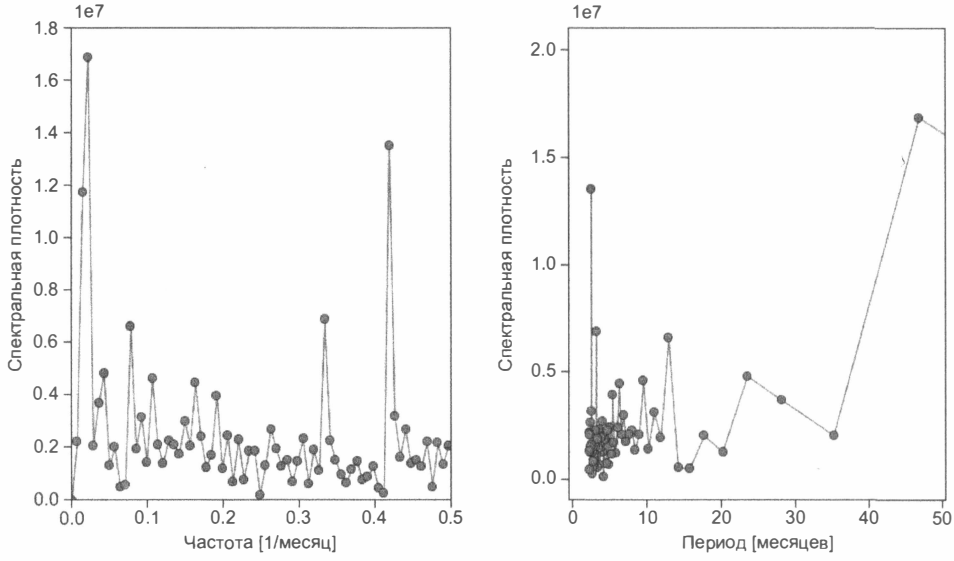
Усовершенствованные признаки временных рядов

Теперь рассмотрим более сложные признаки классических временных рядов. Признаки с *автокорреляцией* показывают статистическую корреляцию временного ряда с его *запаздывающей* версией. Например, такой признак может сопоставить исходный временной ряд с ним же, сдвинутым на одну временную позицию влево (при этом неперекрывающаяся часть удаляется). Сдвиги демонстрируют периодичность и другие статистические структуры, если таковые имеются. Форма автокорреляционной функции (автокорреляция вычисляется по сетке временных сдвигов) фиксирует отличительные признаки структуры временного ряда. В языке Python эта функция находится в модуле `statsmodels`. Иллюстрация 7.7 демонстрирует процесс вычисления автокорреляции и саму функцию для данных по криминальной обстановке в городе Сан-Франциско.

Одним из самых распространенных инструментов проектирования признаков на базе временных рядов является *преобразование Фурье*, в процессе которого временной ряд разбивается на сумму синусоид и косинусоид различных амплитуд. Эти синусоиды и косинусоиды естественным образом возникают во многих реальных наборах данных. Такое преобразование быстро выявляет связанные с временным рядом периодические структуры. Разложение Фурье выполняется специальным дискретным преобразованием, в процессе которого вычисляется *спектральная плотность* временного ряда — то, насколько он коррелирует с синусоидами разных амплитуд, — как функция от амплитуды. Результат этого разложения называется *периодограммой*. Иллюстрация 7.8 показывает периодограмму данных о криминальной обстановке в Сан-Франциско,



Илл. 7.7. Вверху: сопоставление исходного временного ряда и ряда с 12-месячной задержкой дает 12-месячную автокорреляцию. Внизу: функция автокорреляции для данных по криминальной обстановке в Сан-Франциско. Высокая автокорреляция для коротких временных отрезков показывает сильную зависимость уровня преступлений в каждом месяце от показателей предыдущего месяца



Илл. 7.8. Слева: периодограмма данных по криминальной обстановке в Сан-Франциско, показывающая спектральную плотность как функцию частоты. Справа: та же самая периодограмма после того, как по оси x стали откладывать не частоту, а время

рассчитанную с помощью функции `scipy.signal.periodogram` (методы оценки периодограмм встроены в несколько модулей языка Python). Из периодограммы генерируют признаки: например, спектральную плотность при различных амплитудах, сумму спектральных плотностей в определенном диапазоне амплитуд или местоположение максимума спектральной плотности (что описывает основную частоту колебаний временного ряда). Листинг 7.6 демонстрирует пример кода, рассчитывающего периодограмму и генерирующего признаки.

В литературе, посвященной анализу временных рядов, обычно рассматриваются модели для классических рядов. Они описывают каждое значение временного ряда как функцию от его прошлых значений. Десятилетиями эти модели использовались для предсказаний на основе временных рядов. В настоящее время, когда основой анализа временных рядов стало машинное обучение, они зачастую применяются в паре с более сложными ML-алгоритмами, такими как метод опорных векторов, нейронные сети и «случайные леса».

Листинг 7.6. Признаки, полученные из периодограммы

```
import pandas as pd
import numpy as np
import scipy.signal

f, psd = scipy.signal.periodogram(df_ts, detrend='linear')

plt.plot(f, psd,'-ob')
plt.xlabel('frequency [1/month]')
plt.ylabel('Spectral Density')
plt.show()

# Признаки:
period_psd1 = 1./f[np.argmax(psd)]

sdens_gt_12m = np.sum(psd[f > 1./12])

sdens_ratio_12m = float(sdens_gt_12m) / np.sum(psd[f <= 1./12])
```

Вычисляет периодограмму

Признак 1: период самых высоких значений диаграммы вероятностей состояния; для этих данных составляет 47,0 месяца

Признак 2: сумма спектральных плотностей для периодов, превышающих 1/12 месяца

Признак 3: соотношение спектральных плотностей для окон шириной более и менее 1/12 месяца

Вот некоторые из этих моделей:

- *Авторегрессионная модель (AR).* Каждое значение временного ряда моделируется как линейная комбинация последних p значений, где p — параметр, определяемый методом подбора.
- *Модель авторегрессии – скользящего среднего (ARMA).* Каждое значение моделируется как сумма двух полиномиальных функций — модели AR и модели скользящего среднего, которая представляет собой комбинацию предыдущих q погрешностей.
- *GARCH-модель.* Широко применяется в финансовом анализе и описывает такой параметр временных рядов, как белый шум, с помощью модели авторегрессионной условной гетероскедастичности (ARMA).
- *Скрытая марковская модель (HMM).* Вероятностная модель, описывающая наблюдаемые значения временных рядов как производную от неизвестного набора скрытых состояний, которые, в свою очередь, имитируют марковский процесс.

Эти модели позволяют разными способами рассчитывать признаки временных рядов. Вот некоторые из этих способов:

- использование в качестве признаков предсказанных каждой моделью значений (и разницы между предсказаниями);
- использование в качестве признаков оптимальных параметров моделей (например, значений p и q из модели ARMA(p,q));
- вычисление и использование в качестве признака статистического критерия модели (например, среднеквадратичной ошибки).

Такой подход дает возможность комбинировать классические модели временных рядов и последние достижения в области машинного обучения. С каждой стороны можно взять самое лучшее. Если модель ARMA дает точные прогнозы для конкретного временного ряда, успешной будет и использующая ее ML-модель. Если же модель ARMA не очень подходит (а именно так обычно получается с реальными наборами данных), за счет гибкости ML-модели все равно можно получить предсказания высокой точности.

7.3.4. Проектирование признаков для потоков событий

Сейчас мы коротко рассмотрим процесс проектирования признаков для потоков событий. Как было показано в листинге 7.4, данные событий можно преобразовывать в классический временной ряд. Это позволяет применять к точечным процессам все описанные в предыдущих двух разделах процедуры. При этом большая степень детализации дает возможность генерировать дополнительные признаки.

Способом, описанным в разделе 7.1.3, можно вычислить статистику для временных окон и разницы данных для событий. Благодаря тому, что с каждым событием точечного процесса связана отдельная временная метка, указанные показатели можно рассчитать для любого временного окна, вплоть до крайне небольшой детализации. Кроме того, мы получаем доступ к дополнительным характеристикам, например таким, как «время с момента последнего события», «количество событий за последние 48 часов», «среднее время между двумя событиями».

Наконец, если классические временные ряды часто представляются с помощью таких статических моделей, как ARMA и HMM, данные точечных процессов описываются такими моделями, как обычные и негомогенные процессы Пуассона. Эти модели представляют частоту входящих событий как функцию от времени и позволяют предсказывать ожидаемое время наступления следующего события. Более подробное знакомство с этими моделями оставляем вам как задание для самостоятельной работы. Как и в случае моделей классических временных рядов, признаки для решения задач машинного обучения, связанных с точечными процессами, проектируются тремя способами: из предсказаний модели, из параметров модели и из статистического критерия модели.

7.4. Заключение

В этой главе мы рассмотрели процесс генерации признаков из текста и изображений. Эти признаки затем используются в ML-алгоритмах, позволяя строить модели, умеющие «читать» или «видеть» с почти человеческим уровнем восприятия. Вот основные моменты, которые следует запомнить:

- В наборе данных на базе обычного текста нужно преобразовать документы переменной длины в фиксированное количество признаков. Это делается следующими методами:
  - простые методы, например «мешок слов», в которых подсчитывается количество вхождений каждого слова во все рассматриваемые документы;
  - алгоритм tf-idf, учитывающий частоту появления каждого слова в целом корпусе, чтобы при формировании словаря избежать сдвига в сторону распространенных, но неинформативных слов;
  - усовершенствованные алгоритмы для тематического моделирования, такие как латентно-семантический анализ и латентное размещение Дирихле;
  - техники тематического моделирования, позволяющие описывать документы как набор тем, а темы — как набор слов. Это дает возможность сложного семантического представления документов и помогает не только в сфере машинного обучения, но

и, к примеру, при построении усовершенствованных поисковых движков;

- библиотеки scikit-learn и Gensim для языка Python позволяют провести множество интересных экспериментов в области извлечения текста.

- В случае изображений требуется представить их характеристики в виде численных признаков:
  - извлечь из изображения информацию о цветах можно, определив цветовые диапазоны или рассчитав простые статистические характеристики;
  - потенциально полезные метаданные изображения можно получить из самого файла, например, изучив данные EXIF, присутствующие в большинстве файлов изображений;
  - бывают ситуации, когда из изображения требуется извлекать формы или объекты. Это делается следующими методами:
    - простые алгоритмы, осуществляющие распознавание контуров на базе фильтров Собеля или Кэнни;
    - сложные алгоритмы извлечения контуров, такие как гистограмма направленных градиентов;
    - техники уменьшения размерности, такие как метод главных компонент;
    - автоматическое извлечение признаков с помощью глубоких нейронных сетей.
- Данные временных рядов бывают двух типов — классические временные ряды и точечные процессы. Из них можно получить множество признаков для задач с машинным обучением:
  - на основе данных из временных рядов решают два основных типа задач:
    - предсказание будущих значений временного ряда,
    - классификация набора временных рядов;
  - простейшие признаки для классических временных рядов включают в себя вычисление суммарной статистики для рассматриваемых временных окон и разницы значений в пределах таких окон;

- для получения более сложных признаков приходится прибегать к статистическим характеристикам временных рядов с помощью таких инструментов, как автокорреляция и разложение в ряды Фурье;
- для генерации признаков применяются и различные модели классических временных рядов, например AR, ARMA, GARCH и HMM;
- для данных точечных процессов можно получить не только вышеуказанные признаки, но и множество других, так как они обладают большей детализацией;
- для точечных процессов повсеместно применяют обычные и негомогенные процессы Пуассона.

7.5. Терминология

Термин	Определение
Проектирование признаков (feature engineering)	Преобразование входных данных с целью извлечения дополнительных значений и улучшения прогностической точности ML-моделей
Обработка естественного языка (natural language processing)	Область, помогающая компьютеру понять обычную речь
«Мешок слов» (bag of words)	Метод преобразования текста в набор чисел путем подсчета вхождений слова в документ
Шумовые слова (stop words)	Часто встречающиеся слова, бесполезные в качестве признаков (например, артикли, предлоги, глаголы-связки)
Разреженные данные (sparse data)	Данные, состоящие преимущественно из нулей. Такие данные дает большинство NLP-алгоритмов
tf-idf	Частота слова, обратная частоте документа. Метод «мешка слов», нормализованный посредством текста всего корпуса
Латентно-семантический анализ (latent semantic analysis)	Метод обнаружения в документах нужных тем и их связи с набором слов
Латентное размещение Дирихле (latent Dirichlet analysis)	Расширение латентно-семантического анализа, хорошо работающее при решении реальных задач с текстами

Термин	Определение
Расширение содержимого (content expansion)	Процесс добавления данных к исходному содержимому (например, путем перехода по имеющимся в документе ссылкам)
Метапризнаки (meta-features)	Набор признаков, извлекаемых не из содержимого, а из связанных с ним метаданных
Данные EXIF	Стандарт метаданных в изображениях. Включает в себя сведения о фотографии (например, производитель камеры, разрешение, диафрагма)
Распознавание контуров (edge detection)	Процесс, позволяющий убрать шум из большинства изображений
HOG	Гистограмма направленных градиентов. Метод получения признаков изображений, умеющий распознавать формы и объекты
PCA	Метод главных компонент. Способ представления изображений в виде более простых, типичных картинок, в результате чего количество измерений уменьшается. Вместо 100 пикселей изображение можно описать двумя числами — расстоянием до двух главных компонент
Глубокие нейронные сети (deep neural nets)	Недавно появившееся расширение нейронных сетей, эффективное для машинного обучения с аудиовизуальными данными
Классический временной ряд (classical time series)	Набор числовых измерений, сделанный за определенное время
Точечный процесс (point process)	Набор событий за определенный период, каждое с известной временной меткой
Прогноз для временного ряда (time-series forecasting)	Предсказание будущих значений для рассматриваемого временного ряда
Периодограмма (periodogram)	График спектральной плотности мощности для преобразования Фурье как функция от частоты колебаний. Эта техника позволяет определять основные частоты колебаний и применяется для проектирования признаков из временных рядов

8 Пример обработки естественного языка

В этой главе:

- ✓ предсказание тональности рецензий к фильмам на базе реальных данных;
- ✓ подбор возможных вариантов работы с данными и подходящей стратегии моделирования;
- ✓ построение исходной модели с применением базовых NLP-признаков и оптимизация параметров;
- ✓ извлечение более сложных NLP-признаков для увеличения точности моделирования;
- ✓ масштабирование и другие аспекты развертки модели.

В предыдущей главе вы познакомились с техниками усовершенствованного проектирования признаков. Давайте применим их к решению реальной задачи. Нам предстоит построить и оптимизировать модель на базе пользовательских обзоров фильмов.

Как обычно, начнем с анализа имеющихся данных, чтобы получить представление о столбцах признаков и целевом столбце, а также сделать оптимальный выбор проектируемых признаков и будущего ML-алгоритма. Затем на базе простейших алгоритмов извлечения признаков будет построена начальная модель, чтобы показать вам, как получить результат при помощи всего нескольких строк кода. После этого мы более подробно рассмотрим библиотеку алгоритмов извлечения признаков и ML-моделирования, попутно увеличив точность нашей модели. Завершит главу рассказ об аспектах развертки и масштабирования, с которыми приходится сталкиваться при запуске модели в производство.

8.1. Изучение данных и сценарии их применения

В этой главе мы воспользуемся данными конкурса с сайта *Kaggle*, на котором ученые со всего мира соревнуются в решении поставленных различными компаниями задач, пытаясь выиграть призы. Мы же на примере этих данных рассмотрим варианты применения инструментов, с которыми вы познакомились в предыдущих главах, и решим реальную задачу с машинным обучением.

Данные, с которыми мы будем работать, взяты с конкурса «*В "мешок слов" попадает много болтовни*» (*www.kaggle.com/c/word2vec-nlp-tutorial*). Для их скачивания потребуется учетная запись на сайте Kaggle, впрочем, она все равно пригодится, если вы захотите применить свои новые навыки и поучаствовать в каком-нибудь конкурсе!

В следующих разделах мы начнем описывать этот набор данных, рассказывая, что означает каждый столбец и как они были сгенерированы. Затем мы перейдем на более глубокий уровень и рассмотрим атрибуты данных, а также сделаем первоначальные выводы о наборе, с которым нам предстоит работать. Методом мозгового штурма придумаем варианты задач, решение которых возможно при наличии такого набора. Здесь

же вы найдете обзор требований к данным и практические выводы для каждого из потенциальных вариантов применения. В качестве завершающего аккорда будет выбрана одна задача, решению которой мы посвятим остаток главы.

Мы начинаем с описания и изучения данных и только потом выбираем задачу, которую будем решать, но обычно эти операции выполняются в обратном порядке. На практике все начинается с задачи и гипотез или набора вопросов, на которые нужно ответить, и только потом выполняется поиск и анализ данных, которые помогут построить подходящее решение. Это предпочтительная методология, так как сначала приходится тщательно обдумывать задачу и требуемые для ее решения данные и только потом бросать силы на их изучение. Тем не менее бывают случаи, когда предоставляется набор данных и на его основе просят получить какой-нибудь интересный результат.

8.1.1. Первый взгляд на набор данных

Наш набор состоит из отзывов на фильмы с сайта IMDb (*www.imdb.com*) — крупнейшей в мире кинематографической базы данных. Обучающая выборка состоит из 50 000 обзоров, отобранных таким образом, чтобы каждому фильму соответствовало не более 30. Для каждого обзора существует выходная переменная, представленная как двоичный признак, имеющий значение 1, если рейтинг превышает 6, и значение 0 при рейтинге менее 5. Обзоры с промежуточным рейтингом 5–6 в набор данных не попали.

Нам нужно разработать ML-систему для изучения образцов и структуры языка, которым написаны положительные и отрицательные обзоры. Немаловажно, что модель будет обучаться только на текстах рецензий. Контекстные данные, такие как играющие в фильме актеры, режиссер, жанр или год выпуска, учитываться не будут. Возможно, эти данные могли бы увеличить точность предсказаний нашей модели, но в обучающей выборке они отсутствуют.

В дополнение к обучающей выборке предоставляется набор из еще 25 000 обзоров, предназначенный для тестирования. В принципе им можно воспользоваться для проверки производительности модели и оценки качества ее работы с реальными данными. Но сайт Kaggle не снабдил эту

тестовую выборку метками. Поэтому мы сконструируем собственный тестовый набор, разбив предоставленную сайтом Kaggle обучающую выборку в пропорции 70% для обучения и 30% для тестирования.

Крайне важно, чтобы ни один фильм из обучающей выборки не появился в наборе для тестирования[1]. Включение одинаковых обзоров в оба набора приведет к тому, что модель по названиям фильмов изучит, какие из них нравились публике, а какие не вызвали восторга, вместо того чтобы фокусироваться на положительных и негативных тональностях. Но готовой модели предстоит работать с фильмами, названий которых она не знает. А попадание фильмов из обучающей выборки в набор для тестирования создает впечатление, что модель работает лучше, чем это есть на самом деле. По этой причине рекомендуем при конструировании тестового набора использовать временную отсечку, чтобы в него попали более свежие экземпляры, чем в обучающую выборку.

8.1.2. Анализ набора данных

Длина обзоров в нашем наборе варьируется от одного предложения до нескольких страниц текста. Их писали десятки авторов, поэтому состав слов в разных обзорах сильно отличается друг от друга. Требуется построить модель с машинным обучением, которая сможет распознавать и использовать разницу между положительными и отрицательными отзывами, точно предсказывая тональность новых рецензий.

Первым делом нужно внимательно изучить данные и начать обдумывание следующих этапов ML-процесса, таких как выбор типа модели и проектирование признаков. Поэтому рассмотрим 10 самых коротких обзоров, показанных на илл. 8.1. Первая же строчка (id = 10962_3) демонстрирует сложность стоящей перед нами задачи: хотя автор напрямую заявляет, что «фильм ужасный», в рецензии упоминаются «*хорошие* спецэффекты». Но, несмотря на наличие слова «*хорошие*», любой человек сочтет эту рецензию негативной. То есть нам нужно научить ML-модель тому, что даже при наличии положительных слов тональность отзыва задает именно фраза «фильм ужасный»!

[1] В нашей обучающей выборке нет индикатора, сопоставляющего обзор с фильмом. Поэтому мы предполагаем, что эта выборка заранее отсортирована по дате, и делим ее таким образом, чтобы множество обзоров одного и того же фильма совместно попало или в обучающее, или в предназначенное для тестирования подмножество.

id	sentiment	review
10962_3	0	This movie is terrible but it has some good effects.
2331_1	0	I wouldn't rent this one even on dollar rental night.
12077_1	0	Ming The Merciless does a little Bardwork and a movie most foul!
266_3	0	You'd better choose Paul Verhoeven's even if you have watched it.
4518_9	1	Adrian Pasdar is excellent is this film. He makes a fascinating woman.
874_1	0	Long, boring, blasphemous. Never have I been so glad to see ending credits roll.
3247_10	1	I don't know why I like this movie so well, but I never get tired of watching it.
7243_2	0	no comment - stupid movie, acting average or worse... screenplay - no sense at all... SKIP IT!
5327_1	0	A rating of \1\" does not begin to express how dull, depressing and relentlessly bad this movie is."
2469_10	1	This is the definitive movie version of Hamlet. Branagh cuts nothing, but there are no wasted moments.

Илл. 8.1. Десять примеров, взятые из числа самых кратких рецензий. Для каждого случая указывается только идентификатор, тональность в двоичной форме и текст обзора

Сходную ситуацию мы видим и с негативными заявлениями. Такие фразы, как «не устаю пересматривать» и «бесполезных сцен в фильме нет», очевидно указывают на положительную тональность рецензии, несмотря на негативные по своей природе слова. Это показывает, что для корректного прогнозирования тональности следует комбинировать информацию из нескольких (соседствующих) слов.

При изучении более длинных обзоров в глаза бросается тот факт, что они, как правило, написаны многословным, описательным и цветистым языком. Много саркастических, иронических и остроумных фраз. Все это демонстрирует, каким мощным инструментом является машинное обучение, позволяющее извлекать из реальных данных едва уловимые закономерности и давать точные предсказания в неоднозначных обстоятельствах.

8.1.3. Так какой же будет наша задача?

Практикующие машинное обучение иногда начинают решать задачу, не подумав о практической ценности ML-модели. Это ошибочный подход, так как от варианта практического применения зависит структура задачи и ее решение, а также:

- вид целевой переменной (например, двоичная, многоклассовая или реальное число);
- подлежащие оптимизации критерии оценки;

- рассматриваемые алгоритмы обучения;
- какие входные данные будут использоваться, а какие — нет.

Поэтому перед тем как погрузиться в ML-моделирование, нужно определить, какую практическую задачу мы собираемся решить с помощью нашего набора данных. Для каждого потенциального варианта применения нужно ответить на следующие вопросы:

- какую практическую ценность имеет решение?
- какие обучающие данные потребуются?
- какая стратегия моделирования больше всего подходит в выбранной ситуации?
- какие оценочные метрики будут применяться для генерируемых прогнозов?
- хватает ли данных для решения поставленной задачи?

На базе полученных ответов мы сформулируем задачу, решению которой и посвятим остаток главы.

