O'REILLY®

Математика для data science

Управляем данными
с помощью линейной алгебры,
теории вероятностей
и статистики

SPRINT book

Томас Нилд

Essential Math for Data Science

Take Control of Your Data with Fundamental Linear Algebra, Probability, and Statistics

Thomas Nield

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Математика для Data Science

Управляем данными с помощью линейной алгебры, теории вероятностей и статистики

Томас Нилд

SPRINT book 2025

ББК 32.973.233.02 +22.1
УДК 004.6:51
Н66

Нилд Томас

Н66 Математика для Data Science. Управляем данными с помощью линейной алгебры, теории вероятностей и статистики. — Астана: «Спринт Бук», 2025. — 352 с.: ил.

ISBN 978-601-08-4357-8

Освойте математический аппарат, который необходим, чтобы преуспеть в сфере data science, машинного обучения и статистики. Автор книги Томас Нилд поможет вам разобраться в таких дисциплинах, как математический анализ, теория вероятностей, линейная алгебра и статистика, и научиться применять их в контексте таких методов, как линейная регрессия, логистическая регрессия и нейронные сети. Попутно вы узнаете, что представляет собой современная область data science и как использовать полученные знания, чтобы достичь максимального успеха в карьере.

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

ББК 32.973.233.02 +22.1
УДК 004.6:51

ISBN 978-1098102937 англ.

Authorized Russian translation of the English edition of Essential Math for Data Science ISBN 9781098102937

ISBN 978-601-08-4357-8

В какофонии, которую представляет собой современный рынок образовательных услуг в области data science, эта книга выделяется как кладезь четких практических примеров, демонстрирующих основные принципы того, что нужно, чтобы разбираться в данных и извлекать из них ценность. Объясняя основы, книга поможет читателю ориентироваться в любой деятельности в области data science, опираясь на твердое представление о ее составных элементах.

Вики Бойкис (Vicki Boykis),
старший инженер по машинному обучению в Tumblr

Data science опирается на линейную алгебру, теорию вероятностей и математический анализ. Томас Нилд мастерски проводит нас через эти и другие темы, закладывая прочный фундамент для того, чтобы понимать математику, которая стоит за data science.

Майк Икс Коэн (Mike X Cohen),
sincXpress

Мы, специалисты по data science, ежедневно используем сложные модели и алгоритмы. Эта книга быстро разъясняет математику, на которую они опираются, и таким образом помогает легче их понять и внедрить.

Сиддхарт Ядав (Siddharth Yadav),
фрилансер в области data science

Как бы я хотела заполучить эту книгу раньше! Томас Нилд проделал потрясающую работу, доступно и увлекательно изложив сложные математические темы. Свежий подход как к математике, так и к data science: Томас объясняет фундаментальные математические понятия и немедленно демонстрирует, как применять их в машинном обучении. Эту книгу обязательно стоит прочитать всем начинающим специалистам по data science.

Татьяна Эдигер (Tatiana Ediger),
фрилансер в области data science,
разработчик и преподаватель курсов

Оглавление

Предисловие

В последние лет десять люди стали чаще применять математику и статистику в повседневной жизни и работе. Почему так происходит? Связано ли это с растущим интересом к data science, которую журнал Harvard Business Review назвал «самой привлекательной профессией XXI века» (https://oreil.ly/GslO6)? Или нас вдохновляет, как машинное обучение и искусственный интеллект обещают изменить нашу жизнь? А может, это потому, что заголовки новостей пестрят цифрами, опросами и результатами исследований, но мы не знаем, как убедиться, что эти результаты достоверны? Или все дело в беспилотных автомобилях и роботах, которые могут вытеснить людей с рабочих мест в обозримом будущем?

По моему мнению, математика и статистика завоевали всеобщий интерес из-за того, что данные становятся все доступнее, и нам нужны математика, статистика и машинное обучение, чтобы разбираться в этих данных. Да, у нас есть научные инструменты, машинное обучение и другие популярные и заманчивые системы. Мы слепо доверяем этим «черным ящикам», устройствам и программам. Мы не понимаем, как они работают, но все равно полагаемся на них.

Хотя легко поверить, что компьютеры умнее нас (и эту идею активно продвигают на рынке), реальность оказывается прямо противоположной. Это несоответствие может быть опасным на многих уровнях. Действительно ли вы хотите, чтобы алгоритм или искусственный интеллект выносил приговор по уголовному делу или управлял автомобилем, когда никто, включая разработчика, не может объяснить, почему программа приняла то или иное решение? Объяснимость — это следующий рубеж статистических вычислений и искусственного интеллекта. Мы достигнем его только тогда, когда откроем черный ящик и поймем, как работает математика внутри него.

Вероятно, вы спросите: как разработчик может не знать, как работает его собственный алгоритм? Мы поговорим об этом во второй половине книги, когда будем обсуждать методы машинного обучения и объясним, почему необходимо разбираться в математике, на которую они опираются.

Еще одна причина того, что данные накапливаются в колоссальных объемах, — подключенные к сети устройства, которыми мы пользуемся в повседневной

жизни. Мы больше не выходим в интернет только с настольного компьютера или ноутбука. Теперь он вокруг нас в смартфонах, автомобилях и бытовых приборах. За последние два десятилетия это незаметно привело к переменам. Данные превратились из рабочего инструмента в ресурс, который добывают и анализируют ради менее определенных целей. Смарт-часы постоянно собирают сведения о нашем пульсе, дыхании, пройденном расстоянии и других показателях. Затем они загружают эти сведения в облако, чтобы анализировать их вместе с данными от других пользователей. Автомобили, оснащенные компьютерами, собирают данные о наших особенностях вождения, а автопроизводители используют эти данные, чтобы разрабатывать беспилотные автомобили. В аптеках появились даже «умные зубные щетки», которые отслеживают, как мы чистим зубы, и сохраняют эту информацию в облаке. Полезна ли такая информация и важна ли она — это уже другой вопрос!

Мы повсеместно сталкиваемся со сбором данных. Иногда он кажется избыточным, и о сопутствующих проблемах конфиденциальности и этики можно написать целую книгу. Но доступность данных также дает возможность по-новому применять математику и статистику и шире задействовать их за пределами академической среды. Благодаря этому можно больше узнать о человеческом опыте, улучшить эстетические и эксплуатационные качества продуктов, а также оптимизировать бизнес-стратегии. Усвоив идеи, которые изложены в этой книге, вы сможете раскрыть потенциал инфраструктуры, в которой хранятся данные. Я не утверждаю, будто данные и статистические инструменты — это панацея, которая решит все мировые проблемы, но они открыли перед нами новые возможности, которые стоит обратить себе на пользу. Иногда эта польза состоит в том, чтобы понять, что те или иные проекты по работе с данными оказались бесперспективными и усилия лучше потратить на что-то другое.

Растущая доступность данных привела к тому, что data science и машинное обучение стали востребованными профессиональными областями. Если вы стремитесь сделать карьеру в области data science, искуственного интеллекта или инженерии данных, вам просто необходимо разбираться в основах теории вероятностей, линейной алгебры, математической статистики и машинного обучения. Я включил в книгу ровно столько высшей математики, математического анализа и статистики, сколько нужно, чтобы лучше понимать, как работают библиотеки, с которыми вы встретитесь.

Цель этой книги — познакомить читателей с различными областями математики, статистики и машинного обучения, которые пригодятся, чтобы решать задачи из реального мира. В первых четырех главах рассматриваются фундаментальные разделы математики — математический анализ, теория вероятностей, линейная алгебра и математическая статистика. В последних трех главах мы перейдем к машинному обучению. Конечная цель знакомства с ним — собрать воедино

все полученные знания и продемонстрировать, что вы умеете применять на практике библиотеки машинного обучения и статистики, проникая внутрь «черного ящика».

Чтобы запускать примеры из этой книги, вам нужен только компьютер с любой операционной системой (Windows, macOS или Linux) и любая среда разработки Python 3. Основные библиотеки Python, которые нам понадобятся, — это `numpy`, `scipy`, `sympy` и `sklearn`. Если вы не знакомы с Python, то знайте, что это дружелюбный и простой в использовании язык программирования с огромным количеством обучающих материалов. Вот какие из них я рекомендую в первую очередь.

Джоэл Грус (Joel Grus). «Data Science. From Scratch»[1] (O'Reilly)

> Вторая глава этой книги — лучший экспресс-курс по Python, который мне встречался. Даже если вы никогда раньше не писали код, Джоэл проделал фантастическую работу, чтобы вы смогли эффективно освоить Python в кратчайшие сроки. Это также отличная книга для того, чтобы держать ее всегда под рукой и применять к ее материалу свои математические знания!

Дипак Сарда (Deepak Sarda). «Python for the Busy Java Developer» (Apress)

> Если вы разработчик с опытом программирования на статически-типизированных, объектно-ориентированных языках, то эта книга — лучший вариант. Я сам начинал программировать на Java и впечатлен тем, как Дипак рассказывает о возможностях Python и объясняет их Java-разработчикам. Если вы изучали .NET, C++ или другие C-подобные языки, то вам, скорее всего, эта книга тоже подойдет.

Эта книга не сделает вас экспертом и не даст вам знаний на уровне PhD. Я стараюсь не злоупотреблять математическими выражениями из непонятных символов, а выражаться на простом человеческом языке. Тем не менее книга позволит вам увереннее разговаривать о математике и статистике и снабдит необходимыми сведениями, чтобы успешно ориентироваться в этих областях. Я считаю, что самый надежный путь к успеху — это не глубоко разбираться в одной узкой области, а, наоборот, обладать практическими знаниями в нескольких областях. Цель моей книги — именно в том, чтобы дать вам такие знания, и их будет достаточно, чтобы вы обрели уверенность в себе и задавали по-настоящему важные вопросы.

Итак, начнем!

---

[1] Грас Д. «Data Science. Наука о данных с нуля.— 2-е изд., перераб. и доп.».

Условные обозначения

В этой книге используются условные обозначения.

Курсив

Курсивом выделены новые термины и важные понятия.

`Моноширинный шрифт`

Используется для листингов программ, а также обозначает внутри абзацев такие элементы, как имена переменных и функций, базы данных, типы данных, переменные окружения, инструкции и ключевые слова.

`Моноширинный полужирный шрифт`

Выделяет команды или другой текст, который пользователю нужно ввести в точности так, как написано.

`Моноширинный курсив`

Выделяет текст, который нужно заменить значениями, введенными пользователем, или значениями, которые определяются контекстом.

Шрифт без засечек

Используется для обозначения URL, адресов электронной почты, названий кнопок и других элементов интерфейса, каталогов.

Этот рисунок указывает на совет или предложение.

Этот рисунок указывает на общее примечание.

Этот рисунок указывает на предупреждение.

Использование исходного кода примеров

Вспомогательные материалы (примеры кода, упражнения и т. д.) можно скачать по адресу: https://github.com/thomasnield/machine-learning-demo-data. Если у вас возникнут вопросы технического характера или проблемы с примерами кода, направляйте их по электронной почте: bookquestions@oreilly.com.

В общем случае все примеры кода из книги вы можете использовать в своих программах и в документации. Вам не нужно обращаться в издательство за разрешением, если вы не собираетесь воспроизводить существенные части программного кода. Если вы разрабатываете программу и используете в ней несколько фрагментов кода из книги, вам не нужно обращаться за разрешением. Но для продажи или распространения примеров из книги вам потребуется разрешение от издательства O'Reilly. Вы можете отвечать на вопросы, цитируя данную книгу или примеры из нее, но для включения существенных объемов программного кода из книги в документацию вашего продукта потребуется разрешение.

Мы рекомендуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN.

Чтобы получить разрешение на использование значительных объемов программного кода из книги, обращайтесь по адресу permissions@oreilly.com.

Благодарности

Работа над этой книгой продолжалась больше года. Прежде всего я хочу поблагодарить свою жену Кимберли за поддержку во время написания книги, особенно в течение первого года жизни нашего сына Уайатта. Кимберли — замечательная жена и мать, и все, что я сейчас делаю, я делаю ради сына и благополучного будущего нашей семьи.

Я хочу поблагодарить своих родителей за то, что они научили меня преодолевать трудности и никогда не опускать руки. Учитывая тему этой книги, я рад, что они вдохновили меня на то, чтобы серьезно изучать математический анализ в старших классах и колледже. Нельзя написать книгу, если регулярно не выходить из своей зоны комфорта, и в этом мне помогло воспитание, которое я получил от родителей.

Я хочу поблагодарить замечательную команду редакторов и других специалистов издательства O'Reilly, плодотворное сотрудничество с которыми продолжается с тех пор, как я написал свою первую книгу по SQL в 2015 году. С Джилл и Джесс было очень приятно работать над написанием и публикацией этой книги, и я благодарен Джесс за то, что она вспомнила обо мне, когда встал вопрос о том, чтобы издать книгу по математике и data science.

Я хочу поблагодарить своих коллег из Университета Южной Калифорнии, с кафедры авиационной безопасности. Возможность стать первопроходцем в разработке принципов безопасности систем искусственного интеллекта позволила мне узнать то, что доступно немногим, и я с нетерпением жду, чего мы добьемся в ближайшие годы. Арч, ты не перестаешь меня удивлять, и я боюсь, что мир сломается в тот день, когда ты уйдешь на пенсию.

И наконец, я хочу поблагодарить моего брата Дуайта Нилда (Dwight Nield) и моего друга Джона Островера (Jon Ostrower) — партнеров в моем начинании Yawman Flight. Создать стартап очень сложно, и их помощь позволила мне получить драгоценную свободу действий, чтобы написать эту книгу. Джон привел меня в университет, а его достижения в сфере авиационной журналистики просто поразительны (поищите его в сети!). Для меня большая честь, что мои партнеры так же, как и я, увлечены проектом, который я начинал в своем гараже, и я сомневаюсь, что сумел бы воплотить его в жизнь без их помощи.

Всем, кого я пропустил, спасибо за большие и малые дела, которые вы осуществили. Мое любопытство и стремление задавать вопросы гораздо чаще вознаграждались, чем нет, и я не принимаю это как должное. Как говорил Тед Лассо, «будь любопытным и меньше суди»[1].

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@sprintbook.kz (издательство «SprintBook», компьютерная редакция).

Мы будем рады узнать ваше мнение!

О редакторах русского издания

Константин Кноп — преподаватель математики, автор задач и головоломок, член методической комиссии и жюри Всероссийской олимпиады школьников по математике, автор нескольких математических книг.

Ростислав Чебыкин — специалист по прикладной статистике, консультант по математическому аппарату машинного обучения, методист и разработчик онлайн-опросов для бизнеса, докладчик на профессиональных конференциях. Преподавал информационные технологии в РосНОУ и Бауманском учебном центре «Специалист».

[1] Задолго до выхода сериала «Тед Лассо» эту фразу ошибочно приписывали американскому поэту и публицисту Уолту Уитмену (1819–1892), при этом ее настоящее происхождение остается неизвестным. — *Примеч. науч. ред.*

ГЛАВА 1

Обзор начальной математики и математического анализа

В начале первой главы мы разберемся, что такое числа и как устроены переменные и функции в декартовой системе координат. Затем мы рассмотрим возведение в степень и логарифмы, а после этого изучим две основные конструкции математического анализа — производные и интегралы.

Прежде чем углубиться в прикладные области математики — такие, как теория вероятностей, линейная алгебра, математическая статистика и машинное обучение, — имеет смысл рассмотреть некоторые основные понятия базовой математики и математического анализа. Только прошу, не бросайте эту книгу и не разбегайтесь в ужасе! Я расскажу, как вычислять производные и интегралы для функции таким способом, которому вас вряд ли учили в университете. Мы будем делать это не на бумаге, а с помощью языка программирования Python. Даже если вы не знакомы с производными и интегралами, не переживайте.

Я постараюсь изложить эти темы максимально сжато, уделяя внимание только тому, что пригодится в последующих главах и что относится к теме математики для data science.

Это не полноценный курс математики!

Эта книга ни в коем случае не претендует на то, чтобы служить исчерпывающим обзором математики для средних и высших учебных заведений. Если вам нужна именно такая книга, то обратите внимание на «*No Bullshit Guide to Math and Physics*» Ивана Савова (Ivan Savov). Первые несколько глав содержат лучший краткий курс по математике из всех, что я когда-либо видел. Книга «*Mathematics 1001*» Ричарда Элвиса (Richard Elwes) тоже отличается замечательным материалом, причем в виде небольших заметок.

Теория чисел

Что такое числа? Я обещаю не слишком философствовать в этой книге, но не являются ли числа искусственной конструкцией, которую мы сами же и придумали? Почему мы пользуемся цифрами от 0 до 9, а не больше? Почему у нас есть обыкновенные и десятичные дроби, а не только целые числа? Область математики, которая занимается числами и отвечает на вопрос, почему мы определили их так, а не иначе, известна как теория чисел, или высшая арифметика.

Теория чисел уходит корнями в глубокую древность, когда математики открывали различные числовые системы, и объясняет, почему сейчас мы используем числа именно в таком виде. Ниже представлены несколько числовых систем, которые наверняка вам знакомы.

Натуральные числа

Это числа 1, 2, 3, 4, 5 и т. д.[1] Сюда входят только положительные числа, и это самая ранняя известная нам числовая система. Даже пещерные люди выцарапывали на костях и стенах пещер насечки, которые изображали натуральные числа, — так наши предки вели свои первобытные подсчеты.

Целые неотрицательные числа

В дополнение к натуральным числам позднее появилось понятие нуля и соответствующий символ «0». Целые неотрицательные числа — это 0, 1, 2, 3 и т. д. Из Древнего Вавилона пришла идея обозначать особым символом пустые «разряды» в многозначных числах, таких как 10, 1 000 или 1 090. Нули в них указывают на то, что в соответствующем разряде нет значения.

Целые числа

К целым числам относятся натуральные числа, противоположные им отрицательные числа, а также 0. Для нас это понятие — само собой разумеющееся, но древние математики с глубоким недоверием относились к идее отрицательных чисел. Однако если из 3 вычесть 5, то получится −2. Это особенно удобно, когда речь идет о финансах, где мы измеряем прибыль и убытки. В 628 году нашей эры индийский математик Брахмагупта на примере решения квадратного уравнения показал, почему отрицательные числа необходимы для развития арифметики, и только после этого они стали общепринятыми.

---

[1] Иногда к натуральным числам относят и ноль: например, стандарты ISO 80000-2 и ГОСТ Р 54521–2011 определяют два множества натуральных чисел: $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ и $\mathbb{N}^* = \{0, 1, 2, 3, \ldots\}$. В этой книге принят весьма распространенный и удобный на практике подход, согласно которому натуральный ряд начинается с 1. — *Примеч. науч. ред.*

Рациональные числа

Любое число, которое можно представить в виде обыкновенной дроби, например $\frac{2}{3}$, относится к рациональным. Сюда же входят все конечные десятичные дроби и целые числа, потому что их тоже можно выразить в виде обыкновенной дроби, например $\frac{687}{100} = 6,87$ и $\frac{2}{1} = 2$ соответственно. *Рациональными* они называются потому, что одно из значений латинского слова *ratiō* — «отношение, пропорция». Рациональные числа быстро стали необходимы, потому что время, количество вещества и другие величины не всегда можно измерить в дискретных единицах. Например, молоко не обязательно продается в литровой упаковке; иногда его объем приходится измерять в долях литра. Или, скажем, если я бежал 12 минут, я не смогу выразить пройденное расстояние в целых километрах; оно составит $\frac{7}{5}$ километра.

Иррациональные числа

Иррациональные числа нельзя выразить в виде обыкновенной дроби. К ним относятся знаменитое число ϖ, квадратные корни из некоторых чисел (например $\sqrt{2}$), и число Эйлера[1] e, о котором мы узнаем позже. В десятичной записи у этих чисел бесконечное количество знаков после запятой, например 3,14159265358979323846...

История иррациональных чисел довольно интересна. Древнегреческий математик Пифагор считал, что все числа рациональны. Он верил в это так горячо, что создал религию, которая предписывала поклоняться числу 10. «Благослови нас, божественное число, ты, породившее богов и людей!» — молился он и его последователи (почему число 10 было таким особенным, я не знаю). Существует легенда, что один из пифагорейцев по имени Гиппас доказал, что не все числа рациональны, просто продемонстрировав квадратный корень из 2. Это сильно пошатнуло систему верований Пифагора, и в наказание он утопил Гиппаса в море.

Вещественные числа

К вещественным (или действительным) числам относятся все рациональные и все иррациональные числа. На практике в data science любые десятичные дроби, с которыми вы работаете, можно рассматривать как вещественные числа.

Комплексные и мнимые числа

С такими числами можно столкнуться, если извлекать квадратный корень из отрицательного числа. Хотя мнимые и комплексные числа играют свою роль в некоторых типах задач, мы в основном будем держаться от них подальше.

---

[1] В отечественной практике не принято называть *e* числом Эйлера. — *Примеч. ред.*

В data science практически всегда используются натуральные, целые и вещественные числа. Мнимые числа могут встретиться в более сложных случаях — например, при разложении матриц, которых мы коснемся в главе 4.

Комплексные и мнимые числа

Если вы хотите подробнее узнать про мнимые числа, на YouTube есть отличный набор видеороликов «*Imaginary Numbers are Real*» («Мнимые числа реальны», https://oreil.ly/bvyIq).

Порядок выполнения арифметических операций

Надеемся, что вы знакомы с *порядком выполнения* арифметических операций: это порядок, в котором вычисляется каждая часть математического выражения. Напомню, что сначала вычисляется то, что заключено в круглые скобки, затем следует возведение в степень, затем умножение, деление, сложение и вычитание. Операции с одинаковым приоритетом (такие, как несколько сложений подряд) выполняются слева направо.

Например, вычислим значение такого выражения:

$$2\times\frac{(3+2)^2}{5}-4$$

В первую очередь вычислим выражение внутри скобок $(3+2)$, которое дает 5:

$$2\times\frac{(\mathbf{5})^2}{5}-4$$

Затем возведем в квадрат число 5, которое мы только что получили:

$$2\times\frac{\mathbf{25}}{5}-4$$

Далее следуют умножение и деление. Их порядок можно менять местами, потому что деление — это то же самое, что умножение, но с использованием дробей.

Умножая 2 на $\frac{25}{5}$, получаем $\frac{50}{5}$:

$$\frac{50}{5}-4$$

Затем разделим 50 на 5, в результате чего получим 10:

$$\mathbf{10}-4$$

И, наконец, выполняем сложение и вычитание. Естественно, 10 – 4 даст нам 6:

$$10-4=6$$

Конечно, если бы мы сформулировали это выражение на Python, то получили бы значение `6.0`, как показано в примере 1.1.

Пример 1.1. Вычисление выражения на Python

```
my_value = 2 * (3 + 2)**2 / 5 - 4
print(my_value)       # выводит 6.0
```

Следующее замечание может показаться элементарным, но все же об этом важно помнить. Хорошим тоном считается использовать в коде круглые скобки в сложных выражениях, чтобы показать порядок вычисления, даже если формально скобки не обязательны.

В примере 1.2 я сгруппировал дробную часть выражения в круглых скобках, что помогает отделить ее от остального выражения.

Пример 1.2. Использование в Python круглых скобок для наглядности

```
my_value = 2 * ((3 + 2)**2 / 5) - 4
print(my_value)       # выводит 6.0
```

Хотя оба примера синтаксически правильны, второй легче воспринимать человеческим глазом. Когда вы или кто-то другой вносит изменения в код, легко ориентироваться по круглым скобкам, чтобы не нарушить порядок операций. Это обеспечивает дополнительную защиту от ошибок при изменении кода.

Переменные

Если вы программировали на Python или на другом языке, то представляете себе, что такое переменная. В математике *переменная* — это именованная «область» для размещения неопределенного или неизвестного числа.

Например, можно взять переменную x, которая обозначает любое вещественное число, и умножить ее, не уточняя, чему конкретно равно ее значение. В примере 1.3 мы принимаем от пользователя переменную x и умножаем ее на 3.

Пример 1.3. Переменная в Python, над которой производится операция умножения

```
x = int(input("Введите число:\n"))
product = 3 * x
print(product)
```

Для некоторых типов переменных существуют стандартные имена. Ничего страшного, если вы видите их первый раз в жизни! Но некоторые узна́ют греческие

буквы θ (тета), которая обозначает углы, и β (бета), которая обозначает параметр линейной регрессии. Использовать греческие символы в именах переменных в Python довольно неудобно, поэтому мы назовем эти переменные `theta` и `beta`, как показано в примере 1.4[1].

Пример 1.4. Греческие имена переменных в Python

```
beta = 1.75
theta = 30.0
```

Заметим также, что в математике имена переменных могут сопровождаться индексами, чтобы одно и то же имя можно было использовать для нескольких экземпляров переменной. Для практических целей просто считайте, что эти имена принадлежат отдельным переменным. Если вы встречаете переменные x_1, x_2 и x_3, рассматривайте их как три разные переменные, как показано в примере 1.5.

Пример 1.5. Переменные с нижним индексом в Python

```
x1 = 3   # или x_1 = 3
x2 = 10  # или x_2 = 10
x3 = 44  # или x_3 = 44
```

Функции

Функции — это выражения, которые задают соответствие между двумя или более переменными. Более конкретно, функция берет *входные переменные* (которые также называются *независимыми переменными* или *аргументами функции*), подставляет их в выражение, и в результате получается *выходная переменная* (которая также называется *зависимой переменной* или *значением функции*).

Рассмотрим пример простой линейной функции:

$$y = 2x + 1$$

Для любого заданного значения x мы вычисляем выражение с этим x, чтобы найти y. Если $x = 1$, то $y = 3$. Если $x = 2$, то $y = 5$. Если $x = 3$, то $y = 7$ и так далее, как показано в табл. 1.1.

[1] Современный Python формально позволяет использовать в именах переменных более 3 000 символов Unicode; например, θ или мама — допустимые имена. Однако официальная документация по Python рекомендует обходиться только «обычными» символами, которые можно ввести с клавиатуры в латинской раскладке. В большинстве случаев это лучшее решение. — *Примеч. науч. ред.*

Таблица 1.1. Различные значения функции $y = 2x + 1$

x	$2x+1$	y
0	$(2\times0)+1$	1
1	$(2\times1)+1$	3
2	$(2\times2)+1$	5
3	$(2\times3)+1$	7

Функции полезны тем, что они моделируют предсказуемую зависимость между переменными: например, сколько возгораний *y* можно ожидать при температуре *x*. Линейные функции пригодятся в главе 5, где мы будем заниматься линейной регрессией.

Зависимую переменную *y* иногда записывают в другой форме, явно обозначая ее как функцию от *x*, например $f(x)$. Таким образом, вместо того чтобы указывать функцию как $y = 2x + 1$, ее можно выразить так:

$$f(x) = 2x + 1$$

В примере 1.6 показано, как можно объявить математическую функцию в Python и запустить ее.

Пример 1.6. Объявление линейной функции в Python

```
def f(x):
    return 2 * x + 1

x_values = [0, 1, 2, 3]

for x in x_values:
    y = f(x)
    print(y)
```

У функций от вещественных чисел есть порой незаметная, но важная особенность: они часто имеют бесконечное число значений x и соответствующих значений y. Подумайте: сколько значений x можно подставить в функцию $f(x) = 2x + 1$? Почему бы вместо 0; 1; 2; 3 не использовать 0; 0,5; 1; 1,5; 2; 2,5; 3 — как показано в табл. 1.2?

Таблица 1.2. Различные значения функции $y = 2x + 1$

x	$2x+1$	y
0	$(2\times0)+1$	1
0,5	$(2\times0,5)+1$	2
1,0	$(2\times1)+1$	3
1,5	$(2\times1,5)+1$	4
2,0	$(2\times2)+1$	5
2,5	$(2\times2,5)+1$	6
3,0	$(2\times3)+1$	7

Или почему бы не перебирать x с шагом 0,25? Или $\frac{1}{10}$? Эти шаги можно сделать бесконечно малыми, тем самым показывая, что $y = 2x + 1$ — это *непрерывная функция*, где для каждого возможного значения x существует значение y. Это позволяет представить функцию в виде линии, как показано на рис. 1.1.

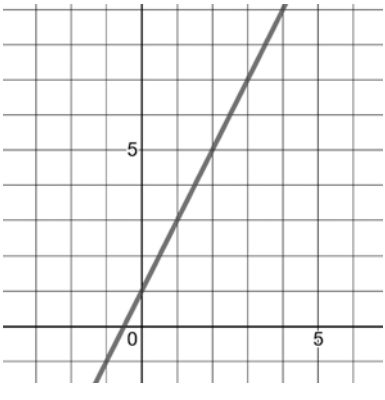


Рис. 1.1. График функции $y = 2x + 1$

Двумерная плоскость с двумя числовыми осями (по одной для каждой переменной), на которой мы строим график, называется *декартовой* (или *прямоугольной*) *системой координат* или *координатной плоскостью*. Мы отслеживаем заданное значение x, затем находим соответствующее значение y и строим пересечения всех таких значений в виде линии. Обратите внимание, что вещественные числа (или десятичные дроби, если вам с ними привычнее) устроены так, что существует

бесконечное количество значений x. Именно поэтому, когда мы строим график функции $f(x)$, получается непрерывная линия без разрывов. На этой линии, как и на любом ее отрезке, находится бесконечное количество точек.

Чтобы строить графики с помощью Python, существует множество библиотек — от Plotly до matplotlib. Во многих задачах на протяжении этой книги мы будем использовать SymPy, и первая задача — построить график функции. SymPy использует matplotlib, поэтому убедитесь, что у вас установлен этот пакет. (В противном случае программа выведет в консоль уродливую диаграмму из текстовых символов.) После этого просто объявите переменную x с помощью функции `symbols()` из SymPy, объявите функцию, а затем постройте график, как показано в примере 1.7 и на рис. 1.2.

Пример 1.7. Построение графика линейной функции с помощью SymPy

```
from sympy import *

x = symbols('x')
f = 2*x + 1
plot(f)
```

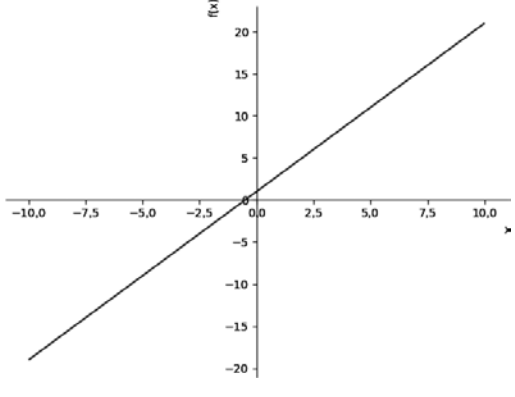


Рис. 1.2. Построение графика линейной функции с помощью SymPy

Пример 1.8 и рис. 1.3 демонстрируют другую функцию: $f(x) = x^2 + 1$.

Пример 1.8. Построение графика квадратичной функции

```
from sympy import *

x = symbols('x')
f = x**2 + 1
plot(f)
```

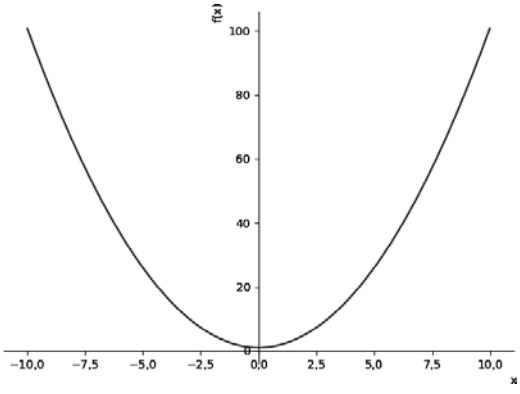


Рис. 1.3. Построение графика квадратичной функции с помощью SymPy

Обратите внимание, что на рис. 1.3 мы получили не прямую линию, а гладкую и симметричную кривую, которая называется параболой. Эта функция непрерывна, но не линейна, потому что ее значения расположены не на прямой линии. С подобными криволинейными функциями математически сложнее работать, но мы научимся некоторым приемам, которые упростят эту работу.

Нелинейные функции

Если график функции — непрерывная, но не прямая линия, такую функцию называют *нелинейной*.

Стоит заметить, что у функции может быть не один, а несколько аргументов. Например, можно рассмотреть функцию с аргументами x и y. Обратите внимание, что здесь, в отличие от предыдущих примеров, y — это не значение функции, а ее аргумент.

$$f(x,y) = 2x + 3y$$

Поскольку у нас теперь две независимые переменные (x и y) и одна зависимая переменная $f(x,y)$, график будет строиться в трех измерениях, чтобы получить не прямую, а плоскость значений, как показано в примере 1.9 и на рис. 1.4.

Пример 1.9. Объявление функции с двумя независимыми переменными в Python

```
from sympy import *
from sympy.plotting import plot3d

x, y = symbols('x y')
f = 2*x + 3*y
plot3d(f)
```

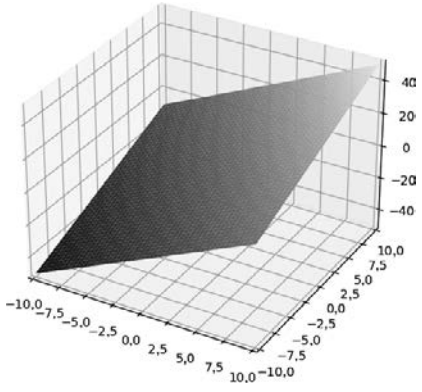


Рис. 1.4. Построение графика трехмерной функции с помощью SymPy

Независимо от того, сколько аргументов у функции, она, как правило, выводит только одну зависимую переменную. Если нужно вычислить несколько зависимых переменных, то для каждой из них, вероятно, понадобится отдельная функция.

Суммы

Я обещал не злоупотреблять математическими выражениями из непонятных символов, однако один символ настолько распространен и полезен, что я не могу не рассказать о нем. Сумма обозначается греческой буквой Σ (сигма) и складывает элементы.

Например, если я хочу перебрать все натуральные числа от 1 до 5, умножить каждое из них на 2 и просуммировать результаты, то вот как это запишется с помощью символа суммы:

$$\sum_{i=1}^{5} 2i = (2\times 1)+(2\times 2)+(2\times 3)+(2\times 4)+(2\times 5) = 30$$

В примере 1.10 показано, как выполнить это суммирование на Python.

Пример 1.10. Выполнение суммирования на Python

```
summation = sum(2*i for i in range(1,6))
print(summation)
```

Обратите внимание, что i — это вспомогательная переменная: она представляет по очереди каждое из чисел, которые мы перебираем. Каждое значение переменной умножается на 2, и результаты затем суммируются. Когда мы перебираем

последовательность данных, можно встретить переменные типа x_i, которые указывают на элемент последовательности с индексом *i*.

Функция range()

Напомним, что функция `range()` в Python не включает последний элемент последовательности. Например, если вызвать `range(1,4)`, то она будет перебирать числа 1, 2, 3. При этом число 4 служит верхней границей последовательности и не входит в перебор.

Еще одно традиционное обозначение — буква *n*, которая выражает количество элементов в последовательности, — например, количество записей в наборе данных. Здесь мы перебираем числовую последовательность размера *n*, умножаем каждое число на 10 и суммируем произведения:

$$\sum_{i=1}^{n} 10x_i$$

В примере 1.11 с помощью Python вычисляется значение этого выражения для набора из четырех чисел. Обратите внимание, что в Python (как и в большинстве языков программирования) элементы обычно индексируются начиная с 0, в то время как в математике принято начинать с индекса 1. Поэтому в функции `range()` мы сместим начало последовательности и начнем с 0.

Пример 1.11. Вычисление суммы элементов на Python

```
x = [1, 4, 6, 2]
n = len(x)

summation = sum(10*x[i] for i in range(0,n))
print(summation)
```

В этом и заключается суть суммирования. В двух словах, символ суммы Σ обозначает, что нужно сложить друг с другом несколько компонентов. При этом с помощью индекса *i* и максимального значения *n* выражается каждая из итераций (шагов перебора), которые входят в процедуру суммирования. Мы будем встречаться с такими конструкциями на протяжении всей книги.

СУММЫ В SYMPY

Не бойтесь вернуться к этой врезке позже, когда вы узнаете подробнее о SymPy. Библиотека SymPy, с помощью которой мы строим графики функций, — это на самом деле библиотека символьной математики. О том, что это значит, мы поговорим позже в этой главе. Но на будущее отметим, что операция суммирования в SymPy выполняется с помощью

функции `Sum()`. В следующем примере кода мы перебираем значения *i* от 1 до *n*, умножаем каждое *i* на 2 и суммируем произведения. Но затем мы используем функцию подстановки `subs()`, чтобы *n* стало равно 5. Это приведет к тому, что программа переберет и умножит все элементы *i* от 1 до 5 и просуммирует результаты:

```
from sympy import *

i,n = symbols('i n')

# перебирает элементы i от 1 до n,
# затем умножает и суммирует
summation = Sum(2*i,(i,1,n))

# задает n равным 5,
# перебирает числа от 1 до 5
up_to_5 = summation.subs(n, 5)
print(up_to_5.doit()) # 30
```

Обратите внимание, что суммирование в SymPy является «ленивым», т. е. оно не вычисляется автоматически и не упрощается. Поэтому, чтобы получить значение выражения, используйте функцию `doit()`.

Возведение в степень

Возведение в степень заключается в том, что число умножается само на само себя определенное количество раз. Чтобы возвести 2 в третью степень (это записывается как 2^3, где 3 стоит в верхнем индексе), нужно перемножить между собой три числа 2:

$$2^3 = 2 \times 2 \times 2 = 8$$

Основание степени — это переменная или величина, которую мы возводим в степень, а *показатель степени* отражает, сколько раз мы умножаем основание само на себя. В выражении 2^3 основанием является 2, а показателем — 3.

У возведения в степень есть несколько интересных свойств. Допустим, мы перемножили x^2 и x^3. Посмотрим, что произойдет, если разложить это выражение на два обычных умножения, а затем объединить показатели степеней:

$$x^2 x^3 = (x \times x)(x \times x \times x) = x^{2+3} = x^5$$

Это называется *правилом произведения степеней*: чтобы перемножить степени с одинаковым основанием, нужно просто сложить их показатели. Обратите

внимание: чтобы применять это правило, основания всех перемножаемых степеней должны быть одинаковыми.

Теперь рассмотрим деление. Что произойдет, если разделить x^2 на x^5?

$$\frac{x^2}{x^5} = \frac{x \times x}{x \times x \times x \times x \times x} = \frac{1}{x \times x \times x} = \frac{1}{x^3} = x^{-3}$$

Как видите, при делении x^2 на x^5 можно сократить два x в числителе и знаменателе, в результате чего получится $\frac{1}{x^3}$. Если один и тот же множитель есть и в числителе, и в знаменателе, его можно сократить.

Но что теперь делать с x^{-3}? Это подходящий момент, чтобы познакомиться с отрицательными показателями степеней. Это всего лишь один из способов выразить, что операция возведения в степень стоит в знаменателе дроби. Например, x^{-3} — это то же самое, что $\frac{1}{x^3}$:

$$\frac{1}{x^3} = x^{-3}$$

Возвращаясь к правилу произведения степеней, можно убедиться, что оно применимо и к отрицательным показателям. Чтобы интуитивно это себе представить, давайте подойдем к задаче по-другому. Деление одной степени на другую можно выразить так: сделать показатель 5 в выражении x^5 отрицательным, а затем умножить то, что получилось, на x^2. Когда нужно прибавить отрицательное число, то фактически выполняется вычитание. Поэтому правило произведения степеней, которое велит складывать показатели, работает и в этом случае:

$$\frac{x^2}{x^5} = x^2 \frac{1}{x^5} = x^2 x^{-5} = x^{2+(-5)} = x^{-3}$$

И напоследок: можете ли вы объяснить, почему любое число в степени 0 равно 1[1]?

$$x^0 = 1$$

Лучший способ разобраться с этим — вспомнить, что если разделить любое число само на себя, то получится 1. Например, алгебраически очевидно, что $\frac{x^3}{x^3}$ сводится к 1. Но это выражение также равняется x^0:

$$1 = \frac{x^3}{x^3} = x^3 x^{-3} = x^{3+(-3)} = x^0$$

Применяя транзитивное свойство равенства (если $a = b$ и $b = c$, то $a = c$), можно сделать вывод, что $x^0 = 1$.

[1] В классической математике это верно для всех оснований степени, кроме 0; не определено. — *Примеч. науч. ред.*

УПРОЩЕНИЕ ВЫРАЖЕНИЙ С ПОМОЩЬЮ SYMPY

Если вам не хочется возиться с тем, чтобы упрощать алгебраические выражения, можно воспользоваться библиотекой SymPy, которая сделает это за вас. Вот как можно упростить предыдущий пример:

```
from sympy import *

x = symbols('x')
expr = x**2 / x**5
print(expr) # x**(-3)
```

Теперь поговорим о дробных показателях степеней. Это альтернативный способ представления корней — например, квадратного корня. Чтобы освежить наши знания, зададим вопрос: «Какое число, умноженное само на себя, даст 4?» Конечно, ответ равен 2. Заметим, что $4^{1/2}$ — это то же самое, что $\sqrt{4}$:

$$4^{1/2} = \sqrt{4} = 2$$

Кубический корень похож на квадратный, но это число, которое умножается на себя три раза, чтобы получить результат. Кубический корень из 8 обозначается $\sqrt[3]{8}$ и отвечает на вопрос: «Какое число, умноженное на себя трижды, дает 8?» Этим числом будет 2, потому что $2 \times 2 \times 2 = 8$. Кубический корень можно выразить как дробный показатель степени: например, $\sqrt[3]{8}$ можно записать как $8^{1/3}$:

$$8^{1/3} = \sqrt[3]{8} = 2$$

Что произойдет, если кубический корень из 8 умножить сам на себя три раза? Корень исчезнет, и останется 8. Если выразить этот кубический корень в виде дробной степени $8^{1/3}$, то станет ясно, что нужно сложить показатели степеней, что в итоге даст 1 и позволит избавиться от корня:

$$\sqrt[3]{8} \times \sqrt[3]{8} \times \sqrt[3]{8} = 8^{1/3} \times 8^{1/3} \times 8^{1/3} = 8^{1/3+1/3+1/3} = 8^1 = 8$$

И последнее свойство: если степень возводится в степень, то показатели перемножаются, а основание остается прежним. Таким образом, $\left(8^3\right)^2$ упрощается до 8^6:

$$\left(8^3\right)^2 = 8^{3 \times 2} = 8^6$$

Если вам любопытно, почему это так, попробуйте разложить это выражение, и убедитесь, что здесь работает правило произведения степеней:

$$\left(8^3\right)^2 = 8^3 \times 8^3 = 8^{3+3} = 8^6$$

И наконец, что означает дробный показатель степени, в котором числитель не равен 1, например $8^{2/3}$? Это всего лишь значит, что надо извлечь кубический корень из 8 и возвести его в квадрат: $\left(8^{1/3}\right)^2$.

$$8^{2/3} = \left(8^{1/3}\right)^2 = 2^2 = 4$$

И наконец, показатель степени может быть иррациональным: например, 8^{π} равно 687,2913. Это может показаться не вполне очевидным, и это неудивительно! Чтобы сэкономить время, мы не будем углубляться в эту тему, потому что она требует обратиться к математическому анализу. Но, по сути, можно возводить числа в иррациональные степени, приближая их рациональными числами. Именно так поступают компьютеры, потому что они в любом случае умеют обращаться только с числами с конечным количеством знаков после запятой.

Например, у числа π бесконечное количество десятичных знаков. Но если оставить только первые 11 цифр (3,1415926535), то π будет приближено рациональным числом 31415926535/10000000000. Восьмерка, возведенная в такую степень, дает нам приблизительно 687,2913, что примерно совпадает с расчетами на калькуляторе:

$$8^{\pi} \approx 8^{\frac{31415926535}{10000000000}} \approx 687,2913$$

Логарифмы

Логарифм — это математическая функция, которая находит показатель степени для определенного числа и основания. На первый взгляд она может показаться не особо интересной, но на самом деле у нее множество применений. Логарифмы встречаются повсюду: от измерения силы землетрясений до управления громкостью бытовой аудиосистемы. Они также часто фигурируют в машинном обучении и data science. Собственно говоря, логарифмы будут ключевым понятием логистической регрессии в главе 6.

Начнем наши размышления с вопроса: «В какую степень нужно возвести число 2, чтобы получилось 8?» Один из способов выразить это математически — использовать в качестве показателя букву *x*:

$$2^x = 8$$

Интуитивно мы знаем ответ: $x = 3$, но хотелось бы выразить эту популярную математическую операцию более элегантным способом. Для этого и предназначена функция $\log()$.

$$\log_2 8 = x$$

Это выражение означает, что основание равно 2, и требуется вычислить показатель степени, в которую нужно возвести это основание, чтобы получить 8. В более общем случае можно перевыразить переменный показатель в виде логарифма:

$$a^x = b$$
$$\log_a b = x$$

С алгебраической точки зрения это способ изолировать x, что становится важным, если требуется найти сам x. В примере 1.12 показано, как вычислить этот логарифм на Python.

Пример 1.12. Вычисление логарифма на Python

```
from math import log

# 2 в какой степени даст 8?
x = log(8, 2)

print(x) # выводит 3.0
```

Если не указать основание функции `log()` на такой платформе, как Python, обычно будет использовано значение по умолчанию. В некоторых областях, например при измерении силы землетрясений, основанием логарифма по умолчанию является 10. Но в data science стандартное основание — число e, и в Python используется именно оно, поэтому вскоре мы поговорим об этом числе.

Как и у степеней, у логарифмов есть несколько свойств, которые проявляются при умножении, делении, возведении в степень и т. д. Чтобы сэкономить ваше время и не отвлекать от главного, я просто приведу их все в табл. 1.3. Основная мысль, которую следует усвоить, заключается в том, что логарифм — это показатель степени, в которую надо возвести заданное основание, чтобы в результате получить заданное число.

Если вам понадобится подробнее изучить особенности логарифмов, используйте в качестве справочного материала табл. 1.3, где свойства степеней и логарифмов перечислены бок о бок.

Таблица 1.3. Свойства степеней и логарифмов

Операция	Свойство степени	Свойство логарифма[1]
Умножение	$x^m \times x^n = x^{m+n}$	$\log(a \times b) = \log(a) + \log(b)$
Деление	$\frac{x^m}{x^n} = x^{m-n}$	$\log\left(\frac{a}{b}\right) = \log(a) - \log(b)$
Возведение в степень	$\left(x^m\right)^n = x^{mn}$	$\log\left(a^n\right) = n \times \log(a)$
Нулевая степень	$x^0 = 1$	$\log(1) = 0$
Обратное число	$x^{-1} = \frac{1}{x}$	$\log\left(x^{-1}\right) = \log\left(\frac{1}{x}\right) = -\log(x)$

Число *e* и натуральные логарифмы

В математике часто можно встретить особое число, которое называется *e*, или *числом Эйлера*. Оно во многом похоже на число ϖ и равно примерно 2,71828. Число *e* широко используется, потому что оно математически упрощает многие задачи. Мы рассмотрим *e* в контексте степеней и логарифмов.

Число *e*

Еще в школе на уроках математики нам задавали упражнения, в которых фигурировало число *e*. В конце концов я спросил учителя: «Мистер Нове, а все-таки, что такое число *e*? Откуда оно взялось?» Помню, что я оставался разочарован примерами про увеличение популяции кроликов и другие природные явления. Надеюсь, что на страницах этой книги мне удастся дать более удовлетворительное объяснение.

> **ПОЧЕМУ ЧИСЛО *e* ТАК ЧАСТО ИСПОЛЬЗУЕТСЯ**
>
> Одно из свойств числа *e* заключается в том, что его показательная функция является производной от самой себя, что облегчает работу с логарифмическими и показательными функциями. О производных мы поговорим позже в этой главе. Во многих случаях, когда основание

[1] Подразумевается, что в каждой формуле в этом столбце все логарифмы берутся по одному и тому же основанию. — *Примеч. науч. ред.*

степени по сути не имеет значения, мы выбираем то, которое приводит к самой простой производной, и это число *e*. Именно поэтому оно служит стандартным основанием во многих функциях в data science.

Вот как мне нравится объяснять число e. Допустим, вы одолжили кому-то 100 долларов под 20 % годовых. Как правило, проценты начисляются ежемесячно, поэтому процент за каждый месяц составит $\frac{0,20}{12} = 0,01666$. Каков будет остаток по кредиту через два года? Для простоты предположим, что до конца этих двух лет выплаты по кредиту не требуются (и не производятся).

Собрав воедино все, что мы узнали о степенях (или подглядев в учебник по финансам), можно получить формулу для расчета процентов. В нее входит остаток по кредиту *A*, начальный капитал *P*, годовая процентная ставка *r*, срок кредита *t* (количество лет) и периодичность начисления процентов *n* (количество месяцев в каждом году). Вот формула:

$$A = P \times \left(1 + \frac{r}{n}\right)^{nt}$$

Таким образом, если ежемесячно начислять проценты, то за два года сумма кредита вырастет до 148,69 доллара:

$$A = P \times \left(1 + \frac{r}{n}\right)^{nt} = 100 \times \left(1 + \frac{0,20}{12}\right)^{12 \times 2} = 148,6914618$$

Чтобы вычислить то же самое на Python, воспользуемся примером 1.13.

Пример 1.13. Вычисление сложных процентов на Python

```
from math import exp

p = 100
r = .20
t = 2.0
n = 12

a = p * (1 + (r/n))**(n * t)

print(a) # выводит 148.69146179463576
```

Но что будет, если начислять проценты ежедневно? Что произойдет тогда? Изменим *n* на 365:

$$A = P \times \left(1 + \frac{r}{n}\right)^{nt} = 100 \times \left(1 + \frac{0,20}{365}\right)^{365 \times 2} = 149,1661279$$

Ничего себе! Если начислять проценты ежедневно, а не ежемесячно, то через два года мы получим на 47,4666 цента больше. Если уж мы так жадничаем, то почему бы не начислять проценты каждый час, как показано далее? Получится ли в этом случае еще больше? В году 8 760 часов, поэтому установим *n* равным этому значению:

$$A = P \times \left(1 + \frac{r}{n}\right)^{nt} = 100 \times \left(1 + \frac{0{,}20}{8\,760}\right)^{8\,760 \times 2} = 149{,}1817886$$

Итак, мы заработали на процентах почти на 2 цента больше! Но не кажется ли вам, что прибыль уменьшается? Давайте попробуем начислять проценты каждую минуту! В году 525 600 минут, поэтому зададим это значение для *n*:

$$A = P \times \left(1 + \frac{r}{n}\right)^{nt} = 100 \times \left(1 + \frac{0{,}20}{525\,600}\right)^{525\,600 \times 2} = 149{,}1824584$$

Выходит, что чем чаще мы начисляем проценты, тем меньше дополнительная прибыль. Если неограниченно продолжать уменьшать периодичность начисления процентов, пока оно не станет непрерывным, к чему это приведет?

Позвольте представить вам число *e* (число Эйлера), которое приблизительно равно 2,71828. Вот формула для непрерывного начисления процентов:

$$A = P \times e^{rt}$$

Возвращаясь к нашему примеру, рассчитаем остаток по кредиту через два года, если проценты начисляются непрерывно:

$$A = P \times e^{rt} = 100 \times e^{0{,}20 \times 2} = 149{,}1824698$$

Это не слишком удивительно, если учесть, что, когда проценты начислялись ежеминутно, остаток составил 149,1824584. Это очень близко к значению 149,1824698, которое получается при непрерывном начислении.

Обычно в Python, Excel и других программах функцию `exp()` используют с числом e в качестве основания. Скоро вы увидите, что *e* применяется настолько широко, что служит основанием по умолчанию как для степенной, так и для логарифмической функций.

Пример 1.14. Вычисление непрерывных процентов на Python

```
from math import exp

p = 100 # начальный капитал
r = 0.20 # годовая процентная ставка
t = 2.0 # количество лет

a = p * exp(r*t)

print(a) # выводит 149.18246976412703
```

Откуда же берется эта константа *e*? Сравните формулы сложных процентов и непрерывных процентов. Они похожи по структуре, но кое в чем различаются:

$$A = P \times \left(1 + \frac{r}{n}\right)^{nt}$$

$$A = P \times e^{rt}$$

Говоря формальным языком, число *e* — это окончательное значение выражения $\left(1+\frac{1}{n}\right)^n$ по мере того, как *n* становится все больше, приближаясь к бесконечности. Попробуйте поэкспериментировать, увеличивая значения *n*. При этом вы заметите следующее:

$$\left(1+\frac{1}{n}\right)^n$$

$$\left(1+\frac{1}{100}\right)^{100} = 2,70481382942$$

$$\left(1+\frac{1}{1000}\right)^{1000} = 2,71692393224$$

$$\left(1+\frac{1}{10000}\right)^{10000} = 2,71814592682$$

$$\left(1+\frac{1}{10000000}\right)^{10000000} = 2,71828169413$$

Чем больше *n*, тем меньше результат отличается от предыдущего, и он сходится примерно к значению 2,71828, которое и есть число *e*. Как мы увидим далее, оно годится не только для того, чтобы изучать популяции и их рост; это число играет ключевую роль во многих областях математики.

В дальнейшем мы будем использовать число e, чтобы строить нормальные распределения в главе 3 и заниматься логистической регрессией в главе 6.

Натуральные логарифмы

Когда в качестве основания логарифма используется число *e*, такой логарифм называется *натуральным*. В некоторых программных продуктах ему соответствует функция `ln()`, а не `log()`. В частности, вместо того чтобы обозначать натуральный логарифм как $\log_e 10$ (степень, в которую нужно возвести число *e*, чтобы получить 10), можно сокращенно записать его как $\ln 10$:

$$\log_e 10 = \ln 10$$

Однако в Python натуральный логарифм задается функцией `log()`. Как уже говорилось ранее, эта функция по умолчанию использует в качестве основания число *e*. Просто опустите второй аргумент, в котором передается основание, и `log()` будет брать логарифм по основанию *e*, как показано в примере 1.15.

Пример 1.15. Вычисление натурального логарифма от 10 на Python

```
from math import log

# в какую степень нужно возвести e, чтобы получить 10?
x = log(10)

print(x) # выводит 2.302585092994046
```

Число *e* встретится во многих местах этой книги. Не стесняйтесь экспериментировать со степенями и логарифмами, используя Excel, Python, Desmos.com или любую другую вычислительную платформу, которая вам нравится. Постройте графики и посмотрите, как выглядят эти функции.

Пределы

Как мы видели на примере числа *e*, иногда возникает любопытный эффект: мы бесконечно увеличиваем или уменьшаем аргумент, а значение функции все время приближается к какому-то числу, но так и не достигает его. Давайте рассмотрим этот случай формально.

Возьмем функцию, изображенную на рис. 1.5.

$$f(x)=\frac{1}{x}$$

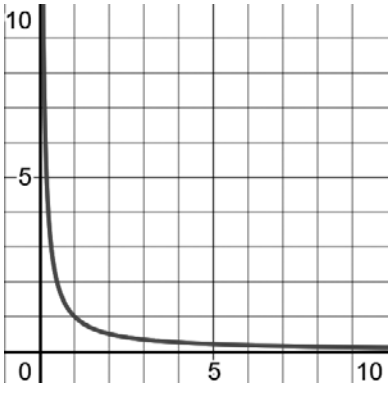


Рис. 1.5. Функция, которая бесконечно приближается к 0, но никогда его не достигает

Мы рассматриваем только положительные значения x. Обратите внимание, что чем больше x, тем ближе значение $f(x)$ к нулю. Как ни странно, $f(x)$ никогда не достигает 0, а только бесконечно приближается к нему.

Чтобы выразить, что функция бесконечно приближается к некоторому значению, но никогда не достигает его, используется предел:

$$\lim_{x\to\infty}\frac{1}{x}=0$$

Это читается так: «При x, стремящемся к бесконечности, функция $\frac{1}{x}$ стремится к 0». Такое поведение функции, при котором она стремится к своему предельному значению, но не достигает его, встречается довольно часто, особенно когда мы углубляемся в производные и интегралы.

С помощью SymPy можно вычислить, к какому значению стремится $f(x)=\frac{1}{x}$, когда x стремится к бесконечности (пример 1.16). Заметим, что символ бесконечности (∞) в SymPy удобно выражается через `oo` (удвоенная латинская буква *o*).

Пример 1.16. Вычисление пределов на Python

```
from sympy import *

x = symbols('x')
f = 1 / x
result = limit(f, x, oo)

print(result) # 0
```

Как вы помните, таким же образом мы нашли число *e*. Оно получается, если неограниченно увеличивать *n* для этой функции:

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n=e=2,71828169413\ldots$$

Забавно, но если вычислять число e с помощью пределов в SymPy (как показано в следующем фрагменте кода), SymPy сразу распознает его как *e*. Чтобы оно отобразилось как число, можно вызвать метод `evalf()`:

```
from sympy import *

n = symbols('n')
f = (1 + (1/n))**n
result = limit(f, n, oo)

print(result) # E
print(result.evalf()) # 2.71828182845905
```

ВОЗМОЖНОСТИ SYMPY

SymPy (https://oreil.ly/mgLyR) — это мощная и фантастическая система компьютерной алгебры для Python, которая использует точные символьные вычисления, а не приближенные расчеты в десятичных дробях. Она полезна в тех ситуациях, когда вы бы решали задачи по математике и математическому анализу на бумаге, но отличается тем, что предлагает знакомый синтаксис языка Python. Вместо того чтобы представлять $\sqrt{2}$ в виде приближенного значения 1,4142135623730951, SymPy сохранит его в формате `sqrt(2)`.

Так почему бы не использовать SymPy для любой математической деятельности? Хотя мы и будем применять эту библиотеку на протяжении всей книги, важно, чтобы с математикой на Python было удобно обращаться, обходясь простыми десятичными числами, потому что так работает scikit-learn и другие библиотеки для data science. Компьютеры обрабатывают десятичные дроби гораздо быстрее, чем символы. SymPy также не всегда справляется, если математические выражения становятся слишком большими. Держите SymPy под рукой, но ни в коем случае не рассказывайте о нем своим детям, которые учатся в школе или университете: ведь эта библиотека может выполнять за них домашние задания по математике.

Производные

Давайте вернемся к функциям и поговорим о них с точки зрения математического анализа. Начнем с производных. *Производная* показывает угол наклона функции, и с ее помощью удобно измерять скорость изменения функции в любой точке.

Почему нас интересуют производные? Они часто используются в машинном обучении и других математических алгоритмах, особенно в методе градиентного спуска. Если угол наклона равен 0, это значит, что мы находимся в минимуме или максимуме функции. Это понятие пригодится нам в дальнейшем, когда мы будем заниматься линейной регрессией (глава 5), логистической регрессией (глава 6) и нейронными сетями (глава 7).

Начнем с простого примера. Рассмотрим функцию $f(x) = x^2$ на рис. 1.6. Насколько круто наклонена ее кривая при $x = 2$?

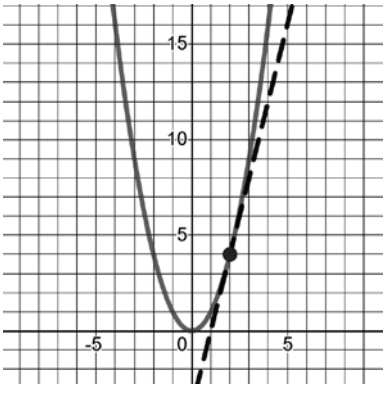


Рис. 1.6. Определение крутизны в заданной точке функции

Обратите внимание, что крутизну можно измерить в любой точке кривой и наглядно представить с помощью касательной. *Касательная* — это прямая, которая «едва касается» кривой в данной точке. Она также показывает уклон функции в этой точке. Чтобы приблизительно определить касательную при заданном значении *x*, можно построить прямую, которая пересекает это значение *x* и *очень близкое* соседнее значение *x* на графике функции.

Возьмем $x = 2$ и соседнее значение $x = 2{,}1$, которые при подстановке в функцию $f(x) = x^2$ дадут $f(2) = 4$ и $f(2{,}1) = 4{,}41$, как показано на рис. 1.7. У результирующей прямой, которая проходит через эти две точки, уклон равен 4,1[1].

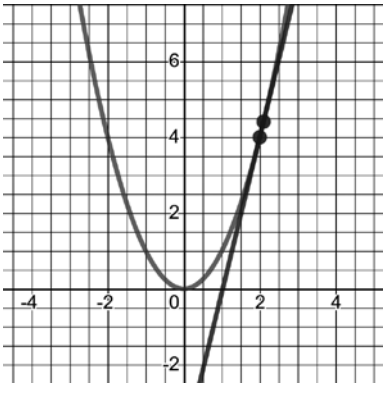


Рис. 1.7. Приблизительный способ расчета уклона

[1] Под *уклоном* здесь подразумевается тангенс угла наклона, или угловой коэффициент касательной в данной точке. — *Примеч. науч. ред.*

Быстро рассчитать уклон *m* между двумя точками можно по простой формуле «подъема на дистанции»:

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{4,41 - 4,0}{2,1 - 2,0} = 4,1$$

Если сделать шаг по *x* между двумя точками еще меньше — например, $x = 2$ и $x = 2,00001$, что даст $f(2) = 4$ и $f(2,00001) = 4,00004$, — то *m* станет *очень* близким к реальному уклону, который равен 4. Таким образом, чем ближе соседнее значение, тем точнее вычисляется уклон в данной точке кривой. Как и в случае со многими другими важными математическими понятиями, мы обнаруживаем нечто примечательное, когда приближаемся к бесконечно большим или бесконечно малым величинам.

В примере 1.17 показано, как можно вычислить производную на Python.

Пример 1.17. Вычисление производной на Python

```
def derivative_x(f, x, step_size):
    m = (f(x + step_size) - f(x)) / ((x + step_size) - x)
    return m

def my_function(x):
    return x**2

slope_at_2 = derivative_x(my_function, 2, .00001)

print(slope_at_2) # выводит 4.000010000000827
```

Хорошая новость заключается в том, что существует более эффективный способ вычислить уклон в любой точке функции. Мы уже использовали SymPy, чтобы строить графики, а теперь я покажу, как с ее помощью можно находить производные, прибегнув к магии символьных вычислений.

Производная от степенной функции типа $f(x) = x^2$ берется так: показатель степени становится коэффициентом при *x*, а прежний показатель уменьшается на 1, в результате чего получается производная $\frac{d}{dx}x^2 = 2x$. $\frac{d}{dx}$ означает *производную по x*, то есть мы строим производную, которая позволяет найти уклон в зависимости от значения *x*. Таким образом, если мы ищем уклон при $x = 2$ и у нас есть производная функции, мы просто подставляем это значение *x*, чтобы получить уклон:

$$f(x) = x^2$$

$$\frac{d}{dx}f(x) = \frac{d}{dx}x^2 = 2x$$

$$\frac{d}{dx}f(2) = 2 \times 2 = 4$$

Если вы хотите глубже изучить правила, по которым можно вычислять производные вручную, то для этого существует множество книг по математическому анализу. Но есть несколько хороших инструментов, которые умеют вычислять производные в символьном виде, — например, библиотека SymPy, которая бесплатна и распространяется с открытым исходным кодом, а также хорошо адаптирована к синтаксису Python. В примере 1.18 показано, как вычислить производную для $f(x) = x^2$ с помощью SymPy.

Пример 1.18. Вычисление производной с помощью SymPy

```
from sympy import *

# Объявляем символ 'x' для SymPy
x = symbols('x')

# Теперь объявляем функцию через обычный синтаксис Python
f = x**2

# Вычисляем производную функции
dx_f = diff(f)
print(dx_f) # выводит 2*x
```

Впечатляет? Итак, если объявить переменную с помощью функции `symbols()` в SymPy, можно объявить функцию от этой переменной посредством обычного синтаксиса Python. После этого можно использовать функцию `diff()`, чтобы вычислить производную. В примере 1.19 мы переносим нашу производную обратно в стандартный Python и просто объявляем ее как еще одну функцию.

Пример 1.19. Вычисление производной на Python

```
def f(x):
    return x**2

def dx_f(x):
    return 2*x

slope_at_2 = dx_f(2.0) # уклон в точке x = 2

print(slope_at_2) # выводит 4.0
```

Если вы собираетесь использовать SymPy дальше, то можете вызвать функцию `subs()`, чтобы заменить переменную *x* на значение 2, как показано в примере 1.20.

Пример 1.20. Использование функции подстановки в SymPy

```
# Вычисляем уклон при x = 2
print(dx_f.subs(x,2)) # выводит 4
```

Частные производные

Еще одно понятие, с которым мы встретимся в этой книге, — *частные производные*, которые будут фигурировать в главах 5, 6 и 7. Это производные от функций, у которых несколько аргументов.

Представьте себе это следующим образом. Вместо того чтобы находить уклон функции на плоскости, мы исследуем уклоны относительно разных переменных в разных направлениях. Для производной по каждой переменной мы предполагаем, что остальные переменные остаются постоянными. Посмотрите на трехмерный график функции $f(x, y) = 2x^3 + 3y^3$ на рис. 1.8, и вы убедитесь, что в каждой его точке существуют уклоны в двух направлениях для двух переменных.

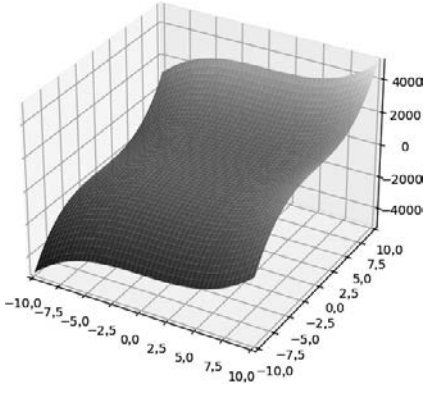


Рис. 1.8. Построение графика трехмерной степенной функции

Возьмем функцию $f(x, y) = 2x^3 + 3y^3$. У нее есть производные по переменным x и y — это $\frac{d}{dx}$ и $\frac{d}{dy}$ соответственно. Они представляют собой величину уклона по каждой переменной на многомерной поверхности. В математике принято называть эти уклоны *градиентами*, когда мы имеем дело с несколькими измерениями. Ниже приведены производные функции по x и y, а также пример кода с использованием SymPy, который их вычисляет:

$$f(x, y) = 2x^3 + 3y^3$$

$$\frac{d}{dx}\left(2x^3 + 3y^3\right) = 6x^2$$

$$\frac{d}{dy}\left(2x^3 + 3y^3\right) = 9y^2$$

Пример 1.21 показывает, как с помощью SymPy вычисляются частные производные по x и y.

Пример 1.21. Вычисление частных производных с помощью SymPy

```
from sympy import *
from sympy.plotting import plot3d

# Объявляем символы x и y в SymPy
x,y = symbols('x y')

# Теперь объявляем функцию через обычный синтаксис Python
f = 2*x**3 + 3*y**3

# Вычисляем частные производные по x и y
dx_f = diff(f, x)
dy_f = diff(f, y)

print(dx_f) # выводит 6*x**2
print(dy_f) # выводит 9*y**2

# Выводим график функции
plot3d(f)
```

Таким образом, для значений $(x, y) = (1,2)$ уклон по x равен $6{\times}1^2 = 6$, а по y — $9{\times}2^2 = 36$.

КАК ВЫЧИСЛЯТЬ ПРОИЗВОДНЫЕ С ПОМОЩЬЮ ПРЕДЕЛОВ

Хотите узнать, какую роль играют пределы при вычислении производных? Если все, что вы изучили до сих пор, не показалось вам чересчур сложным, то смело читайте дальше. Но если вы пока еще перевариваете полученную информацию, то, возможно, стоит вернуться к этой врезке позже.

SymPy позволяет проводить любопытные математические изыскания. Возьмем функцию $f(x) = x^2$; мы приблизительно определили уклон для $x = 2$, проведя прямую через соседнюю точку $x = 2,00001$ с шагом 0,0001. Почему бы не использовать предел, чтобы бесконечно уменьшать шаг s и посмотреть, к какому уклону он стремится?

$$\lim_{s\to 0}\frac{(x+s)^2 - x^2}{(x+s) - x}$$

В нашем примере $x = 2$, поэтому подставим это значение в предыдущее выражение:

$$\lim_{s\to 0}\frac{(2+s)^2 - 2^2}{(2+s) - 2} = 4$$

Если бесконечно приближать значение шага s к нулю, но никогда не достигать его (помните, что соседняя точка прямой не может совпасть с точкой $x = 2$, иначе нельзя будет провести прямую!), можно использовать предел, чтобы убедиться, что уклон сходится к 4, как показано в примере 1.22.

Пример 1.22. Использование пределов для расчета уклона

```
from sympy import *

# Объявляем x и шаг s
x, s = symbols('x s')

# Объявляем функцию
f = x**2

# Вычисляем уклон между двумя точками с шагом s
# Подставляем значения в формулу "подъема на дистанции"
slope_f = (f.subs(x, x + s) - f) / ((x+s) - x)

# Подставляем значение x = 2
slope_2 = slope_f.subs(x, 2)

# Вычисляем уклон при x = 2,
# когда шаг s бесконечно приближается к 0
result = limit(slope_2, s, 0)

print(result) # 4
```

А что, если не присваивать аргументу x конкретного значения? Что произойдет, если в этом случае бесконечно уменьшать размер шага s? Рассмотрим пример 1.23.

Пример 1.23. Использование пределов для вычисления производной

```
from sympy import *

# Объявляем x и шаг s
x, s = symbols('x s')

# Объявляем функцию
f = x**2

# Вычисляем уклон между двумя точками с шагом s
# Подставляем значения в формулу "подъема на дистанции"
slope_f = (f.subs(x, x + s) - f) / ((x+s) - x)

# Вычисляем производную,
# когда шаг s бесконечно приближается к 0
result = limit(slope_f, s, 0)

print(result) # 2*x
```

Таким образом, производная функции равна 2*x. SymPy достаточно сообразителен, чтобы понять, что размер шага не должен достигать нуля, а должен лишь стремиться к нему. Таким образом, из функции $f(x)=x^2$ получается производная $2x$.

Цепное правило

В главе 7, когда мы будем создавать нейронную сеть, нам понадобится специальный математический прием, который называется цепным правилом[1]. Чтобы формировать слои нейронной сети, нам придется выводить производные из каждого слоя. Но пока давайте изучим цепное правило на простом алгебраическом примере. Допустим, даны две функции:

$$y = x^2 + 1$$
$$z = y^3 - 2$$

Обратите внимание, что эти две функции связаны между собой, потому что *y* является выходной переменной в первой функции и аргументом во второй. Это значит, что можно подставить первую функцию *y* во вторую функцию *z* таким образом:

$$z = \left(x^2 + 1\right)^3 - 2$$

Чему же равна производная от *z* по *x*? У нас уже есть подстановка, которая выражает *z* через *x*. Вычислим производную с помощью SymPy, как показано в примере 1.24.

Пример 1.24. Вычисление производной от *z* по *x*

```
from sympy import *

x = symbols('x')

z = (x**2 + 1)**3 - 2
dz_dx = diff(z, x)
print(dz_dx) # 6*x*(x**2 + 1)**2
```

Таким образом, искомая производная от *z* по *x* равна $6x\left(x^2+1\right)^2$

$$\frac{dz}{dx}\left(\left(x^2+1\right)^3-2\right)=6x\left(x^2+1\right)^2$$

Однако давайте теперь попробуем получить то же самое другим способом. Если взять производные функций *y* и *z* по отдельности, а затем перемножить их, то получится производная от *z* по *x*! Попробуем это сделать:

[1] В отечественной литературе этот термин менее неизвестен — у нас принято говорить о правиле дифференцирования сложной функции. — *Примеч. науч. ред.*

$$\frac{dy}{dx}\left(x^2+1\right)=2x$$
$$\frac{dz}{dy}\left(y^3-2\right)=3y^2$$
$$\frac{dz}{dx}=\left(2x\right)\left(3y^2\right)=6xy^2$$

Итак, получилось $6xy^2$. Но это еще не то же самое выражение $6x\left(x^2+1\right)^2$, которое мы получили предыдущим способом. Давайте выразим y через x и подставим это значение, чтобы вся производная $\frac{dz}{dx}$ выражалась только через x без y.

$$\frac{dz}{dx}=6xy^2=6x\left(x^2+1\right)^2$$

Вот мы и получили точно такое же выражение $6x\left(x^2+1\right)^2$ для производной функции!

Это и есть *цепное правило*: если функция y (с аргументом x) входит в другую функцию z (с аргументом y), то чтобы найти производную от z по x, можно перемножить две соответствующие производные:

$$\frac{dz}{dx}=\frac{dz}{dy}\times\frac{dy}{dx}$$

В примере 1.25 показан код на SymPy, который выполняет это преобразование и показывает, что производная, которая вычислена по цепному правилу, равна производной, которая получается при подстановке функции.

Пример 1.25. Вычисление производной $\frac{dz}{dx}$ с использованием цепного правила и без него приводит к одному и тому же результату

```
from sympy import *

x, y = symbols('x y')

# Производная первой функции
# Задаем имя с нижним подчеркиванием, чтобы не было конфликта переменных
_y = x**2 + 1
dy_dx = diff(_y)

# Производная второй функции
z = y**3 - 2
dz_dy = diff(z)

# Вычисляем производную с помощью цепного правила
# и без него, подставляем функцию y
dz_dx_chain = (dy_dx * dz_dy).subs(y, _y)
dz_dx_no_chain = diff(z.subs(y, _y))
```

```
# Цепное правило работает:
# оба варианта дают одинаковый результат
print(dz_dx_chain) # 6*x*(x**2 + 1)**2
print(dz_dx_no_chain) # 6*x*(x**2 + 1)**2
```

Цепное правило — ключевая составляющая обучения нейронной сети с нужными значениями весовых коэффициентов и смещений. Вместо того чтобы по цепочке вычислять производные каждого вложенного узла, можно перемножить производные всех узлов, что с математической точки зрения гораздо проще.

Интегралы

Противоположностью производной является *интеграл*, который определяет площадь под графиком функции на заданном интервале. В главах 2 и 3 мы будем находить площади под распределениями вероятностей. Хотя мы будем использовать не сами интегралы, а кумулятивные функции распределения, которые уже проинтегрированы, полезно знать, как интегралы позволяют находить площади под кривыми. В Приложении А приведены примеры того, как можно применять интегралы к распределениям вероятностей.

Чтобы понять, что такое интеграл, мы рассмотрим интуитивный подход с так называемыми *интегральными суммами* (суммами Римана), который гибко адаптируется к любой непрерывной функции. Прежде всего отметим, что найти площадь области под прямой линией — очень просто. Допустим, что есть функция $f(x) = 2x$, и мы хотим найти площадь под соответствующей линией между точками 0 и 1; на рис. 1.9 эта площадь изображена как закрашенный треугольник.

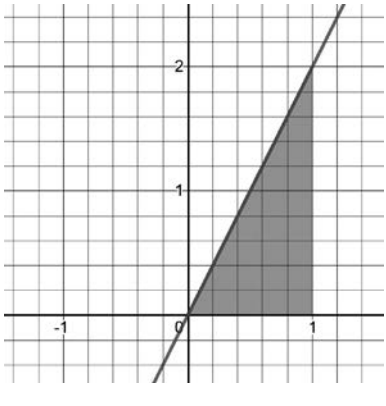


Рис. 1.9. Вычисление площади под графиком линейной функции

Обратите внимание, что мы ищем площадь области, которая ограничена графиком функции и осью x, в интервале от $x = 0,0$ до $x = 1,0$. Если вспомнить основные формулы геометрии, то площадь прямоугольного треугольника равна $A = \frac{1}{2}bh$, где b — длина основания, а h — высота. Визуально можно заметить, что $b = 1$, а $h = 2$. Подставляя эти значения в формулу, мы получаем искомую площадь 1,0:

$$A = \frac{1}{2}bh = \frac{1}{2} \times 1 \times 2 = 1$$

Неплохо получилось, правда? Но давайте рассмотрим функцию, у которой трудно найти площадь под графиком: $f(x) = x^2 + 1$. Чему равна в интервале от 0 до 1 площадь области, которая закрашена на рис. 1.10?

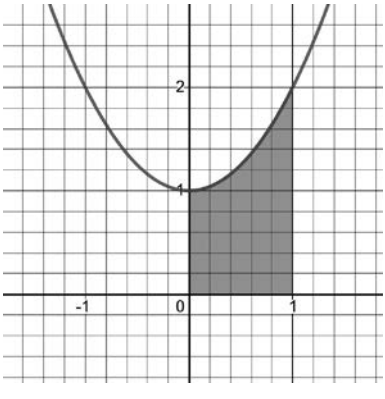


Рис. 1.10. Вычисление площади под нелинейными функциями — непростая задача

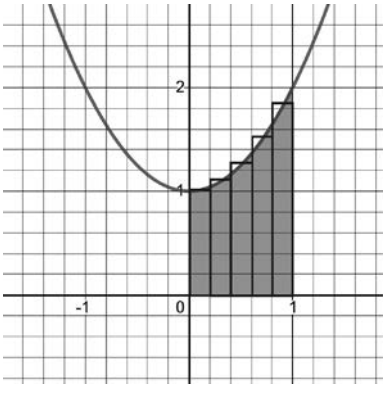


Рис. 1.11. Расположение прямоугольников под кривой для приближенного вычисления площади

Нас снова интересует площадь области между графиком функции и осью x, в интервале от $x = 0$ до $x = 1$. Из-за того, что график криволинейный, здесь нет простой геометрической формулы для площади, но есть один хитрый прием, который поможет ее вычислить.

Что, если расположить под кривой пять прямоугольников одинаковой ширины, как показано на рис. 1.11? Здесь высота каждого прямоугольника — это интервал от оси x до той точки, в которой кривая пересекается с серединой верхней стороны прямоугольника.

Площадь прямоугольника A равна произведению его высоты на ширину, так что можно легко вычислить и просуммировать площади всех прямоугольников. Насколько близким к настоящей площади будет полученное значение? Что, если расположить там 100 прямоугольников? 1 000? 100 000? Если увеличивать количество прямоугольников и уменьшать их ширину, не будем ли мы приближаться к площади под кривой? Да, будем, и это еще один случай, когда мы бесконечно увеличиваем или уменьшаем какую-то величину, чтобы приблизиться к реальному значению.

Давайте попробуем сделать это на Python. Сначала нам понадобится функция, которая приближенно вычисляет интеграл, — мы назовем ее `approximate_integral()`. Аргументы `a` и `b` будут задавать концы интервала — минимальное и максимальное значения x соответственно. `n` — количество прямоугольников под графиком, а `f` — функция, которую мы интегрируем. В примере 1.26 мы реализуем `approximate_integral()`, а затем используем ее, чтобы проинтегрировать функцию $f(x) = x^2 + 1$ по пяти прямоугольникам от 0,0 до 1,0.

Пример 1.26. Приближенное вычисление интеграла на Python

```
def approximate_integral(a, b, n, f):
    delta_x = (b - a) / n  # ширина каждого прямоугольника
    total_sum = 0

    for i in range(1, n + 1):
        midpoint = 0.5 * (2 * a + delta_x * (2 * i - 1))
            # midpoint — координата по x середины верхней стороны прямоугольника
        total_sum += f(midpoint)

    return total_sum * delta_x

def my_function(x):
    return x**2 + 1

area = approximate_integral(a=0, b=1, n=5, f=my_function)
print(area) # выводит 1.33
```

Так мы получаем площадь 1,33. Что произойдет, если взять 1 000 прямоугольников? Попробуем это сделать в примере 1.27.

Пример 1.27. Еще одно приближенное вычисление интеграла на Python

```
area = approximate_integral(a=0, b=1, n=1000, f=my_function)

print(area) # выводит 1.333333250000001
```

Мы получили более точный результат и больше знаков после запятой. А как насчет одного миллиона прямоугольников, как показано в примере 1.28?

Пример 1.28. И еще одно приближенное вычисление интеграла на Python

```
area = approximate_integral(a=0, b=1, n=1_000_000, f=my_function)
print(area) # выводит 1.3333333333332733
```

Видно, что прирост площади с каждым разом уменьшается, и решение, по-видимому, сходится к значению 1,333..., где после запятой идет бесконечное количество троек. Это обозначается 1,(3) и читается «одна целая и три в периоде». Если бы это было рациональное число, то, скорее всего, оно равнялось бы $\frac{4}{3} = 1,(3)$. Чем больше прямоугольников, тем больше непрерывных троек после запятой и тем ближе полученное значение к настоящей площади.

Теперь, когда мы в целом представляем себе, чего мы пытаемся добиться и почему, давайте в примере 1.29 рассмотрим более точный метод с использованием SymPy, которая, оказывается, поддерживает работу с рациональными числами.

Пример 1.29. Интегрирование с помощью SymPy

```
from sympy import *

# Объявляем символ x для SymPy
x = symbols('x')

# Объявляем функцию через обычный синтаксис Python
f = x**2 + 1

# Вычисляем интеграл от функции по x в интервале от x = 0 до x = 1
area = integrate(f, (x, 0, 1))

print(area) # выводит 4/3
```

Отлично! Значит, площадь на самом деле равна значению $\frac{4}{3}$, к которому сходился наш предыдущий метод. К сожалению, стандартный Python (как и многие другие языки программирования) поддерживает только десятичные дроби, но системы компьютерной алгебры, такие как SymPy, позволяют вычислять точные рациональные числа. С помощью интегралов мы будем находить площадь под кривыми в главах 2 и 3, хотя всю работу за нас будет выполнять scikit-learn.

КАК ВЫЧИСЛЯТЬ ИНТЕГРАЛЫ С ПОМОЩЬЮ ПРЕДЕЛОВ

Для самых любопытных я сейчас покажу, как вычислять определенные интегралы с помощью пределов в SymPy. Пожалуйста, если вы еще сомневаетесь, что твердо усвоили весь предыдущий материал, — пропустите эту врезку или вернитесь к ней позже. Но если вы уверены в себе и хотите знать, как вычислять интегралы с помощью пределов, — извольте!

Основная идея во многом повторяет то, чем мы уже занимались: нужно расположить прямоугольники под кривой и бесконечно уменьшать их ширину, пока мы не приблизимся к точной площади. Но, конечно, прямоугольники не могут быть нулевой ширины; ширина должна стремиться к 0, но никогда его не достигать. Это еще один случай, где применяются пределы.

В Академии Хана (Khan Academy) есть отличная статья (https://oreil.ly/sBmCy), которая объясняет, как находить интегральные суммы с помощью пределов, но сейчас мы посмотрим, как это делается в SymPy.

Пример 1.30. Использование пределов для вычисления интегралов

```
from sympy import *

# Объявляем переменные для SymPy
x, i, n = symbols('x i n')

# Объявляем функцию и интервал
f = x**2 + 1
lower, upper = 0, 1

# Вычисляем ширину и высоту каждого прямоугольника с индексом i
delta_x = ((upper - lower) / n)
x_i = (lower + delta_x * i)
fx_i = f.subs(x, x_i)

# Перебираем все n прямоугольников и суммируем их площади
n_rectangles = Sum(delta_x * fx_i, (i, 1, n)).doit()

# Вычисляем площадь,
# устремив число прямоугольников n к бесконечности
area = limit(n_rectangles, n, oo)

print(area) # выводит 4/3
```

Здесь мы определяем ширину каждого прямоугольника `delta_x` и координату начала каждого прямоугольника `x_i`, где i — индекс прямоугольника. `fx_i` — высота прямоугольника с индексом i. Мы объявляем n прямоугольников и суммируем их площади `delta_x * fx_i`, но пока

не получаем конкретного значения площади, потому что мы не задали значение *n*. Вместо этого мы устремляем *n* к бесконечности, чтобы посмотреть, на какой площади сойдется сумма, и получаем ответ $\frac{4}{3}$!

Заключение

В этой главе мы заложили основы для того, чтобы осваивать дальнейший материал. Мы осветили некоторые важные математические понятия, которые относятся к data science, машинному обучению и аналитике, — от теории чисел до логарифмов и интегралов. У вас могут возникнуть вопросы о том, чем так полезны эти понятия. Мы поговорим об этом дальше!

Прежде чем мы начнем разговор о теории вероятностей, уделите немного времени тому, чтобы еще раз просмотреть материал этой главы, а затем выполните упражнения для самопроверки. Читая книгу дальше, вы всегда сможете вернуться к главе 1 и при необходимости освежить ее в памяти, когда начнете применять полученные знания на практике.

Упражнения для самопроверки

1. Является ли число 62,6738 рациональным или иррациональным? Почему?
2. Вычислите значение выражения $10^7 \times 10^{-5}$.
3. Вычислите $81^{1/2}$.
4. Вычислите $25^{3/2}$.
5. Каков будет остаток по кредиту в 1 000 долларов сроком на 3 года под 5 % годовых, если проценты начисляются ежемесячно, а платежей в течение этого периода нет?
6. Выполните предыдущее упражнение при условии, что проценты начисляются непрерывно.
7. Чему равен уклон графика функции $f(x) = 3x^2 + 1$ при $x = 3$?
8. Чему равна площадь под графиком функции $f(x) = 3x^2 + 1$ в интервале от $x = 0$ до $x = 2$?

Ответы — в Приложении Б.

ГЛАВА 2

Теория вероятностей

Что приходит на ум, когда вы задумываетесь о вероятности? Возможно, вы вспоминаете примеры, связанные с азартными играми, — например, вероятность выиграть в лотерею или вероятность того, что на двух игральных костях выпадут шестерки. А может быть, вы представляете себе прогнозирование курса акций, исхода политических выборов или того, прилетит ли ваш самолет вовремя. Наш мир полон неопределенностей, которые нам важно оценить.

Возможно, именно на этом слове следует сосредоточиться: неопределенность. Как можно измерить что-то, в чем мы не уверены?

В конечном счете теория вероятностей — это теоретическая дисциплина, которая изучает, как измерять степень уверенности в том, что событие произойдет. На эту дисциплину опирается статистика, проверка статистических гипотез, машинное обучение и другие темы этой книги. Многие воспринимают вероятность как нечто само собой разумеющееся и считают, что понимают ее. Однако это более многогранная и сложная тема, чем думает большинство людей. Хотя теоремы и понятия теории вероятностей математически обоснованы, с ними начинаются затруднения, когда мы привлекаем данные и переходим к статистике. Мы поговорим об этом в главе 4, которая посвящена статистике и проверке статистических гипотез.

В этой главе мы обсудим, что такое вероятность. Затем мы рассмотрим математические понятия теории вероятностей, формулу Байеса, биномиальное распределение и бета-распределение.

Что такое вероятность

Вероятность — это степень уверенности в том, что событие произойдет, часто выражаемая в процентах. Вот некоторые примеры вопросов, ответ на которые можно оценить как вероятность.

- Какова вероятность того, что, если подбросить монету 10 раз, 7 раз выпадет орел?
- Каковы мои шансы победить на выборах?
- Опоздает или нет мой рейс?
- Насколько я уверен в том, что товар бракованный?

Наиболее распространенный способ выразить вероятность — в процентах, например «Вероятность того, что рейс опоздает, составляет 70 %». Мы будем обозначать эту вероятность $P(X)$, где X — интересующее нас событие. Однако, работая с вероятностью, вы чаще всего будете встречать ее в виде десятичной дроби, которая лежит в интервале между 0 и 1 (в данном случае это дробь 0,7):

$$P(X) = 0,70$$

Правдоподобие похоже на вероятность, и два этих понятия легко спутать (так поступают многие словари). В повседневном общении «вероятность» и «правдоподобие» можно смело использовать как синонимы. Однако следует понимать, чем они различаются. Вероятность количественно оценивает предсказания событий, которые еще не произошли, тогда как правдоподобие измеряет частоту уже произошедших событий. В статистике и машинном обучении мы часто используем правдоподобие (данные о прошлом), чтобы прогнозировать вероятность (данные о будущем).

Важно отметить, что вероятность наступления события должна находиться строго между 0 и 100 %, или между 0 и 1. Логика подсказывает, что вероятность того, что событие не произойдет, можно вычислить, если вычесть его вероятность из 1:

$$P(X) = 0,70$$
$$P(\text{не } X) = 1 - 0,70 = 0,30$$

Есть еще одно различие между вероятностью и правдоподобием. Вероятности всех возможных взаимоисключающих исходов события (имеется в виду, что возможен только один, а не несколько исходов) должны в сумме составлять 1, или 100 %. Однако правдоподобие не подчиняется этому правилу.

В других случаях вероятность может быть выражена в виде *отношения шансов* $O(X)$, например $7:3$, $\frac{7}{3}$ или $2,(3)$.

Чтобы преобразовать отношение шансов $O(X)$ в вероятность $P(X)$, воспользуйтесь следующей формулой:

$$P(X) = \frac{O(X)}{1+O(X)} = \frac{\frac{7}{3}}{1+\frac{7}{3}} = 0,7$$

ШАНСЫ БЫВАЮТ ПОЛЕЗНЫ!

Хотя многим удобнее выражать вероятности в виде процентов или пропорций, отношение шансов — тоже полезный инструмент. Если оно равно 2, это означает, что наступление некоторого события в два раза вероятнее, чем то, что оно не наступит. Иногда это помогает лучше описать уверенность в исходе события, чем процентное соотношение 66,(6) %. Поэтому шансы полезны, чтобы количественно оценить субъективную уверенность, особенно в контексте азартных игр или ставок. Отношение шансов играет важную роль в байесовской статистике (в том числе в вычислении коэффициента Байеса), а также в логистической регрессии, которую мы рассмотрим в главе 6.

Теория вероятностей и статистика

Иногда люди используют термины *теория вероятностей* и *статистика* как взаимозаменяемые. Неудивительно, что эти дисциплины часто путают, однако они все же различаются. *Теория вероятностей* чисто теоретически оценивает, насколько вероятно наступление того или иного события, и не требует привлекать дополнительные данные. *Статистика* же, напротив, не может существовать без данных и использует их, чтобы выявить вероятности, а также предоставляет инструменты для того, чтобы описывать данные.

Давайте подумаем, как предсказать, что при броске одной игральной кости выпадет 4. Если подойти с чисто вероятностной точки зрения, то можно просто сказать, что у кости шесть граней. Мы предполагаем, что каждая из них равновероятна, поэтому вероятность того, что выпадет 4, равна $\frac{1}{6}$, или 16,666 %.

Однако рьяный специалист по статистике может возразить: «Нет! Нам нужно бросить кость несколько раз, чтобы собрать больше данных. Если мы сделаем хотя бы 30 бросков, а лучше — еще больше, только тогда у нас появятся данные, чтобы определить вероятность того, что выпадет 4». Такой подход может показаться неразумным, если мы предполагаем, что кость сделана без дефектов, но что если это не так? В этом случае собрать данные — единственный способ узнать вероятность того, что выпадет 4. О проверке гипотез мы поговорим в главе 3.

Математика вероятностей

Когда мы имеем дело с вероятностью одного-единственного события $P(X)$ — она еще называется *безусловной вероятностью*, — ситуация выглядит довольно просто, как уже говорилось ранее. Но когда мы начинаем комбинировать вероятности разных событий, все становится не таким очевидным.

Вероятность пересечения событий

Допустим, у вас есть симметричная монета и «честная» игральная кость. Необходимо найти вероятность того, что на монете выпадет орел, а на кости — шестерка. Здесь фигурируют две отдельные вероятности двух отдельных событий, но мы хотим найти вероятность того, что оба события произойдут вместе. Это называется *вероятностью пересечения событий*, или *совместной вероятностью*.

Представьте вероятность пересечения событий как логический оператор И. Мы хотим найти вероятность того, что выпадет орел И одновременно выпадет 6. Необходимо, чтобы оба события наступили вместе, — так как же вычислить эту вероятность?