Вариант 1: определение рейтинга новых фильмов

Первый и наиболее очевидный вариант применения набора данных с обзорами фильмов — автоматическое определение рейтинга новых фильмов на базе оставленных к ним отзывов:

- *Какую практическую ценность имеет решение?*

  Мы можем получить впечатляющий инструмент, помогающий решить, стоит ли смотреть тот или иной фильм. Оценка отдельных обзоров — это одно, но очевидно, что намного практичнее оценивать сам фильм по общему рейтингу оставленных на него рецензий. Стабильно высокий трафик таких сайтов, как Rotten Tomatoes, поддерживается благодаря их умению корректно оценивать каждый фильм.

- *Какие обучающие данные потребуются?*

  Здесь не обойтись без текстов рецензий, показателей тональности каждой рецензии и сведений о каждом рецензируемом фильме. Эти три компонента дают возможность построить систему оценки фильмов.

- *Какая стратегия моделирования больше всего подходит в выбранной ситуации?*

  Возможны два подхода.

  (a) Рассматривать каждый фильм как отдельный экземпляр, объединять все обзоры к нему и фиксировать их тональность как средний балл или как многоклассовую модель.

  (b) В качестве экземпляров рассматриваются отдельные рецензии, каждая новая рецензия оценивается по своей положительности, а каждому новому фильму присваивается средний рейтинг.

  Мы предпочитаем вариант (b), так как объединенное рассмотрение рецензий к каждому фильму может привести к путанице, в частности, если на один фильм даются диаметрально противоположные отзывы! Оценка отдельных результатов и их усреднение для получения «мета-оценки» — более однозначный подход.

- *Какие оценочные метрики будут применяться для генерируемых прогнозов?*

  Предположим, у нас есть двоичная переменная для оценки каждого обзора, а ML-алгоритм присваивает обзору балл на основании вероятности того, что этот обзор положительный. Затем мы суммируем баллы от всех рецензий, вычисляя общую оценку фильма. Нас интересует, насколько эта оценка совпадает со средним рейтингом фильма (например, с процентом положительных отзывов), что приводит к появлению такой метрики, как R^2.

  Но можно взять и другую оценочную метрику, которая придаст больший вес верхней части рейтинга. В реальности пользователи модели, скорее всего, будут интересоваться рейтингом фильмов, например, чтобы выбрать, что посмотреть в выходные. Поэтому мы возьмем метрику, позволяющую правильно отбирать верхние позиции из рейтингового списка. Этому условию соответствует, например, доля истинно-положительных значений при малой доле ложно-положительных (скажем, 5 или 10%).

- *Хватает ли данных для решения поставленной задачи?*

  К сожалению, нет. Мы не знаем, какой фильм описывает каждая рецензия!

Вариант 2: оценка каждого отзыва по 10-балльной шкале

Вторая возможность, которую дает имеющийся набор данных, — автоматическая оценка каждого отзыва по шкале от 1 до 10 (принятой на сайте IMDb), основанная на множестве пользовательских отзывов на каждый фильм:

- *Какую практическую ценность имеет решение?*

  Любому новому обзору можно автоматически присвоить рейтинг, то есть исчезает необходимость читать его и оценивать вручную. Это избавляет тех, кто поддерживает сайт IMDb, от большей части ручного труда. Если же пользователи не только пишут обзор, но и ставят фильму оценку, такая система позволяет сделать рейтинг более объективным на основе текста обзора.

- *Какие обучающие данные потребуются?*

  Для этой задачи достаточно текста всех рецензий и оценки по 10-балльной шкале для каждой из них.

- *Какая стратегия моделирования больше всего подходит в выбранной ситуации?*

  И снова у нас есть два варианта:

  (a) Рассмотреть выходную переменную как вещественное число и обучить модель регрессии.

  (b) Рассмотреть выходную переменную как категориальную и обучить модель многоклассовой классификации.

  В нашем случае предпочтительнее вариант (a), так как в отличие от задачи классификации рейтинг рассматривается с помощью числовой шкалы.

- *Какие оценочные метрики будут применяться для генерируемых прогнозов?*

  Для модели регрессии естественной оценочной метрикой будет R^2 или среднеквадратичная ошибка.

- *Хватает ли данных для решения поставленной задачи?*

  И снова нет. В нашем случае каждому обзору сопоставлена только булевская переменная (он может быть положительным или отрицательным), а не детальный числовой рейтинг.

Вариант 3: разделение положительных и отрицательных отзывов

Последним вариантом практического применения будет отделение положительных рецензий от всех остальных:

- *Какую практическую ценность имеет решение?*

  В данном случае речь идет о менее детализированной версии предыдущего варианта, так как каждый обзор автоматически классифицируется как положительный или отрицательный (вместо его оценки по 10-балльной шкале). Такая классификация позволит сайту IMDb отобрать положительные отзывы и с их помощью рекламировать фильмы на первой странице или (что еще лучше) продать их производителям фильма для использования в качестве цитат на постерах.

- *Какие обучающие данные потребуются?*

  Только тексты обзоров и двоичный индикатор для отличия положительного обзора от отрицательного.

- *Какая стратегия моделирования больше всего подходит в выбранной ситуации?*

  Нужно обучить двоичную модель классификации. После этого можно предсказывать рейтинг каждого нового обзора в соответствии с вероятностью того, что отзыв будет положительным.

- *Какие оценочные метрики будут применяться для генерируемых прогнозов?*

  Все зависит от способа использования прогнозов. При автоматическом выборе 10 наиболее положительных обзоров недели (например, для отображения на первой странице сайта IMDb) больше всего подойдет доля истинно-положительных значений при крайне низкой доле

ложно-положительных (скажем, 1%). Если же нам нужно найти все положительные обзоры при полном игнорировании отрицательных (например, для автоматической расстановки тегов, указывающих тональность), подойдут такие метрики, как точность или область под кривой (AUC).

- *Хватает ли данных для решения поставленной задачи?*

  Да! У нас есть обучающая выборка с текстами обзоров и двоичная переменная тональности. Соответственно, остаток главы будет посвящен построению решения для этой задачи.

Напомним, что первым делом мы детально рассмотрели наш набор данных — написанные пользователями отзывы на фильмы с сайта IMDb. Затем мы изучили данные на предмет наличия шаблонов и тенденций. В заключение были предложены варианты практического применения машинного обучения. В каждом случае мы исследовали, как именно машинное обучение помогает в решении поставленной задачи, рассмотрели базовые требования к данным и варианты получения результата.

Пришло время построить ML-модель, отделяющую положительные рецензии от отрицательных.

8.2. Генерация базовых NLP-признаков и построение первого варианта модели

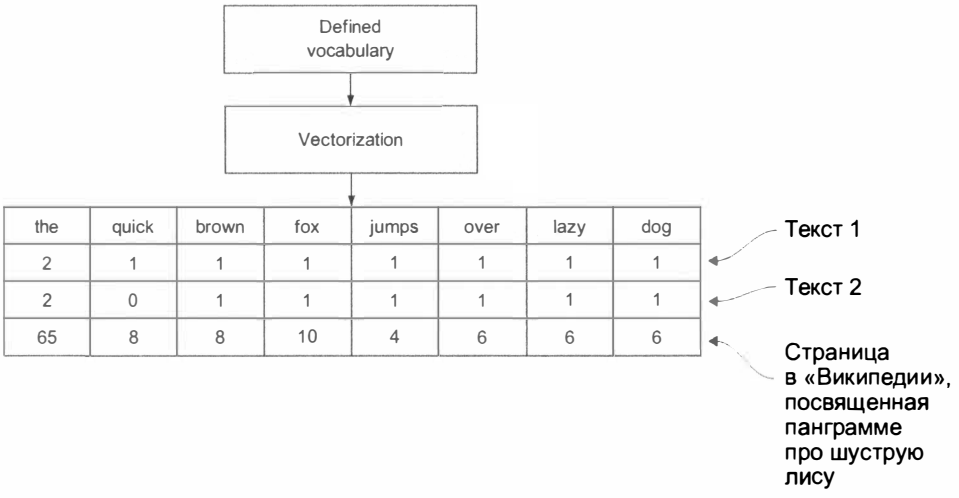
Так как у нас на руках нет ничего, кроме текстов рецензий, для превращения его в подходящий для построения модели набор данных нужно сгенерировать из обычного текста набор признаков. В предыдущей главе мы рассказали о методах извлечения признаков из текста, а сейчас обсудим практические аспекты работы с произвольными текстовыми фрагментами в области машинного обучения. В этом разделе мы рассмотрим:

1. Извлечение признаков из рецензий к фильмам методом «мешка слов».
2. Построение первого варианта модели на базе наивного байесовского классификатора.

3. Совершенствование признаков с помощью алгоритма tf-idf.
4. Оптимизацию параметров модели.

8.2.1. Признаки из «мешка слов»

Напомним, что в предыдущей главе обсуждение процесса генерации признаков из обычного текста началось с простой техники — «мешка слов». В этой технике анализируется весь корпус текста, составляется его полный словарь, и каждый экземпляр в полученном наборе слов преобразуется в список чисел путем подсчета числа вхождений каждого слова в каждый документ. Чтобы освежить вашу память, давайте посмотрим на демонстрирующую данный метод илл. 8.2.



Илл. 8.2. Алгоритм векторизации «мешок слов». Готовый словарь позволяет преобразовать любой новый документ (например, текст 1 и текст 2 на рисунке) в список чисел, указывающих, сколько раз каждое слово встречается в рассматриваемом документе

В листинге 8.1 загружается набор данных, создается разбиение 70%/30% на обучающую и тестовую выборки и используется простой метод подсчета слов для извлечения признаков. Здесь важно понимать, что словарь «мешка слов» нельзя загрязнять словами из тестовой выборки. Набор данных разбивается *до* построения векторизованного словаря именно для того, чтобы получить реалистичную оценку точности модели при работе с ранее *неизвестными* данными.

Листинг 8.1. Признаки, получаемые подсчетом слов в рецензиях на фильмы

```
import pandas
d = pandas.read_csv("movie_reviews/labeledTrainData.tsv", delimiter="\t")

split = 0.7
d_train = d[:int(split*len(d))]
d_test = d[int((1-split)*len(d)):]

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()

features = vectorizer.fit_transform(d_train.review)
test_features = vectorizer.transform(d_test.review)
i = 45000
j = 10
words = vectorizer.get_feature_names()[i:i+10]
pandas.DataFrame(features[j:j+7,i:i+10].todense(), columns=words)
```

Загружает данные

Разбивает данные на обучающую и тестовую выборки

Инициализирует векторизатор, подсчитывающий слова

Обучает словарь и генерирует признаки для обучающей выборки

Генерирует признаки для тестового набора

Подмножество сгенерированных признаков показано на илл. 8.3. Бросается в глаза, что набор данных состоит преимущественно из нулей. Такие наборы называют *разреженными* (sparse), и это распространенный атрибут NLP-данных. С последствиями разреженности приходится сталкиваться, если вы хотите сгенерировать из такого набора признаки для ML-модели, но об этом мы поговорим в следующем разделе.

Слова в словаре

	producer	producer9and	producers	produces	producing	product	production	productions	productive	productively
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	1	0	0	0
4	0	0	0	0	0	0	1	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0

Строки в наборе данных

Ненулевые элементы

Илл. 8.3. Подмножество 7 × 10 признаков, полученных подсчетом слов, которое будет использоваться при построении модели. Полный набор представляет собой разреженную матрицу 17 500 × 65 005 (17 500 документов в обучающей выборке на 65 005 уникальных слов в ней же). Такая матрица возникает для большинства признаков, полученных методом «мешка слов»; отдельные слова из полного словаря редко присутствуют в конкретном документе

8.2.2. Модель на базе наивного байесовского классификатора

Теперь можно воспользоваться полученными из набора данных признаками и построить модель. Есть алгоритмы, которые лучше других работают с разреженными данными. Более того, в некоторые алгоритмы встроена поддержка таких данных, и в общем случае они более эффективны, по крайней мере в том, что касается использования памяти, а зачастую еще и в таких аспектах, как загрузка процессора и время построения модели. В наборе признаков, сгенерированном листингом 8.1, только 0,2% ячеек содержит ненулевые элементы. Плотное представление набора данных значительно увеличивает его размер в памяти.

Наивный байесовский классификатор

Наивный байесовский классификатор представляет собой простой ML-алгоритм, созданный для классификации текстов. Именно в этой области он успешно соперничает с более продвинутыми алгоритмами общего назначения. Его название указывает на тот факт, что формула Байеса применяется к данным с крайне «наивными» предположениями о независимости.

Именно это предположение делает алгоритм слабо применимым для решения общих (плотных) задач, так как признаки крайне редко бывают практически независимыми. Это касается и признаков, полученных из обычного текста, но их зависимость достаточно слаба для хорошей работы алгоритма. Наивный байесовский классификатор — один из немногих ML-алгоритмов, вывод которого занимает всего несколько строк, поэтому мы его вам сейчас покажем.

В этой главе нам нужно классифицировать отзывы путем определения вероятности $p(C_k|x)$ «негативной» ($k = 0$) или «позитивной» ($k = 1$) тональности на базе признаков x экземпляров. Согласно формуле Байеса из теории вероятностей, это записывается так:

$$P(C_k|x) \sim p(C_k)p(x \mid C_k).$$

Здесь $p(x \mid C_k)$ — совместная вероятность признаков x при условии, что экземпляр принадлежит к классу C_k. Благодаря предположению о независимости (наивная часть) можно не рассматривать вероятность для не-

скольких признаков одновременно, и мы получаем обычное произведение вероятностей для каждого признака с учетом класса:

$$p(C_k|x) \sim p(C_k)\, p(x_1 \mid C_k)\, p(x_2 \mid C_k)\, p(\mathrm{x}_4 \mid C_k)\ldots = P(C_k) \times \prod_i^N p(x_i|\, C_k).$$

Так как $p(C_k)$ — маргинальное классовое распределение (общее разбиение обзоров с положительной и отрицательной тональностью), которое мы можем легко получить из имеющихся данных, остается понять, что такое $p(x_i|\, C_k)$. Это выражение означает «вероятность конкретного признака в конкретном классе». К примеру, логично ожидать, что вероятность появления слова «великолепный» в положительном отзыве выше, чем в отрицательном.

Этот показатель можно узнать из данных, подсчитав наличие признака (слова) во всех документах в каждом классе. Распределение вероятности, генерирующее такие показатели, называется *полиномиальным*, и для $p(x_i|\, C_k)$ мы получаем

$$p(x_1 \mid C_k) \sim \prod_i p_{k_i}^{x_i}$$

Вставим это в предыдущее уравнение, а для удобства перейдем в логарифмическое пространство:

$$\begin{aligned}
&\log\left[\, p(C_k \mid x_i)\right] \sim \log\left[\, p(C_k)\prod_i p_{k_i}^{x_i}\right] \\
&= \log\left[\, p(C_k)\right] + \sum_i^n x_i \log(p_{k_i}) \\
&= b + w_k x
\end{aligned}$$

Здесь b — это $\log[p(C_k)]$ (известный из данных), x — признаки экземпляров, которые мы хотим предсказать, а w_k означает $\log(p_{ki})$ — количество раз, когда слово появляется в хорошем или плохом документе, его мы получаем во время построения модели. Из этих формул мы для простоты убрали различные константы, кроме того, при построении алгоритма с нуля следует учитывать многочисленные детали конкретной реализации, но на основной смысл это не влияет.

Наивный байесовский классификатор (особенно полиномиальный) — один из алгоритмов, хорошо справляющихся с классификацией разре-

женных признаков, полученных из обычного текста. Сейчас мы построим модель из признаков, полученных в листинге 8.1.

Листинг 8.2. Построение первой модели для определения тональности обзоров на базе полиномиального наивного байесовского классификатора

```
from sklearn.naive_bayes import MultinomialNB

model1 = MultinomialNB()
model1.fit(features, d_train.sentiment)
pred1 = model1.predict_proba(test_features)
```

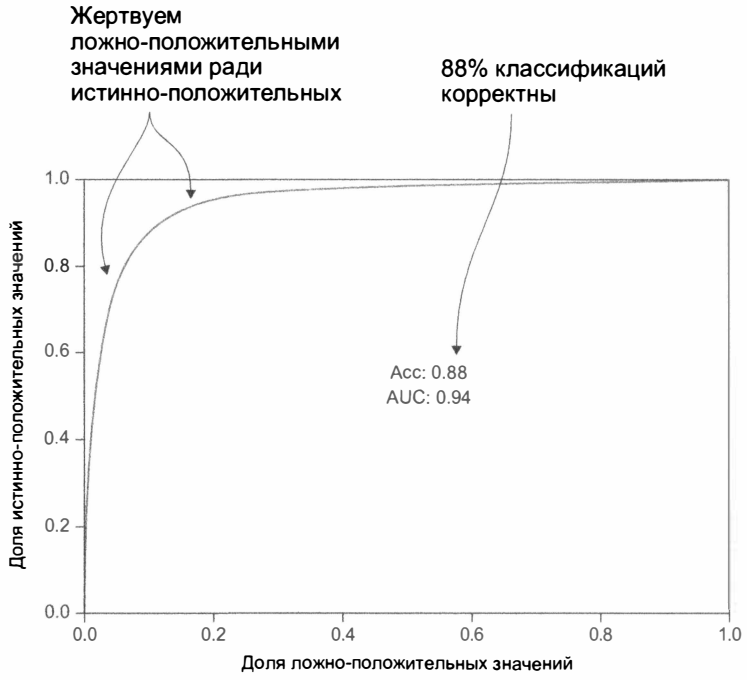
Для оценки производительности такой модели мы зададим функцию в листинге 8.3 и вызовем ее для исходных предсказаний. В качестве оценочных метрик используем общую точность классификации (долю корректно классифицированных документов), кривую рабочей характеристики приемника и соответствующее значение AUC. Все эти метрики рассматривались в главе 4 и использовались во многих примерах.

Листинг 8.3. Оценка первого варианта модели

```
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve

def performance(y_true, pred, color="g", ann=True):
    acc = accuracy_score(y_true, pred[:,1] > 0.5)
    auc = roc_auc_score(y_true, pred[:,1])
    fpr, tpr, thr = roc_curve(y_true, pred[:,1])
    plot(fpr, tpr, color, linewidth="3")
    xlabel("False positive rate")
    ylabel("True positive rate")
    if ann:
        annotate("Acc: %0.2f" % acc, (0.2,0.7), size=14)
        annotate("AUC: %0.2f" % auc, (0.2,0.6), size=14)

performance(d_test.sentiment, pred1)
```

Результат работы этого кода показан на илл. 8.4.

Как видите, производительность нашей примитивной модели вовсе не так плоха, как могло показаться. Корректно классифицировано 88% обзоров, но соотношение между долями истинно-положительных и ложно-положительных значений можно менять в зависимости от того, что предпочтительнее — более сильный шум или лучший уровень распознавания.



Илл. 8.4. Кривая рабочей характеристики приемника для простой модели на базе «мешка слов». На рисунке показаны такие метрики, как точность классификации — доля корректно классифицированных обзоров — и AUC (область под ROC-кривой). Точность показывает, что мы можем ожидать корректной классификации в 88% случаев, но с помощью ROC-кривой можно жертвовать долей ложно-положительных значений (FPR) ради доли истинно-положительных (TPR), и наоборот. Если классификация дает много обзоров, нуждающихся в проверке человеком, можно зафиксировать низкое значение FPR, но это, в свою очередь, уменьшит частоту истинно-положительных распознаваний

Попробуем поработать с новыми обзорами, пропуская их текст через векторизатор и нашу модель и оценивая предсказанную тональность. Начнем с обзора «I love this movie» (Я люблю этот фильм):

```
>>> review = "I love this movie"
>>> print model1.predict(vectorizer.transform([review]))[0]
1
```

Положительной тональности соответствует значение 1, так что угадано верно. Попробуем вариант «This movie is bad» (Это плохое кино):

```
>>> review = "This movie is bad"
>>> print model1.predict(vectorizer.transform([review]))[0]
0
```

Отрицательная тональность обозначается как 0, то есть мы снова получили верную оценку. Теперь попробуем создать для модели неоднозначную ситуацию, написав «I was going to say something awesome, but I simply can't, because the movie is so bad» (Я собирался сказать что-нибудь замечательное, но просто не могу, так как фильм очень плох):

```
>>> review = "I was going to say something awesome, but I simply can't
    because the movie is so bad."
>>> print model1.predict(vectorizer.transform([review]))[0]
0
```

Не получилось, мы все равно получили верное предсказание. Может быть, стоит добавить в отрицательный отзыв больше положительных слов? Возьмем вариант «I was going to say something awesome or great or good, but I simply can't because the movie is so bad» (Я собирался сказать что-нибудь замечательное, великолепное или хорошее, но просто не могу, так как фильм очень плох).

```
>>> review = "I was going to say something awesome or great or good, but I
simply can't because the movie is so bad."
>>> print model1.predict(vectorizer.transform([review]))[0]
0
```

Обмануть эту умную модель не получилось. Наверное, слово «bad» сильно влияет на результат классификации, поэтому попробуем обмануть модель, вставив его в положительный отзыв «It might have bad actors, but everything else is good» (Несмотря на плохих актеров, все остальное хорошо):

```
>>> review = "It might have bad actors, but everything else is good."
>>> print model1.predict(vectorizer.transform([review]))[0]
0
```

Наконец мы смогли хоть как-то обмануть модель. Это маленькое забавное упражнение показывает, насколько хорошо наша модель понимает написанные обычным языком произвольные текстовые фрагменты, относящиеся к рецензиям на кинофильмы. В следующем разделе мы попробуем ее усовершенствовать с помощью дополнительных признаков и подбора более подходящих значений для параметров алгоритма.

8.2.3. Нормализация признаков, полученных из «мешка слов», алгоритмом tf-idf

В предыдущей главе вы познакомились с такой характеристикой, как tf-idf, позволяющей усовершенствовать простые, полученные путем подсчета слов признаки. По сути, эта мера нормализует результаты подсчета слов на основе того, как часто каждое слово появляется в рассматриваемых документах. Распространенные слова получают меньший вес, а относительно редкие — бóльший. Это позволяет более подробно рассмотреть слова (зачастую крайне информативные), реже появляющиеся в наборе данных.

В этом разделе мы применим характеристику tf-idf к нашим признакам и посмотрим, как это повлияет на точность модели. Библиотека scikit-learn сильно упрощает задачу, так как нам достаточно перейти от модуля `CountVectorizer` к модулю `TfidfVectorizer`. Итоговый код показан в следующем листинге.

Листинг 8.4. Использование в модели tf-idf-признаков

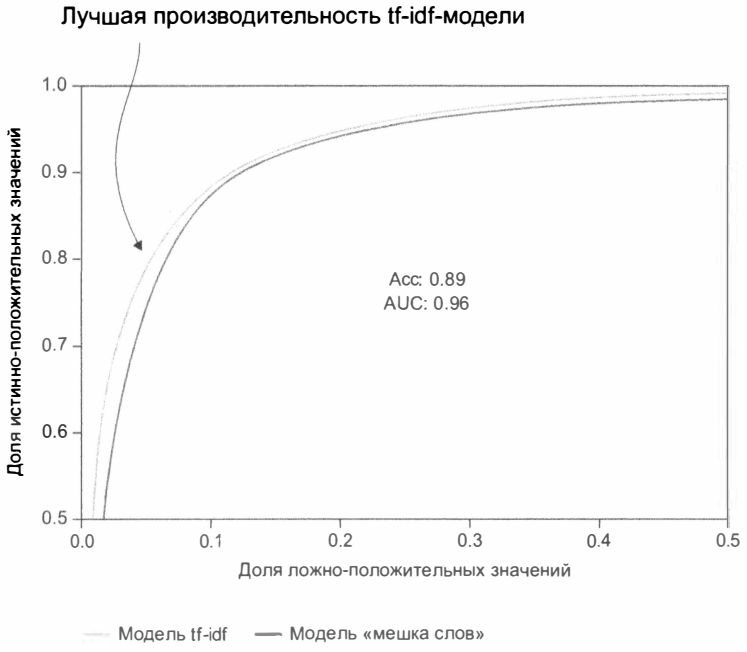
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(d_train.review)

model2 = MultinomialNB()
model2.fit(features, d_train.sentiment)
pred2 = model2.predict_proba(vectorizer.transform(d_test.review))

performance(d_test.sentiment, pred2)
```

Использует для построения признаков векторизатор Tfidf

Обучает модель на базе наивного байесовского классификатора на новых признаках и генерирует прогноз

Выводит результаты

Производительность tf-idf-модели показана на илл. 8.5. Можно увидеть, что tf-idf-признаки слегка увеличивают точность классификации. Конкретно ROC-кривая показывает, что лучше избегать ложно-положительных значений. Представьте, что в обработку поступает множество обзоров, но негативные мы хотим помечать для дальнейшего просмотра человеком. Маленькая доля ложно-положительных классификаций уменьшит количество положительных рецензий, ошибочно переданных на рассмотрение, ускорив работу рецензента.



Илл. 8.5. Кривая рабочей характеристики приемника для tf-idf-модели поверх кривой для предыдущей модели «мешка слов». Мы видим небольшое увеличение как точности классификации, так и параметра AUC (области под ROC-кривой). Точнее говоря, кривая модели tf-idf показывает улучшения в нижней части диапазона FPR; для такого же уровня корректно классифицированных обзоров модель дает меньшую долю ложно-положительных значений. Если в конвейере классификации задействован человек, количество его работы уменьшится

Как у алгоритма, полученного применением меры tf-idf к процессу извлечения признаков из естественного текста, так и у наивного байесовского классификатора есть настройки, позволяющие сконцентрироваться на конкретных деталях нашего набора данных. Мы называем их *гиперпараметрами*. Все дело в том, что переменные (признаки) модели также можно рассматривать как параметры, тогда как параметры алгоритма работают на более высоком уровне. В качестве завершающей попытки увеличения производительности модели важно рассмотреть различные значения таких параметров. Этим мы и займемся в следующем разделе.

8.2.4. Оптимизация параметров модели

Для поиска оптимальных параметров проще всего построить набор моделей с различными параметрами и посмотреть на их метрики производительности. Проблема в том, что параметры нельзя рассматривать независимо друг от друга, — изменение одного параметра может повлиять на оптимальное значение другого. Она решается путем перебора, то есть мы строим модели для различных комбинаций параметров. К сожалению, при большом количестве параметров задача быстро становится неразрешимой, особенно если времени требует построение даже одной модели. Варианты обхода этой проблемы обсуждались в главе 4, но вы, скорее всего, удивитесь, узнав, насколько часто на практике применяется метод перебора. Нужно выработать интуитивное понимание того, какие параметры более независимы друг от друга и какие из них оказывают самый сильный эффект на набор данных конкретного типа. В этом разделе мы рассмотрим упражнение на оптимизацию трех параметров — двух у характеристики tf-idf (`max_features`, `min_df`) и одного у наивного байесовского классификатора (`nb_alpha`).

Первым делом нам потребуется функция, которую мы будем циклически вызывать для построения модели и возвращения параметров и интересующих нас метрик (в данном случае AUC). Код ее определения показан в следующем листинге.

Листинг 8.5. Метод построения модели, используемый для оптимизации параметров

```
def build_model(max_features=None, min_df=1, nb_alpha=1.0):
    vectorizer = TfidfVectorizer(max_features=max_features, min_df=min_df)
    features = vectorizer.fit_transform(d_train.review)
    model = MultinomialNB(alpha=nb_alpha)
    model.fit(features, d_train.sentiment)
    pred = model.predict_proba(vectorizer.transform(d_test.review))
    return {
        "max_features": max_features,
        "min_df": min_df,
        "nb_alpha": nb_alpha,
        „auc": roc_auc_score(d_test.sentiment, pred[:,1])
}
```

Теперь, когда у нас есть функция построения модели, можно запустить конвейер оптимизации, задавая в цикле возможные значения параметров

(выбираемые случайным образом или интуитивно). Эта задача решается в следующем листинге.

Листинг 8.6. Цикл оптимизации параметров

```
from itertools import product

param_values = {
    "max_features": [10000, 30000, 50000, None],
    "min_df": [1,2,3],
    "nb_alpha": [0.01, 0.1, 1.0]
}

results = []
    for p in product(*param_values.values()):
    res = build_model(**dict(zip(param_values.keys(), p)))
    results.append( res )
    print res
```

Задает значения подлежащих оптимизации параметров

Для каждой комбинации значений параметров

Строит модель и сохраняет результат

Рассмотрим оптимизируемые параметры:

- `max_features`. Максимальное количество столбцов со словами, которое будет создавать алгоритм tf-idf. Из имеющихся данных мы знаем, что всего у нас примерно 65 000 столбцов, поэтому мы проверяем сходные числа из этого же диапазона. Значение `None` означает, что будут использоваться все слова.
- `min_df`. Минимальное количество появлений слова в наборе данных, необходимое для включения в число признаков. Это пример потенциальной зависимости параметров, так как количество слов в словаре (и соответственно значение параметра `max_features`) можно изменить, меняя параметр `min_df`.
- `nb_alpha`. Параметр сглаживания наивного байесовского классификатора. Это единственный поддающийся регулировке параметр алгоритма. Для выбора значений нужно узнать смысл этого параметра и как он применяется в других обстоятельствах.

Стоит упомянуть, что в коде из листинга 8.6 используется функция `product` из модуля `itertools`. Этот модуль содержит набор функций Python, упрощающих работу с данными. Указанная функция удобно генерирует все комбинации набора списков (декартово произведение). Результат работы кода из листинга 8.6 показан на илл. 8.6.

	AUC	max_features	min_df	nb_alpha
17	0.955985	30000	3	1.00
18	0.970304	50000	1	0.01
19	0.967335	50000	2	0.01
20	0.963369	50000	3	0.01
21	0.968388	50000	1	0.10
22	0.965854	50000	2	0.10
23	0.962516	50000	3	0.10
24	0.958776	50000	1	1.00
25	0.957700	50000	2	1.00
26	0.956112	50000	3	1.00
27	0.973386	NaN	1	0.01
28	0.967335	NaN	2	0.01

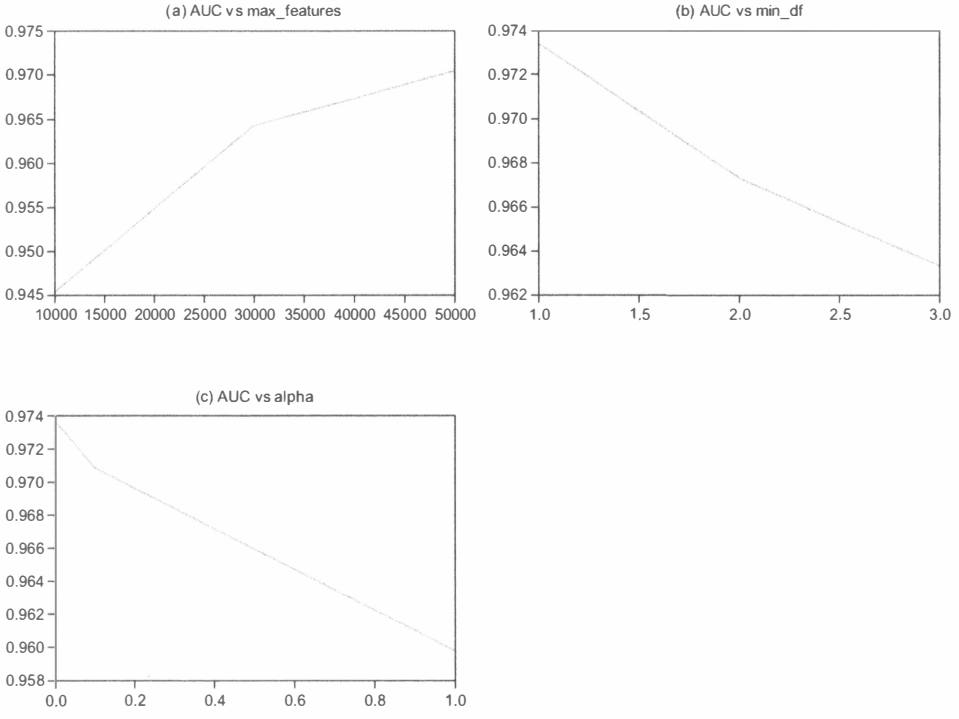
Самое большое значение AUC получено на 27-й итерации

Илл. 8.6. Подмножество результатов из цикла оптимизации параметров. Комбинация параметров на 27-й итерации дает самый лучший результат работы модели

Иллюстрация 8.6 демонстрирует результаты некоторых итераций процесса оптимизации. У нас было всего три параметра и 36 возможных комбинаций их значений, поэтому вычисления заняли не более 10 минут, ведь модель на базе наивного байесовского классификатора обучается относительно быстро, но можно легко представить, сколько времени потребуется на оптимизацию большего числа параметров с большим количеством значений. Впрочем, мы можем начать с широкого диапазона значений, постепенно сужая его и проводя последующую оптимизацию для различных значений. Из таблицы видно, как параметры влияют на AUC модели. Лучшие результаты получены на итерации 27 со следующими значениями:

- `max_features` — None (все слова, значение по умолчанию);
- `min_df` — 1 (значение по умолчанию);
- `nb_alpha` — 0,01.

Итак, нам удалось еще немного улучшить производительность модели, подобрав оптимальное значение для параметра сглаживания наивного байесовского классификатора. Теперь посмотрим на зависимость AUC от каждого параметра (для двух других при этом фиксируются оптимальные значения). Она показана на илл. 8.7.

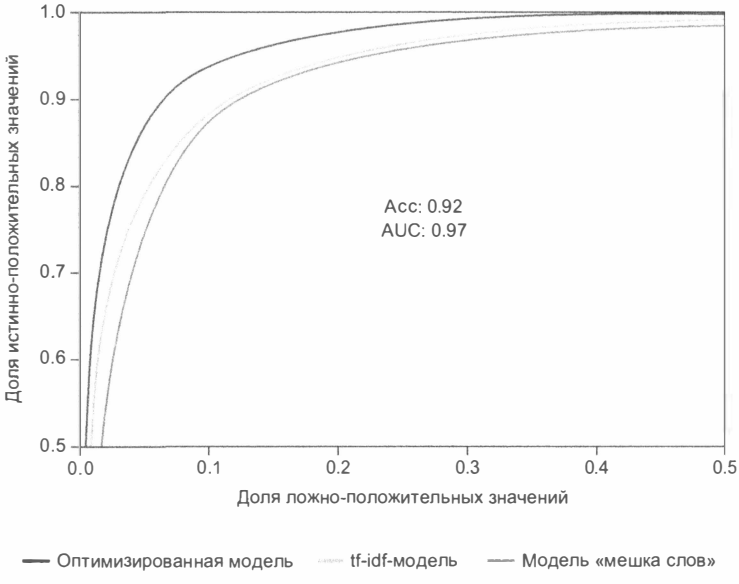


Илл. 8.7. Улучшение AUC при изменении параметров признаков и ML-алгоритма. Мы видим, что (a) более высокое значение max_features улучшает AUC, (b) понижая значение min_df, мы улучшаем AUC и (c) более низкий параметр сглаживания улучшает AUC. Это не означает, что комбинация лучших значений даст самый лучший результат. В комбинации лучше всего работают значения max_features=None (все слова, значение по умолчанию), min_df=1 (минимум, значение по умолчанию), alpha=0,01 (основная причина для улучшения). Самое высокое значение AUC — 0,974. Все показанные графики можно воспроизвести с помощью кода из записной книжки Python

Каждый из этих графиков — только одна точка зрения на процесс изменения AUC, так как для значений этой метрики как функции от всех параметров нам потребовался бы четырехмерный график. Но все равно интересно наблюдать, как модель реагирует на изменение каждого значения. К примеру, чем больше у нас признаков, тем лучше (выиграло

максимально возможное значение). Чем меньше значение параметра `min_df`, тем лучше (выиграло минимально возможное значение). И кроме того, чем меньше параметр `nb_alpha`, тем лучше. Так как теоретически он не имеет нижнего предела, можно попробовать еще сильнее понизить его значение на следующей итерации. Оставляем вам это в качестве самостоятельного упражнения (но мы не смогли найти намного лучшего значения).

Иллюстрация 8.8 демонстрирует ROC-кривые для оптимизированной и всех остальных моделей. Мы видим значительное увеличение производительности для обеих метрик и всех точек ROC-кривой. Это великолепный пример того, как время, потраченное на регулировку гиперпараметров, вознаграждается увеличившейся точностью предсказаний. И последнее, на что следует обратить внимание. Несложно догадаться, что новые варианты параметров модели, в свою очередь, влияют на то, какие признаки и какой алгоритм моделирования (например, подсчета слов или tf-idf) даст наилучший результат. При этом у каждого алгоритма будет свой набор подлежащих оптимизации параметров. Строго говоря,



Илл. 8.8. Сравнение ROC-кривых оптимизированной модели и предыдущих моделей. По оценке на тестовой выборке последняя модель выглядит самой лучшей (на всех точках кривой), а ожидаемая точность значительно выросла

оптимизировать следует все рассматриваемые алгоритмы и их параметры, но в большинстве случаев это физически невозможно, поэтому процесс оптимизации разбивают на этапы. К примеру, сначала мы оптимизируем выбранный NLP-алгоритм, затем ML-модель, а затем подбираем наилучшие параметры. В других проектах этапы оптимизации будут иными. Постепенно, по мере роста опыта, у вас появится чутье на такие вещи.

Показанные на илл. 8.8 ROC-кривые завершают наши эксперименты с первым вариантом модели. Базовый алгоритм и небольшое количество кода позволили построить модель, с хорошей точностью работающую с текстовыми данными. В следующем разделе мы сделаем еще один шаг в области проектирования признаков и моделирования, а также рассмотрим различные аспекты запуска модели в производство.

8.3. Усовершенствованные алгоритмы и тонкости процесса внедрения

В предыдущем разделе мы строили модели, пользуясь простыми признаками и простыми ML-алгоритмами. Точность любой из этих моделей была вполне достаточной для наших целей. Можно попробовать еще больше оптимизировать модель, но всегда приходится искать компромисс между потраченным на оптимизацию временем и потенциальной полезностью инкрементных улучшений точности прогнозирования. Мы рекомендуем изучать важность каждого улучшения, например, сопоставив уменьшение времени работы человека-рецензента и время, которое вы потратите на оптимизацию. Как вы видели, даже самая первая модель во многих случаях была в состоянии уловить тональность обзора и для начала вполне подходила. Зачастую полезнее запустить в работу модель со слегка сниженной точностью предсказаний и по возможности посмотреть на реакцию системы.

Но мы не прислушаемся к этому странному совету, а продолжим процесс оптимизации. Воспользуемся для генерации дополнительных признаков новым инструментом, разработанным компанией Google, — word2vec. После этого заменим алгоритм, так как для новых признаков лучше подойдет «случайный лес».

8.3.1. Word2vec-признаки

Проект word2vec предложила компания Google как относительно новый подход к обработке естественного языка. Сам инструмент word2vec представляет собой модель с машинным обучением, построенную на базе глубоких нейронных сетей — раздела ML, в котором в последнее время получают потрясающие результаты, особенно в областях, связанных с деятельностью человека, например с естественным языком, речью и изображениями.

Для построения word2vec-модели на базе нашей обучающей выборки воспользуемся библиотекой Gensim языка Python, в которую встроена прекрасная реализация word2vec. Мы уже прибегали к ней в главе 7 при изучении латентного размещения Дирихле — другой тематической модели, сходной с word2vec.

Нам потребуется подготовить документ к моделированию, так как Gensim-алгоритмы работают с предложениями (списками слов), а не с произвольными документами. Эта дополнительная работа заодно позволяет посмотреть, что именно попадает в модель. Листинг 8.7 демонстрирует простую функцию токенизации, которая удаляет шумовые слова и знаки препинания, а также преобразует все символы в нижний регистр. Разумеется, все эти операции автоматически выполняются векторизаторами слов из библиотеки scikit-learn; сходную функциональность можно было получить, воспользовавшись инструментарием NLTK для языка Python, но для наглядности мы решили написать функцию с нуля.

Листинг 8.7. Токенизация документа

```
import re, string

stop_words = set(['all', "she'll", "don't", 'being', 'over', 'through',
'yourselves', 'its', 'before', "he's", "when's", "we've", 'had', 'should',
"he'd", 'to', 'only', "there's", 'those', 'under', 'ours', 'has',
"haven't", 'do', 'them', 'his', "they'll", 'very', "who's", "they'd",
'cannot', "you've", 'they', 'not', 'during', 'yourself', 'him', 'nor',
"we'll", 'did', "they've", 'this', 'she', 'each', "won't", 'where',
"mustn't", "isn't", "i'll", "why's", 'because', "you'd", 'doing', 'some',
'up', 'are', 'further', 'ourselves', 'out', 'what', 'for', 'while',
"wasn't", 'does', "shouldn't", 'above', 'between', 'be', 'we', 'who',
"you're", 'were', 'here', 'hers', "aren't", 'by', 'both', 'about', 'would',
```

```
'of', 'could', 'against', "i'd", "weren't", "i'm", 'or', "can't", 'own',
'into', 'whom', 'down', "hadn't", "couldn't", 'your', "doesn't", 'from',
"how's", 'her', 'their', "it's", 'there', 'been', 'why', 'few', 'too',
'themselves', 'was', 'until', 'more', 'himself', "where's", "i've", 'with',
"didn't", "what's", 'but', 'herself', 'than', "here's", 'he', 'me',
"they're", 'myself', 'these', "hasn't", 'below', 'ought', 'theirs', 'my',
"wouldn't", "we'd", 'and', 'then', 'is', 'am', 'it', 'an', 'as', 'itself',
'at', 'have', 'in', 'any', 'if', 'again', 'no', 'that', 'when', 'same',
'how', 'other', 'which', 'you', "shan't", 'our', 'after', "let's", 'most',
'such', 'on', "he'll", 'a', 'off', 'i', "she'd", 'yours', "you'll", 'so',
"we're", "she's", 'the', "that's", 'having', 'once'])

def tokenize(docs):
    pattern = re.compile('[\W_]+', re.UNICODE)
    sentences = []
    for d in docs:
        sentence = d.lower().split(" ")
        sentence = [pattern.sub('', w) for w in sentence]
        sentences.append( [w for w in sentence if w not in stop_words] )
return sentences
```

После преобразования всех символов в нижний регистр разбивает документ на слова

Удаляет все символы, не являющиеся словами, например знаки препинания

Удаляет английские стоп-слова

С этой функцией в токены можно превратить любой список документов, поэтому перейдем к построению нашей первой word2vec-модели. Дополнительную информацию о параметрах используемого алгоритма можно найти в документации на сайте библиотеки Gensim[1].

Листинг 8.8. Word2vec-модель

```
from gensim.models.word2vec import Word2Vec

sentences = tokenize(d_train.review)
model = Word2Vec(sentences, size=300, window=10, min_count=1,
    sample=1e-3, workers=2)
model.init_sims(replace=True)
print model['movie']
#> array([ 0.00794919, 0.01277687, -0.04736909, -0.02222243, …])
```

Генерирует предложения с помощью функции токенизации

Строит и нормализует word2vec-модель

Выводит вектор word2vec-модели для слова «movie»

Вы видите, каким образом одно слово преобразуется в вектор (в рассматриваемом случае он состоит из 300 чисел). Чтобы word2vec-модель смогла сгенерировать признаки для нашего ML-алгоритма, обзоры сле-

[1] *https://radimrehurek.com/gensim/models/word2vec.html*

дует превратить в векторы признаков. Вы уже уместе представлять в векторном виде отдельные слова, а в данном случае требуется представить документ обзора (список слов) как средний вектор всех составляющих его слов. Следующий листинг демонстрирует построение решающей эту задачу функции.

Листинг 8.9. Генерация Word2vec-признаков

```
def featurize_w2v(model, sentences):
    f = zeros((len(sentences), model.vector_size))
    for i,s in enumerate(sentences):
        for w in s:
            try:
                vec = model[w]
            except KeyError:
                continue
            f[i,:] = f[i,:] + vec
        f[i,:] = f[i,:] / len(s)
    return f
```

Инициализирует массив NumPy для векторов признаков

Циклически просматривает каждое предложение, добавляет векторы для каждого слова и берет среднее

Все готово для построения модели на основе сгенерированных word2vec-признаков. Возможно, из нашего обсуждения в разделе 8.2.2 вы помните, что наивный байесовский классификатор хорошо работает с разреженными данными, но не очень подходит для плотных данных. В результате преобразования мы перешли от ~65 000 разреженных признаков, полученных путем подсчета слов, к сотням плотных word2vec-признаков. Модель глубокого обучения изучила темы высокого уровня (листинг 8.8), и теперь каждый документ можно представить как комбинацию тем (листинг 8.9).

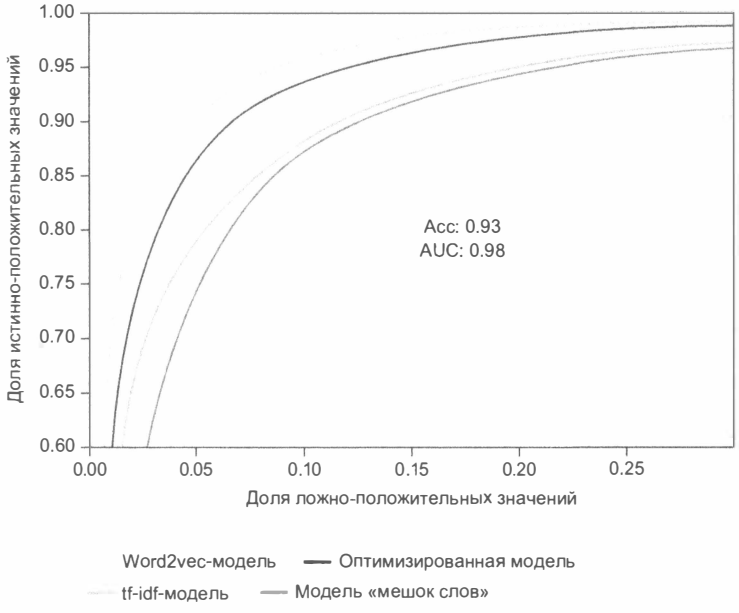
8.3.2. Модель на базе алгоритма «случайный лес»

Наивный байесовский алгоритм, с которым мы работали в предыдущих разделах, несовместим с новыми word2vec-признаками, так как их нельзя представить как порождения полиномиального распределения. Можно поменять распределение и продолжить работу с наивным байесовским алгоритмом, но вместо этого мы обратимся к старому знакомому — алгоритму «случайный лес». В следующем листинге строится модель «случайного леса» из 100 деревьев на базе word2vec-признаков и, как обычно, анализируется ее производительность на тестовой выборке.

Листинг 8.10. Построение модели на базе алгоритма «случайный лес» и word2vec-признаков