У монеты две стороны, а у кости шесть граней, поэтому вероятность того, что выпадет орел, равна $\frac{1}{2}$, а вероятность того, что выпадет шестерка, — $\frac{1}{6}$. Чтобы найти вероятность того, что произойдут оба события (при условии, что они независимы, — подробнее об этом позже!), можно просто перемножить эти две вероятности:

$$P(A \text{ И } B) = P(A) \times P(B)$$

$$P(\text{орел}) = \frac{1}{2}$$

$$P(6) = \frac{1}{6}$$

$$P(\text{орел И } 6) = \frac{1}{2} \times \frac{1}{6} = \frac{1}{12} = 0,08(3)$$

Все достаточно просто, но почему вероятность вычисляется именно так? Многие правила теории вероятностей можно получить, если перебрать все возможные комбинации событий. (Этим занимается комбинаторика — область дискретной математики.) Для нашего случая перечислим все возможные исходы броска монеты и кубика, совместив орлов (О) и решки (Р) с числами от 1 до 6. Жирным шрифтом выделен интересующий нас исход, где мы получаем одновременно орла и 6:

О1 О2 О3 О4 О5 **О6** Р1 Р2 Р3 Р4 Р5 Р6

Обратите внимание, что, если бросать монету и кубик, возможны 12 исходов. Нас интересует только исход «О6» — выпадение орла и шестерки. Поскольку нашему условию удовлетворяет только один исход, а всего их возможно 12, то вероятность выпадения орла и шестерки равна $\frac{1}{12}$.

Вместо того чтобы перебирать все возможные комбинации и подсчитывать те, которые нас интересуют, можно быстро находить совместную вероятность с помощью умножения. Это правило известно как *правило умножения вероятностей*:

$$P(A \text{ И } B) = P(A) \times P(B)$$

$$P(\text{орел И } 6) = \frac{1}{2} \times \frac{1}{6} = \frac{1}{12} = 0,08(3)$$

Вероятность объединения событий

Мы обсудили вероятность пересечения событий, то есть вероятность того, что одновременно наступят два или более событий. А как быть с вероятностью того, что наступит событие *A* или *B*? Когда мы рассматриваем операцию ИЛИ в контексте вероятностей, это называется *вероятностью объединения событий*.

Начнем с *несовместных* событий — таких, которые не могут произойти одновременно. Например, если бросать кость, на ней не могут одновременно выпасть 4 и 6. Может получиться только один исход. Найти вероятность объединения событий для таких случаев очень просто: достаточно всего лишь сложить вероятности отдельных событий. Если нас интересует вероятность того, что при бросании кости выпадет 4 или 6, то она будет равна $\frac{2}{6} = \frac{1}{3}$:

$$P(4) = \frac{1}{6}$$

$$P(6) = \frac{1}{6}$$

$$P(4 \text{ ИЛИ} 6) = \frac{1}{6} + \frac{1}{6} = \frac{1}{3}$$

Но как быть с произвольными событиями, которые могут произойти одновременно? Вернемся к примеру с броском монеты и кости. Какова вероятность того, что выпадет орел ИЛИ 6? Прежде чем поддаться искушению сложить вероятности этих событий, давайте еще раз перечислим все возможные исходы и выделим те, которые нас интересуют:

01 02 03 04 05 06 P1 P2 P3 P4 P5 P6

В данном случае нас интересуют как все исходы, в которых выпал орел, так и все исходы, в которых выпала 6. Этому условию удовлетворяют 7 из 12 возможных исходов, а значит, правильная вероятность объединения равна $\frac{7}{12} = 0,58(3)$.

Но что произойдет, если просто сложить вероятности выпадения орла и шестерки? Мы получим другой (неправильный!) ответ 0,(6):

$$P(\text{орел}) = \frac{1}{2}$$

$$P(6) = \frac{1}{6}$$

$$P(\text{орел ИЛИ } 6) = \frac{1}{2} + \frac{1}{6} = \frac{4}{6} = 0,(6)$$

Почему так? Изучите еще раз все комбинации исходов бросков монеты и кости и посмотрите, нет ли в них чего-то подозрительного. Вы обнаружите, что, когда мы складывали вероятности, мы дважды учли вероятность того, что выпадет шестерка, — и в «О6», и в «Р6»! Если это не совсем очевидно, попробуйте аналогичным способом найти вероятность того, что на монете выпадет орел или на кости — любое число от 1 до 5:

$$P(\text{орел}) = \frac{1}{2}$$

$$P(\text{от 1 до 5}) = \frac{5}{6}$$

$$P(\text{орел ИЛИ от 1 до 5}) = \frac{1}{2} + \frac{5}{6} = \frac{8}{6} = 1,(3)$$

Получилась вероятность 133,(3) %, что, очевидно, неверно, потому что вероятность не может быть больше 100 %, или 1. Проблема опять в том, что мы дважды учли некоторые исходы.

Если поразмыслить над этим чуть дольше, то можно найти логичный способ устранить эту ошибку: вычисляя вероятность объединения событий, вычесть из нее вероятность пересечения. Это правило известно как *правило сложения вероятностей*; оно обеспечивает, чтобы каждое событие учитывалось только один раз:

$$P(A \text{ ИЛИ } B) = P(A) + P(B) - P(A \text{ И } B) = P(A) + P(B) - P(A) \times P(B)$$

Таким образом, если вернуться к нашему примеру с вероятностью выпадения орла или шестерки, необходимо из суммы вероятностей отдельных событий вычесть совместную вероятность того, что выпадет одновременно орел и шестерка:

$$P(\text{орел}) = \frac{1}{2}$$

$$P(6) = \frac{1}{6}$$

$$P(A\ ИЛИ\ B) = P(A) + P(B) - P(A) \times P(B)$$

$$P(\text{орел ИЛИ 6}) = \frac{1}{2} + \frac{1}{6} - \left(\frac{1}{2} \times \frac{1}{6}\right) = \frac{7}{12} = 0,58(3)$$

Заметим, что эта формула применима и к несовместным событиям. Если события являются несовместными и возможен либо исход *A*, либо *B*, но не оба, то вероятность пересечения событий $P(A\ И\ B)$ будет равна 0, а значит, ее можно исключить из формулы. Остается просто сложить вероятности событий, как мы делали до этого.

Таким образом, когда вы вычисляете вероятность объединения двух событий, которые не являются несовместными, обязательно вычитайте совместную вероятность этих событий, чтобы не учитывать некоторые вероятности дважды.

Условная вероятность и формула Байеса

Одно из понятий теории вероятностей, которое легко сбивает людей с толку, — это *условная вероятность*, то есть вероятность того, что наступит событие *A*, при условии, что наступило событие *B*. Обычно она обозначается $P(A \mid B)$.

Допустим, некоторое исследование утверждает, что 85 % людей, больных раком, пили кофе. Как вы на это отреагируете? Не настораживает ли вас этот факт? Не вызывает ли он желания отказаться от любимого утреннего напитка? Давайте сначала определим этот показатель как условную вероятность $P(\text{кофе} \mid \text{рак})$.

Она представляет собой вероятность того, что человек пьет кофе, при условии, что он страдает от рака.

Для примера сравним долю людей с онкологическим диагнозом в США (0,5 % по данным cancer.gov) и долю людей, которые пьют кофе (65 % по данным statista.com):

$$P(\text{кофе}) = 0,65$$

$$P(\text{рак}) = 0,005$$

$$P(\text{кофе} \mid \text{рак}) = 0,85$$

Хм-м... Внимательно изучите эти цифры и задайтесь вопросом, действительно ли проблема заключается в кофе. Обратите внимание, что только 0,5 % населения в определенный момент времени болеют раком. Однако 65 % населения регулярно пьют кофе. Если кофе способствует возникновению рака, то не должны ли наблюдаться гораздо более высокие показатели заболеваемости, чем 0,5 %? Не должна ли доля больных быть ближе к 65 %?

В этом и состоит коварство процентных соотношений. Они могут казаться существенными без какого-либо контекста, и журналисты этим пользуются, чтобы привлечь внимание. «Новое исследование показало, что 85 % больных раком пьют кофе», — гласит заголовок. Конечно, это недобросовестно, потому что журналист взял широко распространенный признак (употребление кофе) и связал его с мало распространенным признаком (заболевание раком).

В условных вероятностях легко запутаться, потому что важно, какое из событий является условием, — а многие ошибочно рассматривают оба события как в чем-то равнозначные. «Вероятность заболеть раком, если вы пьете кофе» — это не то же самое, что «вероятность быть любителем кофе, если у вас рак». Проще говоря, мало кто из кофеманов болеет раком, но многие больные раком пьют кофе.

Если мы хотим изучить, влияет ли кофе на развитие рака, то на самом деле нужно рассмотреть первую из этих условных вероятностей: вероятность того, что кто-то болеет раком, при условии, что он пьет кофе.

$$P(\text{кофе} \mid \text{рак}) = 0,85$$
$$P(\text{рак} \mid \text{кофе}) = ?$$

Как перевернуть условие? Существует простая, но мощная формула, которая называется формулой Байеса, и с ее помощью можно переворачивать условные вероятности:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Если подставить в эту формулу данные, которые у нас есть, можно найти вероятность того, что человек болен раком, при условии, что он пьет кофе:

$$P(\text{рак} \mid \text{кофе}) = \frac{P(\text{кофе} \mid \text{рак}) \times P(\text{рак})}{P(\text{кофе})} = \frac{0,85 \times 0,005}{0,65} = 0,0065$$

В примере 2.1 представлен код на Python, который вычисляет то же самое:

Пример 2.1. Использование формулы Байеса на Python

```
# вероятность того, что человек пьет кофе
p_coffee_drinker = .65

# вероятность того, что человек болен раком
p_cancer = .005

# вероятность того, что человек пьет кофе,
# при условии, что он болен раком
p_coffee_drinker_given_cancer = .85
```

```
# вероятность того, что человек болен раком,
# при условии, что он пьет кофе
p_cancer_given_coffee_drinker = \
    p_coffee_drinker_given_cancer * p_cancer / p_coffee_drinker

# выводит 0.006538461538461539
print(p_cancer_given_coffee_drinker)
```

Таким образом, вероятность того, что человек болен раком, если он пьет кофе, составляет всего 0,65 %! Эта величина сильно отличается от 85 %-ной вероятности того, что человек, который пьет кофе, болен раком. Теперь вы понимаете, почему важно, какое из событий является условием? Именно поэтому формула Байеса так полезна. С ее помощью также можно составлять цепочки из нескольких условных вероятностей, чтобы постоянно обновлять наши ожидания, опираясь на новую информацию.

КОГО МОЖНО НАЗВАТЬ «ЛЮБИТЕЛЕМ КОФЕ»?

Отмечу, что здесь можно было учесть и другие переменные, в частности те, которые определяют человека как «любителя кофе». Если один человек пьет кофе раз в месяц, а другой — каждый день, стоит ли считать обоих «любителями кофе»? Стоит ли одинаково учитывать тех, кто начал пить кофе месяц назад, и тех, кто пьет его уже 20 лет? Как часто и как долго нужно пить кофе, чтобы приобрести статус «любителя кофе» в этом исследовании онкологических заболеваний?

Это важные вопросы, над которыми стоит задуматься, и они показывают, почему данные редко раскрывают полную картину. Если кто-то предъявит вам электронную таблицу пациентов с простым признаком «ДА/НЕТ» в графе о том, пьют ли они кофе, требуйте указать точные критерии! Возможно, понадобится более весомая оценка, например «количество чашек кофе, выпитых за последние три года». Я упростил свой пример и не стал уточнять, по каким конкретно признакам квалифицируются «любители кофе», но имейте в виду, что в реальной работе всегда полезно потянуть за ниточки и распутать клубок данных. Подробнее об этом мы поговорим в главе 3.

Если вы хотите глубже узнать, как работает формула Байеса, обратитесь к Приложению А. Пока же достаточно запомнить, что она помогает перевернуть условную вероятность. Далее мы поговорим о том, как условная вероятность сочетается с вероятностью пересечения и объединения событий.

Наивный байесовский алгоритм

Формула Байеса играет центральную роль в популярном алгоритме машинного обучения, который называется наивным байесовским алгоритмом (Naive Bayes). Джоэл Грус рассказывает об этом в своей книге «Data Science. From Scratch».

Условная вероятность пересечения и объединения событий

Давайте вернемся к вероятности пересечения событий и посмотрим, как она сочетается с условной вероятностью. Допустим, мы хотим найти вероятность того, что кто-то пьет кофе И болен раком. Нужно ли перемножить вероятности $P(\text{кофе})$ и $P(\text{рак})$? Или вместо $P(\text{кофе})$ использовать $P(\text{кофе} \mid \text{рак})$, если эта вероятность известна? Что из этого выбрать?

Вариант 1:

$$P(\text{кофе}) \times P(\text{рак}) = 0,65 \times 0,005 = 0,00325$$

Вариант 2:

$$P(\text{кофе} \mid \text{рак}) \times P(\text{рак}) = 0,85 \times 0,005 = 0,00425$$

Если мы уже установили, что искомая вероятность относится только к больным раком, то не имеет ли смысл использовать $P(\text{кофе} \mid \text{рак})$ вместо $P(\text{кофе})$? Вариант с условной вероятностью более избирательный, и в нем фигурирует условие, которое уже выполнено. Итак, здесь нужно использовать $P(\text{кофе} \mid \text{рак})$, потому что $P(\text{кофе})$ и так входит в совместную вероятность. Таким образом, вероятность того, что кто-то болен раком и любит кофе, равна 0,425 %:

$$P(\text{кофе И рак}) = P(\text{кофе} \mid \text{рак}) \times P(\text{рак}) = 0,85 \times 0,005 = 0,00425$$

Эту совместную вероятность можно вычислить и в противоположном направлении. Чтобы найти вероятность того, что кто-то пьет кофе и болен раком, можно перемножить $P(\text{рак} \mid \text{кофе})$ и $P(\text{кофе})$. Как нетрудно заметить, получится тот же результат:

$$P(\text{рак И кофе}) = P(\text{рак} \mid \text{кофе}) \times P(\text{кофе}) = 0,0065 \times 0,65 = 0,00425$$

А если у нас нет никаких условных вероятностей, то лучшим вариантом будет перемножить вероятности $P(\text{кофе})$ и $P(\text{рак})$, как показано ниже:

$$P(\text{кофе}) \times P(\text{рак}) = 0,65 \times 0,005 = 0,00325$$

Теперь подумайте вот о чем: если событие A никак не влияет на событие B, то что можно сказать об условной вероятности $P(B \mid A)$? В этом случае $P(B \mid A) = P(B)$, то есть вероятность того, что наступит событие B, никак не зависит от того, наступит ли событие A. Поэтому формулу совместной вероятности можно перефразировать так (при этом неважно, зависит ли одно событие от другого):

$$P(A \text{ И } B) = P(B) \times P(A \mid B)$$

И наконец, поговорим об условной вероятности при объединении событий. Если нужно вычислить вероятность того, что произойдет событие A или B, но при этом A может повлиять на вероятность B, то правило сложения меняется так:

$$P(A \text{ ИЛИ } B) = P(A) + P(B) - P(A \mid B) \times P(B)$$

Обратите внимание, что эта формула верна и для несовместных событий. Если события A и B не могут наступить одновременно, то произведение $P(A \mid B) \times P(B)$ обратится в 0.

Биномиальное распределение

В оставшейся части главы мы изучим два распределения вероятностей: биномиальное и бета-распределение. Хотя мы не будем использовать их далее в книге, они сами по себе служат полезными инструментами и принципиально важны для того, чтобы выяснить, как наступают события при определенном количестве испытаний. Кроме того, если освоить эти распределения, то дальше будет легче изучать другие распределения вероятностей, которые мы будем активно использовать в главе 3. Давайте рассмотрим пример, который может встретиться в реальном мире.

Допустим, вы разрабатываете новый турбореактивный двигатель и провели 10 испытаний, в результате получив восемь успешных исходов и два неудачных:

✔ ✔ ✔ ✔ ✔ ✘ ✔ ✘ ✔ ✔

Вы надеялись получить 90 % успешных испытаний, но на основании полученных данных пришли к выводу, что испытания провалились, ведь только 80 % из них оказались успешными. Каждое испытание отнимает много времени и средств, поэтому вы решили, что пора вернуться к чертежной доске и перепроектировать конструкцию.

Однако одна из ваших инженеров настаивает, что необходимо провести дополнительные испытания. «Единственный способ узнать наверняка — это провести

больше испытаний, — утверждает она. — А что, если при большем количестве испытаний окажется, что не менее 90 % из них будут успешными? В конце концов, если подбросить монету 10 раз и получить 8 орлов, это не значит, что монета “настроена” на 80 % орлов».

Недолго думая, вы соглашаетесь с доводами инженера. Даже если честно подбрасывать монету, не всегда будет выпадать одинаковое количество орлов и решек, особенно когда ее подбрасывают всего 10 раз. Скорее всего, выпадет пять орлов, но может выпасть также три, четыре, шесть или семь. Может выпасть даже 10 орлов, хотя это крайне маловероятно. Как же оценить правдоподобность того, что 80 % испытаний завершились успешно, притом что настоящая вероятность успеха равна 90 %?

Одним из инструментов, который может здесь пригодиться, является *биномиальное распределение*. Оно позволяет оценить правдоподобность того, что в серии из n испытаний, вероятность успеха в каждом из которых равна p, может произойти всего k успехов.

Визуально биномиальное распределение выглядит так, как показано на рис. 2.1.

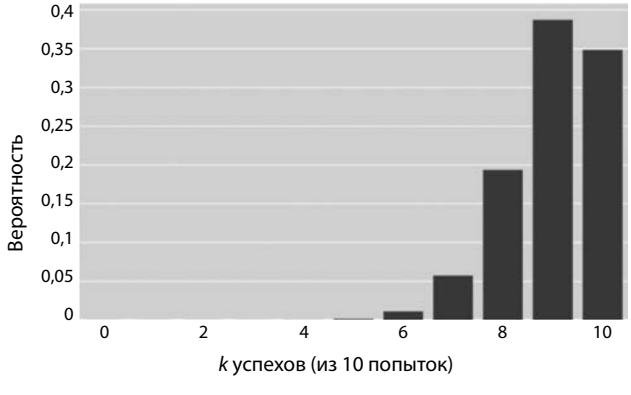


Рис. 2.1. Биномиальное распределение

Здесь для каждого значения k от 0 до 10 показана вероятность того, что k из 10 испытаний будут успешными. Это биномиальное распределение предполагает, что вероятность успеха каждого отдельного испытания p равна 90 % (или 0,9). Если это так, то вероятность получить 8 успехов из 10 испытаний равна 0,1937. Вероятность получить 1 успех из 10 испытаний крайне мала: она равна 0,000000008999, поэтому соответствующий столбик даже не виден.

Чтобы вычислить вероятность восьми или менее успехов, можно сложить значения всех столбиков до восьмого включительно. В результате получится 0,2639.

Как реализовать биномиальное распределение? Можно сделать это с нуля (как показано в Приложении А), а можно воспользоваться библиотеками вроде SciPy. В примере 2.2 показано, как вывести все 11 вероятностей (от 0 до 10 успешных исходов) для нашего биномиального распределения с помощью функции `binom.pmf()` из библиотеки SciPy. PMF (probability mass function) — это математическая *функция вероятности*.

Пример 2.2. Использование библиотеки SciPy для биномиального распределения

```
from scipy.stats import binom

n = 10 # количество испытаний
p = 0.9 # вероятность успеха в каждом испытании

for k in range(n + 1): # количество успехов
    probability = binom.pmf(k, n, p)
    print(f"{k} - {probability}")

# Вывод:
# 0 - 9.999999999999977e-11
# 1 - 8.999999999999976e-09
# 2 - 3.6449999999999933e-07
# 3 - 8.747999999999988e-06
# 4 - 0.00013778099999999974
# 5 - 0.0014880347999999982
# 6 - 0.011160260999999989
# 7 - 0.05739562799999997
# 8 - 0.1937102444999998
# 9 - 0.38742048899999976
# 10 - 0.34867844010000015
```

Как видите, мы задаем n — количество испытаний, p — вероятность успеха для каждого испытания и k — количество успехов, вероятность которого мы хотим найти. Мы перебираем все значения k и для каждого из них вычисляем вероятность того, что мы получим k успехов. Как видно из вывода, наиболее вероятное количество успехов равно девяти.

Но если сложить вероятности того, что произойдет от 0 до 8 успехов, то получится 0,2639. То есть существует вероятность 26,39 % того, что произойдет восемь или меньше успехов, даже если вероятность успеха каждого испытания составляет 90 %. Так что не исключено, что инженер была права: вероятность 26,39 % — не мелочь и вполне возможна.

Однако в этой модели мы сделали одно допущение, которое рассмотрим далее на примере бета-распределения.

Биномиальное распределение с нуля

Обратитесь к Приложению А, чтобы узнать, как построить биномиальное распределение с нуля, не используя библиотеку SciPy.

Бета-распределение

Какое допущение мы сделали, когда моделировали испытания двигателя с помощью биномиального распределения? Есть ли параметр, который мы приняли за истинный и на котором мы построили всю модель? Подумайте хорошенько и читайте дальше.

Слабое место нашего биномиального распределения в том, что мы *предположили*, будто вероятность успеха каждого отдельного испытания составляет 90 %. Это не значит, что наша модель бесполезна. Я просто показал, что если эта базовая вероятность равна 90 %, то вероятность получить 8 или меньше успехов из 10 испытаний составляет 26,39 %. Так что инженер определенно не ошибается в том, что базовая вероятность может составлять 90 %.

Но давайте перевернем вопрос на 180°: а что, если существуют другие значения базовой вероятности, помимо 90 %, которые дают 8 успехов из 10 испытаний? Можно ли получить такой результат, если вероятность успеха каждого испытания равна 80 %? 70 %? 30 %? Зафиксировав условие «8 успехов из 10 попыток», можно ли исследовать вероятности вероятностей?

Вместо того чтобы в поисках ответа на этот вопрос плодить несметное количество биномиальных распределений, воспользуемся одним инструментом. *Бета-распределение* позволяет оценить правдоподобность того, что при a успехов и b неудач базовая вероятность успеха равна тому или иному значению.

График бета-распределения при восьми успехах и двух неудачах представлен на рис. 2.2.

Бета-распределение в Desmos

Если вам интересно поэкспериментировать с бета-распределением, можно воспользоваться графическим калькулятором Desmos (https://oreil.ly/pN4Ep).

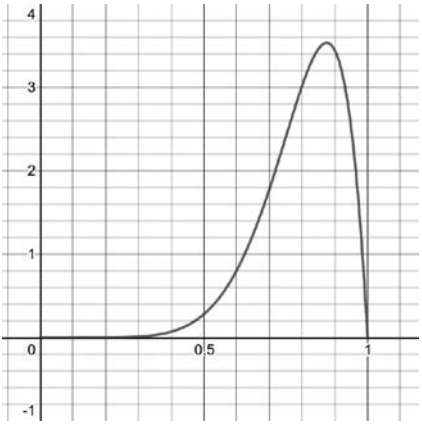


Рис. 2.2. Бета-распределение

Обратите внимание, что ось *x* отражает все базовые вероятности успеха от 0,0 до 1,0 (то есть от 0 до 100 %), а ось *y* — правдоподобность этой вероятности при восьми успехах и двух неудачах. Другими словами, бета-распределение позволяет увидеть вероятность вероятностей при 8 успехах из 10 попыток. Эту величину можно считать метавероятностью, так что не поленитесь потратить время, чтобы понять концепцию!

Вы наверняка заметили, что бета-распределение — непрерывная функция, то есть ее график — непрерывная кривая из дробных значений (в отличие от четких дискретных целых чисел в биномиальном распределении). Это несколько усложняет математические вычисления, потому что значения по оси *y* — не вероятность, а плотность вероятности. Чтобы узнать ту или иную вероятность, нужно вычислить соответствующую площадь под графиком.

Бета-распределение — это один из видов *распределения вероятностей*. Это значит, что площадь под всем графиком равна 1, или 100 %. Чтобы найти вероятность, которая нас интересует, нужно вычислить площадь под кривой в определенном интервале. Например, если мы ищем вероятность того, что 8 успехов из 10 попыток произойдут при базовой вероятности успеха 90 % или более, нам нужно вычислить площадь области между 0,9 и 1, которая равна 0,225, как показано на рис. 2.3.

Как и в случае с биномиальным распределением, бета-распределение можно построить с помощью библиотеки SciPy. У каждого непрерывного распределения вероятностей есть *функция распределения* (*CDF*), которая дает площадь под графиком до заданного значения x. Допустим, нам нужно вычислить площадь в интервале до 90 % (от 0 до 0,9), которая закрашена на рис. 2.4.

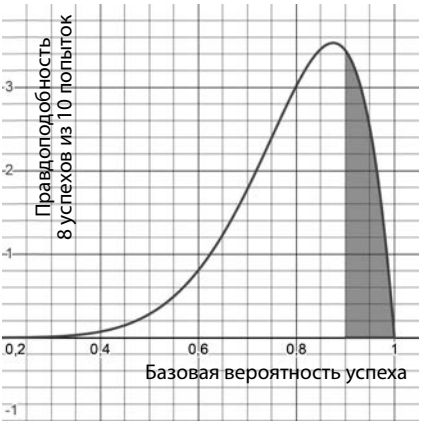


Рис. 2.3. Площадь под графиком в интервале от 90 до 100 % соответствует базовой вероятности успеха 22,5 %

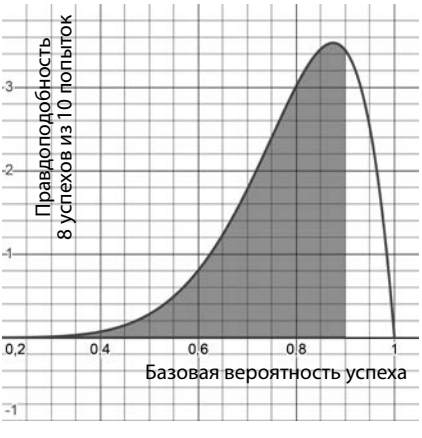


Рис. 2.4. Вычисление площади под графиком в интервале от 0 до 90 % (от 0 до 0,9)

Для этого достаточно воспользоваться SciPy с ее функцией `beta.cdf()`. Все параметры, которые надо указать, — конец интервала x, количество успехов a и количество неудач b, как показано в примере 2.3.

Пример 2.3. Вычисление бета-распределения с помощью SciPy

```
from scipy.stats import beta

a = 8
b = 2
```

```
p = beta.cdf(.90, a, b)

# 0.7748409780000002
print(p)
```

Таким образом, наши расчеты показывают, что с вероятностью 77,48 % базовая вероятность успеха составляет 90 % или менее.

А как вычислить вероятность того, что базовая вероятность успеха равна 90 % или более, как показано на рис. 2.5?

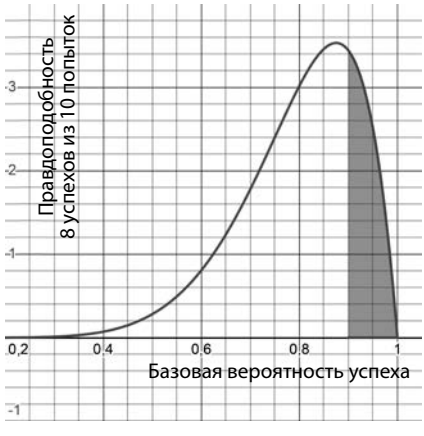


Рис. 2.5. Вероятность успеха, равная 90 % или более

Функция распределения (CDF) определяет площадь только слева от граничного значения, а не справа. Если вспомнить правила теории вероятностей, то одно из них гласит, что у распределения вероятностей общая площадь под кривой равна 1. Если нужно найти вероятность события с другой стороны от граничного значения (выше 0,9, а не ниже), просто вычтите из 1 вероятность того, что базовая вероятность успеха меньше 0,9, и оставшаяся величина будет вероятностью того, что базовая вероятность успеха больше 0,9. На рис. 2.6 показано, как выполняется это вычитание.

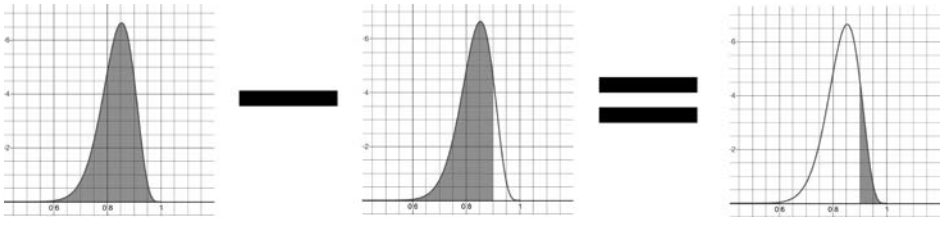


Рис. 2.6. Как найти вероятность того, что базовая вероятность успеха выше 90 %

В примере 2.4 показано, как реализовать это вычитание на Python.

Пример 2.4. Операция вычитания, которая дает площадь области справа от граничного значения в бета-распределении

```
from scipy.stats import beta

a = 8
b = 2

p = 1.0 - beta.cdf(.90, a, b)

# 0.2251590219999998
print(p)
```

Это значит, что при 8 успехах из 10 испытаний двигателя вероятность того, что базовая вероятность успеха составляет 90 % или более, равна всего 22,5 %. Но при этом вероятность того, что она меньше 90 %, оказалась около 77,5 %. Шансы на то, что испытания следует признать успешными, не в нашу пользу, но можно попытаться использовать этот 22,5 %-ный шанс как повод для того, чтобы провести дополнительные испытания, надеясь на то, что нам повезет. Если финансовый директор выделит средства еще на 36 испытаний, из которых 30 окажутся успешными, а 6 — неудачными, то бета-распределение будет выглядеть так, как показано на рис. 2.7.

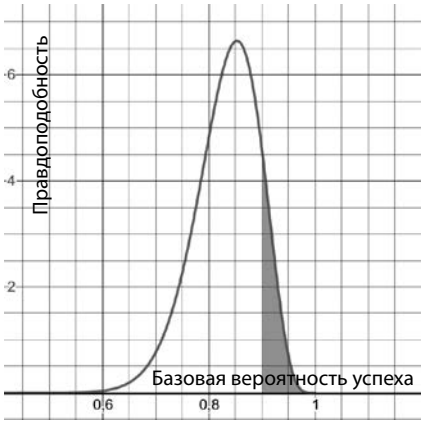


Рис. 2.7. Бета-распределение после 30 успехов и 6 неудач

Обратите внимание, что распределение стало у́же, а значит, мы больше уверены в том, что базовая вероятность успеха находится в меньшем диапазоне. К сожалению, вероятность того, что мы укладываемся в заданный минимальный показатель 90 %, уменьшилась с 22,5 до 13,16 %, как показано в примере 2.5.

Пример 2.5. Бета-распределение с бóльшим количеством испытаний

```
from scipy.stats import beta

a = 30
b = 6

p = 1.0 - beta.cdf(.90, a, b)

# 0.13163577484183686
print(p)
```

На этом этапе, возможно, стоит прекратить испытания, если только вы не хотите дальше экспериментировать с вероятностью 13,16 % и надеяться, что пик распределения сдвинется вправо.

И наконец: как вычислить площадь посередине графика? Например, как найти вероятность того, что базовая вероятность успеха находится в диапазоне от 80 до 90 %, как показано на рис. 2.8?

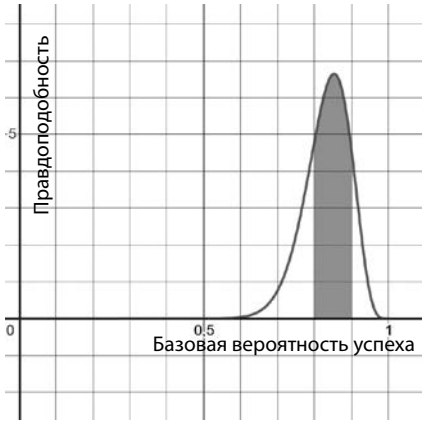


Рис. 2.8. Вероятность того, что базовая вероятность успеха находится в диапазоне от 80 до 90 %

Хорошенько подумайте, как к этому подступиться. Что, если вычесть площадь слева от 0,8 из площади слева от 0,9, как показано на рис. 2.9?

Получим ли мы при этом площадь в интервале от 0,8 до 0,9? Да, и она будет равна 0,3386, что соответствует вероятности 33,86 %. А так можно вычислить площадь на Python (пример 2.6).

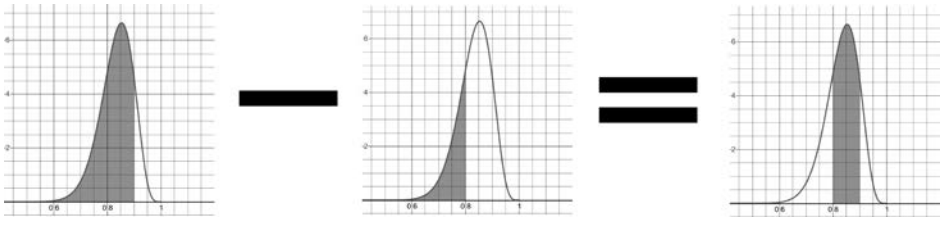


Рис. 2.9. Вычисление площади в интервале от 0,8 до 0,9

Пример 2.6. Вычисление площади в середине бета-распределения с помощью SciPy

```
from scipy.stats import beta
a = 8
b = 2

p = beta.cdf(.90, a, b) - beta.cdf(.80, a, b)

# 0.3386333619999998
print(p)
```

Бета-распределение — замечательный инструмент, с помощью которого можно на основе ограниченного множества наблюдений оценивать вероятность того, что событие произойдет или не произойдет. Оно позволяет рассуждать о вероятностях вероятностей, и его можно корректировать по мере того, как поступают новые данные. Бета-распределение также помогает проверять гипотезы, но в главе 3 мы уделим больше внимания тому, как использовать для этого нормальное распределение и распределение Стьюдента.

Бета-распределение с нуля

О том, как реализовать бета-распределение с нуля, читайте в Приложении А.

Заключение

Из этой главы мы узнали очень много! Мы не только обсудили основы теории вероятностей, логические операторы в ней и формулу Байеса, но и познакомились с распределениями вероятностей, в том числе с биномиальным и бета-распределением. В следующей главе мы встретимся с одним из самых известных распределений — нормальным распределением — и узнаем, как оно связано с проверкой гипотез.

Если вы хотите узнать больше о байесовской вероятности и статистике, то можно начать с отличной книги «*Bayesian Statistics the Fun Way*» Уилла Курта (Will Kurt) (издательство No Starch Press). Кроме того, на платформе O'Reilly (https://oreil.ly/OFbai) доступны интерактивные сценарии Katacoda.

Упражнения для самопроверки

1. Вероятность того, что сегодня пойдет дождь, составляет 30 %, а вероятность того, что зонт, который вы заказали, доставят вовремя, — 40 %. Вы мечтаете сегодня прогуляться под дождем, но для этого нужно, чтобы пошел дождь и чтобы у вас был зонт.

   Какова вероятность того, что пойдет дождь И ваш зонт будет доставлен?

2. Условия те же, что в предыдущем упражнении, но вы собираетесь не на прогулку, а по делам. Это получится, только если не будет дождя или если у вас появится зонт.

   Какова вероятность того, что дождя не будет ИЛИ что ваш зонт будет доставлен?

3. Условия те же, что в предыдущих упражнениях, однако вы выяснили, что в случае дождя вероятность того, что ваш зонт доставят вовремя, составляет всего 20 %.

   Какова вероятность того, что пойдет дождь И ваш зонт будет доставлен?

4. На рейс из Лас-Вегаса в Даллас забронировали места 137 пассажиров. Но надо учитывать, что рейс отправляется из Лас-Вегаса воскресным утром, и, по вашим оценкам, каждый пассажир может не явиться с вероятностью 40 %.

   Вы пытаетесь определить, сколько мест можно перебронировать, чтобы самолет не летел пустым.

   Насколько вероятно, что как минимум 50 пассажиров не полетят?

5. Вы подбросили монету 19 раз. 15 раз выпал орел, а 4 раза — решка.

   Как вы думаете, есть ли при таком сочетании исходов шансы на то, что эта монета не шулерская? Если да, то почему? Если нет, то почему?

Ответы см. в Приложении Б.

ГЛАВА 3

Описательная статистика и статистический вывод

Статистика занимается тем, что собирает и анализирует данные, чтобы обнаружить полезные результаты или предсказать, при каких условиях они могут возникнуть. Большу́ю роль в статистике играет понятие вероятности, потому что мы используем данные, чтобы оценить вероятность того или иного события.

Статистике не всегда достаются заслуженные лавры, но именно она лежит в основе многих инноваций, которые опираются на работу с данными. Машинное обучение как таковое — это статистический инструмент, который находит возможные гипотезы, чтобы установить взаимосвязи между различными переменными в массивах данных. Однако в статистике есть много «слепых пятен», которые сбивают с толку даже профессиональных статистиков. Можно легко увлечься тем, о чем сообщают данные, и забыть уточнить, откуда они взялись. Эта проблема становится все актуальнее, потому что благодаря большим данным, дата-майнингу и машинному обучению статистические алгоритмы все интенсивнее автоматизируются. Поэтому важно владеть прочными фундаментальными знаниями в области статистики и проверки гипотез, чтобы не обращаться с этими автоматическими системами как с «черными ящиками».

В этой главе мы рассмотрим основы статистики и проверки гипотез. Мы начнем с описательной статистики и познакомимся с основными способами, которыми можно обобщать данные. После этого мы перейдем к статистическому анализу, который заключается в том, чтобы на основе выборки попытаться определить свойства генеральной совокупности.

Что такое данные?

Может показаться странным давать определение тому, что такое «данные»: мы используем это понятие сплошь и рядом и воспринимаем его как нечто само собой разумеющееся. Но я считаю, что определение необходимо. Наверное, если спросить первого встречного, что такое данные, он ответит: «Ну, знаете... это данные! То есть... ну, в общем... информация!» — и дальше этого не двинется. Сейчас же данные, судя по всему, позиционируются как «все и вся» — источник не только истины... но и интеллекта! Это сырье для систем искусственного интеллекта, и считается, что чем больше данных, тем больше правды. Поэтому данных никогда не бывает достаточно. Они позволят раскрыть секреты, благодаря которым удастся оптимизировать вашу бизнес-стратегию и, возможно, даже создать искусственный интеллект общего назначения. Но позвольте мне предложить прагматичный взгляд на то, что такое данные. Они важны не сами по себе. Движущая сила всех нововведений и решений — это анализ данных (и способы их получения).

Представьте, что вам показали семейную фотографию. Сможете ли вы проследить историю этой семьи по одной фотографии? А если бы у вас было 20 фотографий? 200 фотографий? 2 000? Сколько фотографий требуется, чтобы реконструировать историю семьи? Нужны ли фотографии, на которых члены семьи запечатлены в разных ситуациях? Поодиночке и вместе? С родственниками и друзьями? Дома и на работе?

Данные, как и фотографии, представляют собой моментальные снимки истории. Они не могут полностью запечатлеть непрерывно меняющуюся реальность и контекст, а также бесконечное количество переменных, от которых зависит эта история. Как мы уже говорили, данные могут быть необъективными. В них бывают пробелы, может не хватать значимых переменных. В идеале нам хотелось бы, чтобы у нас был бесконечный объем данных, которые отражают бесконечное количество переменных, причем настолько подробно, что мы могли бы практически воссоздать реальность и создать альтернативные реальности! Но возможно ли это? В настоящее время — нет. Даже самые мощные суперкомпьютеры в мире, вместе взятые, очень далеки от того, чтобы представить весь мир в виде массива данных.

Поэтому нужно сузить изучаемую область, чтобы цели стали выполнимыми. По нескольким удачным фотографиям, на которых отец семейства играет в гольф, можно легко определить, хороший ли он гольфист. Но попытаться раскрыть всю историю его жизни только по фотографиям может оказаться неосуществимым. Очень многое невозможно зафиксировать в виде моментальных снимков. Эти практические соображения следует применять и в проектах по работе с данными, потому что данные — это всего лишь моментальные снимки, на которых отражено то, что «попало в объектив» в определенный момент времени. Нужно

четко обозначать цели проекта, потому что это помогает следить за тем, чтобы собирать актуальные и полные данные. А если ставить перед собой чересчур широкие и неопределенные задачи, то можно столкнуться с проблемами из-за ложных выводов и неполных наборов данных. Дата-майнинг (добыча данных) имеет смысл в определенных ситуациях, но этим инструментом не стоит злоупотреблять. Мы вернемся к этому вопросу в конце главы.

Даже когда цели четко обозначены, при работе с данными все равно можно столкнуться с проблемами. Вернемся к вопросу о том, можно ли по нескольким удачным фотографиям определить, хорошо ли отец играет в гольф. Может быть, если бы у вас была фотография, на которой он изображен в момент удара, вы бы смогли определить, в хорошей ли форме он тогда был. Или, если вы увидели, как он радуется и принимает поздравления, можно сделать вывод, что он набрал много очков. А может быть, у вас есть фотография его счетной карточки? Но важно иметь в виду, что все эти эпизоды могут быть сфальсифицированы или вырваны из контекста. Может быть, он радовался за кого-то другого, а может, счетная карточка была не его или вовсе поддельной. Как и в случае с фотографиями, данные не отражают контекст и не предлагают объяснений. Это очень важный момент, потому что данные дают подсказки, а не истину. Эти подсказки могут привести к истине, но могут и ввести в заблуждение, если сделать из них ошибочные выводы.

Вот почему так важно уметь интересоваться тем, откуда берутся данные. Задавайте вопросы о том, как были получены данные, кто их собрал и что в них не отражено. Слишком легко зациклиться на том, что сообщают данные, и забыть спросить, откуда они взялись. Еще хуже широко распространенное мнение, будто можно загрузить данные в алгоритм машинного обучения и ждать, что компьютер сам все сделает. Но, как гласит пословица, «мусор на входе — мусор на выходе». Неудивительно, что, по данным сайта VentureBeat (https://oreil.ly/8hFrO), только 13 % проектов в области машинного обучения оказываются успешными. В успешных проектах осмысляются и анализируются как сами данные, так и их источники.

ФУНДАМЕНТАЛЬНАЯ ИСТИНА

В более общем случае пример с семейной фотографией иллюстрирует проблему фундаментальной истины (ground truth) (https://oreil.ly/sa6Ff).

Когда я вел занятия по безопасности систем искусственного интеллекта, мне однажды задали вопрос о том, как улучшить безопасность беспилотных автомобилей. «Когда беспилотному автомобилю не удается распознать пешехода с помощью лидара, нет ли способа обнаружить этот сбой и прекратить движение?» Я ответил, что нет, потому что система не владеет *фундаментальной истиной*, то есть подтвержденным

и исчерпывающим знанием о том, что является истиной. Если автомобиль не распознал пешехода, то как он догадается, что не распознал пешехода? У него нет фундаментальной истины, на которую можно было бы опереться, если только ее не предоставит живой оператор, который вмешается в ситуацию.

Собственно, так и обстоят дела в разработке беспилотных автомобилей. Некоторые датчики, например радары, обеспечивают довольно надежную фундаментальную истину по узким вопросам, например: «Есть ли что-то перед автомобилем?» Но распознавание объектов с помощью камер и лидаров (в неконтролируемых условиях) — это гораздо более расплывчатая задача восприятия с астрономическим числом возможных комбинаций пикселей. Поэтому в данном случае фундаментальной истины просто не существует.

Отражают ли ваши данные фундаментальную истину, которую можно проверить? Полны ли эти данные? Можно ли считать ваши датчики и другие источники информации надежными и точными? Или фундаментальная истина неизвестна?

Описательная статистика и статистический вывод

Что приходит на ум, когда вы слышите слово «статистика»? Вы представили себе, как вычисляют среднее арифметическое, медиану, моду, строят графики и колоколообразные кривые или еще как-то описывают данные? Это наиболее распространенный раздел статистики, который называется *описательной статистикой* и служит для того, чтобы обобщать данные. В конце концов, что практичнее — просмотреть миллион записей с данными или ознакомиться с ними в обобщенном виде? Этот раздел мы рассмотрим в первую очередь.

Статистический вывод (inferential statistics) позволяет выявить признаки более обширной совокупности, часто на основе выборки. У многих складывается неправильное представление о статистическом выводе, потому что этот раздел менее интуитивно понятен, чем описательная статистика.

Зачастую требуется изучить группу, которая слишком велика, чтобы наблюдать ее всю (например, если мы хотим узнать средний рост подростков в Северной Америке), и тогда приходится изучать лишь несколько представителей этой группы, чтобы сделать выводы о ней в целом. Как можно догадаться, осуществить это правильно не так-то просто — ведь мы пытаемся представить всю совокупность с помощью выборки, которая может оказаться нерепрезентативной. Позже мы обсудим эти нюансы.

Совокупности, выборки и смещение

Прежде чем углубляться в описательную статистику и статистический вывод, было бы неплохо дать несколько определений и рассмотреть их на конкретных примерах.

Генеральная совокупность, или просто *совокупность*, — это определенная группа, которую мы хотим изучить, например «все люди старше 65 лет в Северной Америке», «все золотистые ретриверы в Шотландии» или «второкурсники колледжа города Лос-Альтос». Обратите внимание на то, как мы ограничиваем ту или иную совокупность. Одни совокупности охватывают большую группу населения в обширном географическом или возрастном диапазоне, а другие — очень узкие, как, например, второкурсники колледжа в Лос-Альтосе. От того, что вы хотите изучить, зависит, как вы будете ограничивать совокупность.

Выборка — это подмножество совокупности, которое в идеале составлено случайным образом и без смещения и на основе которого мы делаем выводы о целой совокупности. Часто приходится работать с выборками, потому что не всегда можно исследовать всю совокупность. Конечно, некоторые совокупности легче охватить, если они небольшие и доступные. Но собирать данные обо всех людях старше 65 лет в Северной Америке? Вряд ли это целесообразно!

СОВОКУПНОСТИ МОГУТ БЫТЬ АБСТРАКТНЫМИ!