```
features_w2v = featurize_w2v(model, sentences)

model4 = RandomForestClassifier(n_estimators=100, n_jobs=-1)
model4.fit(features_w2v, d_train.sentiment)

test_sentences = tokenize(d_test.review)
test_features_w2v = featurize_w2v(model, test_sentences)
pred4 = model4.predict_proba(test_features_w2v)
performance(d_test.sentiment, pred4, color="c")
```

На илл. 8.9 производительность word2vec-модели на базе алгоритма «случайный лес» сравнивается с показателями предыдущих моделей. Легко заметить, что точность новой модели и в самом деле выше, как в выбранных оценочных метриках, так и во всех точках ROC-кривой.



Илл. 8.9. Кривые рабочей характеристики приемника word2vec-модели и ранее построенных моделей. Легко заметить улучшение для всех точек ROC-кривой, также отражающееся в увеличившейся точности и величине параметра AUC

Производительность последнего варианта модели нас вполне устраивает, так что работу по оптимизации можно прекратить. Но вы можете попро-

бовать другие варианты увеличения точности. С большой вероятностью даже человек будет не в состоянии корректно классифицировать тональность всех обзоров; могут как появиться неверные метки, так и обзоры, понять тональность которых крайне сложно.

Но модель может работать намного лучше, чем в настоящий момент. Поэтому мы составили для вас, наши дорогие читатели, список того, что можно попробовать сделать в порядке приоритетности:

- *Воспользоваться немаркированными данными для построения лучшей тематической модели.*

  Посвященный данным раздел на сайте Kaggle содержит немаркированный набор рецензий, которые можно использовать для обучения. Так как мы реализуем обучение с учителем, на первый взгляд эти данные кажутся бесполезными. Но мы строим word2vec-модель, которая должна изучить нюансы встречающихся в обзорах на сайте IMDb слов, а особенно связи между различными словами и понятиями, соответственно эти данные помогут улучшить ее первый вариант, с признаками на базе обучающей выборки (которые оснащены метками), до того, как вы построите модель целиком.

- *Оптимизировать параметры.*

  Вы видели, как сильно увеличивалась производительность первых вариантов нашей модели после подбора оптимальных значений гиперпараметров. Но мы уже успели построить новую модель (word2vec) и взять новый ML-алгоритм («случайный лес»), так что у вас появилось множество новых параметров, которые можно оптимизировать.

- *Распознать фразы.*

  Библиотека Gensim включает в себя поддержку распознавания фраз, таких как «New York City», которые упускает наша «глупая» функция токенизации отдельных слов. В английском языке распространены понятия, состоящие из нескольких слов, и их имеет смысл включить в функцию генерации предложений.

- *Обработать разные языки.*

  Если не все обзоры написаны на одном языке (в рассматриваемом случае на английском), придется вставлять в различные места на-

шего конвейера обработку различных языков. Первым делом нужно понять, на каком языке написана рецензия, то есть распознать язык (существует несколько библиотек, с разным успехом решающих эту задачу). Затем с помощью этой информации процесс токенизации следует заставить пользоваться различными шумовыми словами и, возможно, разными знаками препинания. В особо неудачных случаях приходится иметь дело с совершенно другой структурой предложений, как, например, в китайском языке, где отличить одно слово от другого можно только по наличию пробела.

Теперь представим, что полученная модель вас устраивает. В реальном мире на этой стадии модель запускают в производство. При этом учитываются следующие аспекты:

- *Насколько велик размер обучающей выборки и не улучшится ли модель, если мы добавим в эту выборку дополнительные данные?*

  Этот аспект влияет на выбор ML-алгоритма, так как готовая модель должна хорошо масштабироваться при росте обучающей выборки. К примеру, наивный байесовский классификатор поддерживает так называемое динамическое обучение, в то время как алгоритм «случайный лес» сложно адаптировать к увеличившемуся набору данных.

- *Каков объем прогнозов и должны ли они делаться в реальном времени?*

  Подробно масштабирование прогнозов по объему и скорости будет обсуждаться в следующей главе, а пока следует учесть, что ответ на этот вопрос тоже влияет на выбор алгоритма и инфраструктуры, в которой будет разворачиваться модель.

8.4. Заключение

В этой главе мы на практическом примере рассмотрели процесс машинного обучения от начала до конца, заодно обсудив основы обработки естественного текста и параметры оптимизации модели.

Вот то, что следует запомнить:

- Важно правильно формулировать задачу. Работу всегда следует начинать с ответа на вопрос: «Какова практическая ценность решения данной задачи?».

- Анализируйте данные, чтобы понять, хватит ли их для решения поставленной задачи.
- Начинайте работу с простых стандартных алгоритмов, позволяющих построить первый вариант модели. В нашем примере тональность рецензий была предсказана с почти 90%-й точностью.
- Точность предсказаний можно увеличивать путем тестирования и оценки альтернативных моделей, а также комбинаций параметров модели.
- Зачастую приходится искать компромисс между различными параметрами модели и оценочными критериями. Мы посмотрели, как для нашей модели оценки тональности компромисс между долями ложно-положительных и ложно-отрицательных значений выражается с помощью ROC-кривой.
- Ультрасовременные техники обработки естественного языка и ML-моделирования, такие как word2vec, — примеры того, как усовершенствованное проектирование признаков улучшает точность работы моделей.
- Выбор алгоритма может зависеть не только от точности модели, а, например, от времени обучения, необходимости добавления новых данных или получения прогнозов в реальном времени.
- Реальные модели всегда можно улучшить.

8.5. Терминология

Термин	Определение
word2vec	Программный инструмент для анализа текстов, выпущенный Google и используемый множеством ультрасовременных систем машинного обучения, работающих с естественным языком
Оптимизация гиперпараметров (hyperparameter optimization)	Различные техники выбора параметров, обеспечивающие максимальную производительность ML-алгоритмов

9 Масштабирование процесса машинного обучения

В этой главе:

- ✓ как определить, когда масштабирование модели помогает увеличить точность и скорость генерации предсказаний;
- ✓ как избежать ненужных вложений в сложные стратегии масштабирования и большую инфраструктуру;
- ✓ способы масштабирования линейных ML-алгоритмов для работы с крупными обучающими выборками;
- ✓ подходы к масштабированию нелинейных ML-алгоритмов — обычно это намного более сложная задача;
- ✓ уменьшение задержки и увеличение скорости генерации прогнозов.

В реальных приложениях с машинным обучением во главу угла зачастую ставится масштабируемость. Бывают системы с машинным обучением, предназначенные для быстрой переработки новых данных и генерации предсказаний, так как через несколько миллисекунд предсказания становятся бесполезными (представьте, к примеру, приложения, в реальном времени работающие с рынком ценных бумаг). С другой стороны, иногда масштабирование требуется в процессе обучения модели, так как ей приходится иметь дело с гигабайтами или даже терабайтами данных.

В предыдущих главах мы в основном имели дело с небольшими наборами данных, что позволяло легко организовать обучение, обработку и моделирование на одном компьютере. Встречаются практические задачи, для решения которых этого вполне достаточно, но куда чаще возникает ситуация, когда требуется масштабирование до нескольких компьютеров, а иногда и до сотен машин в облаке. Эта глава посвящена выбору стратегии масштабирования и изучению связанных с этим процессом технологий.

В первой части речь пойдет об аспектах, которые следует учитывать, сталкиваясь с большим набором данных или с необходимостью генерировать предсказания в большом объеме. Мы рассмотрим способы, позволяющие быстро и с минимальными затратами получить масштабируемое решение, а также расскажем о технологиях, которые придут на помощь в случаях, когда построение такого решения требует больших затрат времени и ресурсов. Следующий раздел более подробно продемонстрирует процесс масштабирования для обучения модели на больших наборах данных. В конце мы поговорим о том, как увеличить объем предсказаний или уменьшить их задержку.

В следующей главе вы примените все это на практике, рассматривая пример решения задачи с большим набором данных.

9.1. Перед началом масштабирования

Тип масштабирования в каждом конкретном случае зависит от особенностей задачи и имеющихся в вашем распоряжении вычислительных мощностей. Мы начнем с описания варианта, который часто требуется современным приложениям с машинным обучением. Будут рассмотрены

аспекты, которые требуется принять во внимание, чтобы понять, какие ограничения имеет ваш ML-код. Потом, когда вы научитесь определять требуемый тип масштабирования, мы познакомим вас со стандартными техниками, позволяющими ML-приложениям справиться со скоростью поступления и объемами данных.

Но вместо того чтобы сразу перейти к рассмотрению конкретных методов масштабирования ML-приложений, мы начнем с обзора ситуации в целом. Используя рабочий процесс машинного обучения в качестве руководства, начнем системное рассмотрение возможностей его масштабирования.

9.1.1. Определяем важные аспекты

Первым делом разобьем рабочий процесс на две части — обучение модели и генерация прогнозов. Какое влияние оказывают ограниченные вычислительные ресурсы в каждом из этих случаев и каким образом они замедляют или даже останавливают работу системы? См. табл. 9.1.

Таблица 9.1. Проблемы с построением модели, которые могут появиться из-за невозможности масштабирования, и их последствия

Проблема масштабирования	Следствие
Слишком большая для модели обучающая выборка	Модель невозможно обучить, а значит, прогнозов просто не будет
Крайне медленное обучение модели из-за больших размеров обучающей выборки	Оптимизация модели становится невозможной (или нецелесообразной), поэтому приходится жертвовать точностью прогнозов

Проблемы, с которыми приходится сталкиваться на стадии построения модели, возникают из-за большого размера обучающих выборок. Иногда данных так много, что модель даже не удается обучить (например, из-за того, что данные не помещаются в памяти). В этой ситуации возможны три варианта действий: (1) найти меньшее подмножество данных для обучения модели, (2) воспользоваться компьютером с большей оперативной памятью или (3) взять алгоритм обучения, который более рационально использует память.

Чуть ниже мы опишем, как можно быстро уменьшить размер набора данных без значительного снижения качества модели. Затем мы обсудим способ масштабирования вычислительных циклов с помощью расширяемых систем данных. После этого вы познакомитесь с масштабируемыми обучающими алгоритмами, которые приводят модель в соответствие с данными, избавляя нас от необходимости искать обходные пути или использовать дополнительное аппаратное обеспечение.

Возможна и ситуация, когда на наборе данных можно обучить только относительно простые модели (например, линейной/логистической регрессии), в то время как более сложные алгоритмы, требующие большего объема памяти (такие, как бустинг), с ними не справляются. В этом случае можно пожертвовать точностью прогнозирования и обучить модель, базирующуюся на более простом алгоритме. Имеет смысл воспользоваться и описанными выше обходными вариантами.

Бывает и так, что огромный размер набора данных сильно замедляет процесс обучения модели и ее оптимизации. Это вынуждает нас, как и в предыдущем сценарии, остановиться на менее точной модели, так как приходится пользоваться более грубой стратегией настройки параметров или вообще от нее отказаться. Но выйти из затруднительного положения можно разверткой дополнительных узлов (горизонтальное масштабирование) и обучением моделей (с различными комбинациями настраиваемых параметров) на разных машинах. Подробно горизонтальное масштабирование будет рассматриваться в разделе 9.1.3.

В процессе генерации предсказаний проблемы с масштабированием возникают из-за слишком быстрого потока данных, процессов генерации предсказаний и проектирования признаков, сильно загружающих процессор, или слишком больших пакетов данных с предсказаниями. Рассмотрим табл. 9.2.

Таблица 9.2. Проблемы с предсказаниями, возникающие из-за отсутствия масштабируемости, и их последствия

Проблема масштабирования	Следствие
Слишком высокая скорость передачи данных (потоковая) для обработки ML-системой	Предсказания запаздывают все сильнее и сильнее, пока их генерация не прекращается

Проблема масштабирования	Следствие
Код проектирования признаков и/или процесса предсказания слишком медленный, чтобы сгенерировать своевременный прогноз	Прогнозы утрачивают свою практическую ценность, особенно в случаях, когда информация требуется в реальном времени
Размер данных (пакета) слишком велик, и модель не может его обработать	Система генерации предсказаний перестает работать

К счастью, все три проблемы можно разрешить одним и тем же способом — увеличив число компьютеров. Преимущество прогнозирования перед обучением модели состоит в том, что в подавляющем большинстве случаев предсказания можно делать независимо для каждого экземпляра данных[1]. Для их генерации в каждый момент времени в памяти нужно держать признаки всего одного экземпляра (и построенной вами ML-модели). В процессе обучения в память, как правило, следует загрузить всю обучающую выборку. Соответственно для решения проблем на стадии генерации предсказаний не требуются более мощные компьютеры; достаточно увеличить их число. Ну и, разумеется, не обойтись без эффективной системы управления данными, которая будет их контролировать.

Если нужно ускорить генерацию предсказаний, обработать больший объем экземпляров или разобраться с медленным процессом проектирования признаков, добавьте в систему дополнительные компьютеры и отправляйте подмножества экземпляров обрабатываться на разные узлы. Затем, при условии распределения обученной модели по всем узлам сети, можно параллельно генерировать предсказания, возвращая их в центральную базу данных.

В разделе 9.3 мы подробно поговорим о том, что нужно делать для масштабирования предсказаний. Существует несколько подходов к построению вычислительных систем для быстрой генерации ML-прогнозов.

[1] Имейте в виду, что далеко не всегда можно получать предсказания для отдельных экземпляров независимо друг от друга. К примеру, модели, создающей прогнозы на базе временных рядов, для генерации одного прогноза могут потребоваться данные множества временных меток.

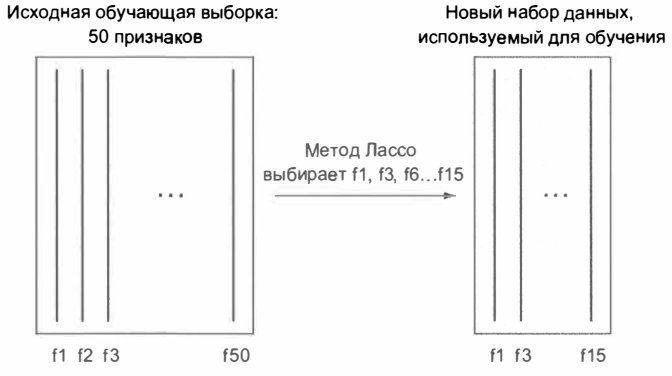
9.1.2. Прореживание обучающей выборки вместо масштабирования?

Бывают ситуации, когда из-за слишком большого размера обучающей выборки и недостаточной мощности процессора обучить модель попросту невозможно. В крайнем случае, *если у вас нет других вариантов*, можно проредить обучающую выборку перед построением модели.

Вообще мы не рекомендуем удалять данные (можно потерять важную информацию), но если этого не избежать, лучше делать это определенными способами. Иногда это даже позволяет улучшить модель, но тут все зависит от лежащего в ее основе алгоритма. Отбрасывать можно как признаки, так и экземпляры. В обоих случаях существует статистически строгий метод уменьшения размера обучающей выборки.

Отбор признаков

Слишком большой размер набора данных часто становится препятствием для вычислений. Скажем, при рассмотрении геномов обучающая выборка может включать в себя данные по миллиону генов (признаки), но всего по сотням пациентов (экземпляры). Как и при анализе текстов, превращение таких данных в n-граммы даст обучающую выборку, содержащую более миллиона признаков. Ее можно уменьшить, отбросив неважные признаки. Этот процесс схематично демонстрируется на илл. 9.1.



Илл. 9.1. Отбор признаков методом Лассо для уменьшения размерности большой обучающей выборки

В главах 4 и 5 мы показали, что в некоторых случаях отбор признаков позволяет улучшить модель. Ведь обучающий алгоритм получает возможность сосредоточиться на важных сигналах, так как его перестают отвлекать не имеющие особого значения признаки. Фактические потери или выгоды процедуры отбора признаков зависят от конкретной ML-модели, а также от того, сколько информации было потеряно, поэтому всегда постфактум нужно тестировать модель, проверяя ее работоспособность. В этом разделе отбор признаков будет рассматриваться в основном как способ работы с большими наборами данных.

Для понижения размерности таких наборов рекомендуется эффективный линейный алгоритм *Лассо Тибширани*, автоматически обнаруживающий подмножество признаков с максимальной прогностической ценностью.

Вычисление всех коэффициентов алгоритма дает представление о порядке признаков в соответствии с их прогностической ценностью. Более того, нам предоставляется оптимальное подмножество признаков применительно к линейной модели.

Если размер набора данных столь велик, что обучить не удается даже модель на базе алгоритма Лассо, можно рассмотреть подмножества экземпляров обучающей выборки (и, возможно, усреднение по проходам алгоритма). Это даст представление о том, какие признаки можно удалить без ухудшения статистической производительности ML-алгоритма.

Недостаток описанного варианта состоит в том, что для оценки важности признаков используется линейная модель. Отбираемые алгоритмом Лассо признаки могут быть связаны с целевой переменной нелинейным соотношением, которое просто не будет обнаружено. Альтернативой являются непараметрические подходы к отбору признаков, например оценка важности с помощью алгоритма «случайный лес», но, как правило, они не масштабируются на большие наборы данных.

Кластеризация экземпляров

Если после отбора признаков размер обучающей выборки *все еще* остается слишком большим для обучения модели, остается произвести отбор экземпляров. В качестве последнего средства можно воспользо-

ваться статистическими алгоритмами кластеризации для идентификации в обучающей выборке избыточных экземпляров и их последующего удаления.

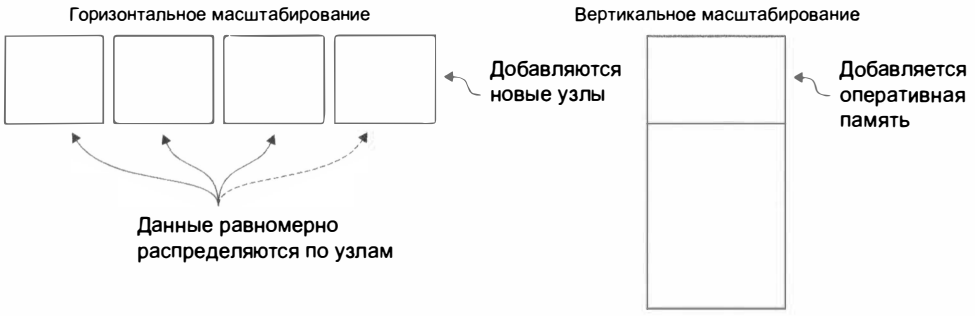
Для этого типа удаления данных рекомендуется агломеративный иерархический алгоритм кластеризации. Он инициализирует каждый экземпляр обучающей выборки как единственный член кластера. Затем ближайшие кластеры попарно объединяются (для определения «близости» используется заранее определенная мера расстояния). Процесс объединения продолжается, пока не выполнится определенное условие (например, не будет получено конкретное количество кластеров). Рекомендуем останавливать его как можно раньше, чтобы не слишком сильно пострадало информационное наполнение данных. Итоговая обучающая выборка включает в себя по одному экземпляру для каждого результирующего кластера.

9.1.3. Масштабируемые системы управления данными

Независимо от стратегии масштабирования рабочего процесса ML, первым делом нужно обработать данные. В последние десятилетия много говорилось о так называемых технологиях обработки больших данных. В нашей книге термин *большие данные* означает набор, слишком большой, чтобы его можно было обработать на одном компьютере за разумное время. Сейчас мы рассмотрим наиболее успешные проекты, связанные с большими данными, и варианты их применения в сфере машинного обучения.

Базовый принцип современных систем обработки больших данных состоит в том, что с увеличением числа данных можно справиться путем добавления компьютеров. Это так называемое *горизонтальное* масштабирование. Альтернативный способ работы с объемными ресурсами — вертикальное масштабирование — состоит в увеличении дискового пространства, памяти и ядер процессора у уже имеющихся компьютеров. Сравнение этих способов показано на илл. 9.2.

Увеличения вычислительных мощностей куда чаще, чем кажется на первый взгляд, достаточно для масштабирования рабочего процесса машинного обучения. Как утверждается в предыдущих разделах, после



Илл. 9.2. Сравнение горизонтального и вертикального масштабирования систем для обработки больших данных. В первом случае в инфраструктуру добавляются новые узлы (компьютеры), что позволяет увеличить количество вычислений, так как нагрузка равномерно распределяется по всем узлам. Примером такой системы является Apache Hadoop. В вертикально масштабируемых системах ресурсы добавляются к уже существующим машинам, позволяя им справляться с большей нагрузкой. На начальных стадиях этот подход более эффективен, но количество ресурсов, которые можно добавить, ограниченно. Пример базы данных, в которой используется такой подход, — SQL-серверы, например PostgreSQL

обработки исходных данных и их подготовки к решению задачи классификации или регрессии их размер уменьшается и становится недостаточным, чтобы оправдать переход к сложной системе, предназначенной для работы с большими данными. Но в случае данных с популярных веб-сайтов, из мобильных приложений, игр или с различных физических датчиков нужно использовать горизонтально масштабируемую систему. И дальше мы будем рассматривать именно такую ситуацию.

Горизонтально масштабируемые системы для работы с большими данными имеют два слоя — для хранения и вычислений. Из *слоя хранения* данные передаются в *слой вычислений*, где и происходит их обработка. Одним из наиболее популярных программных проектов для работы с большими данными является пакет программ Hadoop от Apache, до сих пор широко используемый в промышленности и сфере образования. Его основой послужил новый вариант масштабируемости, разработанный Google и другими компаниями в начале 2000-х.

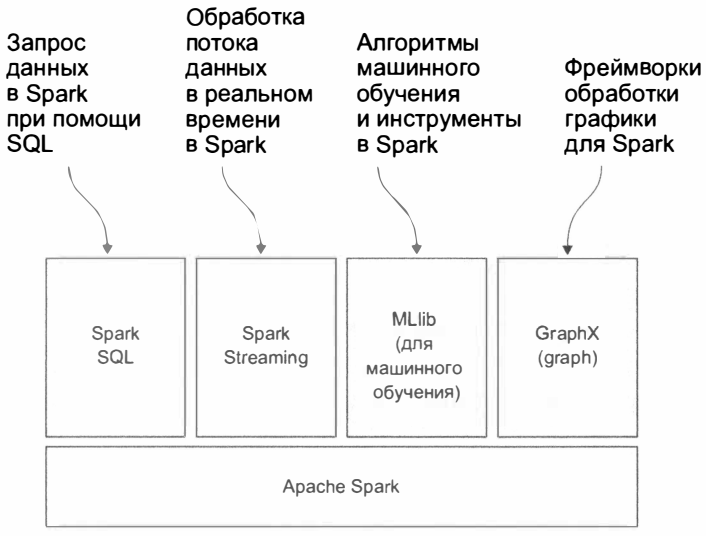
Слой хранения Hadoop называется HDFS (Hadoop Distributed File System — распределенная файловая система Hadoop). Наборы данных распределяются по разным машинам, что дает возможность обрабатывать их в параллельном режиме. Кроме того, каждая порция данных копируется, что снижает вероятность ее потери при программном или аппаратном сбое.

За распределение вычислений по узлам кластера в Hadoop отвечает простой алгоритм *MapReduce*. На шаге «map» этот фреймворк распределяет данные из HDFS между рабочими машинами, как правило, стараясь поддерживать количество строк одинаковым. Это похоже на описанные в предыдущих главах процессы проектирования признаков, в которых в каждую строку данных добавлялись новые столбцы. На шаге «reduce» происходит свертка уже обработанных данных. К этому способу функционирования можно преобразовать многие алгоритмы обработки. После этого такие системы, как Hadoop, берут на себя задачу распределения вычислений между машинами кластера.

В принципе, слои хранения и вычислений в интеграции не нуждаются. Многие организации пользуются системами хранения от провайдеров облачных технологий, например службой S3 в облачной инфраструктуре AWS (Amazon Web Services), совмещенной с фреймворком MapReduce для вычислений. Это выгодно, так как управление большими объемами данных берет на себя AWS, но при этом теряется одна из основ тесной интеграции между HDFS и MapReduce — *локальность данных*. Она увеличивает эффективность системы, так как вычисления производятся на подмножестве данных близко к месту их хранения.

Сообществом Hadoop была разработана библиотека *Mahout*, в которой реализованы многие популярные ML-алгоритмы, работающие с файловой системой HDFS и фреймворком MapReduce. Поэтому если вы пользуетесь Hadoop при решении задач, связанных с машинным обучением, имеет смысл обратить внимание на библиотеку Mahout. Эта библиотека постепенно смещается от упрощенной концепции MapReduce в сторону более сложных подходов к распределенным вычислениям на основе фреймворка *Apache Spark*. Эта более поздняя и широко популярная разработка использует специализированные примитивы для рекуррентной обработки в оперативной памяти, благодаря чему возникает значительный выигрыш в скорости работы для некоторых классов задач. Библиотека MLlib включает в себя набор алгоритмов машинного обучения для Spark. Упрощенная структура этого фреймворка показана на илл. 9.3.

По естественным причинам масштабируемые ML-алгоритмы зачастую линейны. Поэтому в обе библиотеки, Mahout и MLlib, входят по большей части линейные алгоритмы или аппроксимации нелинейных алгоритмов. В следующем разделе мы рассмотрим различные подходы к их масштабированию.

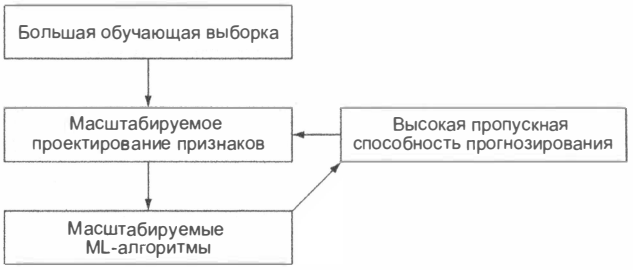


Илл. 9.3. Экосистема Apache Spark для распределенных вычислений на базе ядра Spark. Модуль Spark SQL позволяет работать с таблицами с помощью библиотеки pandas для языка Python или таблиц данных для языка R. Библиотека Spark Streaming обеспечивает обработку данных в реальном времени, что невозможно в Hadoop и классическом Spark с их пакетной обработкой. Библиотека MLlib включает в себя разнообразные ML-алгоритмы, оптимизированные под движок Spark, а библиотека GraphX обеспечивает эффективное вычисление больших графов, таких как граф профилей в социальной сети

9.2. Масштабирование конвейера ML-моделирования

Итак, пришло время заняться практикой и посмотреть, как именно масштабируется конвейер машинного обучения для перехода к большим объемам данных. Предположим, что вы уже выбрали систему обработки больших данных для масштабирования рабочего процесса. На илл. 9.4 показана обновленная версия знакомой нам ML-диаграммы.

В разделе 9.1.3 мы сделали обзор систем, умеющих управлять данными практически любого размера и обрабатывать их. Поскольку все они работают с отдельными экземплярами, обрабатывая их по очереди, описанный в этой книге процесс проектирования признаков в любой из этих систем осуществляется простыми map-вызовами. Теперь посмотрим, как масштабируются в случае больших данных некоторые популярные линейные и нелинейные ML-алгоритмы.



Илл. 9.4. Посвященная моделированию часть диаграммы рабочего процесса ML с масштабируемыми компонентами

9.2.1. Масштабирование обучающих алгоритмов

В начале главы вы видели, что основная проблема масштабирования, возникающая на стадии обучения, связана с возможностью размещения в оперативной памяти очень больших обучающих выборок. Ее можно обойти, например, поискав реализации ML-алгоритмов, которые или (a) требуют меньше места в памяти, чем исходная версия этого же алгоритма, или (b) могут работать в распределенных системах, где каждый узел имеет дело только с подмножеством целого набора данных.

Существуют бесчисленные реализации наиболее распространенных обучающих ML-алгоритмов. Присутствующие во множестве библиотек от scikit-learn до mlpack, эти реализации постоянно увеличивают эффективность использования памяти (одновременно увеличивая размер набора данных, который можно обработать на одном компьютере с фиксированным количеством оперативной памяти). Но пока рост объемов данных опережает успехи программного и аппаратного обеспечения в сфере ML. И в некоторых случаях единственным вариантом становится горизонтальное масштабирование процесса машинного обучения.

Самый распространенный распределенный обучающий алгоритм — это линейная (и логистическая) регрессия. Этот подход используется в пакете Vowpal Wabbit (VW), который служит базой для реализации линейных обучающих алгоритмов на множестве компьютеров. Первым делом подмножества обучающих данных (строки из полного набора) распределяются по разным машинам кластера. Затем на каждой машине циклически выполняется оптимизация предоставленной порции данных, после чего результат отправляется на центральный узел. Там информация объединяется, формируя общее решение. После нескольких итераций

мы получаем модель, которая гарантированно близка к оптимальной (как если бы одна модель была обучена на всех данных одновременно). Следовательно, распределенный способ позволяет обучать линейные модели на терабайтах данных!

Но как мы уже неоднократно упоминали, линейные алгоритмы далеко не всегда адекватно моделируют особенности данных и дают точные предсказания. В таких случаях помогает переход к нелинейным моделям. Но обычно для них требуется больше вычислительных ресурсов и далеко не всегда возможно горизонтальное масштабирование. Это логично, так как нелинейные модели учитывают сложные взаимосвязи между признаками, а значит, в каждом узле нужен набор данных большего размера.

Во многих случаях целесообразнее обновить аппаратное обеспечение или поискать более эффективные реализации выбранного вами алгоритма. Но бывают и ситуации, когда без масштабирования нелинейных моделей не обойтись, и ниже мы поговорим о том, как это делается.

Полиномиальные признаки

Для моделирования нелинейного взаимодействия между признаками часто применяется такой прием, как генерация новых признаков из комбинации существующих и обучение на них линейной модели. Генерация осуществляется путем перемножения признаков в различных сочетаниях: например, *признак 1 умножить на признак 2, признак 2 в квадрате* или *признак 1 умножить на признак 2 умножить на признак 5*. Предположим, наш набор данных состоит из двух признаков: f1 = 4 и f2 = 15. В дополнение к ним мы сгенерируем новые признаки: f1 × f2 = 60, f1 ^ 2 = 16 и f2 ^ 2 = 225. Так как обычно признаков в наборах данных куда больше, чем два, эта техника дает огромное количество новых вариантов. То есть у нас появляются нелинейные комбинации существующих признаков. Они называются *полиномиальными признаками* (polynomial features). Следующий листинг показывает процесс их получения с помощью библиотеки scikit-learn для языка Python. Результаты его выполнения показывают точность после добавления полиномиальных признаков в стандартную модель классификации ирисов Фишера:

```
Точность (линейной модели): 0.95 (+/- 0.12)
Точность (нелинейной модели): 0.98 (+/- 0.09)
```

Листинг 9.1. Превращение линейной модели в нелинейную с помощью полиномиальных признаков

```
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.cross_validation import cross_val_score

iris = datasets.load_iris()

linear_classifier = LogisticRegression()
linear_scores = cross_val_score(linear_classifier, \
    iris.data, iris.target, cv=10)
print " Точность (линейной модели):\t%0.2f (+/- %0.2f)" % \
    (linear_scores.mean(), linear_scores.std() * 2)

pol = PolynomialFeatures(degree=2)
nonlinear_data = pol.fit_transform(iris.data)

nonlinear_classifier = LogisticRegression()
nonlinear_scores = cross_val_score(nonlinear_classifier, \
    nonlinear_data, iris.target, cv=10)
print " Точность (нелинейной модели):\t%0.2f (+/- %0.2f)" % \
    (nonlinear_scores.mean(), nonlinear_scores.std() * 2)
```

Загружает образцы данных. Каждый экземпляр описывает изображение цветка ириса, вид которых модель должна научиться определять

Построение и вывод результатов скользящей проверки линейной модели

Добавляет в набор данных полиномиальные признаки второй степени

Строит линейную модель на нелинейных данных и выводит ее результаты после метода скользящего контроля

Еще одним инструментом для машинного обучения, в который встроено умение извлекать полиномиальные признаки, является библиотека Vowpal Wabbit. Она позволяет моделирование с большими наборами данных на одном компьютере, так как все вычисления выполняются в цикле и *во внешней памяти* (out of core), то есть в памяти хранятся только данные, необходимые на выполняемой итерации. Для работы с неструктурированными и разреженными данными с возможностью масштабирования в VW применяется стохастический градиентный спуск и хеширование признаков. Нелинейные модели библиотека VW создает, добавляя флаги `-q` и `-cubic` для генерации признаков, возведенных во вторую и третью степень и соответствующих полиномиальным признакам, в которых все пары или тройки признаков перемножены друг с другом.

Приближенное представление алгоритмов

Выше мы показали, что полиномиальные признаки позволяют значительно увеличить точность модели, но при этом количество признаков тоже

резко возрастает. Это нецелесообразно делать в случаях, когда входных признаков с самого начала много. Поэтому мы рассмотрим нелинейные алгоритмы, для которых существуют аппроксимации, позволяющие выполнить масштабирование.

Нелинейный алгоритм «случайный лес», который мы уже неоднократно упоминали, крайне популярен. Построенная на его основе модель состоит из множества деревьев решений, и на первый взгляд кажется, что для ее горизонтального масштабирования достаточно построить подмножество деревьев на каждом узле. Но когда данные в узлах значительно отличаются друг от друга, страдает точность модели. Для компенсации можно построить больше деревьев или более грамотно выполнить разбиение данных.

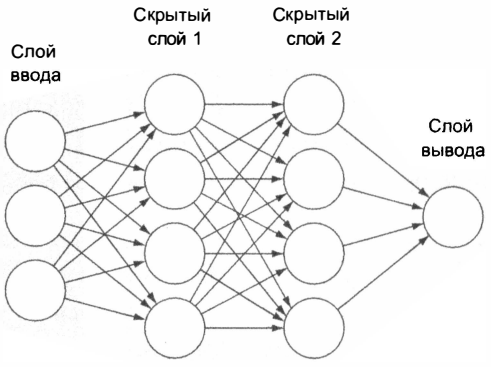
Для масштабирования «случайных лесов» и других алгоритмов также применяется *аппроксимация с помощью гистограммы* — каждый столбец в наборе данных заменяется гистограммой, что, как правило, приводит к значительному уменьшению количества значений. Но если столбцов в гистограмме мало, теряются многие нюансы и страдает производительность модели.

Другой алгоритм, естественным образом допускающий приближенное представление, — метод k-ближайших соседей; существуют специальные структуры деревьев, увеличивающие масштабируемость модели. У метода опорных векторов есть разные варианты аппроксимаций, например Budgeted Stochastic Gradient Descent (BSGD) и Adaptive Multi-hyperplane Machines (AMM).

Глубокие нейронные сети

Недавняя революция в области исследования нейронных сетей породила новый вариант глубокого обучения, дающий модели высокой нелинейности, масштабирующиеся под большие объемы данных. На заре машинного обучения нейронные сети тщательно изучались и широко применялись как в науке, так и в промышленности. Позднее, с появлением более простых для интерпретации алгоритмов, нейронные сети стали использоваться намного реже. Но сейчас, после нескольких важных эволюционных стадий и проникновения в сферу глубокого обучения, нейронные сети снова вошли в обиход и дают впечатляющие результаты при работе с большими и неоднородными данными.

Термин *глубокое обучение* (deep learning) относится к семейству алгоритмов, расширяющих традиционные нейронные сети. Как правило, такие модели состоят из множества скрытых слоев в нейронной сети или из комбинации множества однослойных сетей. Иллюстрация 9.5 демонстрирует вид нейронной сети.



Илл. 9.5. Нейронная сеть из двух скрытых слоев. Имитирующие человеческий мозг нейроны (окружности в каждом слое) соединены с весами, которые определяются в процессе обучения модели. Выходная переменная предсказывается путем прохода входных переменных через взвешенные соединения. В глубоком обучении концепция классической нейронной сети расширяется за счет появления дополнительных скрытых слоев различных форм и степеней связанности

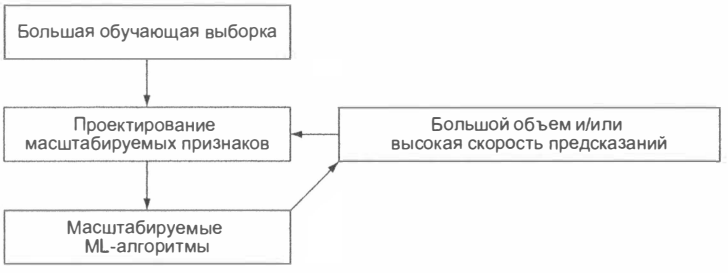
Недостатком глубоких нейронных сетей является долгое время построения и обучения моделей даже на аппаратной базе с графическим процессором. На практике аналогичную производительность можно получить, воспользовавшись, например, алгоритмом «случайный лес», требующим меньшего времени и вычислительных ресурсов. Но, как всегда, все зависит от особенностей поставленной задачи и набора данных. Кроме того, понять, что происходит внутри моделей на базе нейронной сети, крайне сложно. Иногда их называют *моделями черного ящика* (black-box models), ведь приходится доверять результатам статистического анализа, не видя, что происходит внутри. Но опять же, все зависит от варианта применения. Например, при работе с изображениями нейроны берут на себя интуитивное представление различных визуальных структур, которые ведут к конкретным предсказаниям.

Многие методы глубокого обучения показали хорошие результаты при масштабировании на большие объемы данных. Иногда масштабирова-

ние выполняется за счет современных видеокарт (графических процессоров), отвечающих за выполнение определенных вычислений. Для реализации глубокого обучения на языке Python с поддержкой графических процессоров рекомендуем библиотеку *Theano* (*http://deeplearning.net/software/theano/*) или базирующуюся на Theano библиотеку *Keras* (*http://keras.io/*).

9.3. Масштабирование предсказаний

Масштабирование в области машинного обучения не сводится к увеличению размера наборов данных. Представьте себе почтовый сервис, число пользователей которого внезапно достигло нескольких миллионов. Для него была построена модель распознавания спама, умеющая работать с большими наборами данных, но теперь она должна делать сотни миллионов предсказаний в день, то есть более 10 тысяч предсказаний в секунду! Иллюстрация 9.6 показывает общую схему происходящего. В этом разделе мы обсудим способы масштабирования объема предсказаний и скорости их генерации для случаев, когда прогнозы требуются в реальном времени.

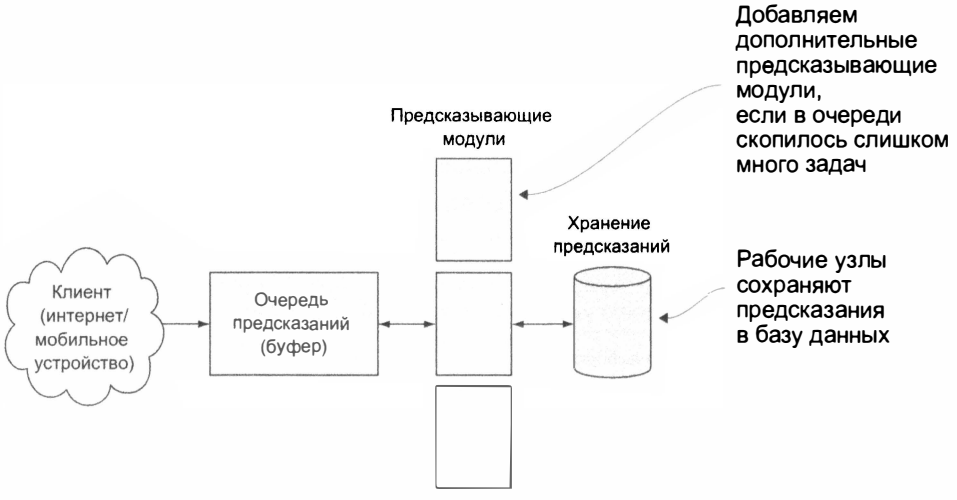


Илл. 9.6. Масштабирование этапа прогнозирования в рабочем процессе ML для увеличения объема и скорости предсказаний

Первым делом рассмотрим архитектуру инфраструктур, подходящих для масштабирования объема предсказаний, чтобы дать клиенту электронной почты возможность обрабатывать большую базу пользователей. Затем поговорим о том, как выполнить масштабирование скорости предсказаний и гарантировать их появление в заданном временном интервале. Это важно, когда ML-модели используются для обратной реакции в режиме реального времени, например при ответах людям в Сети.

9.3.1. Масштабирование объема предсказаний

Для обработки большего количества предсказаний можно воспользоваться существующими в вычислительной архитектуре шаблонами масштабирования рабочих моделей для обработки произвольного количества запросов. Традиционно принято формировать очередь задач, из которой рабочие узлы извлекают задачи, загружают модель (если это нужно), генерируют предсказание и отправляют результаты назад подходящим для конкретного приложения способом. Иллюстрация 9.7 показывает, как может выглядеть архитектура для масштабирования объема предсказаний.



Илл. 9.7. Вариант инфраструктуры для масштабируемой службы предсказаний. Запросы посылаются от заказчика в очередь, которая делегирует их выполнение какому-то из рабочих узлов. Этот узел сохраняет предсказание в базе данных и после завершения работы возвращает клиенту. При переполнении очереди, когда запросов поступает больше, чем узлы могут обработать, добавляются новые узлы

Этот подход, разумеется, требует загрузки модели на всех рабочих узлах и достаточного количества узлов (или решения с автоматическим масштабированием) для обработки всех поступающих запросов. Зная среднее время генерации предсказания одним узлом (`prediction_time`) и скорость поступления запросов (`request_velocity`), можно легко вычислить необходимое количество узлов (`n_workers`):