Важно отметить, что генеральная совокупность может быть теоретической, а не реально существующей. В этих случаях мы обращаемся с реальной совокупностью, как с выборкой из воображаемой совокупности. Вот мой любимый пример: нас интересуют рейсы, которые вылетают из аэропорта между 14 и 15 часами, но в это время нет достаточного количества рейсов, чтобы надежно спрогнозировать, как часто они опаздывают. Поэтому можно рассматривать всю совокупность рейсов как выборку из воображаемой генеральной совокупности всех чисто теоретических рейсов, которые вылетают в период с 14:00 до 15:00.

Задачи такого рода побуждают многих исследователей получать данные с помощью симуляторов. Это может быть полезно, но симуляции редко бывают точными, потому что они учитывают лишь ограниченное количество переменных и опираются на заранее заданные допущения.

Если мы собираемся делать выводы о свойствах совокупности на основе выборки, важно, чтобы выборка была как можно более случайной, иначе выводы могут оказаться искаженными. Приведем пример. Допустим, я студент колледжа Университета штата Аризона. Я хочу выяснить, сколько часов в среднем студенты

колледжей в США еженедельно проводят у экрана телевизора. Я выхожу прямо из общежития и начинаю опрашивать случайных студентов, которые проходят мимо; весь сбор данных занимает несколько часов. В чем же здесь проблема?

Проблема заключается в том, что моя выборка студентов будет *смещенной*, потому что в ней чрезмерно представлена определенная группа студентов в ущерб остальным группам, и это искажает выводы. В моем исследовании генеральная совокупность определена как «студенты колледжей США», а не «студенты колледжа Университета штата Аризона». Я опрашиваю только студентов одного конкретного университета, чтобы сделать выводы обо всех студентах колледжей во всех Соединенных Штатах! Разве это корректно?

Маловероятно, что у студентов во всех колледжах страны будут единообразные предпочтения. Что, если студенты Университета штата Аризона смотрят телевизор гораздо чаще, чем студенты других университетов? Не исказятся ли результаты, если распространять на всю страну выводы, которые сделаны на основе этой выборки? Может быть, это происходит потому, что в городе Темпе, где находится кампус университета, обычно слишком жарко, чтобы выходить на улицу, и поэтому многие проводят время перед телевизором (это подтверждается моим собственным опытом: я много лет прожил в Финиксе — столице Аризоны). Возможно, студенты других колледжей, которые находятся в других климатических зонах, больше гуляют на свежем воздухе и меньше смотрят телевизор.

Это лишь один из источников смещения, из-за которых не стоит ориентироваться на выборку студентов одного университета, если мы хотим изучить совокупность студентов всех Соединенных Штатов. В идеале мне следовало бы случайным образом опросить студентов колледжей в разных университетах по всей стране — тогда я получу более репрезентативную выборку.

Однако смещение не всегда связано с географией. Допустим, я приложил огромные усилия, чтобы опросить студентов по всей территории США. Я организовал кампанию в социальных сетях, чтобы разные университеты распространили опрос в Twitter и Facebook. Таким образом, студенты его увидят и, надеюсь, примут в нем участие. В итоге я получаю сотни ответов о телевизионных пристрастиях студентов из разных уголков страны и радуюсь, что преодолел проблему смещения... или нет?

А что если предположить, что те студенты, которые увидели опрос в социальных сетях, чаще прочих смотрят телевизор? Если они много сидят в социальных сетях, то, скорее всего, они также не против провести свободное время за экраном телевизора. Неудивительно, если на соседних вкладках браузера у них открыты Netflix и Hulu! Этот особый тип смещения, когда та или иная группа с большей вероятностью включает себя в выборку, известен как *смещение из-за самоотбора*.

Проклятье! Неужели тут ничего не поделать? Если задуматься, то смещение данных кажется неизбежным — и зачастую так оно и есть. На наше исследование может повлиять множество *спутывающих переменных* или факторов, которые мы не учли. Проблема смещения данных затратна и труднопреодолима, а машинное обучение особенно уязвимо для нее.

Чтобы избежать этой проблемы, можно действительно случайным образом выбирать студентов из всей совокупности, причем так, чтобы они не могли по собственному желанию попасть в выборку или выйти из нее. Это наиболее эффективный способ бороться со смещением, но, как вы догадываетесь, он требует больших согласованных усилий.

КРАТКАЯ СПРАВКА О ТИПАХ СМЕЩЕНИЯ

Как ни странно, мы, люди, склонны к смещениям. Мы ищем закономерности даже там, где их нет. Возможно, это было эволюционно необходимо, чтобы выживать на заре человечества, потому что поиск закономерностей помогал эффективнее заниматься охотой, собирательством и земледелием.

Существует множество типов смещения, но все они приводят к одному и тому же — к искажению результатов. *Склонность к подтверждению* проявляется, когда вы собираете только те данные, которые подтверждают вашу точку зрения, причем это может происходить даже неосознанно. Например, вы подписываетесь только на те аккаунты в социальных сетях, которые разделяют ваши политические убеждения, что укрепляет вашу позицию, а не дает повод сомневаться в ней.

Мы уже обсудили *смещение из-за самоотбора*, когда определенные категории респондентов с большей вероятностью включат себя в исследование. Глупо приходить в самолет и опрашивать пассажиров, нравится ли им эта авиакомпания по сравнению с другими, чтобы на основании этого оценивать удовлетворенность клиентов среди всех авиакомпаний. Почему? Скорее всего, многие пассажиры выбрали эту авиакомпанию не в первый раз, и это создает смещение из-за самоотбора. Мы не знаем, какая доля пассажиров является повторными клиентами авиакомпании, а ведь они в первую очередь ответят, что предпочитают эту компанию другим. Даже те, кто летит впервые, могут создавать смещение из-за самоотбора, потому что они выбрали этого перевозчика, а наша выборка не охватывает всех перевозчиков.

Ошибка выжившего происходит, когда исследование затрагивает только тех, кто остался в живых, и не учитывает умерших. На мой взгляд,

это самый интересный тип смещения, потому что его примеры разнообразны и неочевидны.

Вероятно, наиболее известный пример ошибки выжившего — случай времен Второй мировой войны, когда британские военно-воздушные силы теряли бомбардировщики из-за немецких обстрелов. Чтобы предохранить самолеты, британские инженеры сначала предложили укреплять их броню в тех местах, где были обнаружены пробоины от снарядов. Однако математик по имени Абрахам Вальд настаивал на том, что это решение абсолютно ошибочно. Он предложил укреплять те места фюзеляжа, где *не было* пробоин. Может, он был сумасшедшим? Отнюдь. Очевидно, что у бомбардировщиков, которые вернулись с задания, не было катастрофических повреждений в местах пробоин. Почему мы в этом уверены? Потому что они вернулись! А что же с самолетами, которые не вернулись? Куда они были поражены? Теория Вальда заключалась в том, что «скорее всего, они были поражены в тех местах, которые остались нетронутыми у самолетов, *которые выжили* и вернулись на базу». Это оказалось верным. Инженеры укрепили те места, где не было пробоин, и выживаемость самолетов и пилотов повысилась. Считается, что это нестандартное наблюдение, помимо прочего, внесло вклад в то, чтобы переломить ход войны в пользу союзников.

Существуют и другие впечатляющие, но менее очевидные примеры ошибки выжившего. Многие консалтинговые компании и книжные издательства любят выявлять качества успешных компаний и руководителей и представлять эти качества как залог будущих успехов. Эти работы — чистой воды ошибка выжившего (а на сайте XKCD есть забавный рисунок на эту тему: https://xkcd.com/1827). В них не учитываются компании и деятели, которые обладали теми же «качествами успеха», но потерпели неудачу и остались неизвестными. Просто мы не слышали о них, потому что они никогда не бывали в центре внимания.

В качестве иллюстрации можно привести пример Стива Джобса. Многие считают, что он был вздорным и вспыльчивым — но при этом создал одну из самых дорогих компаний всех времен. Поэтому некоторые люди убеждены, что вспыльчивость и горячность могут быть связаны с успехом. Опять же, это ошибка выжившего. Я готов поспорить, что существует множество компаний, которыми руководили вспыльчивые люди и которые провалились в безвестности, но мы зацикливаемся на таких историях успеха, как у Apple.

И наконец, в 1987 году было проведено исследование в области ветеринарии, которое показало, что кошки, которые упали с шестого и более

низких этажей, получали более серьезные травмы, чем те, которые упали с большей высоты. Из этого вывели научную теорию о том, будто кошки оптимизируют свое положение в воздухе на уровне примерно пяти этажей, что дает им достаточно времени, чтобы приземлиться на все четыре лапы и получить меньше травм. Но затем газета *Chicago Reader* задала важный вопрос: а что же случилось с мертвыми кошками? Люди вряд ли понесут мертвую кошку к ветеринару, и поэтому нет данных о том, сколько кошек погибло от падения с большой высоты.

Мы достаточно поговорили о совокупностях, выборках и смещении, так что давайте перейдем к математике и описательной статистике. Имейте в виду, что математика и компьютеры не способны распознать смещение в ваших данных. Если вы хороший специалист по data science, то сами должны его обнаружить! Всегда уточняйте, как именно были получены данные, а затем тщательно анализируйте, как конкретно метод сбора данных мог их исказить.

Выборки и смещение в машинном обучении

Подобные проблемы с выборками и смещением характерны и для машинного обучения. И линейная регрессия, и логистическая, и нейронные сети выводят предсказания на основе выборки данных. Если эти данные смещены, то алгоритмы машинного обучения будут делать смещенные выводы.

Этому есть множество документально подтвержденных примеров. В судопроизводстве машинное обучение внедряется с большим трудом, потому что оно неоднократно проявляло смещение во всех смыслах этого слова, ущемляя интересы меньшинств из-за того, что в наборах данных было много представителей этих меньшинств. А в 2017 году компания Volvo испытывала беспилотные автомобили, которые были обучены предотвращать столкновения на наборах данных об оленях и лосях. Однако у искусственного интеллекта не было данных об опыте водителей в Австралии, и поэтому он не мог распознать кенгуру, а тем более отследить их прыжки! Оба эти случая — примеры смещенных данных.

Описательная статистика

Описательная статистика — это область, с которой знакомо большинство людей. Мы коснемся таких элементарных понятий, как среднее арифметическое, медиана и мода, а затем рассмотрим дисперсию, стандартное отклонение и нормальное распределение.

Среднее арифметическое и среднее взвешенное

Среднее арифметическое — это сумма всех чисел из некоторого множества, деленная на их количество. Эта величина полезна тем, что она показывает, где находится «центр тяжести» наблюдаемого множества значений.

Среднее арифметическое вычисляется одинаково как для совокупностей, так и для выборок. В примере 3.1 показано, как на Python вычислить среднее арифметическое для выборки из восьми значений.

Пример 3.1. Вычисление среднего арифметического на Python

```
# Количество домашних животных у каждого респондента
sample = [1, 3, 2, 5, 7, 0, 2, 3]

mean = sum(sample) / len(sample)

print(mean) # выводит 2.875
```

Как видите, мы опросили 8 человек о том, сколько у них домашних животных. Сумма всех значений в выборке равна 23, а количество элементов — 8, что дает нам среднее арифметическое 2,875, потому что $\frac{23}{8} = 2{,}875$[1].

Среднее арифметическое бывает двух типов: выборочное среднее $\bar{x}$ и генеральное среднее μ:

$$\bar{x} = \frac{x_1 + x_2 + x_3 + \ldots + x_n}{n} = \sum \frac{x_i}{n}$$

$$\mu = \frac{x_1 + x_2 + x_3 + \ldots + x_n}{N} = \sum \frac{x_i}{N}$$

Напомним, что символ суммы Σ означает сложение всех элементов. Буквы n и N обозначают размер выборки и генеральной совокупности соответственно, но математически они представляют одно и то же — количество элементов. Аналогично выборочное среднее $\bar{x}$ и генеральное среднее μ вычисляются одинаково, просто они называются по-разному в зависимости от того, с чем мы работаем — с выборкой или с генеральной совокупностью.

Скорее всего, среднее арифметическое вам хорошо знакомо, но кое-чего о нем вы могли не знать: по сути среднее арифметическое — это частный случай так называемого *среднего взвешенного*. Привычное нам среднее арифметическое подразумевает, что все значения одинаково существенны. Но от этого можно отойти, если придать каждому элементу свой весовой коэффициент:

[1] Чтобы вычислить среднее арифметическое, можно также применять функцию `mean` из модуля `statistics`, например: `mean = statistics.mean(sample).` — *Примеч. науч. ред.*

$$\text{среднее взвешенное} = \frac{(x_1 \cdot w_1) + (x_2 \cdot w_2) + (x_3 \cdot w_3) + \ldots + (x_n \cdot w_n)}{w_1 + w_2 + w_3 + \ldots + w_n}$$

Это может пригодиться, если нужно, чтобы одни значения вносили больший вклад в итоговое среднее, чем другие. Распространенный пример, который иллюстрирует среднее взвешенное, — вычисление итогового балла по результатам экзаменов в учебных заведениях. Допустим, нужно сдать три промежуточных экзамена и один выпускной, и мы придаем каждому из трех промежуточных экзаменов весовой коэффициент 20 %, а выпускному — 40 % от итогового балла. В примере 3.2 показано, как в этом случае вычисляется итоговый балл.

Пример 3.2. Вычисление среднего взвешенного на Python[2]

```
# Три промежуточных экзамена с весом 0,20 и выпускной экзамен с весом 0,40
sample = [90, 80, 63, 87]
weights = [0.2, 0.2, 0.2, 0.4]

weighted_mean = sum(s * w for s,w in zip(sample, weights)) / sum(weights)

print(weighted_mean) # выводит 81.4
```

Мы взвешиваем каждый экзаменационный балл, умножая его на соответствующий весовой коэффициент, и делим сумму баллов не на количество значений, а на сумму коэффициентов. Весовые коэффициенты не обязательно должны выражаться в процентах, потому что любые числа, которые используются в этом качестве, в конечном итоге будут приведены в пропорцию. В примере 3.3 мы присваиваем каждому промежуточному экзамену весовой коэффициент 1, а выпускному — 2, чтобы он «весил» вдвое больше, чем каждый из промежуточных. Все равно получается тот же ответ 81,4, потому что эти коэффициенты приводятся в пропорцию.

Пример 3.3. Вычисление среднего взвешенного на Python

```
# Три промежуточных экзамена с весом 1,0 и выпускной экзамен с весом 2,0
sample = [90, 80, 63, 87]
weights = [1.0, 1.0, 1.0, 2.0]

weighted_mean = sum(s * w for s,w in zip(sample, weights)) / sum(weights)

print(weighted_mean) # выводит 81.4
```

[2] Среднее взвешенное также можно вычислить с помощью функции `average()` из модуля numpy. Например, если импортировать этот модуль стандартной инструкцией `import numpy as np`, то среднее взвешенное в этом примере можно получить так: `weighted_mean = np.average(sample, weights=weights).` — *Примеч. науч. ред.*

Медиана

Медиана — это центральное значение в наборе упорядоченных значений. Если упорядочить значения по возрастанию, то медианой будет то значение, которое находится в середине этой последовательности. Если количество значений четное, то медианой считается среднее арифметическое двух центральных значений. Из примера 3.4 видно, что медианное количество домашних животных в нашей выборке равно 7:

```
0, 1, 5, 7, 9, 10, 14
```

Пример 3.4. Вычисление медианы на Python[1]

```
# Количество домашних животных у каждого респондента
sample = [0, 1, 5, 7, 9, 10, 14]

def median(values):
    ordered = sorted(values)
    n = len(ordered)
    mid = int(n / 2) - 1 if n % 2 == 0 else int(n/2)

    if n % 2 == 0:
        return (ordered[mid] + ordered[mid+1]) / 2.0
    else:
        return ordered[mid]

print(median(sample)) # выводит 7
```

Медиана может пригодиться вместо среднего арифметического, если данные перекошены *выбросами* — значениями, которые слишком велики или слишком малы по сравнению с остальными. Вот интересный пример, который поможет усвоить это понятие. В 1986 году среднегодовая начальная зарплата выпускников географического факультета Университета Северной Каролины в Чапел-Хилле (UNC) составляла 250 000 долларов, а для других университетов этот показатель составлял в среднем 22 000 долларов. Должно быть, в UNC просто волшебная учебная программа по географии!

Но что такого прибыльного было на самом деле на географическом факультете UNC? Дело в том, что одним из его выпускников был... Майкл Джордан. Один из самых прославленных игроков NBA действительно окончил географический факультет UNC, однако он начал свою карьеру как баскетболист, а не как географ. Очевидно, что это особый случай, из-за которого возник огромный выброс, а средний доход выпускников значительно сместился.

Именно поэтому в ситуациях с большим количеством выбросов (например, в данных о доходах) медиана может оказаться предпочтительнее, чем среднее

[1] Медиану также можно вычислить с помощью функции `median()` из модуля statistics, например: `print(statistics.median(sample))`. — *Примеч. науч. ред.*

арифметическое. Она менее чувствительна к выбросам и разделяет данные строго посередине, ориентируясь на их относительный порядок, а не на то, где именно они расположены на числовой прямой. Если медиана радикально отличается от среднего, это значит, что набор данных перекошен и содержит выбросы.

> **МЕДИАНА — ЭТО КВАНТИЛЬ**
>
> В описательной статистике есть понятие *квантилей*. Квантиль, по сути, обобщает понятие медианы: он разделяет данные не обязательно посередине, а в произвольном месте. Медиана — это фактически 50 %-ный квантиль, то есть значение, за которым находится 50 % упорядоченных значений. Медиана вместе с 25- и 75 %-ными квантилями называются *квартилями*, потому что они разделяют данные через каждые 25 %.

Мода

Мода — это значение, которое встречается в наборе данных чаще всего. В первую очередь она полезна, когда в наборе данных многократно повторяются одни и те же значения, и необходимо определить, какие из них встречаются наиболее часто.

Если ни одно значение не встречается более одного раза, то мода отсутствует. Если самыми частыми оказываются два значения с одинаковой частотой, то набор данных считается *бимодальным*. В примере 3.5 мы вычисляем моду для набора данных о домашних животных и видим, что он бимодальный, потому что и 2, и 3 встречаются с одинаковой наибольшей частотой.

Пример 3.5. Вычисление моды на Python[2]

```
from collections import defaultdict

# Количество домашних животных у каждого респондента
sample = [1, 3, 2, 5, 7, 0, 2, 3]

def mode(values):
    counts = defaultdict(lambda: 0)

    for s in values:
        counts[s] += 1

    max_count = max(counts.values())
    modes = [v for v in set(values) if counts[v] == max_count]
    return modes

print(mode(sample)) # [2, 3]
```

[2] Моду (или несколько мод) также можно вычислить с помощью функции `multimode()` из модуля statistics, например: `print(statistics.multimode(sample))`. — *Примеч. науч. ред.*

На практике моду используют не слишком часто — за исключением случаев, когда данные повторяются. Это бывает, когда мы работаем с целыми числами, категориями и другими дискретными переменными.

Дисперсия и стандартное отклонение

Когда заходит речь о дисперсии и стандартном отклонении, всплывает много интересных моментов. При работе с этими величинами многих смущает, что для выборки они вычисляются немного не так, как для генеральной совокупности. Я постараюсь сделать все возможное, чтобы доходчиво рассказать об этих различиях.

Дисперсия и стандартное отклонение генеральной совокупности

При описании данных нас часто интересует разница между средним значением и каждой конкретной точкой данных. Эта разница дает представление о разбросе данных.

Допустим, я хочу выяснить, сколько домашних животных принадлежит семерым членам моего рабочего коллектива (обратите внимание, что я определяю его как совокупность, а не как выборку).

Выяснилось, что среднее количество домашних животных, принадлежащих моим коллегам, составляет 6,571. Вычтем это число из каждого значения. Так мы узнаем, насколько далеко находится каждое значение от среднего, как показано в табл. 3.1.

Таблица 3.1. Количество домашних животных, принадлежащих опрошенным сотрудникам

Значение	Среднее	Разница
0	6,571	−6,571
1	6,571	−5,571
5	6,571	−1,571
7	6,571	0,429
9	6,571	2,429
10	6,571	3,429
14	6,571	7,429

Представим эти данные на числовой прямой, где «×» показывает среднее арифметическое (рис. 3.1).

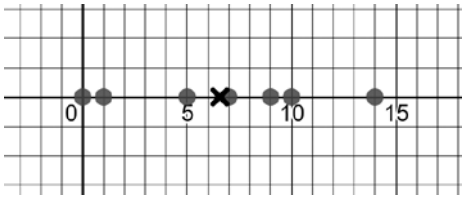


Рис. 3.1. Разброс данных («×» — среднее арифметическое)

Теперь подумаем, чем эта информация может быть полезна. Величины разности дают представление о том, насколько разбросаны данные и насколько далеки значения от среднего. Есть ли способ объединить эти сведения в одно число, чтобы быстро описать степень разброса?

Может возникнуть соблазн взять среднюю разность, но при сложении отрицательные и положительные значения будут аннулировать друг друга. Можно просуммировать абсолютные значения (то есть избавиться от знаков минуса и сделать все значения положительными). Еще эффективнее будет возводить разности в квадрат, перед тем как суммировать. Это не только избавит нас от отрицательных значений (потому что отрицательное число в квадрате становится положительным), но и увеличит большие разности, а с математической точки зрения с полученной величиной будет проще работать (дело в том, что с абсолютными значениями не так удобно вычислять производные). После этого усредним квадраты разностей. В результате получится *дисперсия*, которая показывает, насколько разбросаны данные.

Вот формула, по которой можно вычислить дисперсию:

$$\text{дисперсия} = \frac{(x_1 - \text{среднее})^2 + (x_2 - \text{среднее})^2 + \ldots + (x_n - \text{среднее})^2}{N}$$

Более строго формула дисперсии для генеральной совокупности выглядит так:

$$\sigma^2 = \frac{\Sigma(x_i - \mu)^2}{N}$$

В примере 3.6 дисперсия совокупности для нашего случая с домашними животными вычисляется на Python.

Пример 3.6. Вычисление дисперсии на Python[1]

```
# Количество домашних животных у каждого респондента
data = [0, 1, 5, 7, 9, 10, 14]
```

[1] Дисперсию генеральной совокупности также можно вычислить с помощью функции `pvariance()` из модуля statistics, например: `print(statistics.pvariance(data))`. — *Примеч. науч. ред.*

```
def variance(values):
mean = sum(values) / len(values)
_variance = sum((v - mean) ** 2 for v in values) / len(values)
return _variance

print(variance(data)) # выводит 21.387755102040817
```

Таким образом, дисперсия количества домашних животных, которые принадлежат моим коллегам, равна 21,387755. Хорошо, но что это значит на самом деле? Разумно предположить, что чем больше дисперсия, тем больше разброс, но как соотнести это значение с нашими данными? Это число больше, чем любое из наших наблюдений, потому что мы возводили значения в квадрат и складывали их; результат оказался выражен в совершенно других единицах измерения. Как привести его в соответствие с той же шкалой, в которой находятся исходные данные?

Извлечение квадратного корня — это операция, обратная возведению в квадрат, поэтому возьмем квадратный корень из дисперсии, в результате чего получится *стандартное отклонение* (или *среднеквадратическое отклонение*). Это дисперсия, масштабированная в число, которое выражено в единицах количества домашних животных, и его удобнее себе представлять:

$$\sigma = \sqrt{\frac{\Sigma(x_i - \mu)^2}{N}}$$

Чтобы вычислить стандартное отклонение на Python, воспользуемся функцией `variance()` и извлечем корень из ее результата с помощью функции `sqrt()`. Теперь у нас есть функция `std_dev()`, которая показана в примере 3.7.

Пример 3.7. Вычисление стандартного отклонения на Python[1]

```
from math import sqrt

# Количество домашних животных у каждого респондента
data = [0, 1, 5, 7, 9, 10, 14]

def variance(values):
    mean = sum(values) / len(values)
    _variance = sum((v - mean) ** 2 for v in values) / len(values)
    return _variance

def std_dev(values):
    return sqrt(variance(values))

print(std_dev(data)) # выводит 4.624689730353899
```

[1] Стандартное отклонение генеральной совокупности также можно вычислить с помощью функции `pstdev()` из модуля statistics, например: `print(statistics.pstdev(data))`. — *Примеч. науч. ред.*

Если запустить код из примера 3.7, можно узнать, что стандартное отклонение составляет примерно 4,62 питомца. Эта величина позволяет выразить разброс в тех же единицах измерения, с которых мы начинали, и дисперсию становится немного проще интерпретировать. Некоторые важные применения стандартного отклонения мы рассмотрим в главе 5.

При чем тут квадрат?

Возможно, вас беспокоит вопрос, зачем в формуле дисперсии σ возводится в квадрат (и получается σ^2). Это потому, что дисперсия как бы намекает, что из нее нужно извлечь квадратный корень, чтобы получить стандартное отклонение. Это своеобразное напоминание о том, что вы имеете дело с квадратами величин, и, чтобы вычислить эти величины, нужно извлечь корень.

Выборочная дисперсия и стандартное отклонение

В предыдущем разделе мы говорили о дисперсии и стандартном отклонении генеральной совокупности. А чтобы вычислить эти величины для выборки, в соответствующие формулы необходимо внести важную поправку:

$$Дисперсия: s^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

$$Стандартное\ отклонение: s = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}}$$

Вы уловили разницу? Когда мы усредняем квадраты разностей, мы теперь делим не на общее количество элементов n, а на $n - 1$. Почему так? Это нужно, чтобы уменьшить смещение выборки и не занижать дисперсию совокупности на основе нашей выборки. Когда значение в знаменателе на единицу меньше, мы увеличиваем дисперсию, а значит, фиксируем бóльшую неопределенность в нашей выборке.

ЗАЧЕМ ВЫЧИТАТЬ 1 ИЗ РАЗМЕРА ВЫБОРКИ?

На YouTube-канале Джоша Стармера (Josh Starmer) есть отличная серия видеороликов StatQuest (https://oreil.ly/6S9DO). В одном из них он доходчиво объясняет, почему мы вычисляем дисперсию для выборки по-другому и вычитаем единицу из общего количества элементов.

Если бы данные о домашних животных представляли собой выборку, а не совокупность, нужно было бы сделать соответствующую поправку. В примере 3.8 я модифицировал функции `variance()` и `std_dev()`, чтобы они могли принимать необязательный параметр `is_sample`: если он равен `True`, то из делителя в дисперсии будет вычитаться 1.

Пример 3.8. Вычисление выборочного стандартного отклонения[1]

```
from math import sqrt

# Количество домашних животных у каждого респондента
data = [0, 1, 5, 7, 9, 10, 14]

def variance(values, is_sample: bool = False):
    mean = sum(values) / len(values)
    _variance = sum((v - mean) ** 2 for v in values) /\
        (len(values) - (1 if is_sample else 0))

    return _variance

def std_dev(values, is_sample: bool = False):
    return sqrt(variance(values, is_sample))

print(f"Дисперсия = {variance(data, is_sample=True)}") # 24.952380952380953
print(f"Стандартное отклонение = {std_dev(data, is_sample=True)}")
# 4.995235825502231
```

Обратите внимание, что в примере 3.8 дисперсия и стандартное отклонение больше, чем в предыдущих примерах, где мы рассматривали данные как генеральную совокупность, а не как выборку. Вспомним, что в примере 3.7, когда мы имели дело с совокупностью, стандартное отклонение составляло около 4,62. Но здесь, когда мы работаем с выборкой (и, соответственно, вычитаем 1 из знаменателя дисперсии), оно равно примерно 5,00. Это правильно, потому что выборка может быть смещенной и некорректно представлять совокупность. Поэтому мы увеличиваем дисперсию (а значит, и стандартное отклонение), чтобы повысить оценку того, насколько разбросаны значения. Чем больше дисперсия (и стандартное отклонение), тем меньшую степень доверия к оценке она отражает из-за большего разброса.

Напомним, что среднее арифметическое и дисперсия генеральной совокупности обозначались буквами μ и σ соответственно, а для выборки они обозначаются $\bar{x}$ и s соответственно. Вот формулы стандартного отклонения для выборки и для совокупности:

[1] Дисперсию и стандартное отклонение выборки также можно вычислить с помощью функций `variance()` и `stdev()` из модуля statistics, например: `print(statistics.stdev(data))`. — *Примеч. науч. ред.*

$$s = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}}$$

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

Дисперсия будет равна квадрату этих величин. Таким образом, дисперсия для выборки и для совокупности равна s^2 и σ^2 соответственно:

$$s^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

$$\sigma^2 = \frac{\sum(x_i - \mu)^2}{N}$$

Как уже отмечалось, знак квадрата помогает понять, что нужно извлечь квадратный корень, чтобы получить стандартное отклонение.

Нормальное распределение

В предыдущей главе мы упомянули о распределениях вероятностей, в частности о биномиальном и бета-распределении. Однако самое известное распределение — *нормальное*. Оно также называется *распределением Гаусса* и представляет собой симметричное колоколообразное распределение, где наибольшая плотность сосредоточена вокруг среднего, а разброс определяется как стандартное отклонение. Чем дальше от среднего, тем тоньше становятся «хвосты» по обе стороны кривой.

На рис. 3.2 показано нормальное распределение массы тела золотистого ретривера (в фунтах). Обратите внимание, что бóльшая часть значений расположена около среднего, которое равно 64,43 фунта.

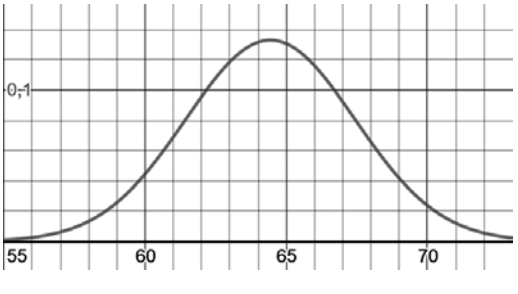


Рис. 3.2. Нормальное распределение

Как обнаружить нормальное распределение

Нормальное распределение часто встречается в природе, науке, технике и других областях. Как же его обнаружить? Допустим, мы измерили массу тела 50 взрослых золотистых ретриверов и нанесли эти данные на числовую прямую, как показано на рис. 3.3.

Рис. 3.3. Выборка из 50 значений массы тела золотистых ретриверов

Обратите внимание, что в центре выборки оказалось больше значений, а по мере удаления влево или вправо их становится меньше. Исходя из нашей выборки, кажется маловероятным, что мы встретим золотистого ретривера с массой 57 или 71 фунт. А что насчет массы 64 или 65 фунтов? Очевидно, она выглядит более правдоподобной.

Есть ли лучший способ наглядно изобразить эту правдоподобность, чтобы понять, какую массу ретривера мы с большей вероятностью увидим в выборке из совокупности? Можно попробовать начертить гистограмму, в которой значения распределяются по числовым интервалам равной длины, а столбики показывают количество значений на каждом интервале. На рис. 3.4 мы строим гистограмму, которая объединяет значения в интервалы по 0,5 фунта.

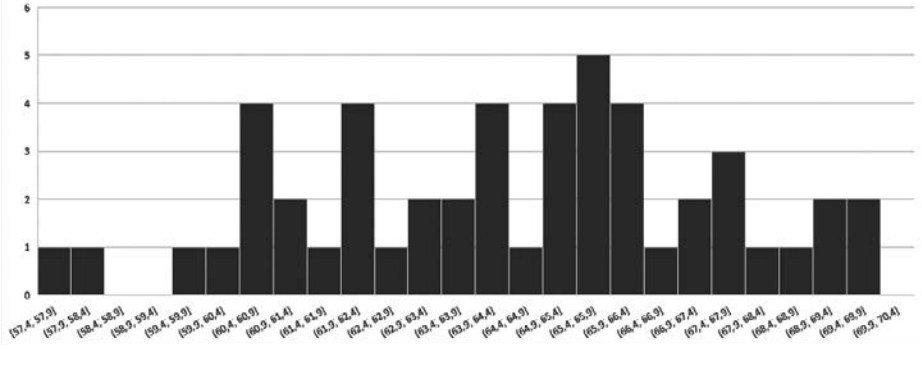


Рис. 3.4. Гистограмма массы тела золотистых ретриверов

Эта гистограмма не отражает никакой существенной структуры данных. Причина этого в том, что выбранные интервалы слишком малы. У нас не настолько большой (и тем более не бесконечный) объем данных, чтобы на каждом

интервале было достаточно точек. Поэтому придется укрупнить интервалы. Давайте сделаем так, чтобы длина каждого интервала составляла три фунта, как показано на рис. 3.5.

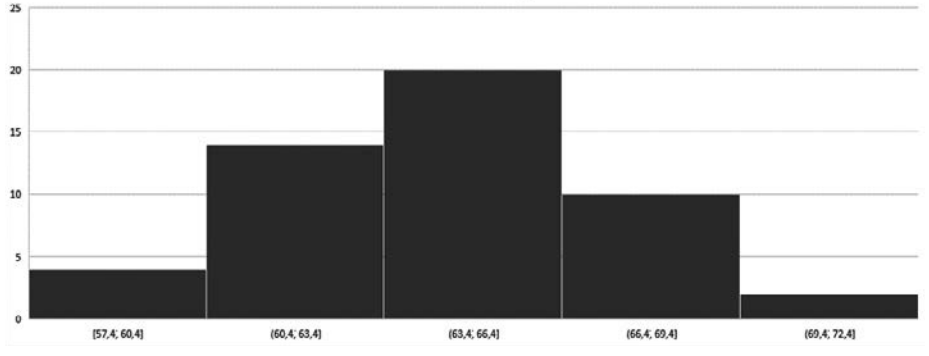


Рис. 3.5. Более информативная гистограмма

Это уже что-то интересное! Как видите, если правильно выбрать размер интервалов (в данном случае каждый из них равен трем фунтам), мы приближаемся к знакомой колоколообразной форме данных. Это не идеальный «колокол», потому что выборки никогда не отражают генеральную совокупность во всей полноте, но эта форма, скорее всего, свидетельствует о том, что наша выборка соответствует нормальному распределению. Если построить гистограмму с подходящими размерами интервалов и отмасштабировать ее так, чтобы площадь под кривой была равна 1 (что требуется для распределения вероятностей), мы увидим примерно колоколообразную кривую, которая представляет нашу выборку. Давайте изобразим ее рядом с исходными точками на рис. 3.6.

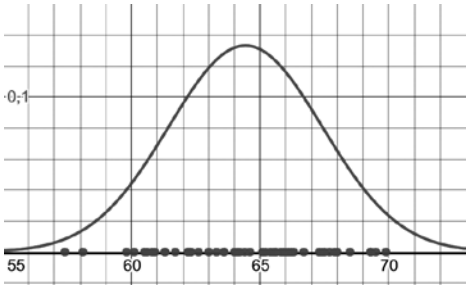


Рис. 3.6. Нормальное распределение, построенное для точек собранных данных

Глядя на эту кривую, можно обоснованно ожидать, что золотистый ретривер, вероятнее всего, будет весить 64,43 фунта (среднее значение), но вряд ли — 55 или 73 фунта. Показатели, которые отстоят от среднего еще дальше, представляются крайне маловероятными.

Свойства нормального распределения

Нормальное распределение так полезно, потому что у него есть ряд важных свойств.

- Оно симметрично: обе стороны зеркально отражают друг друга относительно центральной оси (которая проходит через среднее арифметическое).
- Основная масса значений находится в центре вокруг среднего.
- У него есть разброс, который может быть шире или уже и характеризуется стандартным отклонением.
- «Хвосты» по бокам — это наименее вероятные исходы, которые бесконечно приближаются к нулю, но никогда его не достигают.
- Оно не только довольно точно соответствует многим явлениям в природе и повседневной жизни, но и обобщает даже те задачи, которые не отвечают нормальному распределению, — благодаря центральной предельной теореме, о которой мы скоро поговорим.

Функция плотности вероятности

Стандартное отклонение играет важную роль в нормальном распределении, потому что оно характеризует, насколько это распределение «разбросано». Фактически это один из параметров наряду со средним арифметическим. *Функция плотности вероятности* (PDF, probability density function), которая формирует нормальное распределение, выглядит так:

$$f(x) = \frac{1}{\sigma} \times \sqrt{2\pi} \times e^{-\frac{1}{2}\left(\frac{x-\mu^2}{\sigma}\right)}$$

Мудреное выражение, правда? Здесь даже есть наш старый друг — число *e* из главы 1, а также заковыристый показатель степени. В примере 3.9 представлена эта же формула на Python.

Пример 3.9. Функция нормального распределения на Python

```
# нормальное распределение, возвращает правдоподобие
def normal_pdf(x: float, mean: float, std_dev: float) -> float:
    return (1.0 / (2.0 * math.pi * std_dev ** 2) ** 0.5) * \
        math.exp(-1.0 * ((x - mean) ** 2 / (2.0 * std_dev ** 2)))
```

В этой формуле много всего интересного, но особенно важно то, что она принимает в качестве параметров среднее арифметическое и стандартное отклонение, а также значение *x*, для которого мы ищем правдоподобие.

Как и бета-распределение, описанное в главе 2, нормальное распределение является непрерывным. То есть, чтобы получить вероятность, нужно проинтегрировать его по интервалу значений *x* и найти площадь.

Однако на практике мы будем выполнять эти вычисления с помощью SciPy.

Функция распределения

На графике нормального распределения по вертикальной оси откладывается не вероятность, а правдоподобие того или иного значения. Чтобы найти вероятность, нужно рассмотреть заданный интервал, а затем вычислить площадь под кривой на этом интервале. Допустим, необходимо найти вероятность того, что золотистый ретривер весит от 62 до 66 фунтов. На рис. 3.7 показан интервал, для которого мы хотим найти площадь.

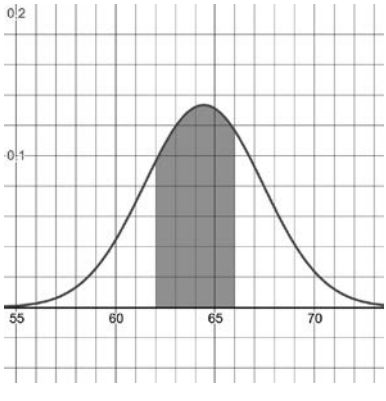


Рис. 3.7. Функция распределения, измеряющая вероятность для интервала от 62 до 66 фунтов

Мы уже решали эту задачу в главе 2 с бета-распределением, и здесь так же, как и для бета-распределения, существует функция распределения (CDF, cumulative distribution function). Будем придерживаться этого же подхода.

Как мы узнали в предыдущей главе, CDF — это площадь под кривой *до* конкретного значения *x* для заданного распределения. Давайте посмотрим на рис. 3.8, как выглядит CDF для нормального распределения массы тела золотистых ретриверов, и изобразим эту функцию вместе с PDF для сравнения.

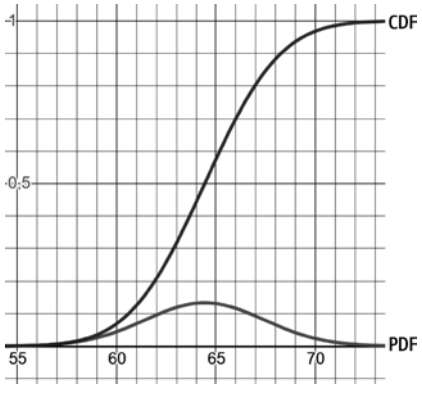


Рис. 3.8. Сравнение функции плотности вероятности (PDF) и функции распределения (CDF)

Обратите внимание, что эти два графика связаны между собой. CDF, которая представляет собой *S*-образную кривую (так называемую *сигмоиду*), показывает площадь под графиком PDF до указанного значения. Обратите внимание на рис. 3.9, что, когда мы рассматриваем область от минус бесконечности до 64,43 (среднее), значение CDF составляет ровно 0,5, или 50 %!

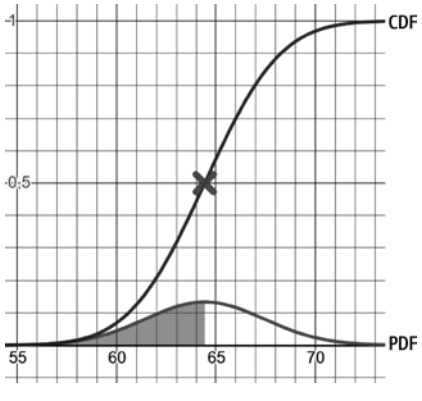


Рис. 3.9. Графики PDF и CDF для вероятности массы тела золотистых ретриверов вплоть до среднего

Мы знаем, что площадь до среднего значения равна 0,5 (то есть 50 %), благодаря тому, что нормальное распределение симметрично, и можно ожидать, что другая сторона колоколообразной кривой тоже будет покрывать 50 % всей площади.

Чтобы найти площадь вплоть до среднего значения массы 64,43 фунта на Python с помощью SciPy, используйте функцию `norm.cdf()`, как показано в примере 3.10.

Пример 3.10. Функция нормального распределения на Python

```
from scipy.stats import norm

mean = 64.43 # среднее
std_dev = 2.99 # стандартное отклонение

x = norm.cdf(64.43, mean, std_dev)

print(x) # выводит 0.5
```

Как и в главе 2, чтобы дедуктивно найти площадь для промежуточного интервала, можно вычесть одну площадь из другой. Если мы хотим найти вероятность встретить золотистого ретривера массой от 62 до 66 фунтов, следует вычислить площадь под кривой до 66 фунтов и вычесть из нее площадь до 62 фунтов, как показано на рис. 3.10.

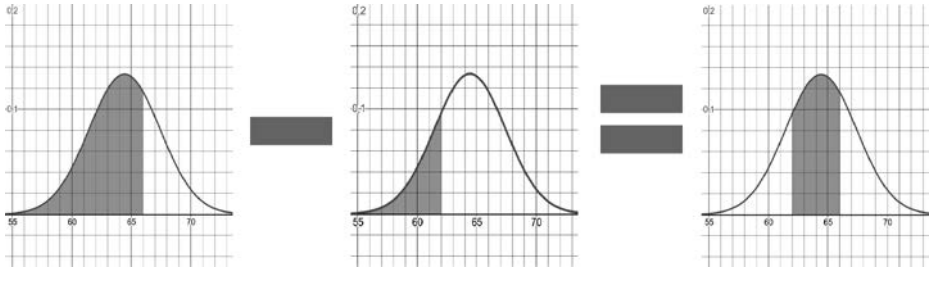


Рис. 3.10. Как найти вероятность промежуточного интервала

Сделать то же самое на Python с помощью библиотеки SciPy очень просто: достаточно вычесть одно значение CDF из другого, как показано в примере 3.11.

Пример 3.11. Вычисление вероятности промежуточного интервала с помощью CDF

```
from scipy.stats import norm

mean = 64.43
std_dev = 2.99

x = norm.cdf(66, mean, std_dev) - norm.cdf(62, mean, std_dev)

print(x) # выводит 0.4920450147062894
```

Итак, вероятность встретить золотистого ретривера массой от 62 до 66 фунтов равна 0,4920, или примерно 49,2 %.

PPF — функция, обратная к функции распределения

Когда мы начнем проверять гипотезы далее в этой главе, то столкнемся с ситуациями, когда известно значение функции распределения и нужно узнать, какое значение x ей соответствует. Конечно, это «CDF наоборот», поэтому нам понадобится функция, обратная по отношению к функцию распределения, которая поворачивает оси на 90°, как показано на рис. 3.11.

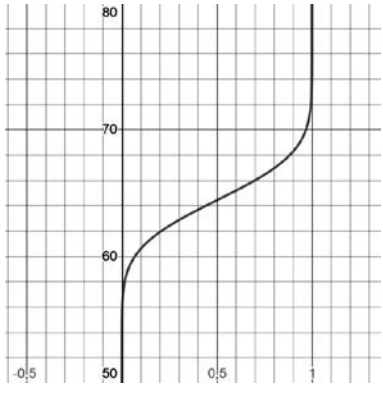


Рис. 3.11. Функция, обратная к функции распределения; она также называется PPF (percent-point function) или квантильной функцией

Таким образом, по заданной вероятности можно найти соответствующее значение x, а в SciPy для этого служит функция `norm.ppf()`. Например, я хочу узнать, какую массу тела имеют 95 % золотистых ретриверов (кроме оставшихся 5 % самых упитанных). Это легко сделать, если использовать «CDF наоборот», как в примере 3.12.

Пример 3.12. Использование функции, обратной к функции распределения, на Python

```
from scipy.stats import norm

x = norm.ppf(.95, loc=64.43, scale=2.99)
    # первый аргумент 0.95 — вероятность
    # loc — среднее, scale — стандартное отклонение

print(x) # 69.3481123445849
```

Таким образом, 95 % золотистых ретриверов весят по 69,348 фунта или меньше.

Кроме того, с помощью PPF можно генерировать случайные числа, которые соответствуют нормальному распределению. Если мы хотим создать модель, которая генерирует тысячу реалистичных масс тела золотистых ретриверов,

можно просто сгенерировать случайное значение в диапазоне от 0 до 1, передать его в PPF и возвратить значение массы, как показано в примере 3.13.

Пример 3.13. Генератор случайных чисел, которые соответствуют нормальному распределению

```
import random
from scipy.stats import norm

for i in range(0, 1000):
    random_p = random.uniform(0.0, 1.0)
    random_weight = norm.ppf(random_p, loc=64.43, scale=2.99)
    print(random_weight)
```

Конечно, NumPy и другие библиотеки умеют генерировать и такие случайные числа, которые не связаны ни с каким распределением, но в этом примере демонстрируется один из случаев, когда PPF (функция, обратная к функции распределения) может быть полезна.

Функция распределения и обратная к ней функция с нуля

О том, как реализовать CDF и PPF с нуля на Python, читайте в Приложении А.

Z-оценки

Нормальное распределение часто масштабируют так, чтобы среднее было равно 0, а стандартное отклонение — 1; в результате получается *стандартное нормальное распределение*. Эта конструкция позволяет легко сравнивать разброс разных нормальных распределений, даже если у них разные средние и дисперсии.