```
n_workers = request_velocity * prediction_time
```

Например, если за одну секунду поступает 10 запросов на предсказания, а рабочий модуль генерирует одно предсказание примерно за 2 секунды, то потребуется по меньшей мере 20 машин. Оптимальным решением в такой ситуации будет возможность регулировать количество рабочих узлов в зависимости от очереди ожидания в каждый момент времени.

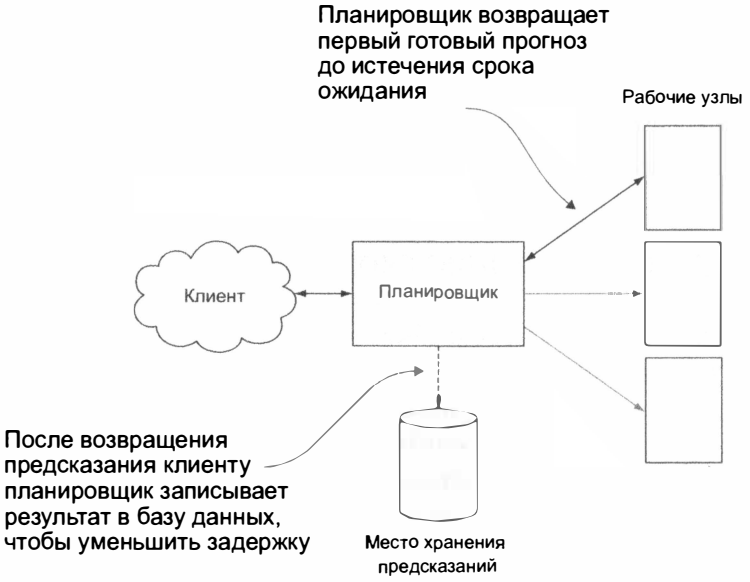
9.3.2. Масштабирование скорости предсказаний

В некоторых случаях предсказание должно возвращаться через определенное время после сделанного клиентом запроса. Скорость реакции важна, например, когда предсказания генерируются в ответ на действия пользователя. Если обратная связь ожидается в режиме реального времени, то несколько секунд ожидания негативно скажутся на впечатлении от работы с системой. Представьте себе поиск в Google, занимающий 20 секунд, — скорее всего, пользователи не станут дожидаться его результатов. А в прогнозах, связанных с финансовыми транзакциями, получение прибыли или потеря денег зачастую зависят от долей секунды.

Ускорение генерации прогнозов достигается разными способами, например обновлением аппаратного обеспечения или переходом к более эффективным алгоритмам или их реализациям. Можно оптимизировать сеть, физически расположив клиента как можно ближе к серверам. Кроме того, не следует запускать другие службы, которые могут увеличить задержку, такие как запись предсказаний в базу данных или ожидание, пока данные запишутся на диск и скопируются по всему кластеру. В следующем примере предполагается, что все эти соображения уже учтены. Рассмотрим два варианта архитектуры для генерации предсказаний в реальном времени.

Первый вариант отличается от представленной в предыдущем разделе архитектуры для масштабирования объема предсказаний большим количеством рабочих станций. Основная идея состоит в отправке запроса на предсказание одновременно нескольким узлам, и первый, кто завершит генерацию, возвращает клиенту результат. Схематично это показано на илл. 9.8.

Другой подход к реализации предсказаний в реальном времени состоит в распределении вычислений. В составе ансамблевых методов есть класс алгоритмов, позволяющих реализовать такой подход. В качестве примера мы снова рассмотрим «случайный лес».

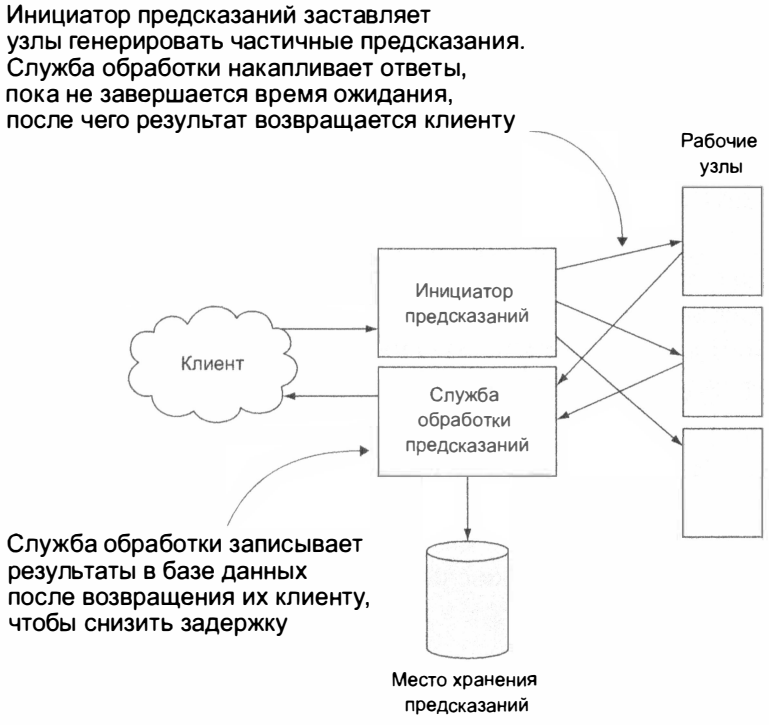


Илл. 9.8. Вариант архитектуры для конвейера предсказаний с низкой задержкой. Планировщик предсказаний отправляет каждое задание на все рабочие узлы, надеясь, что хотя бы один из них вернет результат за требуемое время. Первый же готовый результат планировщик возвращает клиенту и записывает его в журнал или в базу данных для дальнейшего изучения и анализа (в фоновом режиме, возможно, работая над следующим прогнозом)

Модели на базе алгоритма «случайный лес» состоят из ансамбля деревьев решений. В каждом дереве алгоритм делает прогноз и (в случае классификации) считает голоса всех деревьев для определения окончательной вероятности. Скажем, если из 10 деревьев 6 указывают на конкретный экземпляр предсказания, лес возвращает в качестве ответа 6/10, или 60%. Чем выше опрошенное число деревьев, тем точнее и достовернее результаты. Поэтому системы предсказаний в реальном времени часто жертвуют точностью в угоду скорости. Если каждый рабочий узел отвечает за дерево или за список деревьев рассматриваемого леса, его просят сгенерировать прогноз. Как только он заканчивает работу со своими деревьями, результат возвращается в службу, которая собирает прогнозы всех узлов и генерирует финальное предсказание. Обнаружив, что время заканчивается, эта служба возвращает результат, не дожидаясь полного завершения вычислений. Например, если за отведенное время ответ вернули всего 20 из 1000 деревьев, пользователь получит менее точный

прогноз, чем в случае, когда времени хватает на завершение вычислений всеми деревьями.

Схематично эта архитектура представлена на илл. 9.9.



Илл. 9.9. Рекомендуемая архитектура конвейера предсказаний, гарантированно возвращающего результат за определенное время, жертвующая точностью и достоверностью прогноза, если вычисления не успевают завершиться в указанный срок. Инициатор отправляет запросы рабочим узлам, в то время как служба обработки хранит частичные предсказания и готова вернуть их клиенту до истечения времени ожидания

Поддержка описанных выше масштабируемых систем реального времени реализуется разными способами. Для этого подходит, к примеру, часть уже обсуждавшейся экосистемы Apache Spark — Spark Streaming. Здесь вы найдете инструменты и библиотеки, облегчающие построение конвейеров обработки, работающих с потоками данных и генерирующих результаты в реальном времени. Не забудьте, что любые предсказания,

как правило, проходят через тот же самый процесс проектирования признаков, что и обучающие данные на стадии построения модели.

Можно воспользоваться и проектами Apache Storm, Apache Kafka, AWS Kinesis и Turi. Каждый из них имеет свои достоинства и недостатки, зависящие от конкретного варианта применения, поэтому рекомендуем ознакомиться с ними самостоятельно и выбрать наиболее подходящий под ваши задачи инструмент.

9.4. Заключение

В этой главе мы исследовали способы масштабирования систем с машинным обучением для работы с большими данными, сводящиеся к преобразованиям данных или к построению горизонтально масштабируемой инфраструктуры из множества машин. Вот основные положения, которые следует запомнить:

- Иногда систему с машинным обучением нужно масштабировать. Вот основные причины:
  - обучающая выборка слишком велика для обработки на одной машине;
  - слишком долгое обучение модели;
  - слишком большой объем поступающих данных;
  - задержка генерации предсказаний должна быть очень низкой.
- Иногда можно избежать траты времени и ресурсов на построение масштабируемой инфраструктуры:
  - выбрав другой алгоритм, который работает с достаточной скоростью и способен без потери точности решить задачу на одной машине;
  - взяв подмножество данных;
  - прибегнув к вертикальному масштабированию (обновлению существующего аппаратного обеспечения);

- пожертвовав точностью или убрав какие-то ограничения, если это дешевле, чем масштабирование.

- Если горизонтальное масштабирование неизбежно, для настройки инфраструктуры обработки и управления данными можно прибегнуть к таким популярным системам, как:
  - экосистема Hadoop с фреймворком Mahout для машинного обучения;
  - экосистема Spark с библиотекой MLlib для машинного обучения;
  - фреймворк Turi (раньше он назывался GraphLab);
  - технологии обработки потоковых данных Spark Streaming, Apache Storm, Apache Kafka и AWS Kinesis.
- Масштабируя конвейер построения модели, учитывайте следующие моменты:
  - выбирайте масштабируемый алгоритм, например логистическую регрессию или линейный вариант метода опорных векторов;
  - масштабируйте другие (например, нелинейные) алгоритмы с помощью аппроксимаций данных и алгоритма;
  - масштабируемую версию любимого алгоритма можно получить с помощью инфраструктуры для распределенных вычислений.
- Мы можем масштабировать как объем, так и скорость предсказаний. Широко применяются следующие подходы:
  - построение инфраструктуры, которая позволяет менять количество рабочих узлов в зависимости от объема предсказаний;
  - отправка одного и того же запроса на все узлы и возвращение первого же результата для оптимизации скорости прогнозирования;
  - выбор алгоритма, позволяющего распараллеливать предсказания по разным машинам.

9.5. Терминология

Термин	Описание
Большие данные (big data)	Широкий термин, обычно используемый для обозначения задач управления и обработки данных, которые невозможно выполнить на одном компьютере
Горизонтальное/вертикальное масштабирование (horizontal/vertical scaling)	Горизонтальное масштабирование означает увеличение числа компьютеров для обработки большего количества данных. Вертикальное масштабирование сводится к обновлению аппаратного обеспечения существующих компьютеров
Hadoop, HDFS, MapReduce, Mahout	Экосистема Hadoop широко применяется для обработки большого количества данных. HDFS и MapReduce — система распределенного хранения и система параллельных вычислений соответственно. Mahout — связанный с машинным обучением компонент экосистемы Hadoop
Apache Spark, MLlib	Apache Spark — современный проект, пытающийся держать данные в памяти, чтобы достичь большей эффективности, чем в Hadoop. MLlib — поставляемая в составе Spark библиотека для машинного обучения
Локальность данных (data locality)	Выполнение вычислений в месте, где хранятся данные. Передача данных часто становится ограничивающим фактором в проектах с большими данными, поэтому, убрав ее, можно существенно снизить требования к ресурсам
Полиномиальные признаки (polynomial features)	Способ расширения линейных моделей, позволяющий добавить нелинейные взаимодействия между признаками без утраты масштабируемости линейных обучающих алгоритмов
Vowpal Wabbit	Инструмент для эффективного построения ML-моделей на больших наборах данных, избавляющий от необходимости использовать такие системы для работы с большими данными, как Hadoop
Во внешней памяти (out-of-core)	Тип вычислений, при котором в памяти хранятся только данные, участвующие в текущей итерации

Термин	Описание
Аппроксимация гистограммой (histogram approximations)	Аппроксимация обучающей выборки, при которой все столбцы преобразуются в гистограммы
Отбор признаков (feature selection)	Процесс уменьшения размера обучающей выборки путем отбора и сохранения подмножества признаков с наибольшей прогностической ценностью
Лассо	Линейный алгоритм, выбирающий подмножество признаков с наибольшей прогностической ценностью
Глубокие нейронные сети (deep neural nets)	Результат развития нейронных сетей, который масштабируется для больших наборов данных и дает хорошую точность. Зачастую требует больше вычислительных ресурсов, чем другие алгоритмы
Объем/скорость предсказаний (prediction volume/velocity)	Масштабирование объема предсказаний означает возможность обрабатывать больше данных. Масштабирование скорости означает возможность генерировать предсказания за указанный временной интервал
Точность в жертву скорости (accuracy vs. Speed)	При предсказаниях в реальном времени иногда приходится жертвовать точностью ради скорости генерации прогнозов
Spark Streaming, Apache Storm, Apache Kafka, AWS Kinesis	Новые технологии для построения систем, работающих в реальном времени

10 Пример с цифровой рекламой

В этой главе:

- ✓ визуализация и подготовка набора данных;
- ✓ построение модели, предсказывающей вероятность перехода пользователя по рекламному баннеру;
- ✓ сравнение производительности нескольких алгоритмов на фазах обучения и прогнозирования;
- ✓ масштабирование путем уменьшения размерности и параллельной обработки.

В главе 9 были рассмотрены техники масштабирования рабочего процесса машинного обучения. Теперь мы рассмотрим пример их практического применения, а именно попробуем оптимизировать проводимую в Интернете рекламную кампанию. Для начала коротко познакомим вас с непростым миром интернет-рекламы, а также расскажем о данных, на которых все базируется, и о том, какими способами рекламисты пытаются максимально *увеличить возврат инвестиций в рекламу* (ROAS — return on advertising spend). Затем мы применим некоторые техники из главы 9 и построим типичное приложение для работы с большими данными.

Мы используем несколько наборов данных. К сожалению, в открытом доступе не так уж много данных нужного нам типа. Основной набор, с которым мы будем работать, недоступен для скачивания, впрочем, он все равно слишком велик для персонального компьютера.

Единственный набор, который можно скачать и использовать в некоммерческих целях, — это задача с цифровой рекламой с конкурса Kaggle, который спонсировала специализирующаяся на оптимизации производительности рекламных кампаний фирма Criteo. Этот набор состоит из более чем 45 миллионов наблюдений за 39 признаками, из которых 13 — цифровые, а 26 — категориальные. К сожалению, в наборах, используемых на конкурсах по теории анализа и обработки данных, значение признаков зачастую не раскрывается. Переменные имеют имена от V1 до V40, причем V1 — это метка, а с V2 по V40 — признаки. При решении реальных задач у нас есть преимущество в виде информации о том, как именно был измерен каждый признак или что он собой представляет. Но, как показал конкурс, даже без этих сведений можно извлечь из данных прогностическую ценность и создать работающую модель.

Набор данных от Criteo доступен по адресу *https://s3-eu-west-1.amazonaws.com/criteo-labs/dac.tar.gz*.

10.1. Показ рекламы

> Я точно знаю, что трачу половину рекламного бюджета впустую, но не знаю, какую именно.
>
> *Джон Уонамейкер*

Во времена, о которых рассказывает телесериал «*Безумцы*», это высказывание было безусловной истиной. Но как только реклама стала цифровой, появилась возможность исследовать данные, собранные в процессе взаимодействия пользователей с рекламными баннерами, и понять, какие вещи работают, а какие нет.

Существуют различные способы представления рекламы в Интернете. *Макетная реклама* (display ads) появляется на визуализируемых браузерами веб-страницах, посещаемых при помощи обычного компьютера или ноутбука. Так как в мобильных браузерах используются другие правила идентификации пользователей и обработки файлов cookies, отображаемая в них реклама реализуется с помощью других технологий и генерирует совсем другие статистические данные. Встраиваемая в игры и мобильные приложения *естественная реклама* (native ads) и *демонстрируемые перед видео ролики* (pre-roll ads) реализуются с помощью собственной технологии передачи данных и анализируются с привязкой к собственным уникальным процессам. Мы рассмотрим только примеры, связанные с *традиционной* макетной рекламой.

Принятая в этой области терминология по большей части унаследована из печатной рекламы. Сайты, на которых можно заказать рекламу, называются *издательствами* (publications), занимаемая объявлением площадка определяется такими характеристиками, как размер и формат, и называется *рекламным блоком* (ad unit), а расположение на сайте и странице обозначается как *размещение* (placement). Каждая демонстрация объявления называется *показом* (impression). Объявления продаются партиями по 1000 показов, соответственно цена обозначается аббревиатурой CPM (cost per thousand — цена за тысячу контактов).

Когда браузер загружает страницу, например xyz.com, кажется, что все ее содержимое поставляет владелец xyz.com. На самом деле на странице выделены места для объявлений, которые заполняются различными рек-

ламодателями через сложную сеть посредников. Все предоставляющие рекламу серверы ведут журналы, в которые записывается информация о каждом показе: издатель объявления, интернет-адрес пользователя и данные из файлов cookies, которые могут содержать сведения о предыдущих доставках с сервера этого же рекламодателя. Рассмотрим более подробно данные, фиксируемые в процессе рекламной кампании.

10.2. Данные, связанные с цифровой рекламой

Веб-серверы фиксируют следующие данные о запросах пользователя:

- *Адрес клиента.* IP-адрес сделавшего запрос компьютера.
- *Запрос.* URL-адрес и параметры (например, *http://www.abc.com?x=1234&y=abc01*).
- *Статус.* Код ответа сервера; обычно 200 означает успешное выполнение запроса.
- *Ссылающийся домен.* Веб-страница, с которой пользователь перешел на данную страницу.
- *User agent.* Текстовая строка, идентифицирующая выдавший запрос браузер и операционную систему.
- Cookies. Маленький файл, сохраняемый браузером при посещении сайта. При повторном посещении этот файл посылается вместе с запросом.

Кроме того, многие современные рекламные объявления снабжаются маленькой программой на языке JavaScript, которая собирает следующие данные:

- *Viewability.* Демонстрировалось ли объявление и как долго.
- *User ID.* Cookies браузера оставляют уникальные идентификаторы, благодаря которым пользователя можно опознать при повторных визитах.
- *Viewable seconds.* Сколько секунд объявление было видимым.

Иллюстрация 10.1 демонстрирует пример данных, собранных во время рекламной кампании. Информация о видимости извлечена из строки запроса, а `user_id` — случайным образом сгенерированный идентификатор, связывающий пользователя с его предыдущими визитами.

	timestamp	click	viewed	v_secs	user_id	operating_system	pub_domain
0	2015-09-28 09:01:35	False	False	0	9b644f47729749cc80ac9a67df399cb0	Windows	D10037853.com
1	2015-09-21 00:25:42	False	True	3	f5b295de8cf1448c8fde3b4cb1650873	Windows	D10031681.com
2	2015-09-08 00:08:49	False	False	0	06c757b7637647fb96b2d911303d5ed5	Windows	D10013014.com
3	2015-09-15 09:37:24	False	False	0	0dfabf89-5da8-459d-a4f7-3dfea37497f5	Windows	D10013014.com
4	2015-09-25 06:23:47	False	False	0	4171bedc8a99412a980c8521eee86c83	Windows	D10013014.com

Илл. 10.1. Данные по показам. В качестве доменных имен были взяты случайным образом сгенерированные заместители реальных имен

10.3. Проектирование признаков и стратегия моделирования

Наша целевая переменная в этой таблице — *click*. Нужно предсказать вероятность того, что показ завершится щелчком, то есть *переходом по ссылке* (click-through). То есть если пользователь заходит на конкретный сайт, какова вероятность его перехода по рекламному объявлению. Возможны разные формулировки задачи. Можно попытаться предсказать вероятность того, что на объявление среагирует конкретный пользователь, а можно заняться прогнозированием доли переходов (CTR — click-through rate) для каждого издателя объявления.

Как это часто бывает, чтобы определить, как будет выглядеть модель и какие значения мы хотим получить, надо ответить на следующие вопросы: *«Для чего нужен прогноз?»*, *«Каким образом он будет использоваться?»*. Например, рекламодатель может блокировать определенных издателей, соответственно ему нужно идентифицировать рекламу, вероятность перехода по которой минимальна. Уже появились *аукционы рекламных объявлений в реальном времени*, позволяющие рекламодателю приобретать показы, основываясь на предоставляемых рекламной биржей признаках пользователя и объявления, но рекламодатель из

нашего примера еще не приобщился к новейшим технологическим достижениям.

Может возникнуть вопрос, почему не занести в черный список публикации с низким CTR, просто изучив данные за прошедшие периоды. Проблема в том, что если общий CTR кампании составляет примерно 0,1%, ожидаемое число переходов для публикации с небольшим количеством показов равно нулю. Отсутствие переходов не означает, что высокий CTR в данном случае невозможен. В дальнейшем при рассмотрении публикаций небольшого объема в совокупности зачастую наблюдаются показатели CTR выше среднего (поэтому простая блокировка публикаций небольшого объема — плохая стратегия). Нам нужна модель, предсказывающая производительность публикаций вне зависимости от наличия данных за прошедший период.

На первый взгляд может показаться, что задача решается достаточно просто. Мы можем подсчитать показы, переходы и просмотры для пользователей, издателей и операционных систем. Возможно, какой-то эффект оказывает время суток или день недели. Но немного подумав, мы обнаружим, что посещаемые пользователем домены — это признаки, которые описывают пользователя, а сами пользователи, в свою очередь, — признаки домена. Внезапно у нас появилось огромное количество данных и реальная возможность столкнуться с *проклятием размерности* — это выражение используется для описания трудностей работы в *многомерном пространстве*. Изучив данные, вы поймете, что их изобилие является если не проклятием, то, скажем так, палкой о двух концах.

Логическая система, которой мы будем пользоваться, напоминает систему, лежащую в основе *рекомендательных сервисов*, предлагающих фильмы в приложении Netflix, товары на сайте Amazon и рестораны на сайте Yelp. Идея характеризовать пользователей через группу предметов, а предметы через группы пользователей лежит в основе *совместной фильтрации*, при которой пользователи объединяются на базе предпочитаемых ими предметов, а предметы — на базе пользователей, которые их предпочитают. Разумеется, рекомендательные сервисы хотят показать каждому пользователю то, что он, скорее всего, приобретет.

Мы имеем дело с разновидностью этой задачи; вместо множества вещей одно и то же объявление демонстрируется в различных контекстах — публикациях.

Движущий принцип в данном случае состоит в том, что самую высокую вероятность пользовательского отклика (переходов) получат публикации, сходные с теми, которые уже неоднократно провоцировали отклик. А так как сходство основывается на общих пользователях, выбранные публикации привлекут людей, у которых такие же предпочтения, как и у ранее реагировавших на рекламу.

10.4. Размер и форма данных

Начнем с набора из 9 миллионов наблюдений. Он достаточно мал, чтобы поместиться в памяти, и мы сможем быстро рассчитать мощность множества и распределения.

Листинг 10.1. Первое знакомство с данными

```
%matplotlib inline
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_pickle('combined.pickle') ← Загружает данные из сжатого архива

nImps = len(df)
nPubs = len(df.pub_domain.unique())
nUsers = len(df.user_id.unique())

print('nImps={}\nnPubs={}\nnUsers={}'.format(nImps, nPubs, nUsers))

nImps=9098807  ← Количество показов
nPubs=41576    ← Количество издательских доменов
nUsers=3696476 ← Количество разных пользователей

(nPubs * nUsers) / 1000000

153684 ← 153,684 миллиарда ячеек — достаточно большая матрица
```

Размер матрицы пользователь/элемент, поделенный на миллион для читабельности

К счастью, большинство пользователей никогда не посетят большую часть доменов списка, так что матрица пользователь/элемент заполнена неплотно. У нас есть инструменты обработки больших разреженных матриц. Разумеется, никто не заставляет нас представлять пользователей и домены именно в виде строк и столбцов гигантской матрицы, но выяснилось, что некоторые полезные алгоритмы особенно хорошо справляются с задачей, если могут оперировать в памяти такой матрицей.

Но есть еще один момент: 9 миллионов наблюдений, с которыми работает листинг 10.1, — это примерно 0,1% имеющихся данных. Всего нам нужно обработать почти 10 миллиардов показов, и это данные, собранные всего за неделю. Данные о 9 миллионах показов, загруженные в виртуальную машину с 32 Гбайт оперативной памяти в Amazon Web Services (AWS), заняли примерно 53% памяти, и этот показатель явно будет расти.

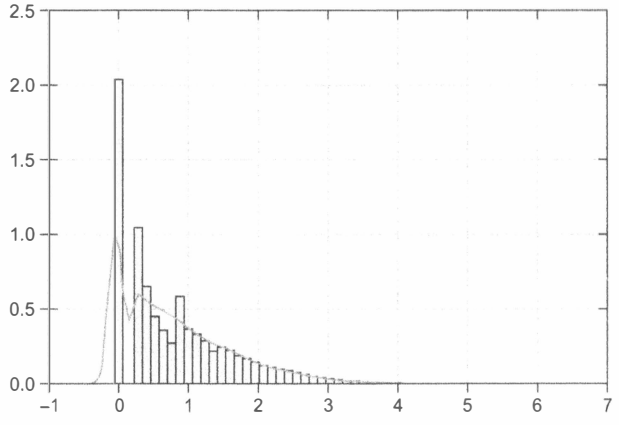
Теперь посмотрим, как данные распределяются по категориальным переменным. В листинге 10.1 мы уже начали этот процесс, рассчитав количество элементов в множествах `pub_domain` и `user_id`.

Листинг 10.2. Распределения

```
import seaborn as sns ◄—— Seaborn — библиотека для статистических визуализаций

nClicks = df.click.value_counts()[True]
print('nClicks={} ({}%)'
.format(nClicks, round(float(nClicks) * 100 / nImps, 2)))
nClicks=10845 (0.12%)

nViews = df.viewed.value_counts()[True]
print('nViews={} ({}%)'.format(nViews,
round(float(nViews) * 100 / nImps, 2)))

nViews=3649597 (40.11%)

df.groupby('pub_domain').size()  ◄— Группирует по доменам
                                    и смотрит на количество
                                    показов на один домен
pub_domain
D10000000.com 321
D10000001.com 117
D10000002.com 124
```

```
D10000003.com 38
D10000004.com 8170
...