Особенно важный факт о стандартном нормальном распределении состоит в том, что оно выражает все значения x через стандартные отклонения. Преобразованные значения называются *Z-оценками* (*Z*-scores), или *стандартизованными оценками*. Чтобы преобразовать значение x в *Z*-оценку, используется простая формула масштабирования:

$$z = \frac{x - \mu}{\sigma}$$

Приведем пример. У нас есть два дома из двух разных районов. В районе A дом в среднем сто́ит 140 000 долларов со стандартным отклонением 3 000 долларов, а в районе B — 800 000 долларов со стандартным отклонением 10 000 долларов.

$$\mu_A = 140000;$$
$$\mu_B = 800000;$$

$$\sigma_A = 3000;$$
$$\sigma_B = 10000.$$

Теперь рассмотрим два дома — по одному из каждого района. Дом из района A стоит 150 000 долларов, а дом из района B — 815 000 долларов. Какой дом дороже по отношению к среднему уровню стоимости дома в своем районе?

$$x_A = 150000;$$
$$x_B = 815000.$$

Если выразить эти две величины через стандартные отклонения, то можно сравнить, на сколько стандартных отклонений каждая из них выше среднего значения для своего района. Для этого воспользуемся формулой *Z*-оценки:

$$z = \frac{x - \text{среднее}}{\text{стандартное отклонение}}$$
$$z_A = \frac{150000 - 140000}{3000} = 3,(3)$$
$$z_B = \frac{815000 - 800000}{10000} = 1,5$$

Таким образом, дом в районе A на самом деле гораздо дороже по меркам своего района, чем дом в районе B, потому что *Z*-оценки этих домов равны 3,(3) и 1,5 соответственно.

В примере 3.14 показано, как можно преобразовать значение *x*, которое происходит из распределения с известными средним и стандартным отклонением, в *Z*-оценку, и наоборот.

Пример 3.14. Преобразование *Z*-оценки в значение *x* и обратно

```
def z_score(x, mean, std):
    return (x - mean) / std

def z_to_x(z, mean, std):
    return (z * std) + mean

mean = 140_000
std_dev = 3_000
x = 150_000

# Преобразование из x в Z-оценку и обратно
z = z_score(x, mean, std_dev)
back_to_x = z_to_x(z, mean, std_dev)

print(f"Z-оценка: {z}") # Z-оценка: 3.333…
print(f"Преобразование обратно в x: {back_to_x}") # X: 150000.0
```

Функция `z_score()` принимает значение *x* и переводит его в количество стандартных отклонений, используя заданные среднее и стандартное отклонение. Функция `z_to_x()` принимает *Z*-оценку и преобразует ее обратно в значение *x*. Можно видеть, что эти функции алгебраически связаны: из одного и того же соотношения одна формула выражает *Z*-оценку, а другая — значение *x*. В этом примере мы преобразовываем значение *x*, равное 150 000, в *Z*-оценку, равную 3,333, а затем преобразовываем ее обратно в *x*.

КОЭФФИЦИЕНТ ВАРИАЦИИ

Коэффициент вариации — полезный инструмент, с помощью которого можно измерять разброс. Он позволяет количественно сравнивать разброс разных распределений. Вычислить его просто: разделите стандартное отклонение на среднее арифметическое. Вот формула, а также пример сравнения для двух районов:

$$cv = \frac{\sigma}{\mu}$$

$$cv_A = \frac{3000}{140000} = 0,0214$$

$$cv_B = \frac{10000}{800000} = 0,0125$$

Как видите, хотя дома в районе А дешевле, чем в В, в районе А больше разброс, а значит, большее ценовое разнообразие.

Статистический вывод

Описательная статистика, которой мы занимались до сих пор, — это то, что всем более или менее понятно. Однако когда мы переходим к статистическому выводу, в полной мере начинают проявляются абстрактные взаимосвязи между выборкой и совокупностью. Эти нюансы не стоит изучать в спешке, а лучше осваивать не торопясь и вдумчиво. Как уже говорилось ранее, люди устроены так, что склонны к смещениям и стремятся быстро делать выводы. Чтобы быть хорошим специалистом в data science, необходимо подавлять это природное желание и задумываться о том, что то или иное явление может объясняться поразному. Иногда вполне допустимо (возможно, даже лучше всего) предположить, что объяснения вообще нет, и вы наблюдаете лишь случайное совпадение.

Начнем с теоремы, которая лежит в основе всего статистического вывода.

Центральная предельная теорема

Нормальное распределение весьма полезно, в том числе потому, что оно часто встречается в природе например, в случае с массой тела взрослых золотистых ретриверов. Однако за пределами природных совокупностей оно проявляется в еще более интересном контексте. Когда мы начинаем исследовать достаточно большие выборки из какой-либо совокупности, то нормальное распределение все равно дает о себе знать, даже если сама эта совокупность ему не подчиняется.

Представим себе, что мы измеряем какой-то параметр совокупности, который распределен абсолютно равномерно случайным образом. Все значения между 0 и 1 равновероятны, и ни одно значение не более предпочтительно, чем любое другое. Но когда мы берем всё бóльшие выборки из этой совокупности, вычисляем среднее арифметическое каждой из них и строим гистограмму, происходит нечто удивительное. Запустите код на Python из примера 3.15 и посмотрите на график, который показан на рис. 3.12.

Пример 3.15. Изучение центральной предельной теоремы на Python

```
# Выборочные средние равномерного распределения образуют нормальное распределение
import random
import plotly.express as px

sample_size = 31
sample_count = 1000

# Центральная предельная теорема в действии:
# 1000 выборок, в каждой из которых по 31 случайному значению
# в диапазоне от 0 до 1
x_values = [(sum([random.uniform(0.0, 1.0)
                  for i in range(sample_size)]) / sample_size)
for _ in range(sample_count)]

y_values = [1 for _ in range(sample_count)]

px.histogram(x=x_values, y = y_values, nbins=20).show()
```

Подождите, а как так вышло, что равномерно случайные числа, которые сгруппировали в выборки по 31 элементу и затем усреднили, смогли образовать распределение, очень похожее на нормальное? Ведь любое значение равновероятно, разве нет? Разве распределение не должно быть плоским, а не колоколообразным?

Дело вот в чем. Отдельные числа в выборках сами по себе не создадут нормального распределения. Их распределение будет плоским, где любое значение равновероятно (так называемое *равномерное распределение*). Но если сгруппировать их в выборки и усреднить, они образуют нормальное распределение.

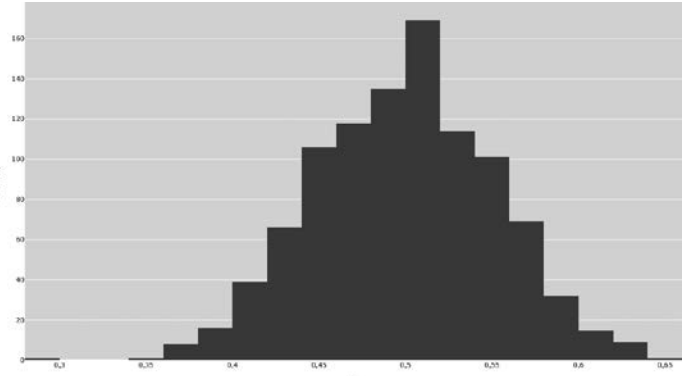


Рис. 3.12. Графическое представление средних выборочных для выборок из 31 элемента

Это следует из *центральной предельной теоремы*, которая утверждает, что если взять достаточно большие выборки одинакового размера из совокупности, вычислить среднее каждой из них и построить их распределение, то получатся любопытные результаты:

1. Среднее арифметическое выборочных средних равно среднему значению совокупности.
2. Если совокупность соответствует нормальному распределению, то и выборочные средние тоже ему соответствуют.
3. Если совокупность не соответствует нормальному распределению, но размер выборок больше 30, то выборочные средние все равно образуют распределение, близкое к нормальному[1].
4. Стандартное отклонение выборочных средних равно стандартному отклонению совокупности, деленному на квадратный корень из размера выборки:

$$\text{стандартное отклонение выборочных средних} = \frac{\text{генеральное стандартное отклонение}}{\sqrt{\text{размер выборки}}}$$

Почему все это важно? Эти результаты позволяют на основе выборок делать полезные выводы о генеральных совокупностях, которые не обязательно нормально распределены. Если в предыдущем коде попробовать меньшие размеры выборок — например, 1 или 2, то нормального распределения не получится. Но если выборки содержат по 31 и более элементов, то видно, что распределение сходится к нормальному, как показано на рис. 3.13.

[1] Точнее говоря, чем больше размер выборок, тем распределение выборочных средних ближе к нормальному. — *Примеч. науч. ред.*

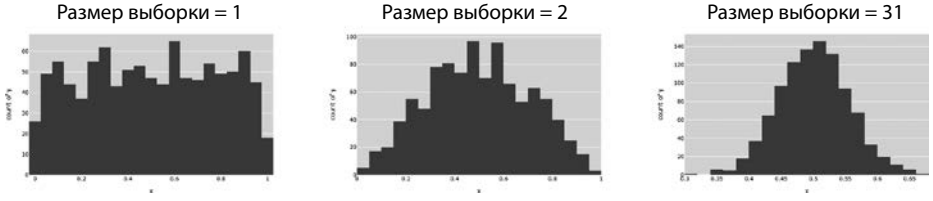


Рис. 3.13. Чем больше размеры выборок, тем распределение ближе к нормальному

Число 31 стало хрестоматийным в учебниках по статистике, потому что при таком размере выборки распределение выборочных показателей часто сходится к распределению совокупности, особенно если исследовать выборочные средние[1]. Когда в выборке меньше 31 элемента, приходится прибегать не к нормальному распределению, а к распределению Стьюдента, хвосты которого тем толще, чем меньше выборки[2]. Мы кратко поговорим об этом позже, но давайте условимся, что, когда пойдет речь о доверительных интервалах и проверке гипотез, мы будем считать, что в наших выборках не меньше 31 элемента.

КАКОВ ОПТИМАЛЬНЫЙ РАЗМЕР ВЫБОРКИ?

Хотя в учебниках часто пишут, что в выборках должно быть не меньше 31 элемента, чтобы выполнялись условия центральной предельной теоремы и получалось нормальное распределение, это не всегда так. Бывают случаи, когда требуется более крупная выборка, — например, если распределение генеральной совокупности асимметрично или мультимодально (то есть имеет несколько пиков, а не один в области среднего значения).

Таким образом, если вы не уверены в том, какому распределению соответствуют исходные данные, то лучше использовать большие выборки. Подробнее об этом можно прочитать в статье Рамси (Ramsey) и Ангера (Unger) «The Central Limit Theorem: What's Large Enough» (https://oreil.ly/IZ4Rk).

---

[1] Во многих источниках число 30 (или 31) считается традиционным минимальным размером выборки, потому что в реальных задачах прикладной статистики чаще всего оказывается достаточно выборок из 30 элементов, чтобы с распределением выборочных средних можно было работать как с нормальным. Однако это просто эмпирическое правило, а не догма; в зависимости от конкретной задачи оптимальный размер выборок может быть больше или меньше 30. — *Примеч. науч. ред.*

[2] В докомпьютерную эпоху прикладная статистика предпочитала иметь дело с нормальным распределением, потому что для него был разработан богатый математический аппарат, широкий ассортимент методов, были широко доступны математические таблицы и другие практические инструменты. Распределение Стьюдента и другие распределения требовали больше кропотливой вычислительной работы. В наше время компьютеры позволяют легко оперировать любыми мыслимыми распределениями вероятностей, так что больше нет насущной необходимости во что бы то ни стало сводить задачи к нормальному распределению. — *Примеч. науч. ред.*

Доверительные интервалы

Возможно, вам приходилось слышать термин «доверительный интервал», который часто сбивает с толку тех, кто начинает изучать статистику. *Доверительный интервал* — это инструмент, который показывает, насколько мы уверены в том, что выборочное среднее (или другой параметр) попадает в ту или иную окрестность генерального среднего.

На основе выборки из 31 золотистого ретривера со средней массой тела 64,408 фунта и стандартным отклонением 2,05 я на 95 % уверен, что генеральное среднее лежит между 63,686 и 65,1296. Откуда это известно? Сейчас я вам покажу, а если вы запутаетесь, вернитесь к этому абзацу и вспомните, чего мы пытаемся добиться. Я выделил эту фразу не просто так!

Сначала я выбираю *доверительный уровень (LOC, level of confidence)* — желаемую вероятность для окрестности генерального среднего. Я хочу быть на 95 % уверен в том, что выборочное среднее попадает в интервал вокруг генерального среднего, который я буду вычислять. Эти 95 % и есть мой доверительный уровень. Можно воспользоваться центральной предельной теоремой и сделать вывод о том, каков искомый интервал. Для этого понадобится *критическое значение Z-оценки*, которое определяет концы симметричного интервала, охватывающего вероятность 95 % в центральной части стандартного нормального распределения, как показано на рис. 3.14.

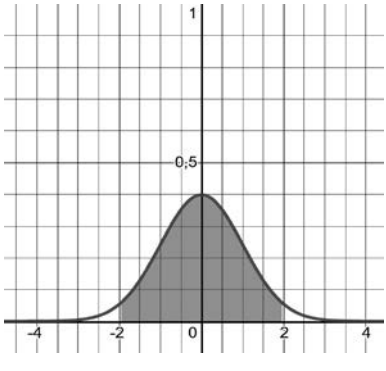


Рис. 3.14. Симметричная вероятность 95 % в центре стандартного нормального распределения

Как вычислить этот симметричный интервал, которому соответствует 0,95 всей площади под графиком? Это проще представить себе на концептуальном уровне, чем вникать в механику расчетов. Возможно, вы подсознательно захотите применить CDF, но потом поймете, что все не так просто.

Сначала нужно использовать PPF (функцию, обратную к CDF). Логика подсказывает, что для того, чтобы получить симметричную область площадью 95 %

в центре распределения, нужно отсечь хвосты, которые покрывают оставшиеся 5 % площади по бокам. Разделив эти 5 % пополам, мы получим по 2,5 % площади с каждой стороны. Таким образом, площади, для которых мы хотим найти значения *x*, равны 0,025 и 0,975, как показано на рис. 3.15.

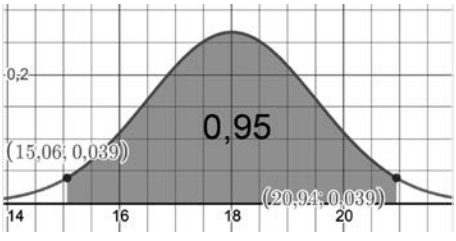


Рис. 3.15. Нужно найти значения *x*, которые соответствуют площадям 0,025 и 0,975

Если найти значения *x* для площадей 0,025 и 0,975, мы получим центральный интервал, которому соответствует 95 % площади. Затем нужно вычислить соответствующие нижнее и верхнее значения *Z*-оценки. Помните, что мы используем стандартное нормальное распределение, поэтому эти значения будут отличаться только знаком. Давайте вычислим это на Python, как показано в примере 3.16.

Пример 3.16. Получение критических *Z*-оценок

```
from scipy.stats import norm

def critical_z_value(p):
    norm_dist = norm(loc=0.0, scale=1.0)
    left_tail_area = (1.0 - p) / 2.0
    upper_area = 1.0 - ((1.0 - p) / 2.0)
    return norm_dist.ppf(left_tail_area), norm_dist.ppf(upper_area)

print(critical_z_value(p=.95))
# (-1.959963984540054, 1.959963984540054)
```

Итак, мы получили ±1,95996; это и есть критические *Z*-оценки, которые ограничивают область с вероятностью 95 % в центре стандартного нормального распределения. Затем с помощью центральной предельной теоремы нужно вычислить *погрешность* (*E*) — такой интервал вокруг выборочного среднего, чтобы в этом интервале находилось генеральное среднее при данном доверительном уровне. Напомню, что в нашей выборке из 31 золотистого ретривера среднее равно 64,408, а стандартное отклонение — 2,05. Погрешность вычисляется по такой формуле:

$$E = \pm z_c \frac{s}{\sqrt{n}}$$

$$E = \pm 1,95996 \times \frac{2,05}{\sqrt{31}} = \pm 0,72164$$

Если добавить погрешность с обеих сторон от выборочного среднего, мы получим доверительный интервал!

$$95\%\text{-ный доверительный интервал} = 64,408 \pm 0,72164$$

В примере 3.17 показано, как вычислить доверительный интервал на Python от начала до конца.

Пример 3.17. Вычисление доверительного интервала на Python

```
from math import sqrt
from scipy.stats import norm

def critical_z_value(p):
    norm_dist = norm(loc=0.0, scale=1.0)
    left_tail_area = (1.0 - p) / 2.0
    upper_area = 1.0 - ((1.0 - p) / 2.0)
    return norm_dist.ppf(left_tail_area), norm_dist.ppf(upper_area)

def confidence_interval(p, sample_mean, sample_std, n):
    # Размер выборки должен быть больше 30
    lower, upper = critical_z_value(p)
    lower_ci = lower * (sample_std / sqrt(n))
    upper_ci = upper * (sample_std / sqrt(n))

    return sample_mean + lower_ci, sample_mean + upper_ci

print(confidence_interval(p=.95, sample_mean=64.408, sample_std=2.05, n=31))
# (63.68635915701992, 65.12964084298008)
```

Интерпретировать это можно так: «На основе выборки из 31 золотистого ретривера со средней массой тела 64,408 фунта и стандартным отклонением 2,05 я на 95 % уверен, что генеральное среднее лежит между 63,686 и 65,1296». Именно так описывается доверительный интервал.

Интересно отметить, что чем больше n, тем у́же доверительный интервал в формуле погрешности! Это логично, потому что чем больше выборка, тем больше мы уверены в том, что генеральное среднее лежит на меньшем участке, — поэтому он и называется доверительным интервалом.

Здесь стоит сделать важное предостережение: чтобы эти расчеты сработали, в выборке должно быть не менее 31 элемента: это связано с центральной предельной теоремой. Если необходимо вычислить доверительный интервал для меньшей выборки, придется использовать распределение с большей дисперсией (более широкие хвосты отражают бо́льшую неопределенность). Для этого служит распределение Стьюдента, которое мы рассмотрим в конце этой главы.

В главе 5 вы узнаете, как использовать доверительные интервалы в линейной регрессии.

p-значения

Когда говорят, что нечто *статистически значимо*, что имеется в виду? Мы часто слышим эти слова, но что они означают с математической точки зрения? Формально это понятие связано с так называемым *p*-значением, которое многим трудно осмыслить. Но я думаю, что в *p*-значениях будет проще разобраться, если проследить, откуда они взялись. Хотя этот пример неидеален, он поможет донести важные идеи.

В 1925 году математик Рональд Фишер (Ronald Fisher) был на вечеринке. Одна из его коллег Мюриэль Бристоль (Muriel Bristol) заявила, что если просто попробует чай, то сможет определить, что было налито в чашку раньше — чай или молоко. Заинтригованный этим утверждением, Фишер тут же поставил эксперимент.

Он приготовил восемь чашек чая, в четыре из которых сначала налил молоко, а в четыре других — чай. Затем он предложил их вперемешку своей коллеге-дегустатору, чтобы она определила порядок наливания каждой из них. Как ни удивительно, Мюриэль классифицировала все чашки правильно, и вероятность того, что ей просто случайно повезло, составила 1 к 70, или 0,01428571.

Эта вероятность 1,4 % и есть та величина, которая обычно называется *p-значением*: это вероятность того, что что-то произошло случайно, а не из-за некой систематической причины. Не вдаваясь в комбинаторные расчеты, можно сказать, что вероятность того, что Мюриэль чисто случайно угадала все чашки, составляет 1,4 %. О чем это говорит?

Когда мы ставим эксперимент — будь то в рамках исследования того, толстеют ли от обезжиренных пончиков или заболевают ли раком от проживания вблизи линий электропередач, — всегда нужно учитывать возможность того, что имеет место случайное совпадение. В эксперименте Фишера была вероятность 1,4 % того, что Мюриэль просто угадала все чашки, и точно так же в других экспериментах всегда есть вероятность того, что случайность, будто игровой автомат, просто предоставила нам удачную комбинацию. Это позволяет сформулировать *нулевую гипотезу* (H_0), согласно которой изучаемая переменная никак не влияет на ход эксперимента, а любые положительные результаты объясняются случайным везением. *Альтернативная гипотеза* (H_1) утверждает, что положительный результат вызван изучаемой переменной, которую называют *управляемой*.

Традиционно порогом статистической значимости считается *p*-значение, равное 5 % (то есть 0,05)[1]. Поскольку 0,014 меньше, чем 0,05, это позволяет отбросить

[1] Как и в случае с минимальным объемом выборки, уровень значимости 5 % — это просто широко распространенная традиция, которой придерживаются во многих статистических отчетах и научных исследованиях. В других случаях выбирают другие уровни значимости (например, 1 или 0,1 %) или пользуются методами, которые обходятся без порогового *p*-значения. — *Примеч. науч. ред.*

нулевую гипотезу о том, что Мюриэль случайно угадала все чашки. Следовательно, есть основания в пользу альтернативной гипотезы: по-видимому, Мюриэль действительно умела определять, что наливают раньше — чай или молоко.

В этом примере с чаепитием мы не учли одно обстоятельство: когда мы вычисляем *p*-значение, нас интересует вероятность не только указанного события, но и всех еще более редких событий. Об этом пойдет речь в следующем примере, где фигурирует нормальное распределение.

Проверка гипотез

Проведенные ранее исследования показали, что среднее время восстановления после простуды составляет 18 дней со стандартным отклонением в 1,5 дня и соответствует нормальному распределению.

Это значит, что с вероятностью около 95 % восстановление займет от 15 до 21 дня, как показано на рис. 3.16 и в примере 3.18.

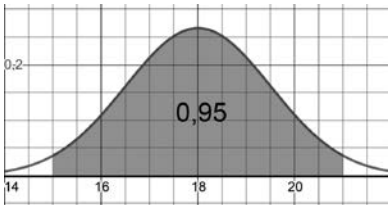


Рис. 3.16. Вероятность восстановления в течение 15–21 дня составляет 95 %

Пример 3.18. Вычисление вероятности восстановления в течение 15–21 дня

```
from scipy.stats import norm

# Среднее время восстановления после простуды — 18 дней,
# стандартное отклонение — 1,5 дня
mean = 18
std_dev = 1.5

# С вероятностью 95 % время восстановления занимает 15–21 день
x = norm.cdf(21, mean, std_dev) - norm.cdf(15, mean, std_dev)

print(x) # 0.9544997361036416
```

Поскольку оставшаяся вероятность составляет 5 %, можно сделать вывод, что существует вероятность 2,5 % того, что восстановление займет больше 21 дня, и 2,5 % того, что оно займет меньше 15 дней. Запомните этот факт, потому что в дальнейшем он будет играть ключевую роль! На него опирается наше *p*-значение.

Теперь предположим, что группа из 40 пациентов получила новое экспериментальное лекарство, и им потребовалось в среднем 16 дней, чтобы восстановиться после простуды, как показано на рис. 3.17.

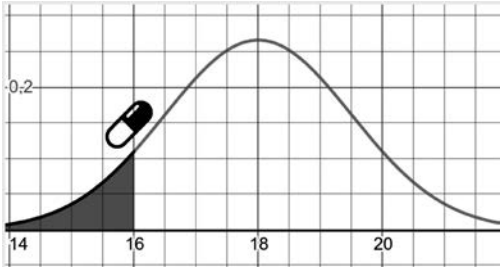


Рис. 3.17. Группе, принимавшей лекарство, потребовалось в среднем 16 дней на восстановление

Помогло ли лекарство? Если хорошенько подумать, то можно понять, что нас интересует вот что: показало ли лекарство статистически значимый результат? Или лекарство не подействовало, а выздоровление за 16 дней оказалось случайным совпадением в испытуемой группе? Первый из этих вопросов определяет альтернативную гипотезу, а второй — нулевую.

Существует два способа узнать значимость: односторонний и двусторонний тест[1]. Начнем с одностороннего.

Односторонний тест

Когда используется *односторонний тест*, нулевую и альтернативную гипотезы обычно формулируют в виде неравенств. Мы выдвигаем гипотезы относительно генерального среднего и рассматриваем варианты, что оно либо больше или равно 18 (нулевая гипотеза H_0), либо строго меньше 18 (альтернативная гипотеза H_1):

$$H_0 : \text{генеральное среднее} \geq 18;$$
$$H_1 : \text{генеральное среднее} < 18.$$

Чтобы отвергнуть нулевую гипотезу, нужно показать, что выборочное среднее пациентов, которые принимали лекарство, не было результатом случайного стечения обстоятельств. Поскольку p-значение 0,05 или меньше традиционно считается статистически значимым, мы возьмем его в качестве порогового (рис. 3.17). Вычислив PPF на Python, как показано на рис. 3.18 и в примере 3.19, мы обнаружим, что площадь 0,05 в левом хвосте примерно соответствует 15,53 дням выздоровления.

[1] В отечественной литературе статистические тесты традиционно назывались *критериями*. В этой книге используется термин *тест*, потому что он стал общепринятым среди русскоязычных специалистов по data science. — *Примеч. науч. ред.*

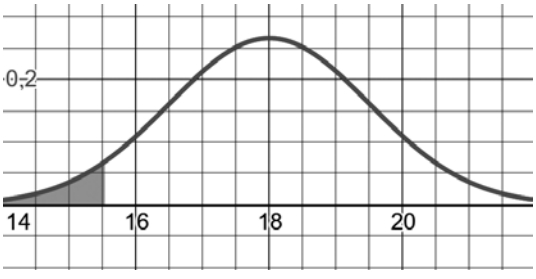


Рис. 3.18. Значение *x*, которому соответствуют первые 5 % площади под кривой

Пример 3.19. Вычисление значения *x*, которому соответствуют первые 5 % площади под кривой[2]

```
from scipy.stats import norm

# Среднее время восстановления после простуды — 18 дней,
# стандартное отклонение — 1,5 дня
mean = 18
std_dev = 1.5

# Какому значению x соответствуют первые 5 % площади под кривой?
x = norm.ppf(.05, mean, std_dev)

print(x) # 15.53271955957279
```

---

[2] В книгу вкралась досадная ошибка: в примерах 3.19–3.22 и в сопутствующем тексте ни расчеты, ни конечный результат не зависят от размера выборки ($n = 40$ пациентов), в то время как здравый смысл подсказывает, что доверительный интервал и *p*-значение должны уменьшаться с ростом *n*.

На самом деле в этих расчетах вместо стандартного отклонения (переменная `std_dev` в коде) должна фигурировать стандартная ошибка среднего $SE_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$. Поскольку стандартное отклонение генеральной совокупности σ неизвестно, его можно оценить выборочным стандартным отклонением *s*.

Исправить ситуацию можно следующим образом:

- *Импортировать функцию квадратного корня:* `from numpy import sqrt`
- Объявить переменную для размера выборки: `n = 40`
- В расчетах вместо `std_dev` использовать `std_dev/sqrt(n)`, например: x = norm.ppf(.05, mean, std_dev/sqrt(n))

Впрочем, в этом случае результаты тестов будут другими: и односторонний, и двусторонний тест покажут статистическую значимость на уровне 95 %, и это позволит отвергнуть нулевую гипотезу и сделать вывод, что лекарство действительно помогает восстанавливаться после простуды.

Можно выйти из положения по-другому: представить себе, будто под `std_dev` подразумевается не выборочное стандартное отклонение s, а стандартная ошибка среднего $SE_{\bar{x}}$. Тогда выборочное стандартное отклонение $s = SE_{\bar{x}} \times \sqrt{n} = 1,5 \times \sqrt{40} \approx 9,5$. В этом случае все расчеты и выводы тестов остаются в силе. — *Примеч. науч. ред.*

Таким образом, если бы среднее выборочное время восстановления в нашей группе пациентов составило 15,53 дня или меньше, то можно было бы считать, что лекарство показало статистически значимый эффект. Однако в нашей выборке среднее составляет 16 дней и не попадает в область отклонения нулевой гипотезы. Таким образом, как показано на рис. 3.19, тест на статистическую значимость не прошел.

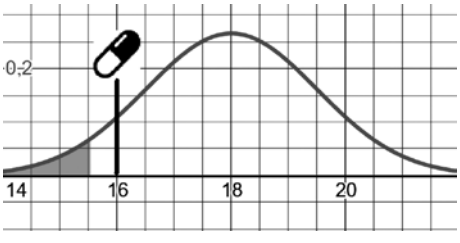


Рис. 3.19. Мы не смогли доказать, что результат испытания лекарства является статистически значимым

Площадь области до отметки «16 дней» — это *p*-значение нашего теста. Оно составляет 0,0912 и вычисляется на Python в примере 3.20.

Пример 3.20. Вычисление одностороннего *p*-значения

```
from scipy.stats import norm

# Среднее время восстановления после простуды — 18 дней,
# стандартное отклонение — 1,5 дня
mean = 18
std_dev = 1.5

# Вероятность для 16 и менее дней
p_value = norm.cdf(16, mean, std_dev)

print(p_value) # 0.09121121972586788
```

Поскольку *p*-значение 0,0912 превышает порог статистической значимости 0,05, нельзя считать испытание лекарства успешным и отвергнуть нулевую гипотезу.

Двусторонний тест

Предыдущий тест называется односторонним, потому что он искал статистическую значимость только в одном хвосте распределения вероятности. Однако зачастую надежнее и грамотнее использовать двусторонний тест. В дальнейшем мы подробно выясним, почему так, но сначала давайте научимся его выполнять.

Для двустороннего теста нулевая и альтернативная гипотезы оформляются в виде утверждений «равно» и «не равно». В нашем испытании лекарства нулевая гипотеза будет предполагать, что среднее время восстановления составляет 18 дней. А альтернативная гипотеза состоит в том, что из-за нового лекарства это время не равно 18 дням:

$$H_0 : \text{генеральное среднее} = 18;$$

$$H_1 : \text{генеральное среднее} \neq 18.$$

Обратите внимание, чем этот тест отличается от одностороннего. Мы формулируем альтернативную гипотезу, чтобы проверить не то, что лекарство уменьшает время восстановления после простуды, а то, что оно оказывает *хоть какой-нибудь* эффект. В частности, мы проверяем, не становится ли восстановление дольше с лекарством, чем без него. Есть ли в этом смысл? Запомним этот момент, чтобы вернуться к нему позже.

Естественно, это значит, что порог статистической значимости *p*-значения распространяется на оба хвоста распределения, а не только на один. Если мы проверяем значимость на уровне 5 %, нужно разделить его пополам и отсечь по 2,5 % от каждого хвоста. Если среднее время восстановления после болезни попадет в одну из отсеченных областей, то тест окажется успешным, и нужно будет отвергнуть нулевую гипотезу (рис. 3.20).

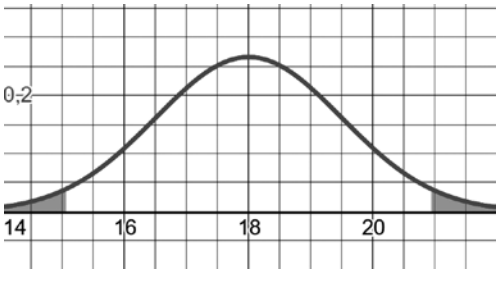


Рис. 3.20. Двусторонний тест

Значения *x* для правого и левого хвоста кривой равны соответственно 15,06 и 20,93. Таким образом, если среднее выборочное меньше левого или больше правого значения, то придется отвергнуть нулевую гипотезу. Эти два значения рассчитываются с помощью PPF, как показано на рис. 3.21 и в примере 3.21. Обратите внимание: чтобы вычислить *x*, от которого начинается правый хвост, мы берем значение 0,95 и добавляем к нему половину порога значимости 0,025, в итоге получая 0,975.

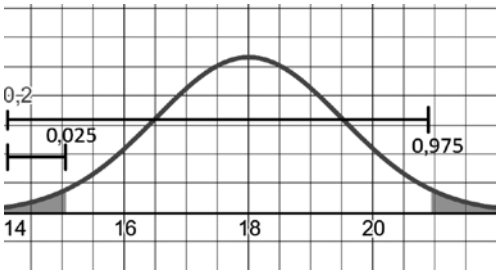


Рис. 3.21. Вычисление области, содержащей 95 % площади под кривой нормального распределения

Пример 3.21. Вычисление интервала для статистической значимости 5 %

```
from scipy.stats import norm

# Среднее время восстановления после простуды — 18 дней,
# стандартное отклонение — 1,5 дня
mean = 18
std_dev = 1.5

# Какому значению x соответствуют первые 2,5 % площади под кривой?
x1 = norm.ppf(.025, mean, std_dev)

# Какому значению x соответствуют первые 97,5 % площади под кривой?
x2 = norm.ppf(.975, mean, std_dev)

print(x1) # 15.060054023189918
print(x2) # 20.93994597681008
```

Выборочное среднее для группы испытуемых равно 16 — то есть не меньше 15,06 и не больше 20,9399. Таким образом, как и в случае одностороннего теста, по-прежнему нельзя отвергать нулевую гипотезу. Эффект от тестируемого лекарства все еще не показывает статистической значимости, что видно на рис. 3.22.

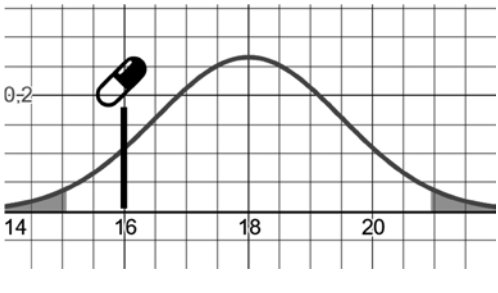


Рис. 3.22. Двусторонний тест не показал статистической значимости

Но что такое *p*-значение? Вот здесь-то и проявляется интересная особенность двусторонних тестов. Нашему *p*-значению соответствует не только область слева от 16, но и симметричная область в правом хвосте. Поскольку 16 на 2 дня меньше среднего, мы также учтем область справа от 20 (на 2 дня больше среднего, рис. 3.23). Таким образом, в закрашенные области входит вероятность порогового или более редкого события с обеих сторон колоколообразной кривой.

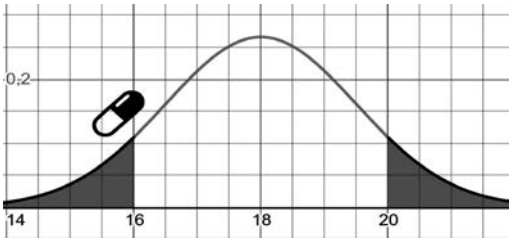


Рис. 3.23. *p*-значение добавляет симметричные области статистической значимости

Суммируя *p*-значения для обеих этих областей, мы получаем итоговое *p*-значение 0,1824. Оно намного больше, чем 0,05, поэтому определенно не проходит по порогу 0,05 (см. пример 3.22).

Пример 3.22. Вычисление двустороннего *p*-значения

```
from scipy.stats import norm

# Среднее время восстановления после простуды — 18 дней,
# стандартное отклонение — 1,5 дня
mean = 18
std_dev = 1.5

# Вероятность восстановиться менее чем за 16 дней
p1 = norm.cdf(16, mean, std_dev)

# Вероятность восстановиться более чем за 20 дней
p2 = 1.0 - norm.cdf(20, mean, std_dev)

# p-значение для обоих хвостов
p_value = p1 + p2

print(p_value) # 0.18242243945173575
```

Зачем же в двустороннем тесте мы добавили симметричную область на противоположном конце распределения? Возможно, это не самая интуитивно понятная мысль, но сначала вспомним, как мы сформулировали гипотезы:

$$H_0 : \text{генеральное среднее} = 18;$$
$$H_1 : \text{генеральное среднее} \neq 18.$$

Если тест должен выбрать один из вариантов «равно 18» и «не равно 18», то нужно учесть любую вероятность, которая соответствует равному или меньшему значению с обеих сторон. Ведь мы пытаемся доказать значимость, а значимыми будут все события, вероятность которых равна пороговому *p*-значению или меньше его. Мы не учитывали эти соображения в одностороннем тесте, где был выбор из вариантов «больше или равно» и «меньше». Но когда мы имеем дело с вариантами «равно» и «не равно», следует уделять внимание обеим сторонам.

Каковы же практические последствия двустороннего тестирования? Как оно влияет на то, отвергаем ли мы нулевую гипотезу? Поставим вопрос так: какой из тестов устанавливает более строгий порог значимости? Можно заметить, что даже если наша цель — показать, что какой-то параметр мог уменьшиться (например, время восстановления у тех, кто принимал лекарство), то переформулировка гипотезы так, чтобы отразить любой эффект (как увеличение, так и уменьшение), порождает более высокий порог значимости. Если порог значимости составляет 0,05, то наш односторонний тест с *p*-значением 0,0912 был ближе к успеху, чем двусторонний тест, в котором *p*-значение оказалось вдвое выше (0,182).

Это значит, что при двустороннем тесте сложнее отклонить нулевую гипотезу, а чтобы признать тест успешным, требуются более серьезные доказательства. Еще подумайте вот о чем: а что, если лекарство могло усугубить простуду и увеличить срок восстановления? Было бы полезно учесть вероятность такого сценария и рассмотреть отклонения в этом направлении. Именно поэтому двусторонние тесты предпочтительнее в большинстве случаев. Они надежнее, чем односторонние, и не смещают гипотезу только в одну сторону.

В главах 5 и 6 мы еще не раз обратимся к проверке гипотез и *p*-значениям.

ОСТЕРЕГАЙТЕСЬ *P*-ХАКИНГА

В научном сообществе все больше внимания уделяется проблеме, которая называется *p-хакингом* (а также *выуживанием* или даже *процеживанием p*-значений), когда исследователи выискивают данные, обеспечивающие статистически значимые *p*-значения (0,05 или меньше). Это несложно сделать с помощью больших данных, машинного обучения и дата-майнинга, когда можно прошерстить сотни или тысячи переменных, пока между ними не обнаружатся статистически значимые связи, которые возникли в результате удачных совпадений.

Почему так много исследователей занимаются *p*-хакингом? Вероятно, многие из них сами не осознают этого. Легко настраивать и подкручивать модель, пропуская «зашумленные» данные и меняя параметры, пока она не даст «правильный» результат. Другие просто испытывают давление со стороны научных и промышленных организаций, которые требуют не объективных, а выгодных результатов.

Например, вас нанимает компания «Вкусный колос», чтобы вы исследовали, не вызывают ли конфеты «Сахарок в шоколаде» диабет. Как вы думаете — если вы предоставите честный анализ, вас пригласят в следующий раз? А что делать, если руководитель поручает вам подготовить прогноз продаж на сумму 15 млн долларов в следующем квартале, чтобы запустить новый продукт? Вы никак не контролируете, какими будут продажи, но от вас требуют модель, которая даст заранее определенный результат. В худшем случае вас могут даже привлечь к ответственности, если прогноз окажется неверным. Это несправедливо, но так бывает!

Именно поэтому дипломатичность и другие социальные навыки могут сыграть решающую роль в карьере специалиста по data science. Если вы можете изложить непростую и неудобную историю так, чтобы она вызвала конструктивный интерес у аудитории, — это уже половина успеха. Только не забывайте об управленческом климате в организации и всегда предлагайте альтернативные решения. Если вы попали в безвыигрышную ситуацию, когда вас просят заниматься *p*-хакингом и могут привлечь к ответственности, если это приведет к неприятным последствиям, то не задумываясь меняйте место работы!

Распределение Стьюдента: работа с малыми выборками

Давайте вкратце рассмотрим, как работать с небольшими выборками (30 элементов или меньше). Это понадобится, когда мы займемся линейной регрессией в главе 5. Независимо от того, рассчитываем ли мы доверительные интервалы или проверяем гипотезы, — если в выборке не больше 30 элементов, то вместо нормального распределения лучше использовать *распределение Стьюдента* (T-distribution). Оно похоже на нормальное, но имеет более широкие хвосты, которые отражают бóльшую дисперсию и неопределенность. На рис. 3.24 показано нормальное распределение (пунктирная линия) и распределение Стьюдента с одной степенью свободы (сплошная линия).

Чем меньше выборка, тем шире хвосты распределения Стьюдента. Но что интересно, по мере приближения к 31 элементу оно становится практически неотличимым от нормального распределения, что наглядно отражает суть центральной предельной теоремы.

В примере 3.23 показано, как найти *критическое значение* распределения Стьюдента для 95 %-ной уверенности. Оно может пригодиться для того, чтобы строить

доверительные интервалы и проверять гипотезы, когда в выборке не более 30 элементов. Концептуально это то же самое, что и критическое *Z*-значение, но мы используем распределение Стьюдента вместо нормального. Чем меньше выборка, тем больше неопределенность, а значит, и интервал.

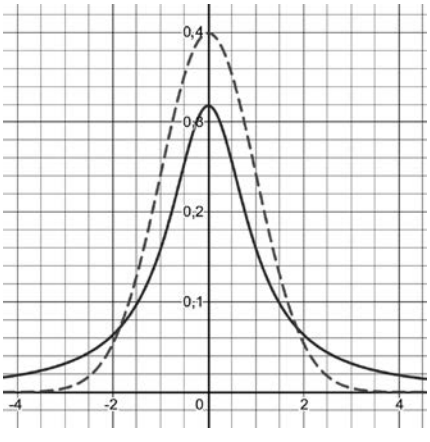


Рис. 3.24. Нормальное распределение и распределение Стьюдента. Обратите внимание на широкие хвосты

Пример 3.23. Вычисление критической области с помощью распределения Стьюдента

```
from scipy.stats import t

# вычисляем интервал критических значений для уверенности 95 %
# (25 элементов в выборке)

n = 25
lower = t.ppf(.025, df=n-1)
upper = t.ppf(.975, df=n-1)

print(lower, upper)
# -2.063898561628021 2.0638985616280205
```

Заметим, что `df` — это количество степеней свободы, и оно должно быть на единицу меньше, чем размер выборки.

НЕ ТОЛЬКО СРЕДНЕЕ АРИФМЕТИЧЕСКОЕ

С помощью доверительных интервалов и проверки гипотез можно оценивать и другие параметры, помимо среднего значения, — например, дисперсию (или стандартное отклонение), а также доли (например, «60 %

респондентов отмечают, что после ежедневной часовой прогулки у них улучшается настроение»). Для этого требуется не нормальное распределение, а другие — например, распределение хи-квадрат. Эти темы выходят за рамки этой книги, но надеюсь, что вы уверенно овладели необходимыми основами, чтобы разобраться в этих областях, если понадобится.

Как бы то ни было, в главах 5 и 6 мы снова встретимся с доверительными интервалами и проверкой гипотез.

Кое-что о больших данных и ошибке меткого стрелка

И последнее, что хотелось бы обсудить под конец этой главы. Как мы уже говорили, когда мы обосновываем свои выводы, всегда следует учитывать роль случайных совпадений. К сожалению, с тех пор, как появились большие данные, машинное обучение и другие инструменты обработки данных, научный метод внезапно превратился в практику, которая работает задом наперед. Это может быть опасно. Почему это так — позвольте продемонстрировать на примере из книги Гэри Смита (Gary Smith) «Standard Deviations» (издательство Overlook Press).

Представим, что я вытягиваю четыре игральные карты из обычной колоды. Это не какая-то игра, и нет никакой другой цели, кроме как вытянуть четыре карты и рассмотреть их. Мне достаются две десятки, тройка и двойка. «Интересно, — говорю я. — Я получил две десятки, тройку и двойку. Нет ли в этом закономерности? Если я вытяну еще четыре карты, будут ли это тоже две последовательные очковые карты и пара? Какая модель лежит в основе этого эксперимента?»

Видите, что я сделал? Я взял совершенно случайное явление и не только предположил закономерности, но и попытался построить на их основе прогностическую модель. Между тем я изначально не ставил своей задачей вытянуть именно такую структуру из четырех карт. Я пронаблюдал ее *после того*, как она образовалась.

Это именно то, чем постоянно грешит дата-майнинг: он выявляет случайно образовавшиеся структуры в случайных событиях. Когда вам доступны огромные объемы данных и быстрые алгоритмы, которые ищут закономерности, нетрудно отыскать результаты, которые выглядят как закономерности, но на самом деле являются случайными совпадениями.

Это все равно, как если бы я стрелял из пистолета по стене, а потом рисовал мишень вокруг отверстия и приглашал друзей, чтобы продемонстрировать свою потрясающую меткость. Глупо, правда? Однако многие деятели в области data

science фактически занимаются этим день ото дня, и эта практика известна как *ошибка меткого стрелка*. Они начинают перемалывать данные без определенной цели, натыкаются на что-то редкое, а затем провозглашают, будто найденное ими каким-то образом создает предсказательную ценность.

Проблема заключается в том, что, согласно закону очень больших чисел, какие-то из редких событий наверняка произойдут — просто мы не знаем, какие именно. Сталкиваясь с редкими событиями, мы обращаем на них особое внимание и даже строим догадки о том, что могло их вызвать. Однако на самом деле суть вот в чем: вероятность того, что каждый конкретный человек выиграет в лотерею, крайне мала, но тем не менее *кто-то* обязательно выиграет. Разве нас удивляет сам факт того, что кто-то оказался победителем? Если никто не предсказал победителя, то ничего значимого не произошло, кроме того, что случайному человеку случайно повезло.

Эти соображения относятся и к корреляциям, которые мы рассмотрим в главе 5. Если у вас есть огромный набор данных с тысячами переменных, разве трудно в нем отыскать статистически значимые результаты с p-значением 0,05? Проще простого! Я найду тысячи таких примеров. Например, я докажу, что количество фильмов с Николасом Кейджем коррелирует с количеством утонувших в бассейне за год (https://oreil.ly/eGxm0).

Поэтому, чтобы не превратиться в «меткого стрелка» и не впасть в заблуждения, связанные с большими данными, старайтесь ответственно проверять гипотезы и собирать данные для этой цели. Если вы все-таки прибегаете к дата-майнингу, отслеживайте свежие данные, чтобы проверить, остаются ли ваши выводы в силе. Наконец, всегда учитывайте возможность случайного совпадения; если какое-то явление не удается объяснить здравым смыслом, то, скорее всего, это случайность.

Мы научились выдвигать гипотезы до того, как собирать данные, но любители дата-майнинга грешат тем, что сначала собирают данные, а потом подбирают к ним гипотезы. Как это ни парадоксально, мы часто действуем более объективно, когда сначала выдвигаем гипотезу, потому что после этого мы ищем данные, чтобы целенаправленно доказать или опровергнуть ее.

Заключение

В этой главе мы изучили много нового, и вы можете гордиться, что продвинулись так далеко. Это была, пожалуй, одна из самых трудных тем в этой книге! Мы не только освоили описательную статистику от среднего арифметического до нормального распределения, но и рассмотрели доверительные интервалы и проверку гипотез.

Надеюсь, что теперь вы воспринимаете данные немного по-другому. Это моментальные снимки отдельных эпизодов, которые не претендуют на то, чтобы полностью отражать реальность. Сами по себе данные не особо полезны; чтобы сделать на их основе значимые выводы, нужно представлять себе контекст, проявлять любопытство и анализировать, откуда они взялись. Мы изучили, как описывать данные, а также как на основе выборки делать выводы о более широкой генеральной совокупности. Наконец, мы рассмотрели некоторые заблуждения, которые характерны для дата-майнинга, если заниматься им чересчур легкомысленно, и узнали, как исправить ситуацию с помощью свежих данных и здравого смысла.

Не волнуйтесь, если вы чувствуете, что вам стоит вернуться и перечитать что-то из этой главы, ведь здесь и правда много материала, который имеет смысл усвоить. Кроме того, если вы хотите добиться успеха в data science и машинном обучении, очень важно приучить себя грамотно тестировать гипотезы. Немногие практикующие специалисты уделяют время тому, чтобы связать статистику и принципы проверки гипотез с машинным обучением, и это вызывает сожаление.

Способность понимать и разъяснять — это передовой рубеж машинного обучения, поэтому не прекращайте учиться и привлекайте изученный материал на протяжении всей этой книги и всей своей карьеры.

Упражнения для самопроверки

1. Вы купили для своего 3D-принтера катушку филамента диаметром 1,75 мм. Вы хотите измерить, насколько диаметр нити действительно близок к 1,75 мм. С помощью штангенциркуля вы пять раз измеряете диаметр в разных местах катушки и получаете такую выборку значений:

   1,78; 1,75; 1,72; 1,74; 1,77.

   Рассчитайте среднее и стандартное отклонение для этой выборки.

2. Производитель утверждает, что средний срок службы смартфона Z-Phone составляет 42 месяца, а стандартное отклонение — 8 месяцев. Предполагая нормальное распределение, какова вероятность того, что данный случайный Z-Phone прослужит от 20 до 30 месяцев?

3. Я сомневаюсь в том, что средний диаметр филамента для моего 3D-принтера равен 1,75 мм, как заявлено в рекламе. Я сделал выборку из 34 измерений штангенциркулем. Выборочное среднее составляет 1,715588, а стандартное отклонение — 0,029252.

   Каков 99 %-ный доверительный интервал для генерального среднего всей катушки с нитью?

4. Ваш отдел маркетинга начал новую рекламную кампанию и хочет узнать, повлияла ли она на объем продаж, который прежде составлял в среднем 10 345 долларов в день со стандартным отклонением 552 доллара. Новая рекламная кампания проводилась в течение 45 дней, и средний объем продаж составил 11 641 долларов в день.

   Повлияла ли реклама на продажи? Почему да или почему нет? (Чтобы получить более достоверную значимость, используйте двусторонний тест.)

Ответы см. в Приложении Б.

ГЛАВА 4

Линейная алгебра

Давайте немного сменим тему: отвлечемся от теории вероятностей и статистики и обратимся к линейной алгебре. Иногда люди путают линейную алгебру с элементарной алгеброй, думая, что это дисциплина о том, как строить линейные графики для алгебраических функций вида $y = mx + b$. Именно поэтому линейную алгебру, вероятно, стоило бы назвать «векторной алгеброй» или «матричной алгеброй», ведь она имеет дело с весьма абстрактными конструкциями. Линейные системы играют в ней определенную роль, но в гораздо более отвлеченном качестве.

Итак, что же такое *линейная алгебра*? Она занимается линейными системами, но представляет их с помощью векторных пространств и матриц. Не волнуйтесь, если вы не знаете, что такое вектор или матрица! Скоро мы дадим им определения и подробно их изучим. На линейную алгебру опираются многие прикладные области математики, статистики, исследования операций, data science и машинного обучения. Когда вы работаете с данными в любой из этих областей, вы используете линейную алгебру — возможно, даже не подозревая этого.

До какого-то момента можно обойтись без изучения линейной алгебры, используя библиотеки машинного обучения и статистики, которые делают всю работу за вас. Но если вы хотите понять, как устроены эти «черные ящики», и эффективнее работать с данными, вам неизбежно придется освоить основы линейной алгебры. Линейная алгебра — это огромная тема, которой посвящены толстые учебники, и, конечно, мы не сможем в совершенстве овладеть ею всего за одну главу книги. Однако полученных знаний будет достаточно, чтобы не бояться линейной алгебры и представлять себе, какую роль она играет в data science. Кроме того, далее в этой книге — в частности, в главах 5 и 7 — у нас будет возможность применить линейную алгебру на практике.

Что такое вектор?

Говоря простыми словами, *вектор* — это стрелка в пространстве, у которой есть определенное направление и длина и которая часто представляет фрагмент данных. Это главный строительный блок линейной алгебры: в частности, на основе векторов формируются матрицы и линейные преобразования. В самой простой форме вектор не привязан к конкретному местоположению, поэтому можно представлять себе, что он начинается в начале декартовой системы координат — в точке (0, 0).

На рис. 4.1 показан вектор $\vec{v}$, который обозначает перемещение на три единицы по горизонтали и на две по вертикали.

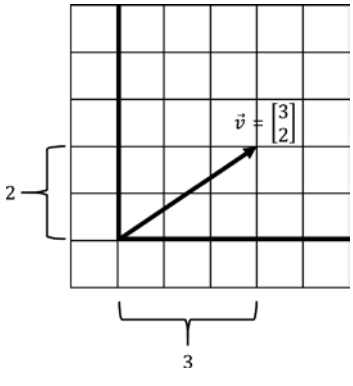


Рис. 4.1. Простой вектор

Еще раз подчеркнем, что задача вектора — наглядно представить фрагмент данных. Если у нас есть данные о том, что площадь дома составляет 18 000 квадратных футов, а его стоимость — 260 000 долларов, это можно выразить в виде вектора [18 000, 260 000], отложив 18 000 единиц по горизонтали и 260 000 — по вертикали.

Математически вектор можно выразить так:

$$\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

Объявить вектор можно с помощью простой коллекции Python — например, списка, как показано в примере 4.1.

Пример 4.1. Объявление вектора в Python с помощью списка

```
v = [3, 2]
print(v)
```

Однако когда речь заходит о математических вычислениях с векторами — особенно если заниматься задачами машинного обучения — лучше использовать библиотеку NumPy, потому что в этой области она эффективнее, чем обычный Python. Для операций линейной алгебры также можно применять библиотеку SymPy, и мы будем время от времени обращаться к ней в этой главе, когда окажется неудобно работать с десятичными числами. Впрочем, на практике вы, скорее всего, будете иметь дело в основном с NumPy, поэтому в первую очередь мы будем придерживаться именно ее.

Чтобы объявить вектор, вызовите функцию `array()` из NumPy и передайте ей коллекцию элементов, как показано в примере 4.2.

Пример 4.2. Объявление вектора в Python с помощью NumPy

```
import numpy as np
v = np.array([3, 2])
print(v)
```

Python медленный, а его числовые библиотеки — нет

Язык Python считается медленным в вычислительном отношении, потому что он не компилируется в низкоуровневый машинный код или байт-код, как Java, C#, C и т. д. Код на Python динамически интерпретируется во время выполнения. Однако числовые и научные библиотеки Python работают быстро. Такие библиотеки, как NumPy, обычно пишутся на низкоуровневых языках — в первую очередь на C и C++, — и поэтому они эффективны по части вычислений. Python выступает в роли связующего звена, интегрируя эти библиотеки для ваших задач.

Векторы очень широко применяются на практике. В физике вектор часто понимают как направление и размер какой-либо величины. В математике вектор задает направление и длину, и его можно ассоциировать с движением на координатной плоскости. В области работы с данными вектор — это массив чисел, в котором хранятся данные. Именно с этой интерпретацией нам как профессионалам в сфере data science предстоит познакомиться ближе всего. Однако важно никогда не забывать о визуальном представлении, чтобы не представлять себе вектор как эзотерический набор чисел. Без визуальной модели практически невозможно усвоить такие фундаментальные понятия линейной алгебры, как линейная зависимость и определители.

На рис. 4.2 для примера представлены еще несколько векторов. Обратите внимание, что некоторые из них направлены в сторону отрицательных значений по осям *X* и (или) *Y*. Векторы с отрицательными координатами пригодятся, когда мы будем выполнять операции над ними: по сути, с их помощью можно не только складывать, но и вычитать векторы.

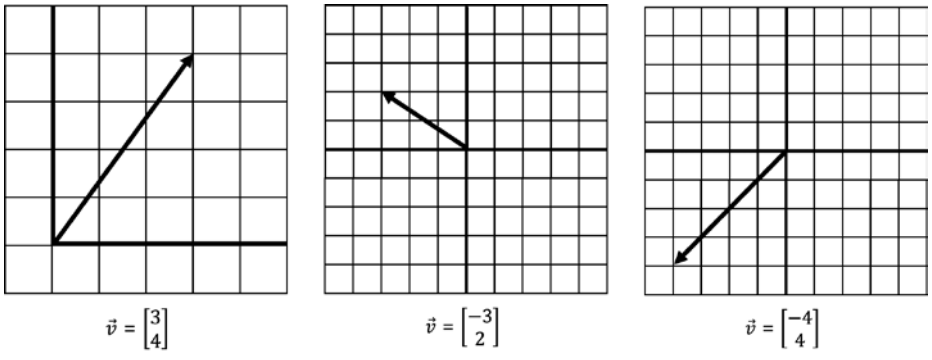


Рис. 4.2. Примеры различных векторов

ЧЕМ ПОЛЕЗНЫ ВЕКТОРЫ?

Многие из тех, кто сталкивается с векторами (и линейной алгеброй в целом), не понимают, чем они полезны. Векторы — весьма абстрактное понятие, но у них есть множество вполне ощутимых применений. Если вы разбираетесь в векторах и линейных преобразованиях, вам будет легче работать с компьютерной графикой. Например, потрясающая библиотека визуализации Manim (https://oreil.ly/Os5WK) описывает эффекты анимации и трансформации на языке векторов. В области статистики и машинного обучения данные часто импортируются и преобразуются в числовые векторы, чтобы с ними можно было работать. Средства поиска решения, такие как встроенный инструмент Excel или Python PuLP, используют линейное программирование, где векторы позволяют максимизировать решение, соблюдая заданные ограничения. В видеоиграх и авиасимуляторах с помощью векторов и линейной алгебры моделируется не только графика, но и физика. Я думаю, что векторы так трудно освоить не потому, что непонятно, как их применять, а скорее потому, что их применения настолько разнообразны, что трудно увидеть в них что-то общее.

Обратите внимание, что векторы могут существовать более чем в двух измерениях. Вот пример трехмерного вектора с координатами *x*, *y* и *z:*

$$\vec{v} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 2 \end{bmatrix}$$

Чтобы построить этот вектор, необходимо продвинуться на четыре единицы в направлении *X*, одну в направлении *Y* и две в направлении *Z*; это представлено на рис. 4.3. Обратите внимание, что теперь вектор изображен не на двумерной сетке, а в трехмерном пространстве с тремя осями: *X*, *Y* и *Z*.

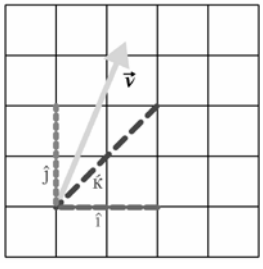


Рис. 4.3. Трехмерный вектор

Естественно, этот трехмерный вектор можно представить на Python с помощью трех числовых значений, как показано в примере 4.3.

Пример 4.3. Объявление трехмерного вектора в Python с помощью NumPy

```
import numpy as np
v = np.array([4, 1, 2])
print(v)
```

Наглядно изображать векторы более чем в трех измерениях — непростая задача, и в этой книге мы не будем тратить на нее силы. Но многомерные векторы по-прежнему тривиально представляются с помощью чисел. В примере 4.4 показано, как в Python объявить пятимерный вектор.

$$\vec{v} = \begin{bmatrix} 6 \\ 1 \\ 5 \\ 8 \\ 3 \end{bmatrix}$$

Пример 4.4. Объявление пятимерного вектора в Python с помощью NumPy

```
import numpy as np
v = np.array([6, 1, 5, 8, 3])
print(v)
```

Сложение векторов

Сами по себе векторы не особо примечательны. Вектор отражает направление и величину, как бы задавая перемещение в пространстве. Но когда мы начинаем соединять векторы с помощью операции *сложения векторов*, все становится интереснее. Фактически мы объединяем в один вектор перемещения, которые соответствуют двум векторам.

Допустим, у нас есть два вектора $\vec{v}$ и $\vec{w}$, которые показаны на рис. 4.4. Как их сложить?

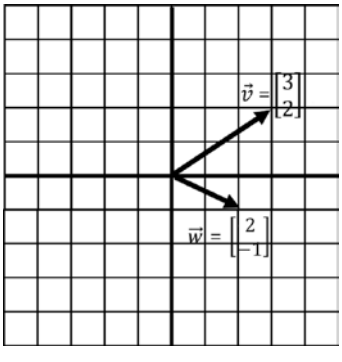


Рис. 4.4. Сложение двух векторов

О том, какая польза от этого сложения, мы поговорим чуть позже. Но если бы мы захотели соединить эти два вектора, включая их направление и величину, как бы это выглядело? В числовом выражении это элементарно: сложив координаты *x* двух векторов, мы получаем координату *x* нового вектора, и аналогично поступаем с координатами *y* — см. пример 4.5.

$$\vec{v} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}; \vec{w} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

$$\vec{v} + \vec{w} = \begin{bmatrix} 3+2 \\ 2+(-1) \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

Пример 4.5. Сложение двух векторов на Python с помощью NumPy

```
from numpy import array

v = array([3,2])
w = array([2,-1])

# сумма векторов
v_plus_w = v + w

# вывод результата
print(v_plus_w) # [5, 1]
```

Но как это выглядит на координатной плоскости? Чтобы визуально сложить два вектора, нужно приставить начало одного вектора к концу другого и проследовать до конца последнего вектора (рис. 4.5). Точка, в которой вы окажетесь, — это конец нового вектора, который представляет собой результат сложения двух векторов.

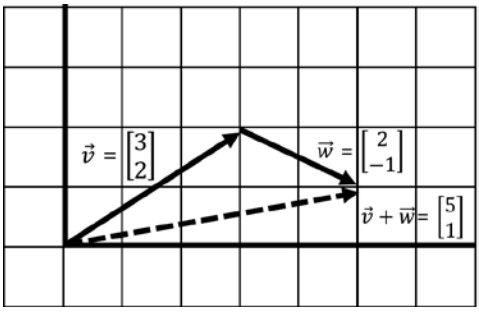


Рис. 4.5. Сложение двух векторов; результат — новый вектор

Как показано на рис. 4.5, когда мы доходим до конца последнего вектора $\vec{w}$, мы получаем новый вектор $\begin{bmatrix}5\\1\end{bmatrix}$ — это результат сложения $\vec{v}$ и $\vec{w}$. На практике это может происходить, когда нужно просто сложить данные. Если бы нам понадобилась общая стоимость жилья и его общая площадь в определенном районе, то для этого мы сложили бы несколько векторов в один по такому же принципу.

Обратите внимание, что неважно, в каком порядке складывать $\vec{v}$ и $\vec{w}$. То есть сложение векторов — это *коммутативная* операция, где порядок слагаемых не имеет значения. Если сначала изобразить вектор $\vec{w}$, а затем присоединить к нему $\vec{v}$, в результате получится тот же самый вектор $\begin{bmatrix}5\\1\end{bmatrix}$, как показано на рис. 4.6.

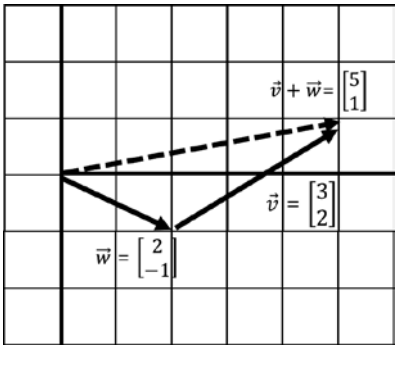


Рис. 4.6. Сложение векторов — коммутативная операция

Умножение вектора на число

Умножение вектора на число (или *масштабирование вектора*) увеличивает или уменьшает длину вектора. Длина умножается на число, которое называется *скаляром*. На рис. 4.7 вектор $\vec{v}$ умножается на 2, отчего становится вдвое длиннее.

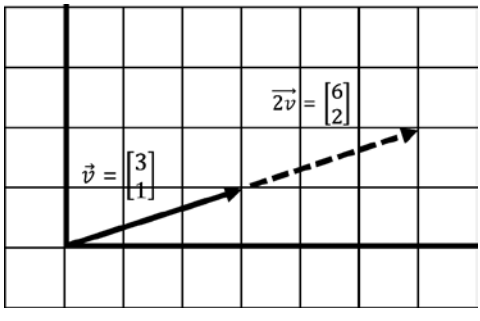


Рис. 4.7. Умножение вектора на число

С точки зрения математики каждая координата вектора умножается на одно и то же число:

$$\vec{v} = \begin{bmatrix}3\\1\end{bmatrix}$$

$$2\vec{v} = 2\begin{bmatrix}3\\1\end{bmatrix} = \begin{bmatrix}3\times 2\\1\times 2\end{bmatrix} = \begin{bmatrix}6\\2\end{bmatrix}$$

Чтобы выполнить эту операцию на Python, достаточно умножить вектор на скаляр, как показано в примере 4.6.

Пример 4.6. Умножение вектора на число на Python с помощью NumPy

```
from numpy import array

v = array([3,1])

# умножение вектора на число
scaled_v = 2.0 * v

# вывод результата
print(scaled_v) # [6. 2.]
```

На рис. 4.8 вектор $\vec{v}$ умножается на 0,5, отчего становится вдвое короче.

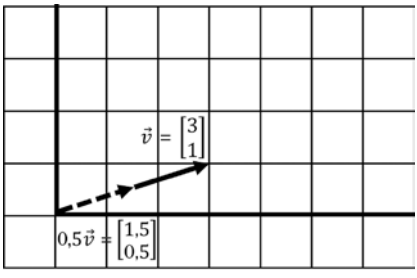


Рис. 4.8. Уменьшение длины вектора в два раза

ОПЕРАЦИИ С ДАННЫМИ — ЭТО ОПЕРАЦИИ С ВЕКТОРАМИ

Любые операции с данными можно представить с помощью векторов — даже простейшее вычисление среднего арифметического.

Посмотрите, какую роль здесь играет умножение вектора на число. Допустим, мы хотим вычислить среднюю стоимость и среднюю площадь дома для целого района. Если сложить векторы, которые соответствуют каждому дому, получится один гигантский вектор, содержащий общую стоимость и общую площадь всех домов. Затем мы разделим его на количество домов N, то есть умножим на $1 / N$. Результатом будет вектор, который содержит среднюю стоимость и среднюю площадь дома.

Важно отметить, что при умножении вектора на число его направление сохраняется; меняется только длина. Но из этого правила есть одно исключение, которое показано на рис. 4.9. Когда вектор умножается на отрицательное число, его направление меняется на противоположное.

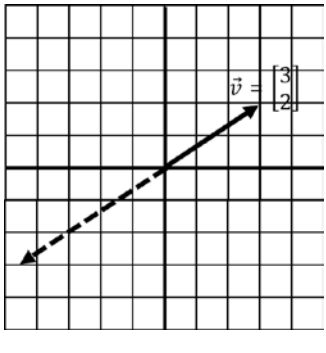


Рис. 4.9. При умножении на отрицательный скаляр направление вектора меняется на противоположное

Однако, если задуматься, при умножении на отрицательное число ориентация вектора, по сути, не меняется, потому что он остается на той же прямой. Это приводит нас к такому ключевому понятию, как линейная зависимость.

Линейная оболочка и линейная зависимость

Две операции — сложение двух векторов и умножение вектора на число — позволяют реализовать простую, но плодотворную идею. С помощью этих операций можно из двух векторов создать любой результирующий вектор, какой мы захотим. На рис. 4.10 показаны шесть примеров того, как можно умножить каждый из векторов $\vec{v}$ и $\vec{w}$ на какое-либо число, а затем сложить результаты. (Это называется *линейной комбинацией* векторов $\vec{v}$ и $\vec{w}$.) Каждый из векторов $\vec{v}$ и $\vec{w}$ имеет свое фиксированное направление (он может разве что развернуться на 180°, если умножить его на отрицательное число), но если составлять из них линейные комбинации, то можно получить *любой* новый вектор.

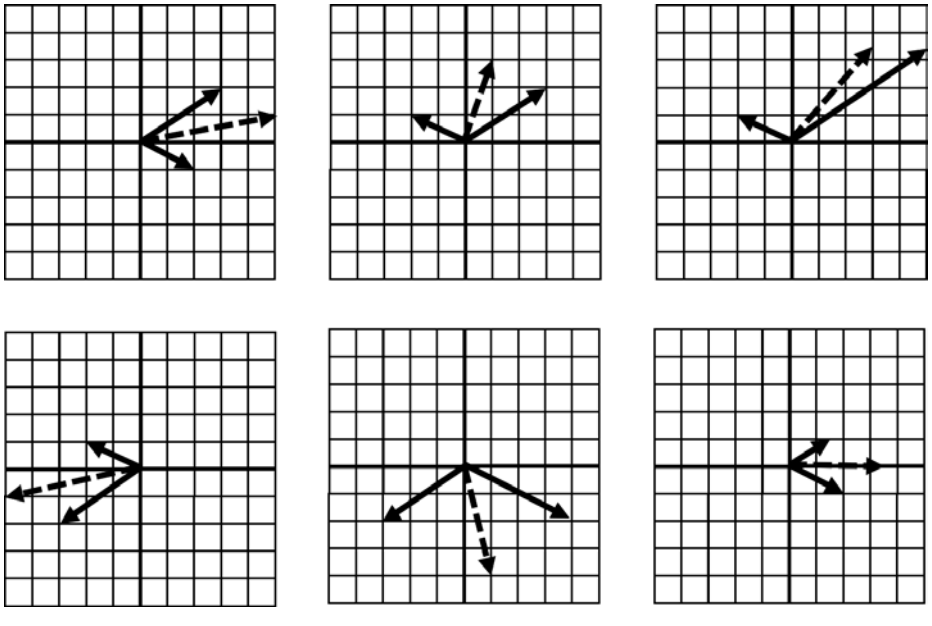


Рис. 4.10. Линейная комбинация двух векторов позволяет создать любой новый вектор

Это пространство возможных векторов называется *линейной оболочкой*, и в большинстве случаев она позволяет сконструировать абсолютно любой новый вектор, просто умножая исходные векторы на числа и складывая их. Если два вектора направлены в две независимые стороны, мы говорим, что они *линейно независимы*.

Но в каком случае ассортимент создаваемых векторов ограничен? Подумайте над этим и читайте дальше.

Что происходит, когда два вектора направлены в одну и ту же сторону или лежат на одной прямой? Любая комбинация этих векторов тоже будет лежать на этой прямой, и линейная оболочка не выходит за пределы этой линии. На какие бы числа мы ни умножали векторы, их сумма в итоге будет лежать на той же прямой. В таком случае говорят, что векторы *линейно зависимы*, как показано на рис. 4.11.

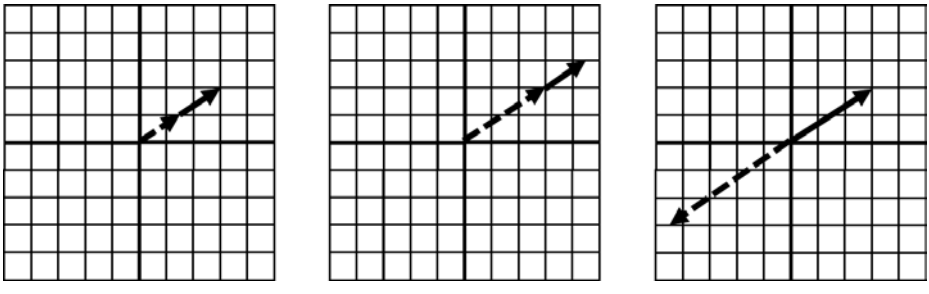


Рис. 4.11. Линейно зависимые векторы

Вся линейная оболочка состоит из одной-единственной прямой, потому что на ней лежат оба исходных вектора. А значит, с помощью линейных комбинаций из них нельзя образовать произвольный вектор, который не был бы направлен вдоль этой же прямой.

Если мы имеем дело с линейно зависимым множеством векторов в трех и более измерениях, мы часто оказываемся «заперты» на плоскости с меньшей размерностью. На рис. 4.12 показан пример того, как линейная оболочка замыкается на двумерной плоскости, хотя заданы трехмерные векторы.

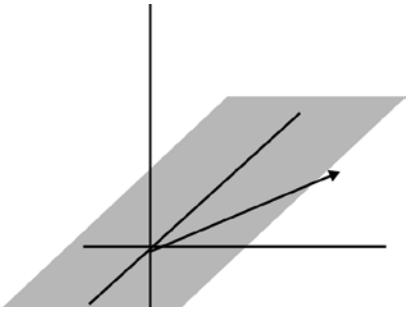


Рис. 4.12. Линейная зависимость в трехмерном пространстве. Обратите внимание, что линейная оболочка ограничена двумерной плоскостью

Позже мы научимся проверять линейную зависимость с помощью простого инструмента, который называется определителем. Но почему важно, являются ли два вектора линейно зависимыми? Дело в том, что если они линейно зависимы, многие задачи становятся сложными или вовсе неразрешимыми. Например, когда позже в этой главе мы будем рассматривать системы уравнений, то убедимся, что в линейно зависимом наборе уравнений переменные могут исчезнуть, отчего задача станет неразрешимой. Но если система линейно независима, то возможность гибко создать любой нужный вектор из двух или более векторов оказывается ключевым фактором в поиске решения!

Линейные преобразования

Линейные комбинации векторов играют крайне важную роль в линейной алгебре. Каждый вектор направлен в свою фиксированную сторону, но если умножить векторы на числа и сложить результаты, то можно получить любой вектор на плоскости. Если исходные векторы линейно независимы, такой комбинированный вектор может быть направлен в любую сторону и иметь любую длину, которую мы зададим. Этот принцип лежит в основе линейных преобразований, когда один вектор воздействует на другой и преобразовывает его подобно тому, как это делает функция.

Базисные векторы

Допустим, что заданы два простых вектора $\hat{i}$ и $\hat{j}$. Они известны как *базисные векторы*, с помощью которых описываются преобразования других векторов. Обычно они имеют длину 1 и направлены перпендикулярно друг другу в положительную сторону осей *X* и *Y*, как показано на рис. 4.13.

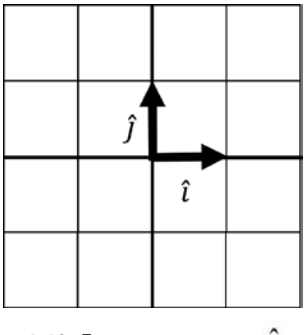


Рис. 4.13. Базисные векторы $\hat{i}$ и $\hat{j}$

Базисные векторы удобно понимать как строительные блоки, с помощью которых можно сформировать или преобразовать любой вектор. Набор базисных векторов называется *базисом* и выражается в виде матрицы 2×2, где первый столбец — вектор $\hat{i}$, а второй — $\hat{j}$:

$$\hat{i} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \hat{j} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\text{базис} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Матрица — это набор векторов (например, $\hat{i}$, $\hat{j}$), в котором может быть произвольное количество строк и столбцов и который позволяет удобно представлять данные. Масштабируя и складывая $\hat{i}$ и $\hat{j}$, можно получить любой вектор на плоскости. Для начала просто сложим эти два вектора и получим вектор $\vec{v}$, который показан на рис. 4.14.

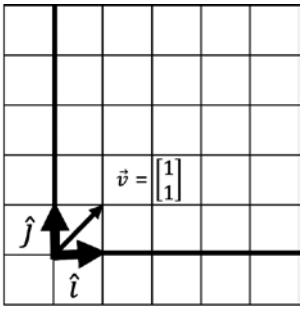


Рис. 4.14. Создание вектора из базисных векторов

Теперь я хочу, чтобы вектор $\vec{v}$ оканчивался в точке [3, 2]. Как будет выглядеть $\vec{v}$, если $\hat{i}$ умножить на 3, а $\hat{j}$ — на 2? Сначала отмасштабируем их по отдельности, как показано ниже:

$$3\hat{i} = 3\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

$$2\hat{j} = 2\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

Если растянуть пространство в этих двух направлениях, что произойдет с вектором $\vec{v}$? Он растянется вслед за $\hat{i}$ и $\hat{j}$. Это называется *линейным преобразованием*, когда мы преобразовываем вектор (растягиваем, сжимаем, сдвигаем или поворачиваем его), модифицируя базисные векторы. В данном случае (рис. 4.15) мы масштабировали векторы $\hat{i}$ и $\hat{j}$ и таким образом растянули пространство вместе с нашим вектором $\vec{v}$.

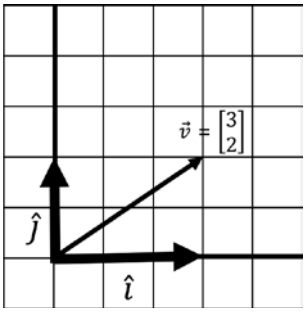
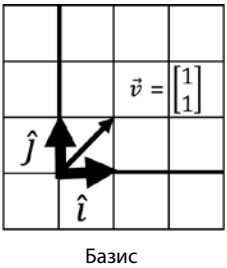


Рис. 4.15. Линейное преобразование

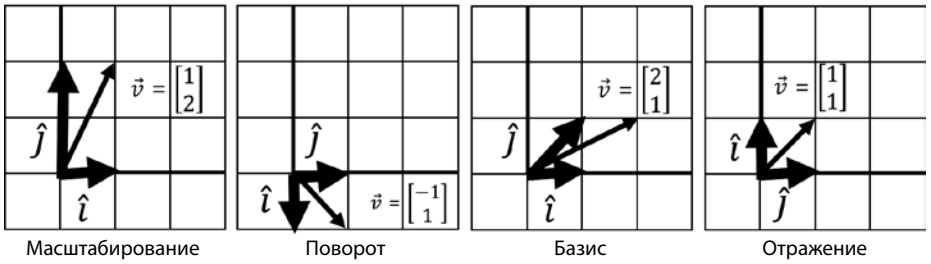
Но где оканчивается $\vec{v}$? Легко увидеть, что его конец располагается в точке $\begin{bmatrix}3\\2\end{bmatrix}$. Вспомним, что $\vec{v}$ — это сумма векторов $\hat{i}$ и $\hat{j}$. Поэтому, чтобы вычислить, где оказался вектор $\vec{v}$, нужно просто взять отмасштабированные $\hat{i}$ и $\hat{j}$ и сложить их:

$$\vec{v}_{\text{новый}} = \begin{bmatrix}3\\0\end{bmatrix} + \begin{bmatrix}0\\2\end{bmatrix} = \begin{bmatrix}3\\2\end{bmatrix}$$

Обычно линейные преобразования сводятся к четырем действиям над базисными векторами; эти действия показаны на рис. 4.16.



Базис



Масштабирование Поворот Базис Отражение

Рис. 4.16. Четыре вида линейных преобразований

Эти четыре линейных преобразования лежат в основе всей линейной алгебры. Масштабирование вектора (то есть умножение вектора на число) приводит к тому, что он растягивается или сжимается. Сдвиг перекашивает координатную плоскость так, что прямоугольники превращаются в параллелограммы. Поворот вращает векторное пространство вокруг начала координат, а отражение приводит к тому, что базисные векторы $\hat{i}$ и $\hat{j}$ меняются местами.

Важно отметить, что в линейной алгебре нельзя использовать преобразования, которые переводят прямые линии в кривые или превращают векторы в загогулины. Вот почему она называется линейной, а не нелинейной алгеброй!

Умножение матрицы на вектор

Все, что мы узнали до сих пор, приводит нас к следующему важному понятию линейной алгебры. Если отслеживать, куда переходят $\hat{i}$ и $\hat{j}$ после преобразования, можно не только конструировать новые векторы, но и модифицировать существующие. Если вы хотите по-настоящему понимать линейную алгебру, задумайтесь, почему создание векторов и их преобразование — это фактически одно и то же. Это все вопрос относительности, если учитывать, что базисные векторы служат основой результирующего вектора как до, так и после преобразования.

Вот как выглядит формула для преобразования вектора $\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix}$ с помощью базиса из векторов $\hat{i}$ и $\hat{j}$, которые представлены в виде матрицы $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$:

$$\begin{bmatrix} x_{\text{новый}} \\ y_{\text{новый}} \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

Базисному вектору $\hat{i}$ соответствует первый столбец матрицы $\begin{bmatrix} a \\ c \end{bmatrix}$, а вектору $\hat{j}$ — второй столбец $\begin{bmatrix} b \\ d \end{bmatrix}$. Оба базисных вектора объединяются в матрицу, которая по-прежнему представляет собой набор векторов, выраженный в виде двумерной (или многомерной) таблицы чисел. Такое преобразование вектора с помощью базисных векторов известно как *умножение матрицы на вектор*. Поначалу эта конструкция может показаться искусственной, однако эта формула — компактный способ записать операцию, которая масштабирует и складывает векторы $\hat{i}$ и $\hat{j}$ точно так же, как мы ранее складывали векторы, и применяет преобразование к произвольному вектору $\vec{v}$.

Так что по сути матрица — это преобразование, которое выражено в виде базисных векторов.

Чтобы выполнить это преобразование на Python с помощью NumPy, нужно объявить базисные векторы в форме матрицы, а затем применить ее к вектору $\vec{v}$

с помощью метода `dot()` (пример 4.7). Этот метод выполняет операцию масштабирования и сложения над матрицей и вектором по схеме, которую мы описали выше. Результат операции называется *суммой скалярных произведений*, и мы будем изучать ее на протяжении всей этой главы.

Пример 4.7. Умножение матрицы на вектор с помощью NumPy

```
from numpy import array

# создаем базисную матрицу из векторов î и ĵ
basis = array(
    [[3, 0],
     [0, 2]]
)

# объявляем вектор v
v = array([1,1])

# создаем новый вектор как сумму скалярных произведений
new_v = basis.dot(v)

print(new_v) # [3 2]
```

Размышляя в терминах базисных векторов, я предпочитаю выражать их в явном виде, а затем объединять в матрицу. Только учтите, что матрицу приходится *транспонировать* — то есть менять местами столбцы и строки. Это связано с тем, что функция `array()` библиотеки NumPy делает из каждого вектора строку, а не столбец, как нам нужно. Транспонирование в NumPy продемонстрировано в примере 4.8.

Пример 4.8. Явное представление базисных векторов и их применение в качестве преобразования

```
from numpy import array

# объявляем базисные векторы î и ĵ
i_hat = array([2, 0])
j_hat = array([0, 3])

# создаем базисную матрицу из векторов î и ĵ
# ее нужно транспонировать, чтобы строки стали столбцами
basis = array([i_hat, j_hat]).transpose()

# объявляем вектор v
v = array([1,1])

# создаем новый вектор как сумму скалярных произведений
new_v = basis.dot(v)

print(new_v) # [3 2]
```

Вот еще один пример. Пускай вектор $\vec{v}$ — это $\begin{bmatrix}2\\1\end{bmatrix}$, а $\hat{i}$ и $\hat{j}$ — соответственно $\begin{bmatrix}1\\0\end{bmatrix}$ и $\begin{bmatrix}0\\1\end{bmatrix}$. Затем мы преобразуем $\hat{i}$ и $\hat{j}$ в $\begin{bmatrix}2\\0\end{bmatrix}$ и $\begin{bmatrix}0\\3\end{bmatrix}$. Что при этом произойдет с вектором $\vec{v}$? Решая эту задачу вручную по формуле, мы получим следующее:

$$\begin{bmatrix}x_{\text{новый}}\\y_{\text{новый}}\end{bmatrix}=\begin{bmatrix}a & b\\c & d\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}ax+by\\cx+dy\end{bmatrix}$$

$$\begin{bmatrix}x_{\text{новый}}\\y_{\text{новый}}\end{bmatrix}=\begin{bmatrix}2 & 0\\0 & 3\end{bmatrix}\begin{bmatrix}2\\1\end{bmatrix}=\begin{bmatrix}(2\times2)+(0\times1)\\(2\times0)+(3\times1)\end{bmatrix}=\begin{bmatrix}4\\3\end{bmatrix}$$

В примере 4.9 представлено то же самое решение на Python.

Пример 4.9. Преобразование вектора с помощью NumPy

```
from numpy import array

# объявляем базисные векторы î и ĵ
i_hat = array([2, 0])
j_hat = array([0, 3])

# создаем базисную матрицу из векторов î и ĵ
# ее нужно транспонировать, чтобы строки стали столбцами
basis = array([i_hat, j_hat]).transpose()

# объявляем вектор v
v = array([2,1])

# создаем новый вектор как сумму скалярных произведений
new_v = basis.dot(v)

print(new_v) # [4 3]
```

Теперь вектор $\vec{v}$ оканчивается в точке $\begin{bmatrix}4\\3\end{bmatrix}$. На рис. 4.17 показано, как выглядит это преобразование.

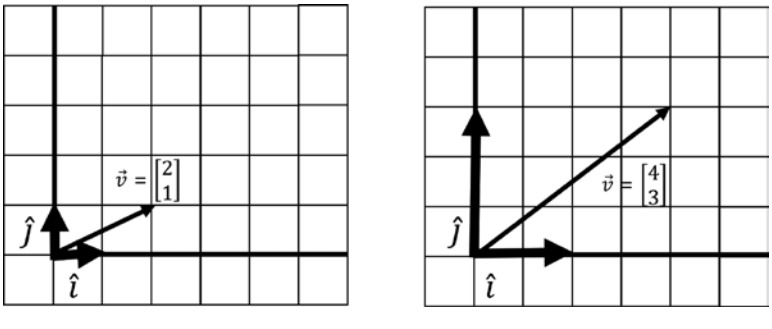


Рис. 4.17. Линейное преобразование с растягиванием

А следующий пример выводит обсуждение на новый уровень. Возьмем вектор $\vec{v}$ с координатами $\begin{bmatrix}2\\1\end{bmatrix}$. Векторы $\hat{i}$ и $\hat{j}$ сначала соответствуют точкам $\begin{bmatrix}1\\0\end{bmatrix}$ и $\begin{bmatrix}0\\1\end{bmatrix}$, но затем преобразуются и переходят в точки $\begin{bmatrix}2\\3\end{bmatrix}$ и $\begin{bmatrix}2\\-1\end{bmatrix}$. Что произойдет с вектором $\vec{v}$? Рассмотрим рис. 4.18 и пример 4.10.

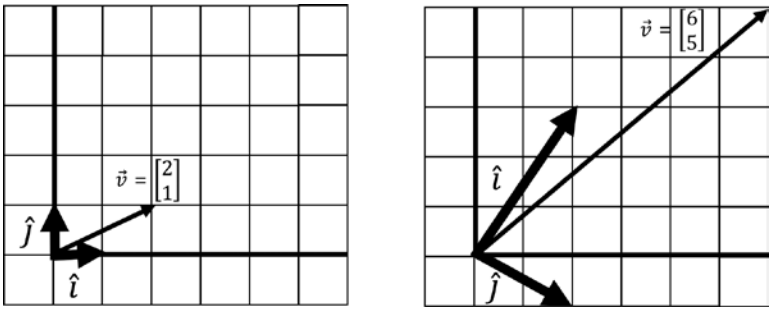


Рис. 4.18. Линейное преобразование с поворотом, сдвигом и отражением пространства

Пример 4.10. Более сложное преобразование

```
from numpy import array

# объявляем базисные векторы î и ĵ
i_hat = array([2, 3])
j_hat = array([2, -1])

# создаем базисную матрицу из векторов î и ĵ
# ее нужно транспонировать, чтобы строки стали столбцами
basis = array([i_hat, j_hat]).transpose()

# объявляем вектор v
v = array([2,1])

# создаем новый вектор как сумму скалярных произведений
new_v = basis.dot(v)

print(new_v) # [6 5]
```

В этом примере произошло очень много всего. Мы не только масштабировали $\hat{i}$ и $\hat{j}$ и растянули вектор $\vec{v}$, но и сдвинули, повернули и отразили пространство. О том, что оно отразилось, говорит тот факт, что $\hat{i}$ и $\hat{j}$ поменялись местами в ориентации по часовой стрелке, и позже в этой главе мы научимся обнаруживать этот эффект с помощью определителей.

БАЗИСНЫЕ ВЕКТОРЫ В ТРЕХМЕРНОМ ПРОСТРАНСТВЕ И ЗА ЕГО ПРЕДЕЛАМИ

Возможно, вас интересует, как представить себе преобразования векторов в трех и более измерениях. Понятие базисных векторов неплохо расширяется на многомерные конструкции. Например, если мы рассматриваем трехмерное векторное пространство, то в нем есть базисные векторы $\hat{i}$, $\hat{j}$ и $\hat{k}$. Просто продолжайте добавлять новые буквы для каждого нового измерения (рис. 4.19).

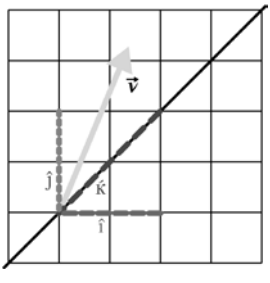


Рис. 4.19. Базисные векторы в трехмерном пространстве

Стоит также отметить, что некоторые линейные преобразования могут увеличивать или уменьшать количество измерений в векторном пространстве. Именно это делают неквадратные матрицы (где количество строк не равно количеству столбцов). Чтобы сэкономить время, мы не будем углубляться в эту тему. Такие преобразования прекрасно объяснены и продемонстрированы на образовательном YouTube-канале 3Blue1Brown (https://oreil.ly/TsoSJ).

Умножение матриц

После того как мы научились умножать матрицу на вектор, пришло время узнать, как перемножить две матрицы. Считайте, что *умножение матриц* заключается в том, что к векторному пространству применяются несколько преобразований. Каждое преобразование похоже на функцию: сначала мы применяем то, которое находится на самом глубоком уровне вложенности, а затем каждое следующее — по порядку от более глубоких уровней к менее глубоким.

Вот как можно применить поворот, а затем сдвиг к любому вектору $\vec{v}$ со значением $\begin{bmatrix} x \\ y \end{bmatrix}$:

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}$$

Можно объединить эти два преобразования с помощью следующей формулы, которая позволяет наложить одно преобразование на другое. Каждый элемент итоговой матрицы — это скалярное произведение строки первой матрицы и соответствующего столбца второй матрицы:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} e & f \\ g & h \end{bmatrix}=\begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

С помощью этой формулы два отдельных преобразования (вращение и сдвиг) можно превратить в одно:

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} (1\times 0)+(1\times 1) & (-1\times 1)+(1\times 0) \\ (0\times 0)+(1\times 1) & (0\times -1)+(1\times 0) \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}$$

Чтобы воспроизвести эти вычисления на Python с помощью NumPy, можно просто объединить две матрицы с помощью метода `matmul()` или оператора `@` (см. пример 4.11). Затем это объединенное преобразование можно применить к вектору $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

Пример 4.11. Композиция двух преобразований

```
from numpy import array

# Преобразование 1
i_hat1 = array([0, 1])
j_hat1 = array([-1, 0])
transform1 = array([i_hat1, j_hat1]).transpose()

# Преобразование 2
i_hat2 = array([1, 0])
j_hat2 = array([1, 1])
transform2 = array([i_hat2, j_hat2]).transpose()

# Композиция преобразований
combined = transform2 @ transform1

# Проверка
print(f"Объединенная матрица:\n {combined}")

v = array([1, 2])
print(combined.dot(v)) # [-1 1]
```

Используйте matmul() и @ вместо dot()

Чтобы перемножать матрицы, в общем случае лучше применять метод `matmul()` и его сокращение `@`, чем метод `dot()` из библиотеки NumPy. `matmul()` лучше обращается с матрицами более высокой размерности и эффективнее транслирует элементы.

Если вам интересно вникнуть в подробности реализации, начните с этого вопроса на сайте StackOverflow (https://oreil.ly/YX83Q).

Обратите внимание, что если бы мы применили к вектору $\vec{v}$ каждое преобразование по отдельности, то получился бы тот же результат. Если заменить последнюю строку кода на три строки, которые представлены ниже, то преобразования будут применяться последовательно, но все равно получится новый вектор с координатами $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$:

```
rotated = transform1.dot(v)
sheered = transform2.dot(rotated)
print(sheered) # [-1 1]
```

Обратите внимание, что важно, в каком порядке применять преобразования! Если применить `transformation1` после `transformation2`, то получится другой результат — $\begin{bmatrix} -2 \\ 3 \end{bmatrix}$, как показано в примере 4.12. Таким образом, произведение матриц — не коммутативная операция: если поменять порядок действий, то в общем случае результат тоже изменится!