f = df.groupby('pub_domain').size()
f.describe()

count 41576.000000
mean 218.847580
std 6908.203538
min 1.000000
25% 2.000000
50% 5.000000
75% 19.000000
max 1060001.000000

sns.distplot(np.log10(f));
```

На илл. 10.2 мы видим, что количество показов велико только на небольшом числе доменов. Чтобы показать графическое распределение, мы взяли не повторения, а их десятичный логарифм (благодаря основанию 10, ось x можно представить как 10^0, 10^1, 10^2...).



Илл. 10.2. Гистограмма данных по показам демонстрирует сильный сдвиг в распределении числа показов по доменам издателей

Возможно, самая важная информация, которую мы тут видим, — это относительная редкость переходов. Их всего 0,12%, или 0,0012. Это

достаточно хорошая общая доля переходов. Но нам требуется больше данных, чтобы получить достаточно примеров для построения модели. И это вполне стандартная ситуация. Часто приходится предсказывать относительно редкие явления. Возможность обрабатывать огромные наборы данных с помощью современных технологий позволила применить машинное обучение к новому классу задач.

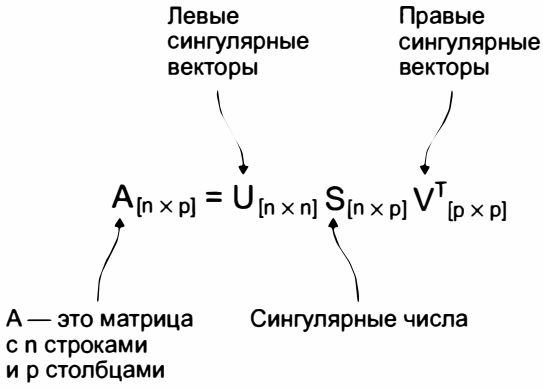
Сходный сдвиг имеет и распределение показов для переменной `user_id`. На среднего пользователя приходится 2,46 показа, при медиане 1, то есть среднее увеличивается за счет немногих, активно просматривающих рекламу.

10.5. Сингулярное разложение

В главах 3 и 7 метод главных компонент, или PCA, упоминался как техника машинного обучения без учителя, часто применяемая для уменьшения размерности и извлечения признаков. Если считать каждого пользователя признаком публикации, с которой он взаимодействует, то мы получим примерно 3,6 миллиона признаков на публикацию, 150 миллиардов значений для изучаемых нами данных. Очевидно, что количество признаков следует уменьшить, и, к счастью, это легко достижимо.

Метод главных компонент включает в себя несколько алгоритмов, и в том числе *сингулярное разложение*, или SVD (singular value decomposition). Существуют разные способы его математического объяснения и интерпретации, но мы не будем слишком углубляться в тонкости линейной алгебры. К счастью, как и для рассмотренного в главе 7 латентно-семантического анализа, для SVD существует отличная реализация в библиотеке scikit-learn. Впрочем, матричной алгебры мы тоже слегка коснемся. Если вы когда-либо перемножали матрицы, то знаете, насколько важен их размер. Если $A_{[n \times p]}$ означает матрицу из n строк и p столбцов, ее можно умножить на другую матрицу, состоящую из p строк и q столбцов (например, $B_{[p \times q]}$). Результатом такой операции станет матрица из n строк и q столбцов (назовем ее $C_{[n \times q]}$). И оказывается, любую матрицу можно представить в виде трех компонент, которые называются левым и правым сингулярными векторами и сингулярными числами соответственно.

В нашем примере: если n — число пользователей, каждый из которых представлен строкой в матрице A, а p — число издателей, каждый из которых представлен столбцом, то:



Что интересно, сингулярные числа дают нам информацию о важности признаков, представленных левым и правым сингулярными векторами (это строки в U и V^T). В частности, сингулярные векторы показывают независимые расширения к соответствующим векторам признаков. Рассмотрим, что следует из взаимозависимых или *ковариантных* признаков. Для простоты возьмем всего два идентичных признака — A и B. После того как модель рассмотрела признак A, признак B уже не может дать ей новой информации. Поэтому для построения прогнозирующих моделей требуются независимые признаки, каждый из которых хоть в какой-то степени является прогностическим фактором для целевой переменной. Множество слабых прогностических факторов при условии, что построенные на них прогнозы отличаются от случайного угадывания, в комбинации дает хороший результат. Но такой *эффект ансамбля* возможен только на независимых признаках.

Выполним сингулярное разложение наших рекламных данных и посмотрим на полученные сингулярные числа.

Листинг 10.3. Сингулярное разложение рекламных данных

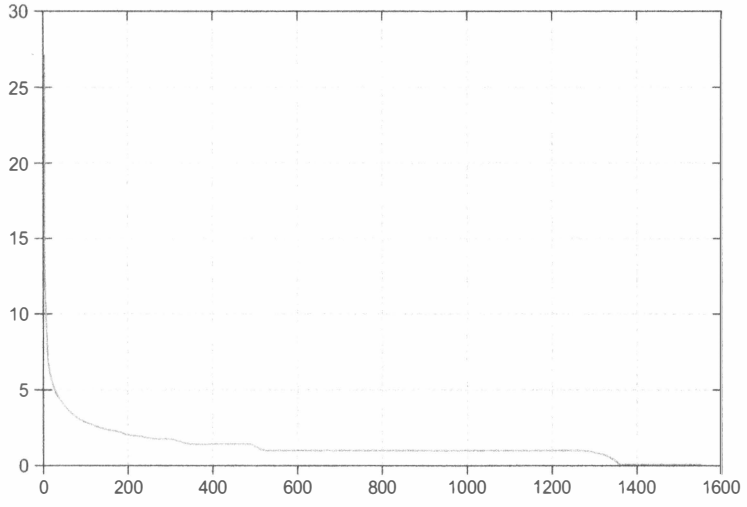
```
user_idx, pub_idx = {}, {}
for i in range(len(users)):
    user_idx[users[i]] = i
for i in range(len(pubs)):
    pub_idx[pubs[i]] = i
```

Замещает целочисленными индексами символические ключи для пользователя и домена

```
nTrainUsers = len(df.user_id.unique())
nTrainPubs = len(df.pub_domain.unique())
V = sp.lil_matrix((nTrainUsers, nTrainPubs))
def matput(imp):
if imp.viewed:
            V[user_idx[imp.user_id], pub_idx[imp.pub_domain]] = 1

df5[df5.click == True].apply(matput, axis=1)

# run svds (svd for sparse matrices)

u, s, vt = svds(V, k = 1550)

plt.plot(s[::-1])
```

Создает разреженную матрицу взаимодействий пользователь/домен

Мы добавили в листинг параметр k, обозначающий *максимальное количество сингулярных чисел*, чтобы ограничить расчеты 1550 самых больших сингулярных чисел. Иллюстрация 10.3 демонстрирует их абсолютные значения; мы видим примерно 1425 ненулевых значений и свыше 450 наиболее независимых признаков, все остальные в изрядной степени ковариантны. И это не удивительно. Хотя у нас более 3 миллионов пользователей, большинство из них взаимодействует с очень маленьким количеством доменов. Скажем, 136 тысяч пользователей были зафиксированы всего один раз (и это на *ebay.com*). Так что если вектор каждого пользователя считать признаком издателя, у *ebay.com* будет 136 000 идентичных признаков.



Илл. 10.3. Сингулярные числа для рекламных данных

Сингулярное разложение помогло нам уменьшить более чем 3 миллиона признаков до примерно 7 тысяч. Это соотношение 400:1. Теперь можно намного лучше представить, сколько ресурсов нам потребуется. Давайте посмотрим на способы оптимизации ресурсов, необходимых для обучения наших моделей.

10.6. Оценка и оптимизация ресурсов

Итак, мы вычислили такие характеристики наших данных, как число объектов в множествах и их распределения, а также выполнили проектирование признаков. В этом разделе мы оценим нашу задачу с точки зрения соотношения между вычислительной нагрузкой и имеющимися ресурсами.

Для оценки требований к ресурсам нужно сделать ряд измерений. До этого мы использовали виртуальную машину m4.2xlarge Amazon EC2. Кратко поясним, что это означает. Аббревиатура EC2 означает веб-сервис, предоставляющий вычислительные мощности в облаке. Он входит в инфраструктуру Amazon Web Services. Каждый экземпляр представляет собой виртуальный сервер с выделенным процессором, оперативной памятью и диском или твердотельным накопителем для хранения данных в сети. Обозначение *m4.2xlarge* соответствует серверу с восемью ядрами и 32 Гбайт памяти. Дисковое пространство выделяется отдельно. Наша виртуальная машина имеет 1 терабайт в хранилище EBS (elastic block storage). Создаваемые по этой технологии диски расположены на виртуальных накопителях и настроены таким образом, что нашему экземпляру выделен объем 1 Тбайт. Мы установили туда операционную систему Linux. При необходимости мы можем увеличить количество ядер, память или даже зарезервировать себе дополнительные экземпляры.

Теперь посмотрим на рабочую загрузку. Необработанные данные в файлах транзакций находятся в сетевом хранилище *Simple Storage Service* (Amazon S3), которое спроектировано под хранение больших объемов данных. Но доступ к ним осуществляется куда медленнее, чем в случае, когда данные хранятся на локальном диске вашего компьютера. Каждый файл содержит примерно миллион записей. Из хранилища S3 можно считывать приблизительно 30 тысяч записей в секунду. Если обрабатывать по одной записи за раз, на обработку 10 миллиардов записей по-

требуется примерно 92 часа. Скачивание из S3 можно ускорить на 75% путем параллельной загрузки (на одном экземпляре), что снизит время ожидания до 23 часов.

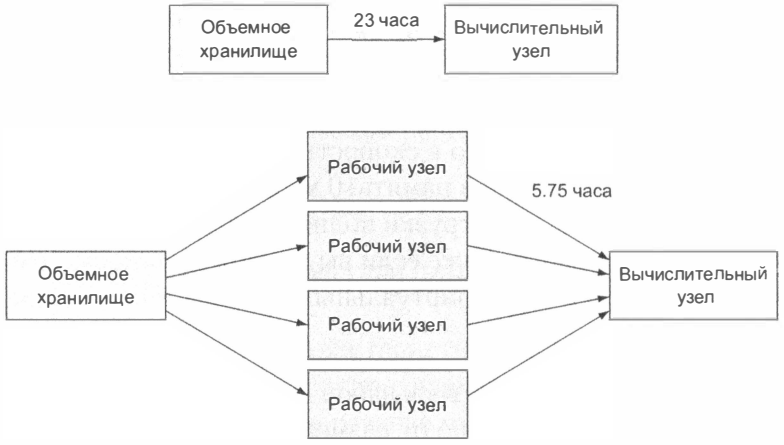
Но проблема состоит не только в скорости передачи данных. Выше мы упоминали, что загруженные в память10 миллионов записей занимают 53% из 32 Гбайт. То есть для загрузки полного набора данных нам потребуется 1,7 терабайт памяти. Даже если вы можете себе такое позволить, на сервисе Amazon просто нет виртуальных машин с таким количеством оперативной памяти.

К счастью, загружать в память весь набор данных не нужно. Более того, наши запросы зависят не только от размера данных, но и от их формы. Под формой мы подразумеваем кардинальность основных ключей. В наших 10 миллиардах записей всего около 10 миллионов пользователей и примерно 300 тысяч доменов, то есть матрица пользователь/домен содержит приблизительно 3 триллиона элементов. Но при заполнении разреженной матрицы значения оказываются только в 0,01% ячеек, так что наши 3 триллиона уменьшаются до 300 миллионов. Предположив, что на одно значение требуется одно 64-битное число с плавающей точкой, мы получим, что для матрицы пользователь/домен будет достаточно всего 2,5 из наших 32 Гбайт.

Снизить время обработки помогут параллельные процессы. Иллюстрация 10.4 демонстрирует применение рабочих станций (в нашем случае это дополнительные виртуальные машины EC2) для параллельного скачивания данных.

Рабочие узлы не только загружают данные из хранилища S3. Независимо друг от друга каждый из них строит разреженную матрицу пользователей и доменов. Затем вычислительный узел объединяет их в одну матрицу.

В главе 9 вы познакомились с такими технологиями обработки больших данных, как Hadoop, MapReduce и Apache Spark. Описанный в этом разделе процесс представляет собой сильно упрощенную версию того, что происходит в системе распределенных вычислений MapReduce. Большая задача разбивается на небольшие фрагменты, каждый из которых передается рабочему узлу. После того, как все узлы завершат расчеты, результаты объединяются и в таком виде возвращаются инициатору запроса. Фреймворк Hadoop оптимизирует этот процесс несколькими способами.



Илл. 10.4. Параллельная обработка ускоряет процесс начального получения данных

Во-первых, рабочие узлы не запрашивают данные по сети, а хранят свою часть данных у себя. А Hadoop раздает задачи таким образом, чтобы по возможности каждый узел работал с данными, которые уже есть на его локальном диске. Фреймворк Spark идет еще дальше, заставляя рабочие узлы загружать данные в память, чтобы для выполнения назначенных им задач не требовались операции ввода/вывода.

Задача, которую мы пытаемся решить, достаточно объемна, чтобы потребовалась параллельная обработка, но вряд ли имеет смысл тратить силы на установку одного из вышеперечисленных фреймворков. Рабочий процесс нужно запускать раз в день, и, увеличив количество виртуальных машин, мы сможем завершить его за час или даже быстрее. Но легко можно представить себе приложение, которое требует более частого запуска различных процессов, при которых хранение исходных данных в памяти в течение многих циклов обработки на много порядков уменьшит производительность.

10.7. Моделирование

Нам нужна модель, которая будет предсказывать CTR для каждого домена. Мы взяли в качестве признаков пользовательские взаимодействия и уменьшили размерность пространства признаков путем сингулярного

разложения. После этого возможны различные варианты действий. Первую модель мы построим на базе метода *k-ближайших соседей* (KNN — k-nearest neighbors). Несмотря на свою простоту, модель получится на удивление эффективной.

Попробуем также решить задачу регрессии на базе алгоритма «*случайный лес*». «Случайные леса» представляют собой обучение, в основе которого лежат деревья решений; выбирается множество случайных подмножеств данных и случайных подмножеств признаков, и для каждого из них строятся решающие деревья.

10.8. Метод k-ближайших соседей

Иллюстрация 10.5 демонстрирует упрощенные версии матрицы пользователь/элемент и матрицы расхождений. Обратите внимание, что диагональ последней состоит из одних нулей, так как пользовательский вектор каждого домена (столбец матрицы пользователь/элемент) идентичен сам себе, а значит, находится на нулевой дистанции от себя. Как видите, расстояние между векторами pub3, pub4 и pub7 вполне ожидаемо равно нулю, так как соответствующие столбцы матрицы пользователь/элемент идентичны. Также бросается в глаза, что расстояние от вектора pub1 до pub5 такое же, как расстояние от вектора pub5 до pub1. Иными словами, наши расхождения симметричны. Что интересно, некоторые алгоритмы рекомендательных сервисов не определяют расстояние симметрично. Элемент A может быть подобен элементу B, но при этом элемент B не подобен элементу A.

Подсчитаем сходство (на самом деле расхождение или расстояние) между всеми парами доменов, используя один из доступных критериев. Возьмем наиболее распространенный вариант — *евклидову метрику*.

После того как мы попарно рассчитаем расстояния, нужно будет определить предсказанный показатель CTR для каждого издателя. В методе KNN целевая переменная вычисляется путем усреднения значений целевых переменных k-ближайших соседей. При этом предполагается, что каждый пример наблюдения будет по большей части одинаковым с его ближайшими соседями. На этом этапе появляются важные вопросы. Во-первых, как выбрать значение параметра k? Сколько соседей мы

будем рассматривать? Кроме того, больший вес принято присваивать ближайшим соседям, обычно умножая рассчитанное среднее значение целевой переменной на весовой множитель $1/расстояние$ или $|1/расстояние|^2$.

Матрица пользователь/элемент

	pub1	pub2	pub3	pub4	pub5	pub6	pub7
user1	0	0	1	1	0	1	1
user2	1	1	0	0	1	1	0
user3	1	0	0	0	0	1	0
user4	0	0	0	0	0	0	0
user5	1	1	1	1	1	1	1

Матрица расхождений

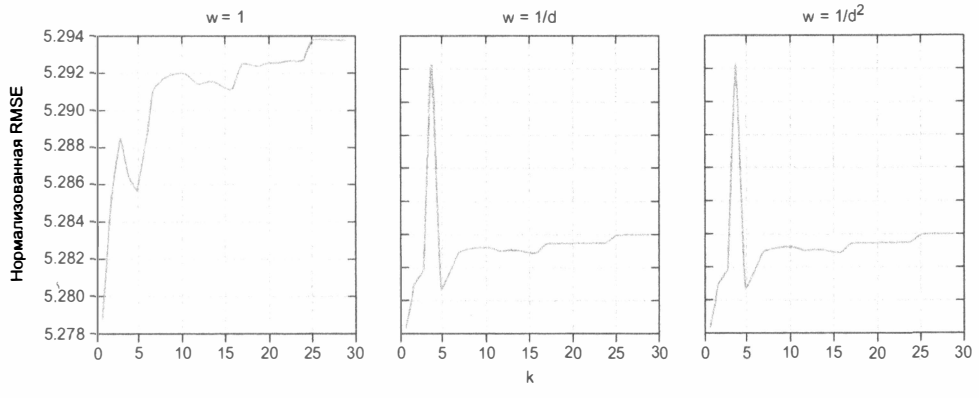
	pub1	pub2	pub3	pub4	pub5	pub6	pub7
pub1	0.0	1.0	1.7	1.7	1.0	1.0	1.7
pub2	1.0	0.0	1.4	1.4	0.0	1.4	1.4
pub3	1.7	1.4	0.0	0.0	1.4	1.4	0.0
pub4	1.7	1.4	0.0	0.0	1.4	1.4	0.0
pub5	1.0	0.0	1.4	1.4	0.0	1.4	1.4
pub6	1.0	1.4	1.4	1.4	1.4	0.0	1.4
pub7	1.7	1.4	0.0	0.0	1.4	1.4	0.0

Илл. 10.5. Матрица расхождений или расстояний показывает, до какой степени взаимодействия пользователей совпадают или отличаются. В нашем примере рассматривается бинарная матрица пользователь/элемент, показывающая, взаимодействовал ли конкретный пользователь с конкретным издателем

Листинг 10.4 показывает вычисление предсказанных значений для диапазона возможных k с помощью модуля `NearestNeighbors` из библиотеки scikit-learn. Мы пробуем три формулы для взвешенных величин, каждая для 20 значений k. Иллюстрация 10.6 показывает, что самые лучшие предсказания получаются при одном или двух ближайших соседях, а усреднение по большому диапазону не дает видимых улучшений. Скорее всего, это связано с разреженностью данных, в которых ближайшие соседи зачастую располагаются на большом расстоянии друг от друга. Обратите внимание, что зависимость от величины k тоже невелика. В любом случае нормализованная среднеквадратичная ошибка для тестового набора предсказаний находится в диапазоне 5%. Совсем неплохо!

Листинг 10.4. Предсказание методом KNN

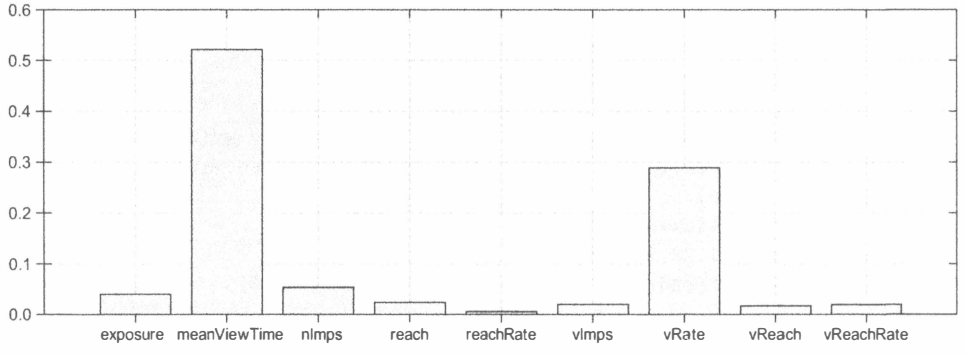
```
\ from sklearn.neighbors import NearestNeighbors

weightFunctions = {
    'f1': lambda x: [1 for i in range(len(x))],   <-- Равные веса
    'f2': lambda x: 1 / x,                        <-- 1/расстояние
    'f3': lambda x: 1 / x ** 2                    <-- 1/расстояние в квадрате
}

for idx, f in enumerate(weightFunctions):         <-- Для каждой из трех схем определения весов вычисляем предсказанные целевые значения для k = 1 до 20
    rmseL = []
    wf = weightFunctions[f]
    for nNeighbors in range(1,20, 1):
        neigh = NearestNeighbors(nNeighbors)      <-- Инициализация

        neigh.fit(VT)                             <-- Находит k-ближайших соседей; VT — это транспонированная матрица пользователь/элемент для обучающей выборки
        act = pd.Series()
        pred= pd.Series()

        for i in range(TT.shape[0]):              <-- TT — это транспонированная матрица пользователь/элемент для тестовой выборки
            d = neigh.kneighbors(tt[i,:], return_distance=True)
            W = pd.Series([v for v in d[0][0]])
            y = pd.Series(pubsums.iloc[d[1][0]].CTR)
            act.append(pd.Series(tsums.iloc[i].CTR))
            pred.append(pd.Series(np.average(y, weights = wf(W))))
    mse = act.sub(pred).pow(2).mean() / (pred.max() - pred.min())
    mseL.append(rmse)
    plt.subplot(130+idx+1)
    plt.plot(range(1,20,1), mseL)
    plt.tight_layout(pad=2.0)
```



Илл. 10.6. Среднеквадратичная ошибка для трех весовых функций и значений k от 1 до 30

10.9. «Случайные леса»

На обучающей фазе «случайных лесов» циклически берутся фрагменты данных и подвергаются процессу, который называется *бэггинг* (bootstrap aggregating). Для каждого фрагмента строится дерево решений на базе случайным образом выбранного подмножества признаков. Чтобы обеспечить возможность генерации прогнозов для новых данных, каждое дерево решений оценивается независимо от остальных, а результаты усредняются (в случае регрессии) или каждое дерево «голосует» (в случае классификации). Для многих приложений более производительны другие алгоритмы, например растущие деревья (boosted trees) или метод опорных векторов, но преимуществом «случайных лесов» является легкость их применения, интерпретации и понимания, а также возможность обучения множества деревьев в параллельном режиме. Еще раз воспользуемся библиотекой scikit-learn (илл. 10.7).



Илл. 10.7. Важность переменных в задаче регрессии на базе алгоритма «случайный лес»

Листинг 10.5. Регрессия на базе алгоритма «случайный лес»

```
from sklearn.ensemble import RandomForestRegressor
from sklearn import cross_validation

features = ['exposure', 'meanViewTime', 'nImps', 'reach', 'reachRate',
            'vImps', 'vRate', 'vReach', 'vReachRate']

X_train, X_test, y_train, y_test = cross_validation.train_test_split(
        df[features], df.CTR, test_size=0.40, random_state=0)
```

В качестве признаков просто совокупные значения по каждому издателю

Разбивает данные на тестовые и обучающие, признаки и целевые переменные, обучает модель на 60% данных и откладывает 40% для тестирования

```
reg = RandomForestRegressor(n_estimators=100, n_jobs=-1)
model = reg.fit(X_train, y_train)

scores = cross_validation.cross_val_score(model, X_train, y_train)
print(scores, scores.mean())

([ 0.62681533, 0.66944703, 0.63701492]), 0.64442575999999996)

model.score(X_test, y_test)