Пример 4.12. Применение преобразований в обратном порядке

```
from numpy import array

# Преобразование 1
i_hat1 = array([0, 1])
j_hat1 = array([-1, 0])
transform1 = array([i_hat1, j_hat1]).transpose()

# Преобразование 2
i_hat2 = array([1, 0])
j_hat2 = array([1, 1])
transform2 = array([i_hat2, j_hat2]).transpose()

# Композиция преобразований: сначала сдвиг, потом поворот
combined = transform1 @ transform2

# Проверка
print(f"Объединенная матрица:\n {combined}")

v = array([1, 2])
print(combined.dot(v)) # [-2 3]
```

Считайте, что каждое преобразование — это функция, и мы применяем их по порядку от самой внутренней к самой внешней, как вложенные вызовы функций.

ЛИНЕЙНЫЕ ПРЕОБРАЗОВАНИЯ НА ПРАКТИКЕ

Возможно, вы спросите, какое отношение все эти линейные преобразования и матрицы имеют к data science и машинному обучению. Ответ — самое непосредственное! Линейные преобразования лежат в основе всех математических манипуляций с данными — от импорта данных до числовых операций в линейной регрессии, логистической регрессии и нейронных сетях.

Однако на практике вам редко придется тратить время на то, чтобы геометрически визуализировать данные в виде векторных пространств и линейных преобразований. Вы будете в основном иметь дело с многомерными задачами, где визуализация не очень помогает. Но помнить о геометрической интерпретации полезно хотя бы для того, чтобы понимать, что делают эти на первый взгляд мудреные числовые операции! Иначе вам просто пришлось бы зубрить шаблоны операций без какого-либо контекста. Геометрическое представление также помогает усвоить новые понятия линейной алгебры, такие как определители.

Определители

С помощью линейных преобразований мы «растягиваем» или «сжимаем» пространство, и может оказаться важным, в какой степени это происходит. Возьмем для примера область векторного пространства на рис. 4.20. Что с ним произойдет, если отмасштабировать векторы $\hat{i}$ и $\hat{j}$?

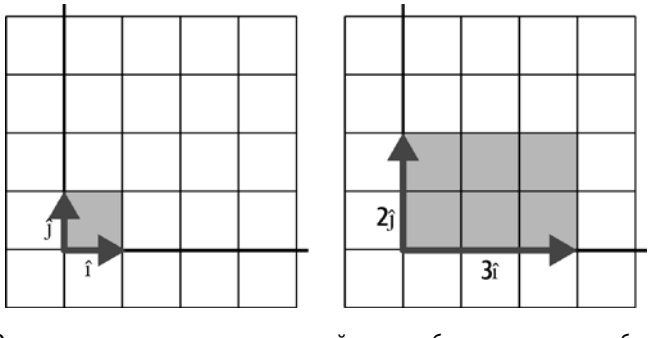


Рис. 4.20. Определитель показывает, как линейное преобразование масштабирует область

Обратите внимание, что площадь увеличилась в 6 раз, и этот множитель известен как *определитель* (или *детерминант*). Определитель показывает, как изменяется масштаб выбранной области векторного пространства при линейных преобразованиях, и может дать полезную информацию о преобразовании.

В примере 4.13 показано, как вычислить определитель на Python.

Пример 4.13. Вычисление определителя

```
from numpy.linalg import det
from numpy import array

i_hat = array([3, 0])
j_hat = array([0, 2])

basis = array([i_hat, j_hat]).transpose()

determinant = det(basis)

print(determinant) # выводит 6.0
```

Простые сдвиги и повороты не должны влиять на определитель, потому что площадь при этом не меняется. На рис. 4.21 и в примере 4.14 показан простой сдвиг, в результате которого определитель остается равным 1.

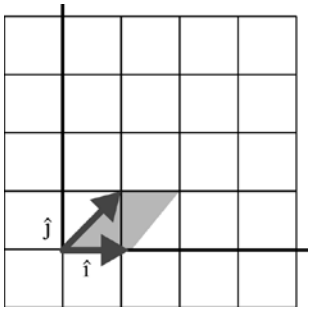


Рис. 4.21. Простой сдвиг не меняет определитель

Пример 4.14. Определитель в случае сдвига

```
from numpy.linalg import det
from numpy import array

i_hat = array([1, 0])
j_hat = array([1, 1])

basis = array([i_hat, j_hat]).transpose()
```

```
determinant = det(basis)

print(determinant) # выводит 1.0
```

Но при масштабировании определитель увеличится или уменьшится, потому что при этом увеличивается или уменьшается площадь области. Если отразить пространство (при этом $\hat{i}$ и $\hat{j}$ поменяются местами в ориентации по часовой стрелке), то определитель станет отрицательным. На рис. 4.22 и в примере 4.15 показан определитель для преобразования, которое не только масштабирует векторное пространство, но и отражает его.

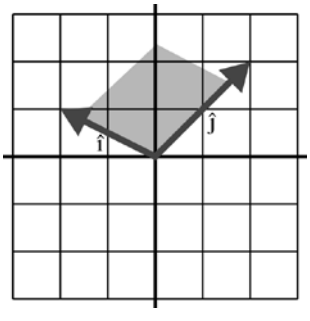


Рис. 4.22. При отражении определитель становится отрицательным

Пример 4.15. Отрицательный определитель

```
from numpy.linalg import det
from numpy import array

i_hat = array([-2, 1])
j_hat = array([1, 2])

basis = array([i_hat, j_hat]).transpose()

determinant = det(basis)

print(determinant) # выводит -5.0
```

Поскольку этот определитель меньше нуля, можно быстро понять, что произошло отражение. Однако, безусловно, самая важная информация, которую сообщает определитель, — это то, является ли преобразование линейно зависимым. Если определитель равен 0, это значит, что пространство было сжато до меньшей размерности.

На рис. 4.23 представлены два линейно зависимых преобразования, при которых двумерное пространство сжимается в одно измерение, а трехмерное — в два. Площадь и объем соответственно в обоих случаях равны 0!

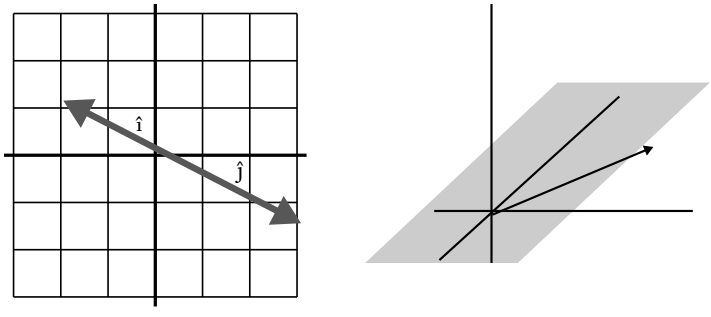


Рис. 4.23. Линейная зависимость в двух- и трехмерном пространствах

В примере 4.16 показан код для первого случая, где все двумерное пространство сжимается в одномерную числовую прямую.

Пример 4.16. Нулевой определитель

```
from numpy.linalg import det
from numpy import array

i_hat = array([-2, 1])
j_hat = array([3, -1.5])

basis = array([i_hat, j_hat]).transpose()

determinant = det(basis)

print(determinant) # выводит 0.0
```

Таким образом, полезно проверять, не равен ли определитель нулю, чтобы выявить случаи, когда преобразование линейно зависимо. Если вы столкнетесь с этим, то, скорее всего, это будет означать, что вам досталась сложная или неразрешимая задача.

Особые виды матриц

Существуют несколько особых разновидностей матриц, о которых стоит знать.

Квадратная матрица

Квадратная матрица — это матрица, в которой количество строк равно количеству столбцов, например:

$$\begin{bmatrix} 4 & 2 & 7 \\ 5 & 1 & 9 \\ 4 & 0 & 1 \end{bmatrix}$$

Такие матрицы в основном представляют линейные преобразования, а также необходимы для многих операций, например для спектрального разложения матриц.

Единичная матрица

Единичная матрица — это квадратная матрица, на главной диагонали которой стоят единицы, а остальные значения равны 0, например:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Что особенного в единичных матрицах? Если у вас есть такая матрица, это значит, что вы фактически отменили преобразование и вернулись к исходным базисным векторам. Это сыграет большую роль, когда мы будем решать системы уравнений в следующем разделе.

Обратная матрица

Обратная матрица обращает (то есть отменяет) преобразование, которое выполнила другая матрица. Допустим, у нас есть матрица A:

$$A = \begin{bmatrix} 4 & 2 & 4 \\ 5 & 3 & 7 \\ 9 & 3 & 6 \end{bmatrix}$$

Матрица, обратная к A, обозначается A^{-1}. В следующем разделе мы узнаем, как вычислить обратную матрицу с помощью SymPy или NumPy, однако вот как выглядит A^{-1}:

$$A^{-1} = \begin{bmatrix} -\frac{1}{2} & 0 & \frac{1}{3} \\ 5{,}5 & -2 & -\frac{4}{3} \\ -2 & 1 & \frac{1}{3} \end{bmatrix}$$

Если перемножить A^{-1} и A, получится единичная матрица. В следующем разделе, который посвящен системам уравнений, мы рассмотрим это преобразование с помощью NumPy и SymPy.

$$\begin{bmatrix} -\frac{1}{2} & 0 & \frac{1}{3} \\ 5,5 & -2 & -\frac{4}{3} \\ -2 & 1 & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 4 & 2 & 4 \\ 5 & 3 & 7 \\ 9 & 3 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Диагональная матрица

Диагональная матрица похожа на единичную: ее главная диагональ состоит из ненулевых значений, а остальные значения равны 0. Диагональные матрицы полезны в некоторых вычислениях, потому что они представляют простые скаляры, которые применяются к векторному пространству. Эта конструкция встречается в некоторых операциях линейной алгебры.

$$\begin{bmatrix} 4 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

Треугольная матрица

У *треугольной матрицы* все ненулевые элементы вместе с главной диагональю образуют треугольник: то есть все значения, которые находятся выше (или ниже) этой диагонали, равны 0.

$$\begin{bmatrix} 4 & 2 & 9 \\ 0 & 1 & 6 \\ 0 & 0 & 5 \end{bmatrix}$$

Треугольные матрицы полезны во многих задачах вычислительной математики, потому что с их помощью обычно проще решать системы уравнений. Они также используются в некоторых задачах разложения матриц, — например, таких, как LU-разложение (https://oreil.ly/vYK8t).

Разреженная матрица

Иногда встречаются матрицы, которые состоят в основном из нулей и содержат совсем немного ненулевых элементов. Такие матрицы называются *разреженными*, и с чисто математической точки зрения они не слишком интересны. Но в компьютерном представлении их можно реализовать эффективнее, чем обычные матрицы. Разреженная матрица не будет занимать место, чтобы хранить кучу нулей, а будет отслеживать только ненулевые ячейки.

$$\text{разреженная матрица: } \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Если вы работаете с большими разреженными матрицами, их можно создавать в явном виде с помощью функции `sparse`.

Системы уравнений и обратные матрицы

Одна из основных задач линейной алгебры — решение систем уравнений. Кроме того, это хороший материал, чтобы изучить обратные матрицы. Допустим, вам даны следующие уравнения, и нужно найти x, y и z:

$$\begin{cases} 4x + 2y + 4z = 44 \\ 5x + 3y + 7z = 56 \\ 9x + 3y + 6z = 72 \end{cases}$$

Можно попробовать вручную поэкспериментировать с алгебраическими операциями, чтобы выразить каждую переменную, но если вы хотите, чтобы эту задачу решил компьютер, вам понадобится сформулировать ее на языке матриц, как показано ниже. Внесите коэффициенты в матрицу A, значения в правой части уравнений — в матрицу B, а неизвестные — в матрицу X:

$$A = \begin{bmatrix} 4 & 2 & 4 \\ 5 & 3 & 7 \\ 9 & 3 & 6 \end{bmatrix}; B = \begin{bmatrix} 44 \\ 56 \\ 72 \end{bmatrix}; X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

В матричном виде система уравнений имеет вид $AX = B$. Требуется найти такую матрицу X, которая преобразовывает матрицу A так, чтобы в результате получилась матрица B:

$$AX = B$$

$$\begin{bmatrix} 4 & 2 & 4 \\ 5 & 3 & 7 \\ 9 & 3 & 6 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 44 \\ 56 \\ 72 \end{bmatrix}$$

Нужно «сократить» A, чтобы выразить X и получить значения x, y и z. Для этого можно домножить обе части уравнения на обратную матрицу к A, которая обозначается A^{-1}. Это можно выразить алгебраически:

$$AX = B$$

$$A^{-1}AX = A^{-1}B$$

$$X = A^{-1}B$$

Мы будем вычислять A^{-1} на компьютере, а не вручную. Вот матрица, обратная к A:

$$A^{-1} = \begin{bmatrix} -\frac{1}{2} & 0 & \frac{1}{3} \\ 5{,}5 & -2 & -\frac{4}{3} \\ -2 & 1 & \frac{1}{3} \end{bmatrix}$$

Обратите внимание, что если умножить A^{-1} на A, то получится единичная матрица — то есть матрица, которая состоит из одних нулей, за исключением единиц по диагонали. В линейной алгебре умножение на единичную матрицу — это все равно что обычное умножение на 1: оно не влияет на множитель и фактически выражает неизвестные x, y и z по отдельности:

$$A^{-1}A = \begin{bmatrix} -\frac{1}{2} & 0 & \frac{1}{3} \\ 5{,}5 & -2 & -\frac{4}{3} \\ -2 & 1 & \frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} 4 & 2 & 4 \\ 5 & 3 & 7 \\ 9 & 3 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Чтобы увидеть эту единичную матрицу в действии на Python, лучше использовать библиотеку SymPy, а не NumPy. Дело в том, что NumPy отображает элементы матрицы в виде десятичных дробей, отчего единичная матрица получится не столь наглядной. Но если вычислить ее в символьном виде, как показано в примере 4.17, вы увидите чистый алгебраический вывод. Обратите внимание, что для умножения матриц в SymPy используется оператор *, а не @.

Пример 4.17. Изучение обратной и единичной матриц с помощью SymPy

```
from sympy import *

# 4x + 2y + 4z = 44
# 5x + 3y + 7z = 56
# 9x + 3y + 6z = 72

A = Matrix([
    [4, 2, 4],
    [5, 3, 7],
    [9, 3, 6]
])

# умножение A⁻¹ на A дает единичную матрицу
inverse = A.inv()
identity = inverse * A

# выводит Matrix([[-1/2, 0, 1/3], [11/2, -2, -4/3], [-2, 1, 1/3]])
print(f"Обратная матрица: {inverse}")

# выводит Matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
print(f"Единичная матрица: {identity}")
```

На практике погрешность, которая свойственна числам с плавающей точкой, не сильно влияет на результат, поэтому, чтобы найти X, вполне можно использовать NumPy. В примере 4.18 показано соответствующее решение.

Пример 4.18. Решение системы уравнений с помощью NumPy

```
from numpy import array
from numpy.linalg import inv

# 4x + 2y + 4z = 44
# 5x + 3y + 7z = 56
# 9x + 3y + 6z = 72

A = array([
    [4, 2, 4],
    [5, 3, 7],
    [9, 3, 6]
])

B = array([
    44,
    56,
    72
])

X = inv(A).dot(B)

print(X) # [ 2. 34. -8.]
```

Таким образом, $x = 2$, $y = 34$, а $z = -8$. В примере 4.19 показано, как решить эту же систему уравнений на SymPy.

Пример 4.19. Решение системы уравнений с помощью SymPy

```
from sympy import *

# 4x + 2y + 4z = 44
# 5x + 3y + 7z = 56
# 9x + 3y + 6z = 72

A = Matrix([
    [4, 2, 4],
    [5, 3, 7],
    [9, 3, 6]
])

B = Matrix([
    44,
    56,
    72
])

X = A.inv() * B

print(X) # Matrix([[2], [34], [-8]])
```

А так решение выглядит в математической записи:

$$X = A^{-1}B$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & 0 & \frac{1}{3} \\ 5,5 & -2 & -\frac{4}{3} \\ -2 & 1 & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 44 \\ 56 \\ 72 \end{bmatrix} = \begin{bmatrix} 2 \\ 34 \\ -8 \end{bmatrix}$$

Надеемся, теперь вы получили представление об обратных матрицах и о том, как с их помощью можно решать системы уравнений.

Системы уравнений в линейном программировании

Этот метод решения систем уравнений используется и в линейном программировании, где ограничения задаются с помощью неравенств, а целевая функция минимизируется или максимизируется.

На YouTube-канале PatrickJMT есть много хороших видео по линейному программированию (https://bit.ly/3aVyrD6). Мы также обзорно рассмотрим его в Приложении А.

На практике вам вряд ли понадобится вычислять обратные матрицы вручную — за вас это может сделать компьютер. Но если есть такая необходимость или вам просто любопытно, то вам стоит познакомиться с методом Гаусса – Жордана, который также известен как метод полного исключения неизвестных. На YouTube-канале PatrickJMT (https://oreil.ly/RfXAv) есть несколько видеороликов, которые демонстрируют этот метод.

Собственные векторы и собственные значения

Разложение матрицы заключается в том, что матрица выражается в виде произведения других матриц, подобно разложению чисел на множители (например, 10 можно представить как 2×5).

Разложение матриц может пригодиться для того, чтобы находить обратные матрицы и вычислять определители, а также в линейной регрессии. В зависимости от поставленной задачи матрицу можно разложить по-разному. В главе 5 мы воспользуемся одним из методов разложения матрицы — а именно QR-разложением — чтобы выполнить линейную регрессию.

Но в этой главе мы сосредоточимся на распространенном методе, который называется спектральным разложением матриц. Он часто используется в машинном обучении и лежит в основе метода главных компонент. Впрочем, здесь у нас нет возможности подробно рассмотреть каждое из этих применений. Пока просто запомните, что спектральное разложение позволяет разбивать матрицы на компоненты, с которыми легче работать в разных задачах машинного обучения. Учтите, что этот метод подходит только для квадратных матриц.

В спектральном разложении участвуют две конструкции: собственные значения, которые обозначаются буквой λ (лямбда), и собственные векторы, которые обозначаются буквой v, как показано на рис. 4.24.

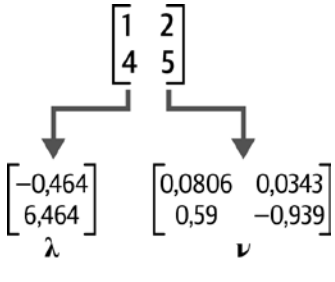


Рис. 4.24. Собственные векторы и собственные значения

Собственные векторы и собственные значения квадратной матрицы A удовлетворяет такому соотношению:

$$Av = \lambda v$$

У матрицы размером $n \times n$ существует n собственных векторов и соответствующих им собственных значений, однако не у всех матриц они выражаются действительными числами. Иногда собственные векторы и собственные значения бывают комплексными.

Пример 4.20 демонстрирует, как вычислить собственные векторы и собственные значения для заданной матрицы A с помощью NumPy.

Пример 4.20. Вычисление собственных векторов и собственных значений с помощью NumPy

```
from numpy import array, diag
from numpy.linalg import eig, inv

A = array([
    [1, 2],
    [4, 5]
])

eigenvals, eigenvecs = eig(A)

print("СОБСТВЕННЫЕ ЗНАЧЕНИЯ")
print(eigenvals)
print("\nСОБСТВЕННЫЕ ВЕКТОРЫ")
print(eigenvecs)

"""
СОБСТВЕННЫЕ ЗНАЧЕНИЯ
[-0.46410162 6.46410162]

СОБСТВЕННЫЕ ВЕКТОРЫ
[[-0.80689822 -0.34372377]
 [ 0.59069049 -0.9390708 ]]
"""
```

Как же восстановить матрицу A, зная ее собственные векторы и собственные значения? Вспомните эту формулу:

$$Av = \lambda v$$

Чтобы выразить из этой формулы A, ее нужно преобразовать в такой вид:

$$A = Q\Lambda Q^{-1}$$

В этой новой формуле Q — собственные векторы, Λ — собственные значения в диагональной форме, а Q^{-1} — матрица, обратная к Q. Диагональная форма означает, что координаты вектора расположены на главной диагонали матрицы, а все остальные значения этой матрицы равны 0 — подобно единичной матрице.

В примере 4.21 приводится полный цикл преобразования матрицы на Python: матрица подвергается спектральному разложению, а затем восстанавливается обратно.

Пример 4.21. Разложение и восстановление матрицы с помощью NumPy

```
from numpy import array, diag
from numpy.linalg import eig, inv

A = array([
    [1, 2],
    [4, 5]
])

eigenvals, eigenvecs = eig(A)

print("СОБСТВЕННЫЕ ЗНАЧЕНИЯ")
print(eigenvals)
print("\nСОБСТВЕННЫЕ ВЕКТОРЫ")
print(eigenvecs)

print("\nВОССТАНОВЛЕННАЯ МАТРИЦА")
Q = eigenvecs
R = inv(Q)

L = diag(eigenvals)
B = Q @ L @ R

print(B)

"""
СОБСТВЕННЫЕ ЗНАЧЕНИЯ
[-0.46410162 6.46410162]

СОБСТВЕННЫЕ ВЕКТОРЫ
[[-0.80689822 -0.34372377]
 [ 0.59069049 -0.9390708 ]]

ВОССТАНОВЛЕННАЯ МАТРИЦА
[[1. 2.]
 [4. 5.]]
"""
```

Как видите, восстановленная матрица совпадает с той, с которой мы начали.

Заключение

Линейная алгебра может быть безумно абстрактной, она полна загадок и идей для размышления. Вам может показаться, что вся эта тема — одна большая западня, и вы будете правы! Тем не менее ею стоит продолжать интересоваться, если вы хотите построить долгую и успешную карьеру в области data science. Линейная алгебра лежит в основе статистических вычислений, машинного обучения и других прикладных областей data science. В конечном счете на линейную алгебру опирается вся информатика в целом. Безусловно, до какого-то момента можно обходиться без этих знаний, но рано или поздно вы столкнетесь с тем, что вам не хватает понимания, чтобы справиться с той или иной задачей.

Вы можете задаться вопросом, какое отношение изученный материал имеет к практической data science, — ведь все эти векторы и матрицы могут показаться чисто теоретическими понятиями. Не волнуйтесь: мы будем встречаться с практическими применениями линейной алгебры на протяжении всей книги. Но теория и геометрические интерпретации важны для того, чтобы понимать, что происходит «под капотом», когда вы работаете с данными. Если вы визуально представляете себе линейные преобразования, это поможет освоить более сложные понятия, которые могут встретиться на вашем пути.

Если вы хотите больше узнать о линейной алгебре, то вам не найти лучшего ресурса, чем серия видеороликов на YouTube-канале «Essence of Linear Algebra» («Главное о линейной алгебре») от 3Blue1Brown (https://oreil.ly/FSCNz). Также полезны видео по линейной алгебре на канале PatrickJMT (https://oreil.ly/Hx9GP).

А если вы хотите освоить NumPy, рекомендую прочитать книгу Уэса Маккинни (Wes McKinney) «Python for Data Analysis»[1]. В ней не так много внимания уделяется линейной алгебре, зато это прекрасное практическое руководство о том, как работать с наборами данных с помощью NumPy, pandas и Python.

Упражнения для самопроверки

1. Вектор $\vec{v}$ имеет значение $\begin{bmatrix}1\\2\end{bmatrix}$, но затем происходит преобразование, которое переводит вектор $\hat{i}$ в точку $\begin{bmatrix}2\\0\end{bmatrix}$, а вектор $\hat{j}$ — в точку $\begin{bmatrix}0\\1{,}5\end{bmatrix}$. Где окажется $\vec{v}$?

[1] Маккинни, Уэс. «Python и анализ данных: Первичная обработка данных с применением pandas, NumPy и Jupiter. 3-е изд.».

2. Вектор $\vec{v}$ имеет значение $\begin{bmatrix}1\\2\end{bmatrix}$, но затем происходит преобразование, которое переводит вектор $\hat{i}$ в точку $\begin{bmatrix}-2\\1\end{bmatrix}$, а вектор $\hat{j}$ — в точку $\begin{bmatrix}1\\-2\end{bmatrix}$. Где окажется $\vec{v}$?
3. Линейное преобразование переводит базисные векторы $\hat{i}$ и $\hat{j}$ в точки $\begin{bmatrix}1\\0\end{bmatrix}$ и $\begin{bmatrix}2\\2\end{bmatrix}$ соответственно. Чему равен определитель этого преобразования?
4. Можно ли заменить два или более последовательных линейных преобразования на одно? Почему да или почему нет?
5. Решив систему уравнений, найдите *x*, *y* и *z*:

$$\begin{cases}3x+1y+0z=54\\2x+4y+1z=12\\3x+1y+8z=6\end{cases}$$

6. Является ли следующая матрица линейно зависимой? Почему да или почему нет?

$$\begin{bmatrix}2 & 1\\6 & 3\end{bmatrix}$$

Ответы см. в Приложении Б.

ГЛАВА 5

Линейная регрессия

Один из самых популярных методов анализа данных заключается в том, чтобы на основе наблюдаемых точек данных построить прямую, которая отображает связь между двумя или более переменными. *Регрессия* пытается подогнать функцию к наблюдаемым данным, чтобы спрогнозировать новые данные. *Линейная регрессия* подгоняет к данным прямую линию, пытаясь установить линейную связь между переменными и предсказать новые данные, которые еще предстоит наблюдать.

Возможно, лучше будет посмотреть на иллюстрацию, чем просто прочитать описание. На рис. 5.1 приведен пример линейной регрессии.

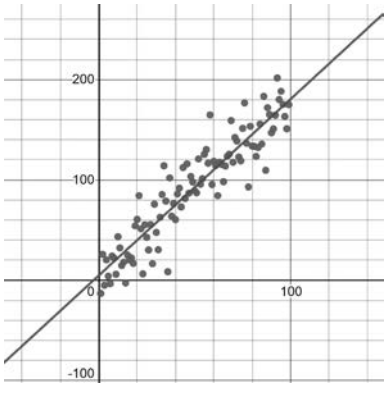


Рис. 5.1. Пример линейной регрессии, которая подгоняет прямую к наблюдаемым данным

Линейная регрессия — это рабочая лошадка data science и статистики. Она не только задействует понятия, которые мы изучили в предыдущих главах, но и закладывает фундамент для последующих тем, таких как нейронные сети (глава 7) и логистическая регрессия (глава 6). Этот относительно простой метод существует уже более двухсот лет и в наши дни считается одной из форм машинного обучения.

Специалисты по машинному обучению обычно выполняют валидацию по-своему, разделяя данные на обучающую и тестовую выборки, а специалисты по статистике чаще используют такие метрики, как интервалы прогнозирования и корреляция, чтобы оценить статистическую значимость. Мы рассмотрим оба подхода, чтобы помочь вам преодолеть постоянно увеличивающийся разрыв между обеими дисциплинами и, таким образом, лучше подготовиться к тому, чтобы усидеть на двух стульях.

РАЗВЕ РЕГРЕССИЯ — ЭТО МАШИННОЕ ОБУЧЕНИЕ?

Машинное обучение охватывает много методов, но один из самых распространенных в настоящее время — обучение с учителем, и регрессия играет в нем важную роль. Именно поэтому линейную регрессию считают одной из форм машинного обучения. Путаница в том, что специалисты по статистике часто называют свои регрессионные модели *статистическим обучением*, в то время как в области data science и машинного обучения они называются *машинным обучением*.

Хотя обучение с учителем часто сводится к регрессии, машинное обучение без учителя больше связано с кластеризацией и выявлением аномалий. В обучении с подкреплением нередко сочетается обучение с учителем и симуляция, которая быстро генерирует искусственные данные.

Мы изучим еще две формы машинного обучения с учителем в главе 6, которая посвящена логистической регрессии, и в главе 7, где пойдет речь о нейронных сетях.

Простая линейная регрессия

Я хочу изучить связь между возрастом собаки и тем, сколько раз ее водили к ветеринару. У меня есть искусственная выборка из 10 случайных собак. Я предпочитаю объяснять сложные методы с помощью простых наборов данных (реальных или искусственных), чтобы вам было легче понять сильные и слабые стороны метода, не забивая себе голову сложными данными. Давайте построим график этого набора данных, как показано на рис. 5.2.

Здесь явно просматривается *линейная зависимость*: когда одна из переменных увеличивается или уменьшается, другая тоже увеличивается или уменьшается примерно пропорционально. Чтобы обозначить зависимость, в области этих точек можно провести прямую, как показано на рис. 5.3.

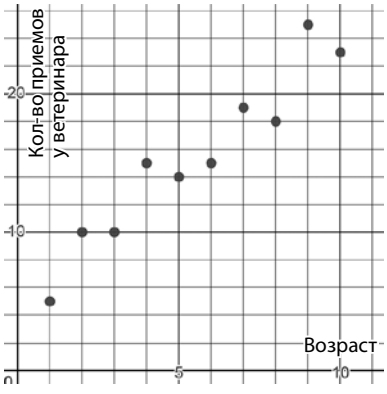


Рис. 5.2. Возраст и количество приемов у ветеринара для выборки из 10 собак

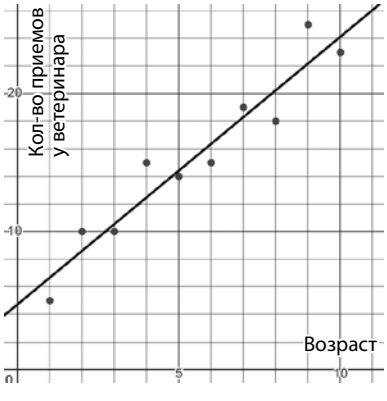


Рис. 5.3. Подгонка прямой к данным

Позже в этой главе я объясню, как подогнать такую прямую, а также как оценить качество подгонки. А пока давайте разберемся, в чем польза от линейной регрессии. Она позволяет прогнозировать данные, с которыми мы раньше не сталкивались. В нашей выборке нет собаки в возрасте 8,5 года, но мы можем посмотреть на эту прямую и предположить, что за свою жизнь такую собаку 21 раз отведут к ветеринару. Для этого достаточно убедиться, что если $x = 8,5$, то $y = 21,218$ (рис. 5.4). Еще одно преимущество линейной регрессии — в том, что она позволяет анализировать переменные на предмет возможных взаимосвязей и выдвигать гипотезы о том, что между коррелирующими переменными есть причинно-следственная связь.

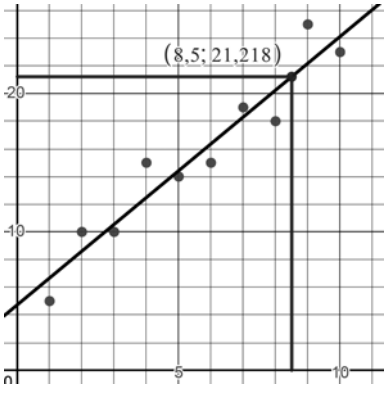


Рис. 5.4. Прогнозирование с помощью линейной регрессии: для собаки возрастом 8,5 года предполагается около 21,2 приема у ветеринара

В чем же недостатки линейной регрессии? Нельзя рассчитывать на то, что каждая точка данных будет лежать *точно* на этой прямой. В конце концов, реальные данные зашумлены, они никогда не бывают идеальными и не придерживаются прямой линии. Их график может быть вообще не похож на прямую! Вокруг прямой будет область погрешности, где точки располагаются выше или ниже линии. Мы рассмотрим это с математической точки зрения, когда будем говорить о p-значениях, статистической значимости и интервалах прогнозирования, которые описывают, насколько можно полагаться на регрессию. Еще одна загвоздка заключается в том, что линейная регрессия не позволяет делать прогнозы вне диапазона имеющихся данных — то есть нельзя прогнозировать значения y для $x < 0$ и $x > 10$, потому что у нас нет данных на этих интервалах.

Не забывайте о смещении выборки!

Следует изучить на предмет смещения исходные данные и то, как они были собраны. Данные собирались только в одной ветеринарной клинике? В нескольких случайных клиниках? Нет ли смещения из-за самоотбора, когда в выборку попадают только собаки, которых водят к ветеринару? Если собаки были отобраны в одном и том же географическом регионе, может ли это повлиять на данные? Возможно, в жарком пустынном климате собакам чаще требуется ветеринар из-за теплового истощения и укусов змей, и из-за этого количество посещений в нашей выборке оказалось завышенным.

Как уже говорилось в главе 3, стало модным преподносить данные как истину в последней инстанции. Однако данные — это всего лишь выборка из совокупности, и нужно позаботиться о том, насколько она репрезентативна. Важно интересоваться тем, о чем говорят данные, но не менее (если не более) важно интересоваться, откуда они взялись.

Простая линейная регрессия с помощью scikit-learn

В этой главе нам предстоит многое узнать о линейной регрессии, но для начала давайте обзорно познакомимся с ней на примере программного кода.

С линейной регрессией умеют работать многие платформы — от Excel до Python и R. В этой книге мы продолжим придерживаться Python и начнем с библиотеки scikit-learn, которая сделает всю работу за нас. Позже в этой главе мы узнаем, как построить линейную регрессию с нуля, чтобы усвоить такие важные методы, как градиентный спуск и метод наименьших квадратов.

В примере 5.1 показано, как с помощью scikit-learn выполнить простейшую линейную регрессию без валидации на выборке из 10 собак. В этом коде мы импортируем данные с помощью pandas (https://oreil.ly/xCvwR), преобразовываем их в массивы NumPy, проводим линейную регрессию с помощью scikit-learn и выводим график с помощью Plotly.

Пример 5.1. Линейная регрессия с помощью scikit-learn

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Импортируем точки данных из внешнего источника
df = pd.read_csv("https://bit.ly/3goOAnt", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1]

# Извлекаем столбец с выходными значениями (все строки, только последний
столбец)
Y = df.values[:, -1]

# Подгоняем прямую к точкам
fit = LinearRegression().fit(X, Y)

m = fit.coef_.flatten()
b = fit.intercept_.flatten()
print(f"m = {m}") # 1.93939394[1]
print(f"b = {b}") # 4.73333333

# Выводим график
plt.plot(X, Y, 'o') # диаграмма рассеяния
plt.plot(X, m*X+b) # линия регрессии
plt.show()
```

[1] Код на Python выводит коэффициенты в виде чисел с плавающей точкой, однако на самом деле в этой регрессии коэффициенты в точности равны $m = 1\frac{31}{33}$ и $b = 4\frac{11}{15}$. — *Примеч. науч. ред.*

Еще раз подытожим, что происходит в этом коде. Сначала мы импортируем данные из файла CSV, который размещен на GitHub (https://bit.ly/3cIH97A). Эти данные состоят из двух столбцов, которые мы с помощью pandas разделяем на наборы данных *X* и *Y*. Затем мы подгоняем модель `LinearRegression` к входным данным *X* и выходным данным *Y* с помощью функции `fit()`. После этого мы получаем коэффициенты *m* и *b*, которые описывают подогнанную линейную функцию.

На графике, как и следовало ожидать, получается подогнанная линия, которая проходит поблизости от точек данных, как показано на рис. 5.5.

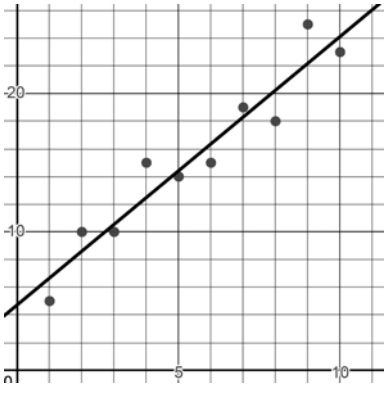


Рис. 5.5. scikit-learn подгоняет линию регрессии к имеющимся данным

Как узнать, какая линия лучше всего подойдет к этим точкам? Обсудим это дальше.

Остатки и квадратичные отклонения

Каким образом статистические инструменты, такие как библиотека scikit-learn, подбирают линию, которая подходит к данным точкам? Этот вопрос сводится к двум другим, на которые опирается все машинное обучение:

- Что значит «наилучшая подгонка»?
- Как добиться «наилучшей подгонки»?

На первый вопрос есть исторически устоявшийся ответ: нужно минимизировать квадраты, а точнее, сумму квадратов остатков. Давайте разберемся, что это значит. Проведите поблизости от точек любую прямую. *Остатки* — это

арифметические разности между точками данных и ближайшими по вертикали точками на прямой, как показано на рис. 5.6.

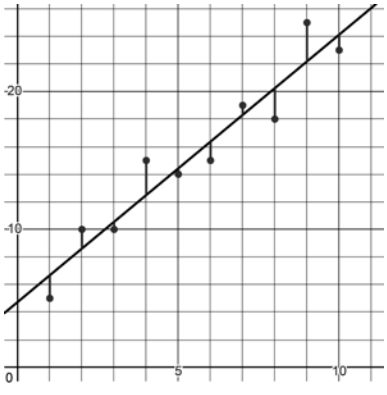


Рис. 5.6. Остатки — это расхождение между прямой и точками данных

Если точка расположена выше прямой, ей соответствует положительный остаток, а если ниже — отрицательный. Другими словами, остаток — это разность между прогнозируемым значением y (которое лежит на прямой) и фактическим значением (из набора данных). Остатки также называются *отклонениями*, или *ошибками*, потому что они отражают, насколько наши прогнозы расходятся с данными.

В примере 5.2 показано, как вычислить остатки между имеющимися 10 точками и прямой $1,93939394x+4,73333333$, а в примере 5.3 перечислены полученные остатки.

Пример 5.2. Вычисление остатков между прямой и точками данных

```
import pandas as pd

# Импортируем точки данных из внешнего источника
points = pd.read_csv("https://bit.ly/3goOAnt", delimiter=",").itertuples()

# Задаем прямую
m = 1.93939394
b = 4.73333333

# Вычисляем остатки
for p in points:
    y_actual = p.y
    y_predict = m*p.x + b
    residual = y_actual - y_predict
    print(residual)
```

Пример 5.3. Остатки для каждой точки

```
-1.6727272727272728  # -1³⁷/₅₅
1.3878787878787868   # 1⁶⁴/₁₆₅
-0.5515151515151508  # -⁹¹/₁₆₅
2.509090909090908    # 2²⁸/₅₅
-0.43030303030302974 # -⁷¹/₁₆₅
-1.369696969696971   # -1⁶¹/₁₆₅
0.6909090909090914   # ³⁸/₅₅
-2.24848484848485    # -2⁴¹/₁₆₅
2.8121212121212125   # 2¹³⁴/₁₆₅
-1.1272727272727288  # -1⁷/₅₅
```

Когда мы подгоняем прямую к 10 точкам данных, скорее всего, нужно минимизировать эти остатки, чтобы итоговый разрыв между прямой и точками был как можно меньше. Но как измерить «итоговый разрыв»? Лучше всего оценить его как *сумму квадратов* всех остатков — то есть возвести каждый остаток в квадрат и сложить их. Из каждого фактического значения y мы вычтем прогнозируемое значение y (то есть координату y соответствующей точки на прямой), а затем возведем в квадрат и просуммируем все полученные разности.

ПОЧЕМУ НЕ ИСПОЛЬЗУЮТСЯ АБСОЛЮТНЫЕ ВЕЛИЧИНЫ?

Возможно, вы зададитесь вопросом, зачем возводить остатки в квадрат, перед тем как их складывать. Почему бы просто не сложить остатки, не возводя их в квадрат? Это не сработает, потому что положительные и отрицательные значения аннулируют друг друга. Ну а что если сложить абсолютные значения, отбросив все знаки минуса? Звучит многообещающе, но абсолютные значения неудобны с математической точки зрения. Если быть точнее, они плохо сочетаются с производными, которые мы вскоре будем использовать для градиентного спуска. Вот почему мы оцениваем общие потери с помощью квадратов остатков.

На рис. 5.7 показано, как визуально представить себе эти вычисления: с каждым остатком связан квадрат, длина стороны которого равна остатку. Мы суммируем площади всех квадратов и позднее узнаем, как найти оптимальные значения m и b, чтобы минимизировать эту сумму.

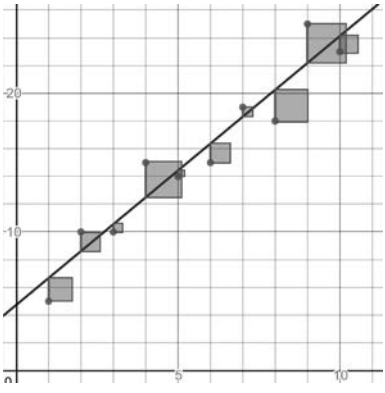


Рис. 5.7. Сумма квадратов остатков в наглядном представлении. Это сумма площадей всех квадратов, длина стороны каждого из которых равна остатку

Давайте модифицируем наш код, как показано в примере 5.4, чтобы найти сумму квадратов.

Пример 5.4. Вычисление суммы квадратов остатков для заданной прямой и данных

```
import pandas as pd

# Импортируем точки данных из внешнего источника
points = pd.read_csv("https://bit.ly/2KF29Bd").itertuples()

# Задаем прямую
m = 1.93939394
b = 4.73333333

sum_of_squares = 0.0

# Вычисляем сумму квадратов
for p in points:
    y_actual = p.y
    y_predict = m*p.x + b
    residual_squared = (y_predict - y_actual)**2
    sum_of_squares += residual_squared

print(f"Сумма квадратов = {sum_of_squares}")
# Сумма квадратов = 28.096969696969715 = 28 32/330
```

Следующий вопрос: как без помощи библиотеки типа scikit-learn найти значения *m* и *b*, которые дадут минимальную сумму квадратов? Мы рассмотрим это в следующем разделе.

Поиск оптимальной прямой

Итак, у нас есть способ измерить, насколько хорошо та или иная прямая подходит к точкам данных: сумме квадратов остатков. Чем меньше это число, тем лучше подогнана прямая. Как же теперь найти нужные значения коэффициентов m и b, которые минимизируют сумму квадратов?

Существует несколько алгоритмов, которые пытаются найти оптимальный набор значений. Худший из них — *метод грубой силы*: вы генерируете миллионы случайных значений m и b и выбираете такое их сочетание, при котором сумма квадратов оказывается наименьшей. Это вряд ли сработает, потому что ни за какое разумное время не получится подобрать даже приблизительное решение. Нам понадобится более систематизированный подход. Я разберу пять методов, которые здесь можно использовать: аналитическое решение, метод обратных матриц, разложение матриц, градиентный спуск и стохастический градиентный спуск. Существуют и другие алгоритмы — например, поиск восхождением (все они описаны в приложении А), но здесь мы будем придерживаться самых распространенных методов.

ОБУЧЕНИЕ МОДЕЛИ — ЭТО ПОДГОНКА РЕГРЕССИИ

Именно это и обозначает слово «обучение» в контексте машинного обучения: мы предоставляем исходные данные и целевую функцию (например, сумму квадратов), а алгоритм находит нужные коэффициенты m и b, чтобы оптимизировать эту функцию. Таким образом, когда мы «обучаем» модель машинного обучения, мы на самом деле минимизируем функцию потерь.

Аналитическое решение

Некоторые читатели могут спросить, существует ли формула (так называемое *аналитическое решение*), по которой можно точно рассчитать линейную регрессию. Ответ — да, но она эффективна только для простой линейной регрессии с одной входной переменной. Для многих задач машинного обучения с несколькими переменными и большим объемом данных не существует эффективного аналитического решения. Чтобы масштабировать задачу, можно использовать методы линейной алгебры, о которых мы поговорим в ближайшее время. Кроме того, мы познакомимся с алгоритмами поиска, такими как стохастический градиентный спуск.

Для простой линейной регрессии с одной входной и одной выходной переменной коэффициенты m и b можно вычислить по формуле, которая приводится ниже. В примере 5.5 показано, как выполнить эти вычисления на Python.

$$m = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2}$$

$$b = \frac{\sum y}{n} - m\frac{\sum x}{n}$$

Пример 5.5. Вычисление коэффициентов m и b для простой линейной регрессии

```
import pandas as pd

# Импортируем данные
points = list(pd.read_csv('https://bit.ly/2KF29Bd', delimiter=",").itertuples())

n = len(points)

m = (n*sum(p.x*p.y for p in points) - sum(p.x for p in points) *
    sum(p.y for p in points)) / (n*sum(p.x**2 for p in points) -
    sum(p.x for p in points)**2)

b = (sum(p.y for p in points) / n) - m * sum(p.x for p in points) / n

print(m, b)
# 1.9393939393939394 4.7333333333333325
```

Эти формулы для вычисления коэффициентов m и b можно вывести методами математического анализа, и позже в этой главе мы немного поработаем с SymPy на тот случай, если у вас возникнет желание узнать, как они выводятся. Пока же можно просто подставить в формулу количество точек данных n, а также перебрать значения x и y, чтобы получить результат.

В дальнейшем мы будем изучать методы, которые больше ориентированы на современную практику работы с большими объемами данных. Аналитические решения, как правило, плохо масштабируются.

Вычислительная сложность

Почему аналитические формулы плохо масштабируются на большие наборы данных? Дело в так называемой *вычислительной сложности*: это понятие из информатики, которое позволяет оценить, сколько ресурсов уйдет на то, чтобы выполнить алгоритм, в зависимости от размера задачи. С этим понятием полезно ознакомиться, и я рекомендую два отличных видеоролика на YouTube на эту тему: «P vs. NP and the Computational Complexity Zoo», (https://oreil.ly/TzQBl), «What Is Big O Notation?», (https://oreil.ly/EjcSR)

Метод обратных матриц

В дальнейшем я буду иногда обозначать коэффициенты буквами β_1 и β_0 вместо *m* и *b* соответственно. С такими обозначениями вам предстоит чаще сталкиваться в профессиональной среде, так что сейчас самое время к ним привыкать.

Хотя мы посвятили линейной алгебре целую главу, применять ее может оказаться непосильной задачей, если вы еще неуверенно ориентируетесь в математике и data science. Именно поэтому в большинстве примеров из этой книги используется обычный Python или библиотека scikit-learn. Однако я буду обращаться к линейной алгебре там, где это имеет смысл, — просто чтобы показать, как она бывает полезна. Если этот раздел покажется вам слишком сложным, пропустите его и вернитесь сюда позже.

Чтобы подогнать линейную регрессию, можно использовать транспонированные и обратные матрицы, которые мы изучали в главе 4. Далее мы вычислим вектор коэффициентов *b*, взяв за основу матрицу значений входных переменных *X* и вектор значений выходных переменных *y*. Не уходя глубоко в математический анализ и доказательства из линейной алгебры, приведем готовую формулу:

$$b = \left(X^T \cdot X\right)^{-1} \cdot X^T \cdot y$$

Как вы наверняка заметили, здесь над матрицей *X* выполняются операции транспонирования и поиска обратной матрицы, а также происходит умножение матриц. В примере 5.6 показано, как по этой формуле вычислить коэффициенты регрессии *m* и *b* с помощью NumPy.

Пример 5.6. Использование обратных и транспонированных матриц для подгонки линейной регрессии