0.6135074515145226

plt.rcParams["figure.figsize"] = [12.0, 4.0]
plt.bar(range(len(features)), model.feature_importances_, align='center')
_ = plt.xticks(range(len(features)), features)
```

Запускает модель на тестовой выборке

Разбивает обучающую выборку для оценки модели методами скользящего контроля

Запускает модель регрессии на базе алгоритма «случайный лес» со 100 деревьями; параметр n_jobs сообщает алгоритму, что нужно использовать все доступные ядра

Оптимизированная модель для решения задачи регрессии на базе алгоритма «случайный лес» дает нам приемлемые предсказания параметра CTR, но они далеко не так хороши, как прогнозы на базе алгоритма KNN. В принципе, можно исследовать комбинации этих двух, а возможно, и других моделей. Методы, в которых применяются объединенные модели, называются *ансамблевыми* (ensemble methods). «Случайные леса» сами по себе являются ансамблевым методом, так как бэггинг позволяет генерировать множество моделей. Объединение разных моделей осуществляется путем *стекинга* (stacked generalization). Этот процесс заключается в том, что предсказания множества моделей превращаются в признаки, которые комбинируются путем обучения и получения предсказаний от еще одной модели, обычно на базе алгоритма логистической регрессии.

10.10. Другие практические моменты

Итак, мы рассмотрели ограничения, с которыми приходится сталкиваться при работе с большими данными: высокая размерность, недостаток вычислительных ресурсов и места для хранения, а также недостаточная

скорость передачи данных по сети. Описанный выше процесс можно применять к разным видам цифровой рекламы — мобильной, видео и встроенной в игры или приложения. Аукционы рекламных объявлений и персонализация на уровне отдельных пользователей требуют внимания к совсем другим аспектам. Каждый раз все зависит от данных, с которыми нужно работать, да и модели, идеально работающие в одной ситуации, могут совершенно не подойти к другой.

В нашем примере рассматривается огромный набор данных за прошедшие периоды. Но наш подход, напоминающий рекомендательные сервисы, сталкивается с так называемой проблемой *холодного старта* (cold-start problem). При появлении в системе нового пользователя или нового продукта исторические данные отсутствуют, и основы для построения ассоциации попросту нет. В нашей задаче небольшое количество новых пользователей не имеет значения, но когда кампания начинается с нуля, данных за прошедшие периоды нет по определению. А модели, построенные на базе сходных кампаний, могут оказаться неэффективными.

Наличие множества доступных инструментов и моделей дает большое преимущество при решении практических задач. Чем больше и сложнее среда, тем выше польза от хорошо систематизированного и встроенного в когерентный автоматизированный рабочий конвейер инструментария в виде наборов для генерации признаков, уменьшения размерности данных, обучения, прогнозирования и оценки моделей.

Мы рассмотрели показательный пример бизнеса, в котором внешние факторы могут уменьшить эффективность прогнозов построенных нами моделей. По мере развития технологий и способов ведения бизнеса меняются и подходы. Увеличение числа мобильных устройств коренным образом изменило общую картину. Рекламные аукционы в режиме реального времени заставили изменить уровень, на котором выполняется оптимизация. Новые виды мошенничества, блокираторы рекламы, новые браузеры и новые интернет-технологии меняют ход моделирования. В реальном мире модели строятся, тестируются, внедряются, измеряются, перестраиваются, повторно тестируются, заново внедряются и еще раз измеряются.

Цифровая реклама — это бизнес, в который вкладывают миллиарды долларов. Для брендов, которые ею пользуются, оптимизация, позволя-

ющая хоть немного снизить непроизводительные траты, уже означает значительный возврат инвестиций. Деньги экономит даже уменьшение числа бесполезных показов, если же в результате появится новый клиент, вы получите куда больше, чем просто снижение затрат на рекламу, и это оправдает все усилия, потраченные на преодоление множества трудностей в этом динамичном бизнесе.

10.11. Заключение

В этой главе мы рассмотрели задачу машинного обучения, немного выйдя за рамки выбора алгоритмов, обучения и тестирования моделей. Хотя эти вещи составляют основу такой дисциплины, как машинное обучение, их успешное выполнение зачастую зависит от практических аспектов и неизбежных компромиссов. Вот основные моменты, на которые стоит обратить внимание:

- Первым делом всегда нужно понять суть моделируемого бизнеса или деятельности, их цели и способы измерения этих целей. Важно подумать и о том, как можно воспользоваться полученными прогнозами, чтобы заранее понять, какие настройки и оптимизации можно сделать на базе предоставленных моделью аналитических оценок.

- Различные стратегии генерации признаков могут дать разные рабочие наборы данных. Поэтому полезно рассмотреть широкий диапазон возможностей. В первой модели мы значительно увеличили набор признаков, а затем уменьшили его с помощью сингулярного разложения. Во второй модели воспользовались простой группировкой. Лучший подход, как обычно, определяется особенностями конкретной задачи и данных.

- После изучения подмножеств данных появилась возможность оценить вычислительные ресурсы, которые потребуются для их анализа. В рассматриваемом примере трудности были связаны не с ML-алгоритмами, а со способами накопления исходных данных и их группировки в подходящей для моделирования форме. Это вполне стандартная ситуация, и при оценке необходимых ресурсов важно учитывать задачи, которые будут решаться перед запуском нашего конвейера и в процессе его работы.

- Часто лучше всего работает не одна модель, а целый ансамбль, предсказания которого группируются с помощью еще одной прогнозирующей модели. На практике приходится выбирать между самыми лучшими ансамблями и затратами на создание, управление и поддержку комплексного рабочего процесса.
- В реальности зачастую существует несколько вариантов постановки задачи. Вы видели это на примере задачи с рекламой, и такая ситуация стандартна в любой сложной области.
- Базовая динамика моделируемых явлений, как правила, непостоянна. Меняются бизнес, рынки, поведения и условия. Работая с реальными ML-моделями, нужно постоянно следить за их производительностью и периодически начинать все с самого начала.

10.12. Терминология

Термин	Определение
Рекомендательный сервис (recommender)	Класс ML-алгоритмов, используемый для предсказания привлекательности различных элементов для пользователя
Совместная фильтрация (collaborative filtering)	Рекомендательные алгоритмы, которые работают, характеризуя пользователей через предпочитаемые ими элементы, а элементы — через предпочтения общих пользователей
Ансамблевый метод (ensemble method)	Стратегия машинного обучения, в которой объединяются независимые предсказания различных моделей
Эффект ансамбля (ensemble effect)	Использование комбинации нескольких моделей для получения лучшей прогностической производительности, чем у каждой из них в отдельности
Метод k-ближайших соседей (k-nearest neighbors)	Алгоритм, основывающий предсказания на ближайших наблюдениях в обучающем пространстве
Евклидова метрика (Euclidean distance)	Один из множества способов измерения расстояний в пространстве признаков. В двумерном пространстве это знакомая вам формула расстояния между двумя точками

Термин	Определение
«Случайный лес» (random forest)	Ансамблевый метод, который при решении задач классификации или регрессии обучает множество решающих деревьев на подмножествах обучающих данных и признаков и генерирует предсказания на базе объединенных результатов моделирования
Бэггинг (bagging)	Процесс обучения на случайных подвыборках с повторениями из обучающей выборки, используемый «случайными лесами» и другими алгоритмами
Стекинг (stacking)	Использование алгоритма машинного обучения (часто это логистическая регрессия) для объединения предсказаний, полученных от других алгоритмов, и формирования «окончательного» прогноза

10.13. Подводим итоги

Основной целью написания этой книги была попытка понятным и интересным способом показать, каким образом применяется на практике машинное обучение. Мы хотели, чтобы вы научились распознавать ситуации, когда его можно использовать для решения стоящих перед вами задач. Вот основные рассмотренные нами моменты:

- Методы машинного обучения превосходно подходят для решения определенных, ориентированных на данные задач.
- Основной рабочий процесс машинного обучения включает в себя подготовку данных, построение модели, оценку модели, оптимизацию и прогнозирование.
- На стадии подготовки данных важно убедиться в том, что собрано достаточное для решения поставленной задачи количество корректных данных, визуализировать и проанализировать эти данные, решить проблему с отсутствующими данными, записать категориальные признаки, выполнить проектирование признаков и тщательно следить, не появляются ли систематические ошибки.

- В машинном обучении используется множество моделей. Они делятся на линейные и нелинейные, параметрические и непараметрические, обучающиеся с учителем и без, а также на модели классификации и регрессии.
- Оценка и оптимизация модели включают в себя циклический скользящий контроль, замеры производительности и регулировку параметров.
- Проектирование признаков позволяет задействовать знания предметной области и пользоваться неструктурированными данными. Зачастую оно коренным образом улучшает производительность моделей.
- Масштабирование связано не только с размером данных. В это понятие входит распределение работы, скорость приема новых данных, время обучения и время прогнозирования, и все это в контексте нужд обслуживаемого бизнеса.

Математический и вычислительный аспекты машинного обучения изучаются уже более 50 лет, но до недавнего времени все ограничивалось академическими кругами и небольшим числом сложных приложений. Рост числа работающих в Интернете компаний и распространение данных выпустили джинна из бутылки. Бизнес, правительственные круги и исследователи каждый день открывают новые варианты применения машинного обучения. Именно им посвящена наша книга, в которой базовая математика и информатика фигурируют ровно в тех количествах, которые требуются, чтобы объяснить не только то, чем именно занимаются практики машинного обучения, но и *как* именно они это делают. Основной упор сделан на техники и процессы, применение которых не зависит от алгоритма, масштаба или приложения. Надеемся, что мы сняли завесу тайны с машинного обучения и помогли увидеть, каким образом его можно использовать для решения важных задач.

Прогресс происходит волнообразно. Компьютерная автоматизация изменила наше общество. Интернет изменил нашу жизнь и культуру. И можно ожидать, что современное машинное обучение — предвестник следующей волны. Что это будет — предсказуемый подъем, волна-убийца или цунами? Пока рано делать выводы, но принятие машинного обучения не просто происходит, оно ускоряется. Одновременно мы видим потрясающий прогресс в развитии инструментов машинного обучения.

Компьютерные системы развиваются новыми способами, по мере того как мы обучаем их все более абстрактным навыкам. Они учатся видеть, слышать, говорить, переводить с одного языка на другой, водить наши автомобили и предвосхищать наши нужды и желания в сферах товаров, услуг, знаний и отношений.

Артур Кларк сказал, что любая достаточно развитая технология неотличима от магии (третий закон Кларка). Когда машинное обучение только появилось, оно напоминало магию. Но по мере его повсеместного распространения мы начинаем понимать, что это инструмент. Видя множество примеров его применения, мы можем обобщить их (в человеческом смысле слова) и вообразить другие варианты, не вдаваясь во все детали его функционирования. Как и другие продвинутые технологии, некогда считавшиеся чудесами, машинное обучение начинает восприниматься как естественное явление, более тонкое и красивое, чем любая магия.

Приложение. Популярные алгоритмы машинного обучения

Название	Тип	Применение	Линейный/ нелинейный	Требует нормализации
Линейная регрессия	Регрессия	Моделирует скалярную целевую переменную с одним или несколькими количественными признаками. Хотя регрессия вычисляет линейную комбинацию, признаки можно преобразовать с помощью нелинейных функций, если мы знаем или можем угадать соотношение между ними. R: www.inside-r.org/r-doc/stats/lm Python: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression	Линейный	Да
Логистическая регрессия	Классификация	Распределяет наблюдения по категориям на базе количественных признаков; предсказывает целевой класс или вероятности целевых классов. R: www.statmethods.net/advstats/glm.html Python: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html	Линейный	Да

Название	Тип	Применение	Линейный/ нелинейный	Требует нормализации
Метод опорных векторов	Классификация/регрессия	Классификация, основанная на разделении в многомерном пространстве. Предсказывает целевые классы. Вероятности целевых классов требуют дополнительных вычислений. Регрессия использует подмножество этих данных, а производительность сильно зависит от данных. R: https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf Python: http://scikit-learn.org/stable/modules/svm.html	Линейный	Да
Метод опорных векторов с ядром	Классификация/регрессия	Метод опорных векторов с поддержкой различных нелинейных моделей. R: https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf Python: http://scikit-learn.org/stable/modules/svm.html	Нелинейный	Да
Метод k-ближайших соседей	Классификация/регрессия	Целевые переменные вычисляются на базе «ближайших» к ним тестовых экземпляров, найденных по формуле определения расстояний (например, по евклидовой метрике). В задаче классификации считаются «голоса» обучающих целевых переменных. В задаче регрессии их значения усредняются. Предсказания базируются на «локальном» подмножестве данных, но для некоторых наборов они демонстрируют высокую точность. R: https://cran.r-project.org/web/packages/class/class.pdf	Нелинейный	Да

Название	Тип	Применение	Линейный/ нелинейный	Требует нормализации
		Python: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html		
Деревья решений	Классификация/регрессия	Обучающая выборка рекурсивно разбивается на подмножества по результатам тестирования значения атрибута, что порождает предсказывающие целевую переменную деревья решений. Дает понятные модели, но алгоритмы «случайный лес» и бустинг практически всегда обеспечивают более низкий уровень ошибок. R: www.statmethods.net/advstats/cart.html Python: http://scikit-learn.org/stable/modules/tree.html#tree	Нелинейный	Нет
«Случайный лес»	Классификация/регрессия	«Ансамбль» решающих деревьев дает более точное предсказание, чем одно дерево. В задаче классификации деревья решений «голосуют». В задаче регрессии данные ими значения усредняются. R: https://cran.r-project.org/web/packages/randomForest/randomForest.pdf Python: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html	Нелинейный	Нет
Бустинг	Классификация/регрессия	В методах с множеством деревьев алгоритм бустинга уменьшает ошибку генерализации, регулируя их веса. Он присваивает больший вес неверно классифицированным экземплярам или (в задаче регрессии) экземплярам с большими остатками.	Нелинейный	Нет

Название	Тип	Применение	Линейный/ нелинейный	Требует нормализации
		R: https://cran.r-project.org/web/packages/gbm/gbm.pdf https://cran.r-project.org/web/packages/adabag/adabag.pdf Python: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html		
Наивный байесовский алгоритм	Классификация	Простой масштабируемый алгоритм классификации, применяемый в основном в задачах классификации текстов (например, определение спама). Предполагает независимость признаков друг от друга (поэтому называется наивным), чего практически никогда не встречается, но в некоторых случаях алгоритм работает на удивление хорошо. Он базируется на теореме Байеса, но не является байесовским в статистическом значении этого термина. R: https://cran.r-project.org/web/packages/e1071/ Python: http://scikit-learn.org/stable/modules/classes.html#modulesklearn.naive_bayes	Нелинейный	Да
Нейронные сети	Классификация/регрессия	Используются для оценки неизвестных функций на основе большого количества входных данных через алгоритм обратного распространения. В общем случае превосходит сложностью и требованиями к вычислительным ресурсам все остальные методы, но в определенных случаях дает отличные результаты. Основа множества методов глубокого обучения	Нелинейный	Да

Название	Тип	Применение	Линейный/ нелинейный	Требует нормализации
		R: https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf https://cran.r-project.org/web/packages/nnet/nnet.pdf Python: http://scikit-learn.org/dev/modules/neural_networks_supervised.html http://deeplearning.net/software/theano/		
Vowpal Wabbit	Классификация/регрессия	Пакет ML, разработанный Джоном Лэнгфордом из Yahoo Research (теперь работает в Microsoft). Включает в себя различные алгоритмы, в том числе обычный метод наименьших квадратов и однослойные нейронные сети. Так как им можно пользоваться в Сети, исчезает необходимость хранить в памяти все данные. Известен быстрой обработкой больших наборов данных. Пакет Vowpal Wabbit обладает уникальным форматом ввода и в общем случае запускается из командной строки, а не через API. https://github.com/JohnLangford/vowpal_wabbit/wiki		
XGBoost	Классификация/регрессия	Хорошо оптимизированная и масштабируемая версия алгоритма растущих деревьев решений. https://xgboost.readthedocs.org/en/latest/		

Хенрик Бринк, Джозеф Ричардс, Марк Феверолф

Машинное обучение

Перевела с английского И. Рузмайкина

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*Н. Римицан*
Литературный редактор	*А. Андриенко*
Художник	*С. Заматевская*
Корректоры	*Н. Викторова, В. Сайко*
Верстка	*Л. Егорова*

Изготовлено в России. Изготовитель: ООО «Питер Пресс».
Место нахождения и фактический адрес: 192102, Россия, город Санкт-Петербург,
улица Андреевская, дом 3, литер А, помещение 7Н. Тел.: +78127037373.

Дата изготовления: 05.2017. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12
Книги печатные профессиональные, технические и научные.

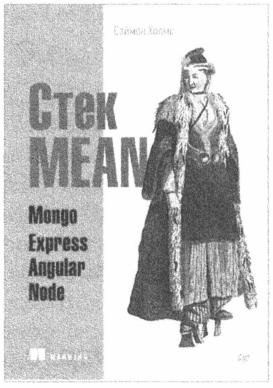
Подписано в печать 18.05.17. Формат 70 · 100/16. Бумага офсетная. Усл. п. л. 27,090. Тираж 1000. Заказ 3196.

Отпечатано в АО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор».
142300, Московская область, г. Чехов, ул. Полиграфистов, 1.
Сайт: www.chpd.ru, E-mail: sales@chpd.ru, тел. 8(499)270-73-59

СЕРИЯ «БИБЛИОТЕКА ПРОГРАММИСТА»

С. Холмс

СТЕК MEAN. MONGO, EXPRESS, ANGULAR, NODE



Обычно при веб-разработке на всех уровнях стека используются разные языки программирования. База данных MongoDB, фреймворки Express и AngularJS и технология Node.js вместе образуют стек MEAN – мощную платформу, на всех уровнях которой применяется всего один язык: JavaScript. Стек MEAN привлекателен для разработчиков и бизнеса благодаря простоте и экономичности, а конечные пользователи любят MEAN-приложения за их скорость и отзывчивость.

ПИТЕР®

СЕРИЯ «БИБЛИОТЕКА ПРОГРАММИСТА»

А. Бхаргава

ГРОКАЕМ АЛГОРИТМЫ. ИЛЛЮСТРИРОВАННОЕ ПОСОБИЕ ДЛЯ ПРОГРАММИСТОВ И ЛЮБОПЫТСТВУЮЩИХ

Алгоритмы – это всего лишь пошаговые инструкции решения задач, и большинство таких задач уже были кем-то решены, протестированы и проверены. Можно, конечно, погрузится в глубокую философию гениального Кнута, изучить многостраничные фолианты с доказательствами и обоснованиями, но хотите ли вы тратить на это свое время? Откройте великолепно иллюстрированную книгу и вы сразу поймете, что алгоритмы – это просто. А грокать алгоритмы – это веселое и увлекательное занятие.

ПИТЕР®

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР®
WWW.PITER.COM

ИЗДАТЕЛЬСКИЙ ДОМ «ПИТЕР» предлагает профессиональную, популярную и детскую развивающую литературу

Заказать книги оптом можно в наших представительствах

РОССИЯ

Санкт-Петербург: м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-83, 703-73-72; e-mail: sales@piter.com

Москва: м. «Электрозаводская», Семеновская наб., д. 2/1, стр. 1, 6 этаж
тел./факс: (495) 234-38-15; e-mail: sales@msk.piter.com

Воронеж: тел.: 8 951 861-72-70; e-mail: hitsenko@piter.com

Екатеринбург: ул. Толедова, д. 43а; тел./факс: (343) 378-98-41, 378-98-42;
e-mail: office@ekat.piter.com; skype: ekat.manager2

Нижний Новгород: тел.: 8 930 712-75-13; e-mail: yashny@yandex.ru; skype: yashny1

Ростов-на-Дону: ул. Ульяновская, д. 26
тел./факс: (863) 269-91-22, 269-91-30; e-mail: piter-ug@rostov.piter.com

Самара: ул. Молодогвардейская, д. 33а, офис 223
тел./факс: (846) 277-89-79, 277-89-66; e-mail: pitvolga@mail.ru,
pitvolga@samara-ttk.ru

БЕЛАРУСЬ

Минск: ул. Розы Люксембург, д. 163; тел./факс: +37 517 208-80-01, 208-81-25;
e-mail: og@minsk.piter.com

Издательский дом «Питер» приглашает к сотрудничеству авторов:
тел./факс: (812) 703-73-72, (495) 234-38-15; e-mail: ivanovaa@piter.com
Подробная информация здесь: http://www.piter.com/page/avtoru

Издательский дом «Питер» приглашает к сотрудничеству зарубежных торговых партнеров или посредников, имеющих выход на зарубежный рынок: *тел./факс: (812) 703-73-73; e-mail: sales@piter.com*

Заказ книг для вузов и библиотек:
тел./факс: (812) 703-73-73, доб. 6243; e-mail: uchebnik@piter.com

Заказ книг по почте: *на сайте www.piter.com; тел.: (812) 703-73-74, доб. 6216; e-mail: books@piter.com*

Вопросы по продаже электронных книг: *тел.: (812) 703-73-74, доб. 6217; e-mail: kuznetsov@piter.com*

КНИГА-ПОЧТОЙ

ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР» МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ:

- на нашем сайте: **www.piter.com**
- по электронной почте: **books@piter.com**
- по телефону: **(812) 703-73-74**

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ:

- Наложенным платежом с оплатой при получении в ближайшем почтовом отделении.
- С помощью банковской карты. Во время заказа вы будете перенаправлены на защищенный сервер нашего оператора, где сможете ввести свои данные для оплаты.
- Электронными деньгами. Мы принимаем к оплате Яндекс.Деньги, Webmoney и Kiwi-кошелек.
- В любом банке, распечатав квитанцию, которая формируется автоматически после совершения вами заказа.

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ДОСТАВКИ:

- Посылки отправляются через «Почту России». Отработанная система позволяет нам организовывать доставку ваших покупок максимально быстро. Дату отправления вашей покупки и дату доставки вам сообщат по e-mail.
- Вы можете оформить курьерскую доставку своего заказа (более подробную информацию можно получить на нашем сайте www.piter.com).
- Можно оформить доставку заказа через почтоматы (адреса почтоматов можно узнать на нашем сайте www.piter.com).

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ:

- фамилию, имя, отчество, телефон, e-mail;
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру;
- название книги, автора, количество заказываемых экземпляров.

БЕСПЛАТНАЯ ДОСТАВКА:

- курьером по Москве и Санкт-Петербургу при заказе на сумму **от 2000 руб.**
- почтой России при предварительной оплате заказа на сумму **от 2000 руб.**

ВАША УНИКАЛЬНАЯ КНИГА

Хотите издать свою книгу? Она станет идеальным подарком для партнеров и друзей, отличным инструментом для продвижения вашего бренда, презентом для памятных событий! Мы сможем осуществить ваши любые, даже самые смелые и сложные, идеи и проекты.

МЫ ПРЕДЛАГАЕМ:

- издать вашу книгу
- издание книги для использования в маркетинговых активностях
- книги как корпоративные подарки
- рекламу в книгах
- издание корпоративной библиотеки

Почему надо выбрать именно нас:

Издательству «Питер» более 20 лет. Наш опыт – гарантия высокого качества.

Мы предлагаем:

- услуги по обработке и доработке вашего текста
- современный дизайн от профессионалов
- высокий уровень полиграфического исполнения
- продажу вашей книги во всех книжных магазинах страны

Обеспечим продвижение вашей книги:

- рекламой в профильных СМИ и местах продаж
- рецензиями в ведущих книжных изданиях
- интернет-поддержкой рекламной кампании

Мы имеем собственную сеть дистрибуции по всей России, а также на Украине и в Беларуси. Сотрудничаем с крупнейшими книжными магазинами. Издательство «Питер» является постоянным участником многих конференций и семинаров, которые предоставляют широкую возможность реализации книг.

Мы обязательно проследим, чтобы ваша книга постоянно имелась в наличии в магазинах и была выложена на самых видных местах.

Обеспечим индивидуальный подход к каждому клиенту, эксклюзивный дизайн, любой тираж.

Кроме того, предлагаем вам выпустить электронную книгу. Мы разместим ее в крупнейших интернет-магазинах. Книга будет сверстана в формате ePub или PDF – самых популярных и надежных форматах на сегодняшний день.

Свяжитесь с нами прямо сейчас:

Санкт-Петербург – *Анна Титова, (812) 703-73-73, titova@piter.com*
Москва – *Сергей Клебанов, (495) 234-38-15, klebanov@piter.com*

САЛД

САНКТ-ПЕТЕРБУРГСКАЯ
АНТИВИРУСНАЯ
ЛАБОРАТОРИЯ
ДАНИЛОВА

www.SALD.ru
8 (812) 336-3739

Антивирусные
программные продукты

В последние годы машинное обучение вышло на уровень большого бизнеса: компании активно используют его для зарабатывания денег, прикладные исследования бурно развиваются, а неугомонные разработчики используют любую возможность повысить свой уровень владения этой тематикой.

Данная книга рассчитана на тех, кто хочет решать самые разнообразные задачи при помощи машинного обучения. Как правило, для этого нужен Python, поэтому в примерах кода используется этот язык, а также библиотеки pandas и scikit-learn. Вы познакомитесь с основными понятиями ML, такими как сбор данных, моделирование, классификация и регрессия, а главное, получите практический опыт обработки реальных данных.