```
import pandas as pd
from numpy.linalg import inv
import numpy as np

# Импортируем данные
df = pd.read_csv("https://bit.ly/3goOAnt", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1].flatten()

# Добавляем столбец-заполнитель из единиц, чтобы сгенерировать пересечение
с осью
X_1 = np.vstack([X, np.ones(len(X))]).T

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]
```

```
# Вычисляем угловой коэффициент и ординату пересечения с осью Y
b = inv(X_1.transpose() @ X_1) @ (X_1.transpose() @ Y)
print(b) # [1.93939394, 4.73333333]

# Прогнозируем значения y
y_predict = X_1.dot(b)
```

Это непросто понять интуитивно, но обратите внимание, что мы создаем столбец из одних единиц и размещаем его рядом со столбцом *X*. Это нужно для того, чтобы получить β_0 — точку пересечения с осью *Y*. Поскольку в этом столбце все значения равны 1, он фактически генерирует ординату пересечения в дополнение к угловому коэффициенту β_1.

Разложение матриц

Когда у вас много данных и много измерений, компьютеры могут начать захлебываться и выдавать нестабильные результаты. Это подходящий сценарий для разложения матриц, о котором мы узнали в главе 4, посвященной линейной алгебре. В данном случае мы возьмем матрицу *X*, добавим к ней дополнительный столбец из единиц, чтобы сгенерировать пересечение β_0, как в предыдущем примере, а затем разложим матрицу на две другие матрицы *Q* и *R*:

$$X = Q \cdot R.$$

Не углубляясь в математический анализ, приведем формулу, по которой с помощью *Q* и *R* можно вычислить коэффициенты β_1 и β_0 в форме матрицы *b*:

$$b = R^{-1} \cdot Q^T \cdot y.$$

В примере 5.7 показано, как эта формула *QR*-разложения позволяет выполнить линейную регрессию на Python с помощью NumPy.

Пример 5.7. Линейная регрессия с помощью *QR*-разложения

```
import pandas as pd
from numpy.linalg import qr, inv
import numpy as np

# Импортируем данные
df = pd.read_csv("https://bit.ly/3goOAnt", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1].flatten()
```

```
# Добавляем столбец-заполнитель из единиц, чтобы сгенерировать пересечение
с осью
X_1 = np.vstack([X, np.ones(len(X))]).transpose()

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]

# Вычисляем угловой коэффициент и пересечение с осью Y
# с помощью QR-разложения
Q, R = qr(X_1)
b = inv(R).dot(Q.transpose()).dot(Y)

print(b) # [1.93939394, 4.73333333]
```

Чтобы выполнить линейную регрессию, во многих научных библиотеках используется именно метод *QR*-разложения, потому что он легче справляется с большими объемами данных и работает устойчивее. Что значит «устойчивее»? *Вычислительная устойчивость* (https://oreil.ly/A4BWJ) говорит о том, насколько хорошо алгоритм минимизирует ошибки, а не усугубляет их при приближенных вычислениях. Помните, что компьютеры работают с десятичными дробями ограниченной точности и вынуждены выполнять приближенные вычисления, поэтому важно, чтобы наши алгоритмы не деградировали из-за того, что в этих вычислениях накапливаются ошибки.

Тяжело усваивать материал?

Не переживайте, если вам покажутся слишком сложными примеры того, как вычислять линейную регрессию с помощью линейной алгебры! Я всего лишь хотел продемонстрировать, как линейную алгебру можно применять на практике. В дальнейшем мы сосредоточимся на других методах, которые могут вам пригодиться.

Градиентный спуск

Градиентный спуск — это метод оптимизации, где используются производные и итерации, чтобы подобрать набор параметров, при котором минимизируется или максимизируется значение целевой функции. Чтобы понять, что такое градиентный спуск, давайте проведем незатейливый мысленный эксперимент, а затем применим его результаты на простом примере.

Мысленный эксперимент с градиентным спуском

Представьте, что вы находитесь ночью в горной местности, и у вас есть фонарик. Вы пытаетесь добраться до самой низкой точки местности. Вы обозреваете

ближайший участок склона вокруг себя еще до того, как начнете движение. Вы делаете каждый следующий шаг в том направлении, где склон круче всего уходит вниз. На крутых склонах вы делаете большие шаги, а на пологих — маленькие. В конце концов вы окажетесь в нижней точке, где местность плоская, то есть ее уклон равен 0. Звучит неплохо, правда? Такое движение с фонариком известно как *градиентный спуск*, когда мы двигаемся в ту сторону, где склон круче понижается.

В машинном обучении зачастую приходится задумываться обо всех возможных суммах квадратов потерь, с которыми можно столкнуться при различных параметрах, наподобие крутизны горного склона. Чтобы минимизировать потери, мы перемещаемся по ландшафту потерь. Для этого у градиентного спуска есть привлекательная особенность: частная производная служит тем самым фонариком, который позволяет видеть крутизну ближайшего склона для каждого параметра (в данном случае m и b, или β_0 и β_1). Мы двигаемся в тех направлениях для m и b, где угловой коэффициент максимален. Для бóльших угловых коэффициентов мы делаем более широкий шаг, а для меньших — более узкий. Ширину этого шага можно вычислить, взяв определенную долю от углового коэффициента, — она известна как *скорость обучения*. Чем выше скорость обучения, тем быстрее будет работать система за счет снижения точности. Но чем ниже скорость, тем больше времени займет обучение и тем больше итераций потребуется.

Выбирать скорость обучения — все равно что выбирать между муравьем, человеком или великаном, который спускается по склону. Муравей (маленькая скорость обучения) будет делать крошечные шаги и потратит неприемлемо много времени, чтобы добраться до подножия горы, но сделает это точно. Великан (большая скорость обучения) может постоянно перешагивать через минимум и никогда в него не попасть, сколько бы шагов он ни сделал. У человека (умеренная скорость обучения) ширина шага, вероятно, наиболее сбалансирована: она позволяет достичь оптимального баланса между скоростью и точностью обнаружения минимума.

Научимся ходить, прежде чем бегать

Для функции $f(x)=(x-3)^2+4$ найдем значение x, при котором значение функции минимально. Хотя эту задачу легко решить алгебраически, давайте воспользуемся градиентным спуском.

На рис 5.8 показано, что мы пытаемся сделать. Мы хотим «шагнуть» по оси x в ту сторону, где достигается минимум функции (то есть угол наклона равен 0).

В примере 5.8 используется функция `f(x)` и ее производная по x — `dx_f(x)`. Напомним, что в главе 1 мы рассказывали о том, как вычислять производные с помощью SymPy. После того как мы нашли производную, можно перейти к градиентному спуску.

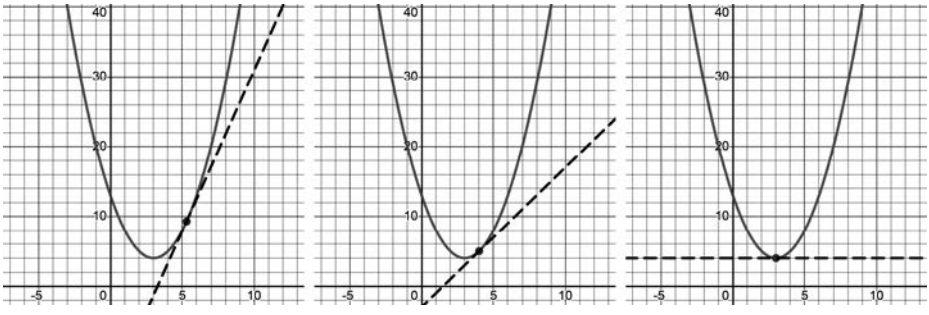


Рис. 5.8. Движение к локальному минимуму, где угол наклона достигает 0

Пример 5.8. Градиентный спуск для нахождения минимума параболы

```
import random

def f(x):
    return (x - 3) ** 2 + 4

def dx_f(x):
    return 2*(x - 3)

# Скорость обучения
L = 0.001

# Количество итераций градиентного спуска
iterations = 100_000

# Начинаем со случайного значения x
x = random.randint(-15,15)

for i in range(iterations):

    # Вычисляем угловой коэффициент
    d_x = dx_f(x)

    # Обновляем x, вычитая из него произведение скорости обучения на угловой
коэффициент
    x -= L * d_x

print(x, f(x)) # выводит 2.999999999999889 4.0
```

Если построить график функции (как показано на рис. 5.8), можно увидеть, что ее минимум находится в точке $x = 3$, и предыдущий пример подошел очень близко к этому значению. Скорость обучения — это фактически доля от углового коэффициента, которая вычитается из значения x на каждой итерации. Бóльшие угловые коэффициенты приводят к бóльшим шагам, а меньшие — к меньшим.

После достаточного количества итераций x окажется в самой нижней точке функции, где наклон равен 0 (или весьма близко к этой точке).

Градиентный спуск и линейная регрессия

Теперь вам, наверное, интересно, как применить эту процедуру к линейной регрессии. Идея остается прежней, только теперь в роли переменных выступают m и b (они же β_0 и β_1), а не x. Почему так? Дело в том, что в простой линейной регрессии нам уже известны значения x и y, потому что они доступны в составе обучающих данных. Переменные, которые нужно найти, — это на самом деле параметры m и b. Они задают оптимально подогнанную прямую, с помощью которой можно предсказывать новые значения y по значениям x.

Как вычислить угловые коэффициенты («крутизну склона») для m и b[1]? Нам понадобятся частные производные по каждой из этих переменных. От какой функции брать производные? Помните, что наша цель — минимизировать потери, поэтому целевой функцией будет сумма квадратов отклонений. Таким образом, мы ищем производные от этой функции по m и b.

В примере 5.9 фигурируют две частные производные — по m и по b. (Вскоре мы узнаем, как выполнить те же вычисления с помощью SymPy.) В этом примере мы ищем m и b методом градиентного спуска: 100 000 итераций при скорости обучения 0,001 будет достаточно. Обратите внимание, что чем меньше скорость обучения, тем медленнее оно происходит и тем больше итераций требуется. Однако если задать слишком высокую скорость, то алгоритм будет работать быстро, но даст плохое приближение. Когда говорят, что алгоритм машинного обучения «обучается» или «тренируется», это значит, что он на самом деле просто подгоняет регрессию, как здесь.

Пример 5.9. Градиентный спуск для линейной регрессии

```
import pandas as pd

# Импортируем данные
points = list(pd.read_csv("https://bit.ly/2KF29Bd").itertuples())

# Строим модель
m = 0.0
b = 0.0
```

[1] Не запутайтесь с угловыми коэффициентами! Когда мы рассматриваем готовую прямую регрессии $y = mx + b$, то m служит угловым коэффициентом (другими словами, это тангенс угла наклона прямой по отношению к положительному направлению оси X). Но здесь задача состоит в том, чтобы найти саму прямую регрессии — то есть вычислить ее коэффициенты m и b. Для этого мы минимизируем функцию потерь $\sum_i \left((mx_i + b) - y_i \right)^2$ и исследуем наклоны (то есть угловые коэффициенты) графика этой функции в направлениях m и b. — *Примеч. науч. ред.*

```
# Скорость обучения
L = .001

# Количество итераций
iterations = 100_000

n = float(len(points)) # Количество элементов в наборе данных

# Градиентный спуск
for i in range(iterations):

    # Угловой коэффициент в направлении m
    D_m = sum(2 * p.x * ((m * p.x + b) - p.y) for p in points)

    # Угловой коэффициент в направлении b
    D_b = sum(2 * ((m * p.x + b) - p.y) for p in points)

    # Обновляем m и b
    m -= L * D_m
    b -= L * D_b

print(f"y = {m}x + {b}")
# y = 1.9393939393939548x + 4.733333333333227
```

Что ж, неплохо! Эта аппроксимация довольно близка к аналитическому решению. Но в чем же загвоздка? То, что мы нашли «оптимально подогнанную прямую», минимизировав сумму квадратов, еще не означает, что наша линейная регрессия удачна. Если минимизировать сумму квадратов — гарантирует ли это, что получится отличная модель для прогнозов? Не совсем. После того как мы узнали, как подогнать линейную регрессию, давайте сделаем шаг назад, взглянем на общую картину и прежде всего поймем, обеспечит ли эта регрессия качественные прогнозы.

Но прежде чем заняться этим, ненадолго отвлечемся на альтернативное решение — с помощью библиотеки SymPy.

Градиентный спуск для линейной регрессии с помощью SymPy

Если вы хотите узнать, как с помощью SymPy удалось получить обе производные для функции суммы квадратов (по m и b), то взгляните на соответствующий код в примере 5.10.

Пример 5.10. Вычисление частных производных по m и b

```
from sympy import *

m, b, i, n = symbols('m b i n')
x, y = symbols('x y', cls=Function)
```

```
sum_of_squares = Sum((m*x(i) + b - y(i)) ** 2, (i, 0, n))

d_m = diff(sum_of_squares, m)
d_b = diff(sum_of_squares, b)

print(d_m)
print(d_b)

# ВЫВОД
# Sum(2*(b + m*x(i) - y(i))*x(i), (i, 0, n))
# Sum(2*b + 2*m*x(i) - 2*y(i), (i, 0, n))
```

Эта программа выводит две частные производные — по m и по b. Обратите внимание, что функция `Sum()` перебирает элементы набора и складывает их (в данном случае это все точки данных), а x и y мы рассматриваем как функции, которые возвращают координаты точки по индексу i.

В математической записи, где $e(x)$ означает функцию потерь в виде суммы квадратов остатков, эта функция и ее частные производные по m и b выглядят так:

$$e(x) = \sum_{i=0}^{n} \left((mx_i + b) - y_i \right)^2$$

$$\frac{d}{dm} e(x) = \sum_{i=0}^{n} 2 \left(b + mx_i - y_i \right) x_i$$

$$\frac{d}{db} e(x) = \sum_{i=0}^{n} \left(2b + 2mx_i - 2y_i \right)$$

Чтобы задействовать наш набор данных и провести линейную регрессию методом градиентного спуска, придется выполнить несколько дополнительных шагов, как показано в примере 5.11. В производные `d_m` и `d_b` нужно подставить значения `n`, `x(i)` и `y(i)` для всех точек данных. В результате должны остаться только переменные `m` и `b`, оптимальные значения которых нужно найти с помощью градиентного спуска.

Пример 5.11. Линейная регрессия с помощью SymPy

```
import pandas as pd
from sympy import *

# Импортируем данные
points = list(pd.read_csv("https://bit.ly/2KF29Bd").itertuples())

m, b, i, n = symbols('m b i n')
x, y = symbols('x y', cls=Function)

sum_of_squares = Sum((m*x(i) + b - y(i)) ** 2, (i, 0, n))
```

```
d_m = diff(sum_of_squares, m) \
    .subs(n, len(points) - 1).doit() \
    .replace(x, lambda i: points[i].x) \
    .replace(y, lambda i: points[i].y)

d_b = diff(sum_of_squares, b) \
    .subs(n, len(points) - 1).doit() \
    .replace(x, lambda i: points[i].x) \
    .replace(y, lambda i: points[i].y)

# Компилируем производные с помощью lambdify, чтобы ускорить вычисления
d_m = lambdify([m, b], d_m)
d_b = lambdify([m, b], d_b)

# Строим модель
m = 0.0
b = 0.0

# Скорость обучения
L = .001

# Количество итераций
iterations = 100_000

# Градиентный спуск
for i in range(iterations):

    # Обновляем m и b
    m -= d_m(m,b) * L
    b -= d_b(m,b) * L

print(f"y = {m}x + {b}")
# y = 1.939393939393954x + 4.733333333333231
```

Как показано в примере 5.11, для обеих функций частных производных стоит вызвать функцию `lambdify()`, чтобы преобразовать их из SymPy в оптимизированные функции Python. В результате при градиентном спуске вычисления будут гораздо быстрее. Полученные функции Python совместимы с NumPy, SciPy или любыми другими числовыми библиотеками, которые SymPy обнаружит в вашей системе. После того как функции преобразованы, можно выполнять градиентный спуск.

Наконец, если вам любопытно, как выглядит функция потерь для этой простой линейной регрессии, то в примере 5.12 показан код на SymPy, который подставляет значения `x`, `y` и `n` в функцию потерь, а затем строит ее график в зависимости от `m` и `b` (см. рис. 5.9). Алгоритм градиентного спуска приводит к самой нижней точке этого графика.

Пример 5.12. График функции потерь для линейной регрессии

```
from sympy import *
from sympy.plotting import plot3d
import pandas as pd

points = list(pd.read_csv("https://bit.ly/2KF29Bd").itertuples())
m, b, i, n = symbols('m b i n')
x, y = symbols('x y', cls=Function)

sum_of_squares = Sum((m*x(i) + b - y(i)) ** 2, (i, 0, n)) \
    .subs(n, len(points) - 1).doit() \
    .replace(x, lambda i: points[i].x) \
    .replace(y, lambda i: points[i].y)

plot3d(sum_of_squares)
```

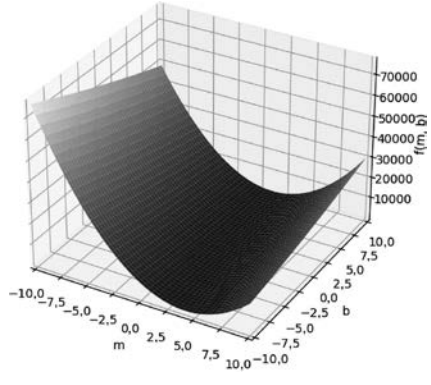


Рис. 5.9. График функции потерь для простой линейной регрессии

Переобучение и дисперсия

Подумайте вот о чем: если бы мы хотели полностью минимизировать потери, то есть уменьшить сумму квадратов до 0, как бы мы поступили? Есть ли другие варианты, кроме линейной регрессии? Одно из решений, которое приходит в голову, — это построить кривую, которая соединяет все точки. Действительно, почему бы просто не соединить точки отрезками, как показано на рис. 5.10, и не использовать эту модель для прогнозов? Это даст нам нулевые потери!

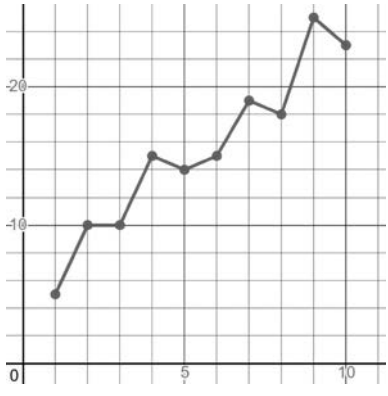


Рис. 5.10. Регрессия путем простого соединения точек приводит к нулевым потерям

И правда, зачем мы так долго возились с линейной регрессией, а не поступили по-простому? Помните, что наша стратегическая цель — не минимизировать сумму квадратов, а делать точные предсказания на новых входных данных. Эта модель «рисования по точкам» сильно *переобучена*, то есть она слишком точно подстраивает регрессию под обучающие данные и в результате будет плохо работать с новыми данными. Переобученная модель чувствительна к выбросам, которые находятся далеко от остальных точек, а значит, в прогнозах будет высокая *дисперсия*. Хотя в этом примере точки расположены относительно близко к прямой линии, проблема будет намного хуже, если работать с другими наборами данных, где наблюдается больший разброс и выбросы. Поскольку из-за переобучения увеличивается дисперсия, прогнозируемые значения могут оказаться где ни попадя!

Переобучение — это запоминание

Когда вы слышите, будто регрессия «запомнила» (или «вызубрила») данные, а не обобщила их, — речь идет о переобучении.

Как вы уже догадались, модель строится затем, чтобы найти эффективные обобщения, а не зубрить данные. Иначе регрессия превратится просто в базу данных, которая годится только для того, чтобы искать в ней уже имеющиеся значения.

Именно поэтому в машинном обучении к модели добавляется смещение, а линейная регрессия считается сильно смещенной моделью. Это не то же самое, что смещение в данных, о котором мы подробно говорили в главе 3. *Смещение модели* означает, что мы отдаем предпочтение определенной схеме (например, тому, чтобы поддерживать прямую линию) в противовес тому, чтобы изгибать

график и точно подгонять модель под данные. Смещенная модель оставляет некоторое пространство для маневра, благодаря чему можно минимизировать потери на новых данных и получить более точные прогнозы вместо того, чтобы минимизировать потери на данных, на которых модель была обучена. Можно сказать, что, если добавить смещение в модель, мы предотвращаем переобучение, ради чего допускаем *недообучение*, то есть меньшую подгонку к обучающим данным.

Как вы понимаете, эта задача требует сложного балансирования, потому что она ориентируется на две противоположные цели. По сути в машинном обучении мы говорим: «Я хочу подогнать регрессию под свои данные, но не хочу подгонять ее *слишком сильно*. Мне нужен некоторый запас для прогнозов на новых данных, которые будут отличаться от имеющихся».

Лассо-регрессия и гребневая регрессия

Еще два довольно популярных варианта линейной регрессии — метод «лассо» и гребневая регрессия. Гребневая регрессия добавляет к линейной регрессии дополнительное смещение в виде штрафа, отчего та хуже подгоняется к данным. Лассо-регрессия пытается изолировать шумные переменные, благодаря чему она полезна, когда нужно автоматически удалить переменные, которые могут быть нерелевантными.

Как бы то ни было, нельзя просто механически применить линейную регрессию к тем или иным данным, сделать прогнозы на ее основе и считать, что все в порядке. Регрессия может оказаться переобученной, даже если смещение сводится к прямой линии. Поэтому всегда нужно анализировать модель на предмет как переобучения, так и недообучения, и искать золотую середину между этими крайностями. Если найти такую середину не получается, всю модель придется отбросить.

Стохастический градиентный спуск

В контексте машинного обучения вам вряд ли доведется выполнять градиентный спуск так, как мы делали это до сих пор, когда обучали модель на основе всех обучающих данных (так называемый *пакетный градиентный спуск*). На практике вы, скорее всего, будете заниматься *стохастическим градиентным спуском*, который на каждой итерации обучается только на одной выборке из набора данных. При *мини-пакетном градиентном спуске* на каждой итерации используется несколько выборок (например, по 10 или 100 точек данных).

Зачем использовать только часть данных на каждой итерации? Специалисты по машинному обучению видят в таком подходе несколько преимуществ. Во-первых, он значительно сокращает вычисления, потому что на каждой итерации приходится перебирать не весь обучающий набор данных, а только его фрагмент. Второе преимущество — меньшее переобучение. Если на каждой итерации алгоритм обучения получает только часть данных, то функция потерь постоянно меняется, поэтому спуск не останавливается на ее минимуме. В конце концов, именно минимизация потерь приводит к переобучению, поэтому стоит внести некоторую случайность, чтобы модель осталась немного недообученной (но будем надеяться, что не чересчур).

Конечно, при этом аппроксимация становится менее точной, поэтому нужно действовать осторожно. Именно поэтому в ближайшее время мы поговорим о том, как разделять данные на обучающую и тестовую выборки, а также о других метриках, с помощью которых можно оценивать надежность линейной регрессии.

Пример 5.13 демонстрирует, как выполнить стохастический градиентный спуск на Python. Если задать размер выборки больше 1, получится мини-пакетный градиентный спуск.

Пример 5.13. Стохастический градиентный спуск для линейной регрессии

```
import pandas as pd
import numpy as np

# Импортируем данные
data = pd.read_csv("https://bit.ly/2KF29Bd", header=0)

X = data.iloc[:, 0].values
Y = data.iloc[:, 1].values

n = data.shape[0] # количество строк

# Строим модель
m = 0.0
b = 0.0

sample_size = 1 # размер выборки
L = .0001 # скорость обучения
epochs = 1_000_000 # количество итераций для градиентного спуска

# Стохастический градиентный спуск
for i in range(epochs):
    idx = np.random.choice(n, sample_size, replace=False)
    x_sample = X[idx]
    y_sample = Y[idx]
```

```
    # Текущее прогнозируемое значение Y
    Y_pred = m * x_sample + b

    # Производная функции потерь по m
    D_m = (-2 / sample_size) * sum(x_sample * (y_sample - Y_pred))

    # Производная функции потерь по b
    D_b = (-2 / sample_size) * sum(y_sample - Y_pred)

    m = m - L * D_m # обновляем m
    b = b - L * D_b # обновляем b

    # Выводим параметры каждой итерации
    if i % 10000 == 0:
        print(i, m, b)

print(f"y = {m}x + {b}")
```

Когда я запустил код, то получил линейную регрессию $y = 1,9382830354181135x + 4,753408787648379$. Практически наверняка ваши результаты будут немного отличаться, потому что стохастический градиентный спуск не сходится к конкретному минимуму, а заканчивается где-то в его окрестности.

Случайность — это плохо?

Если вам некомфортно иметь дело со случайностью и получать разные ответы каждый раз, когда вы запускаете код, — что поделать! Так устроен мир машинного обучения, оптимизации и стохастических алгоритмов! Многие алгоритмы аппроксимации основаны на случайности, и хотя некоторые из них чрезвычайно полезны, другие, как и следовало ожидать, могут работать небрежно и приводить к неудовлетворительным результатам.

Многие воспринимают машинное обучение и искусственный интеллект как инструмент, который дает объективные и точные ответы, но это очень далеко от истины. Машинное обучение обеспечивает приближенные результаты с той или иной степенью неопределенности, часто не опираясь на фундаментальную истину. Его легко использовать не по назначению, если не знать, как оно работает, и было бы неправильно не учитывать его недетерминированный и приблизительный характер.

Хотя случайность позволяет создавать весьма эффективные инструменты, ею часто злоупотребляют. Не стоит заниматься *p*-хакингом и манипулировать затравочными значениями и случайностью в надежде получить «хороший» результат; лучше приложить усилия к тому, чтобы проанализировать данные и модель.

Коэффициент корреляции

Взгляните на диаграмму рассеяния и соответствующую линейную регрессию на рис. 5.11. Почему в данном случае линейная регрессия может плохо сработать?

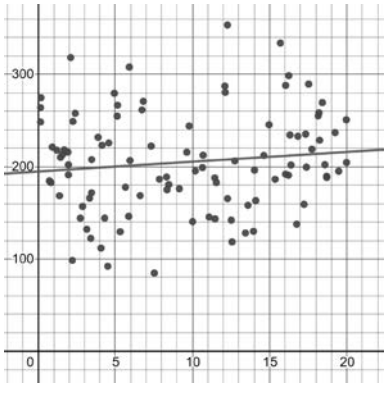


Рис. 5.11. Диаграмма рассеяния данных с высокой дисперсией

Проблема в том, что данные отличаются высокой дисперсией. Если данные сильно разбросаны, дисперсия может увеличиться до такой степени, что прогнозы получатся менее точными и менее полезными из-за больших остатков. Конечно, такую смещенную модель, как линейная регрессия, можно заставить не отклоняться и меньше реагировать на дисперсию. Однако недообучение тоже будет портить прогнозы, потому что данные очень сильно разбросаны. Хотелось бы численно оценить, насколько «промахиваются» наши прогнозы.

Как же измерить эти остатки в целом? Как получить представление о том, насколько велика дисперсия в данных? Позвольте представить вам *линейный коэффициент корреляции*, который также называется *коэффициентом корреляции Пирсона* и оценивает степень взаимосвязи между двумя переменными в виде значения от −1 до 1. Если коэффициент корреляции близок к 0, это значит, что взаимосвязи нет. Чем ближе он к 1, тем сильнее *положительная корреляция*, когда при увеличении одной переменной пропорционально увеличивается и другая. Если коэффициент близок к −1, это означает сильную *отрицательную корреляцию*, когда при увеличении одной переменной другая пропорционально уменьшается.

Коэффициент корреляции часто обозначается буквой *r*. У сильно разбросанных данных на рис. 5.11 он равен 0,1201. Поскольку это значение гораздо ближе к 0, чем к 1, можно сделать вывод, что данные слабо связаны между собой.

На рис. 5.12 приведены еще четыре диаграммы рассеяния и указаны соответствующие коэффициенты корреляции. Обратите внимание, что чем больше

расположение точек похоже на прямую линию, тем сильнее корреляция, а беспорядочно разбросанные точки дают слабую корреляцию.

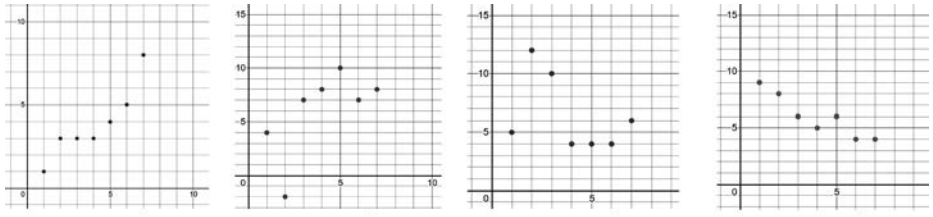


Рис. 5.12. Коэффициенты корреляции для четырех диаграмм рассеяния

Как вы наверняка догадались, коэффициент корреляции помогает понять, можно ли предположить связь между двумя переменными. Если наблюдается сильная положительная или отрицательная корреляция, то переменные могут пригодиться для линейной регрессии. Но если корреляции нет, они грозят просто добавить шум и ухудшить точность модели.

Как вычислить коэффициент корреляции на Python? Давайте воспользуемся простым набором данных из 10 точек (https://bit.ly/2KF29Bd), с которым мы уже работали. Чтобы быстро и просто проанализировать корреляции для всех пар переменных, можно использовать функцию `corr()` из библиотеки pandas. Она выводит коэффициенты корреляции между переменными в каждой паре из набора данных — в нашем случае это только переменные `x` и `y`. Такая конструкция называется *корреляционной матрицей* и вычисляется в примере 5.14.

Пример 5.14. Вычисление коэффициентов корреляции для каждой пары переменных с помощью библиотеки pandas

```
import pandas as pd

# Загружаем данные в датафрейм pandas
df = pd.read_csv("https://bit.ly/2KF29Bd", delimiter=",")

# Выводим коэффициенты корреляции между переменными
correlations = df.corr(method="pearson")
print(correlations)

# ВЫВОД:
#   x        y
# x 1.000000 0.957586
# y 0.957586 1.000000
```

Коэффициент корреляции 0,957586 между *x* и *y* указывает на сильную положительную связь между этими двумя переменными. Можно не обращать внимания на те ячейки матрицы, где *x* или *y* сопоставляются сами с собой и корреляция равна 1[1]. Очевидно, что если сопоставлять переменную саму с собой, то корреляция будет идеальной и ее коэффициент составит ровно 1, потому что каждое значение переменной точно совпадает само с собой. Когда переменных больше двух, в корреляционной матрице будет больше строк и столбцов — по количеству переменных, которые нужно сопоставить и скоррелировать.

Если подставить в код другой набор данных с большой дисперсией (где данные сильнее разбросаны), вы убедитесь, что коэффициент корреляции уменьшается, что указывает на более слабую взаимосвязь.

КАК ВЫЧИСЛИТЬ КОЭФФИЦИЕНТ КОРРЕЛЯЦИИ

Для тех, кто интересуется математикой, приведу формулу, по которой вычисляется коэффициент корреляции:

$$r = \frac{n\sum xy - \left(\sum x\right)\left(\sum y\right)}{\sqrt{n\sum x^2 - \left(\sum x^2\right)}\sqrt{n\sum y^2 - \left(\sum y^2\right)}}$$

Чтобы реализовать эту формулу на Python, я предпочитаю использовать однострочные циклы `for`, которые суммируют элементы. В примере 5.15 показано, как вычислить коэффициент корреляции с нуля на Python.

Пример 5.15. Вычисление коэффициента корреляции с нуля на Python

```
import pandas as pd
from math import sqrt

# Импортируем данные
points = list(pd.read_csv("https://bit.ly/2KF29Bd").itertuples())
n = len(points)

numerator = n * sum(p.x * p.y for p in points) - \
    sum(p.x for p in points) * sum(p.y for p in points)

denominator = sqrt(n*sum(p.x**2 for p in points) - sum(p.x for p in points)**2)
\
    * sqrt(n*sum(p.y**2 for p in points) - sum(p.y for p in points)**2)
```

[1] В неформальной речи для краткости иногда говорят «корреляция равна N», имея в виду, что *коэффициент корреляции* равен *N*. — *Примеч. науч. ред.*

```
corr = numerator / denominator

print(corr)

# ВЫВОД:
# 0.9575860952087218
```

Статистическая значимость

В линейной регрессии нужно учитывать еще один аспект: не является ли корреляция данных случайной? В главе 3 мы познакомились с проверкой гипотез и *p*-значениями, а теперь рассмотрим эти понятия на примере линейной регрессии.

> **БИБЛИОТЕКА STATSMODEL**
>
> Хотя в рамках этой книги мы не собираемся осваивать еще одну библиотеку, стоит упомянуть, что если вы хотите заниматься статистическим анализом, то имеет смысл обратить внимание на библиотеку statsmodel (https://oreil.ly/8oEHo).
>
> В scikit-learn и других библиотеках машинного обучения нет инструментов, которые позволяли бы оценивать статистическую значимость и строить доверительные интервалы; причины этого мы обсудим в другой врезке. Мы будем реализовывать эти инструменты самостоятельно. Но знайте, что подходящая библиотека существует, и ее стоит попробовать!

Начнем с основополагающего вопроса: может ли быть так, чтобы линейная зависимость в данных наблюдалась в результате случайного совпадения? Как добиться 95 %-ной уверенности в том, что корреляция между двумя переменными значима, а не случайна? Если это напомнило вам проверку гипотез из главы 3, то знайте, что это она и есть! Нам нужно не просто вычислить коэффициент корреляции, но и количественно оценить, насколько мы уверены в том, что он возник не случайно.

Здесь мы оцениваем не среднее арифметическое, как в главе 3 на примере с испытанием лекарства, а коэффициент корреляции генеральной совокупности на основе выборки. Коэффициент корреляции совокупности мы обозначим греческой буквой ρ (ро), а выборки — r. Как и в главе 3, у нас будет нулевая гипотеза H_0 и альтернативная H_1:

$$H_0 : \rho = 0 (\text{нет взаимосвязи});$$

$$H_1 : \rho \neq 0 (\text{есть взаимосвязь}).$$

Нулевая гипотеза H_0 заключается в том, что между двумя переменными нет связи, или, говоря более формально, коэффициент корреляции равен 0. Альтернативная гипотеза H_1 заключается в том, что связь есть, и она может быть как положительной, так и отрицательной. Поэтому альтернативная гипотеза формулируется как $\rho \neq 0$, чтобы учесть и положительную, и отрицательную корреляцию.

Вернемся к набору данных из 10 точек, который представлен на рис. 5.13. Насколько правдоподобно, что эти точки образовались случайно и при этом их конфигурация напоминает линейную зависимость?

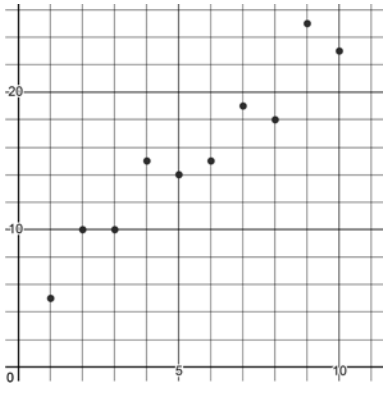


Рис. 5.13. Насколько правдоподобно, что мы по случайному стечению обстоятельств получим такие данные, что их диаграмма рассеяния напоминает линейную зависимость?

В примере 5.14 мы уже вычислили для этого набора данных коэффициент корреляции, который составил 0,957586. Это сильная и убедительная положительная корреляция. Но по-прежнему нужно оценить, не объясняется ли она случайным везением. Давайте проверим нашу гипотезу с помощью двустороннего теста на доверительном уровне 95 % и выясним, есть ли связь между этими двумя переменными.

В главе 3 упоминалось распределение Стьюдента, которое отличается от нормального более широкими хвостами, отражающими бóльшую дисперсию и неопределенность. Для проверки гипотез в контексте линейной регрессии мы будем использовать распределение Стьюдента, а не нормальное. Сначала построим график распределения Стьюдента и выделим на нем 95 %-ный критический интервал, как показано на рис. 5.14. В нашей выборке 10 записей, поэтому у распределения будет 9 степеней свободы ($10-1=9$).

Критическое значение составляет примерно ±2,262, и его можно вычислить на Python, как показано в примере 5.16. Соответствующий критический интервал охватывает 95 % площади в центральной части распределения Стьюдента.

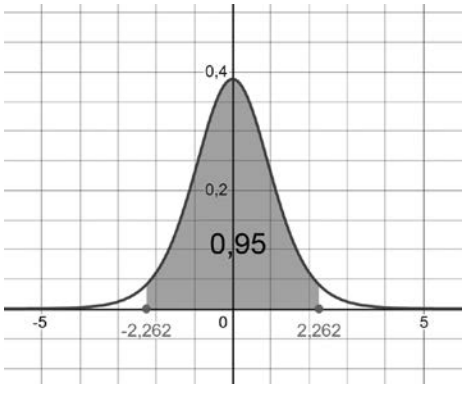


Рис. 5.14. Распределение Стьюдента с 9 степенями свободы

Пример 5.16. Вычисление критического значения для распределения Стьюдента

```
from scipy.stats import t

n = 10
lower_cv = t(n-1).ppf(.025)
upper_cv = t(n-1).ppf(.975)

print(lower_cv, upper_cv)
# -2.2621571628540997 2.2621571628540993
```

Если тестовое значение (так называемая *t*-статистика) окажется вне интервала (–2,262, 2,262), можно будет отвергнуть нулевую гипотезу. Чтобы рассчитать *t*-статистику, воспользуемся следующей формулой. Здесь r — коэффициент корреляции, а n — размер выборки:

$$t = \frac{r}{\sqrt{\frac{1-r^2}{n-2}}}$$

$$t = \frac{0,957586}{\sqrt{\frac{1-0,957586^2}{10-2}}} = 9,339958$$

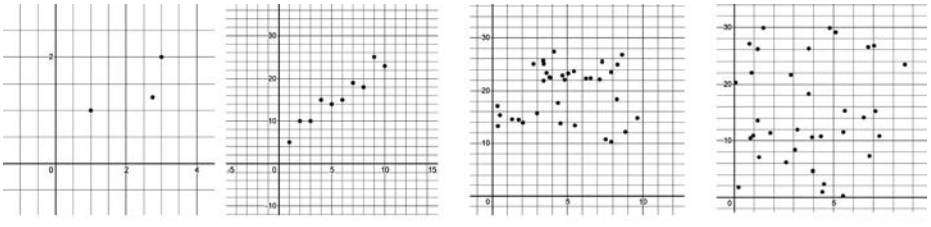
Давайте реализуем весь тест на Python, как показано в примере 5.17. Если тестовое значение находится за пределами критического интервала на доверительном уровне 95 %, мы признаем, что наша корреляция не была случайной.

Пример 5.17. Проверка значимости для данных с предполагаемой линейной зависимостью

```
from scipy.stats import t
from math import sqrt

# Размер выборки
n = 10

lower_cv = t(n-1).ppf(.025)
upper_cv = t(n-1).ppf(.975)

# Коэффициент корреляции
# на основании данных https://bit.ly/2KF29Bd
r = 0.9575860952087218

# Выполняем тест
test_value = r / sqrt((1-r**2) / (n-2))

print(f"Тестовое значение (t-статистика): {test_value}")
print(f"Критический интервал: {lower_cv}, {upper_cv}")

if test_value < lower_cv or test_value > upper_cv:
    print("Корреляция обоснована, отвергаем H₀")
else:
    print("Корреляция не обоснована, нельзя отвергнуть H₀")

# Вычисляем p-значение
if test_value > 0:
    p_value = 1.0 - t(n-1).cdf(test_value)
else:
    p_value = t(n-1).cdf(test_value)

# Двусторонний тест, поэтому умножаем на 2
p_value = p_value * 2
print(f"p-значение: {p_value}")
```

Тестовое значение составляет примерно 9,39958, что определенно находится за пределами диапазона (−2,262, 2,262), поэтому можно отвергнуть нулевую гипотезу и признать, что наша корреляция не случайна. Это связано с тем, что *p*-значение весьма значимо: 0,000005976. Это намного ниже нашего порога в 0,05, так что мы имеем дело не с совпадением: корреляция обоснована. Вполне логично, что *p*-значение так мало, потому что расположение точек очень похоже на прямую. Крайне маловероятно, что они выстроились в ряд случайно.

На рис. 5.15 показаны другие наборы данных с соответствующими коэффициентами корреляции и *p*-значениями. Изучите каждый из них. Как вам кажется, какой набор наиболее полезен для прогнозирования? В чем проблемы с другими наборами?



Корреляция	0,771454	Корреляция	0,957586095	Корреляция	0,100984708	Корреляция	0,037875905
p-значение	0,34913	*p*-значение	0,000005976	*p*-значение	0,569732057	*p*-значение	0,831544971

Рис. 5.15. Различные наборы данных с соответствующими коэффициентами корреляции и *p*-значениями

После того как вы изучили наборы данных на рис. 5.15, давайте посмотрим, что удалось выяснить. Самая левая диаграмма демонстрирует высокую положительную корреляцию, но на ней всего три точки. Нехватка данных приводит к тому, что *p*-значение возрастает до 0,34913 и увеличивается вероятность того, что данные образовались случайно. Это логично, потому что если у нас есть всего три точки, в них нетрудно усмотреть линейную модель, но это не намного лучше, чем две точки, через которые можно просто провести прямую. Отсюда вытекает важное правило: чем больше у вас данных, тем меньше *p*-значение, особенно если эти данные тяготеют к прямой линии.

Вторая диаграмма — это набор данных, который мы только что рассмотрели. В нем всего 10 точек, но они образуют линейную форму настолько отчетливо, что наблюдается не только сильная положительная корреляция, но и чрезвычайно низкое *p*-значение. Когда *p*-значение так мало, можно биться об заклад, что вы исследуете спланированный и хорошо контролируемый процесс, а не социологическое или природное явление.

На двух правых диаграммах на рис. 5.15 не удается обнаружить линейную зависимость. Их коэффициент корреляции близок к 0, что указывает на отсутствие взаимосвязи, а *p*-значения убедительно свидетельствуют о том, что данные носят случайный характер.

Общее правило таково: чем больше у вас данных, расположение которых стабильно напоминает прямую, тем более статистически значимым будет *p*-значение для корреляции. Чем меньше данных или чем шире они рассеяны — тем выше будет *p*-значение, и придется заключить, что корреляция возникла по случайному стечению обстоятельств[1].

Коэффициент детерминации

Давайте познакомимся с важным понятием, которое часто встречается в статистике, а также в регрессиях в машинном обучении. *Коэффициент детерминации*, который обозначается r^2, измеряет, какая доля дисперсии одной переменной объясняется дисперсией другой переменной. Он также является квадратом коэффициента корреляции r. Чем ближе r к идеальной корреляции (−1 или 1), тем ближе r^2 к 1. По сути, r^2 показывает, насколько сильно две переменные связаны друг с другом.

Давайте продолжим изучать данные на рис. 5.13. В примере 5.18 мы возьмем код из примера 5.14, который вычислял коэффициенты корреляции, и просто возведем их в квадрат.

Пример 5.18. Матрица коэффициентов детерминации, построенная с помощью pandas

```
import pandas as pd

# Загружаем данные в датафрейм pandas
df = pd.read_csv('https://bit.ly/2KF29Bd', delimiter=",")

# Выводим коэффициенты детерминации между переменными
coeff_determination = df.corr(method='pearson') ** 2
print(coeff_determination)

# ВЫВОД:
#          x         y
# x 1.000000 0.916971
# y 0.916971 1.000000
```

[1] В реальных задачах зависимость между переменными может быть нелинейной, когда одна переменная тесно связана с другой, но их диаграмма рассеяния похожа не на прямую, а, например, на параболу, экспоненту, синусоиду или другую математическую кривую. В этом случае, если не преобразовывать данные, линейная регрессия не поможет выявить настоящую зависимость, линейный коэффициент корреляции может быть близок к 0, а соответствующее *p*-значение может оказаться высоким. Если вас интересует взаимосвязь между двумя переменными, первым делом смотрите на диаграмму рассеяния, которая подскажет наличие и характер взаимосвязи. Чтобы изучать нелинейную связь, можно преобразовывать одну или обе переменных и применять другие статистические методы, которые не рассматриваются в этой главе. — *Примеч. науч. ред.*

Коэффициент детерминации 0,916971 интерпретируется так: 91,6971 % дисперсии переменной *x* объясняется переменной *y* (и наоборот), а оставшиеся 8,3029 % — это шум, который вызван другими неучтенными переменными. 0,916971 — это довольно хороший коэффициент детерминации, который показывает, что *x* и *y* объясняют дисперсию друг друга. Но в системе могут участвовать и другие переменные, которые отвечают за оставшиеся 0,083029. Помните, что корреляция не равносильна причинно-следственной связи, и другие переменные тоже могут вносить свой вклад в наблюдаемую взаимосвязь.

Корреляция — это не причинно-следственная связь!

Важно отметить: хотя мы уделяем много внимания тому, чтобы вычислять корреляцию и строить метрики на ее основе, помните, что *корреляция — это не причинно-следственная связь*! Вероятно, вы уже слышали эту мантру, но я хочу объяснить, почему она так популярна среди специалистов по статистике.

Если мы наблюдаем корреляцию между *x* и *y* — это само по себе не повод судить, что *x* является причиной *y*. На самом деле может быть, что *y* является причиной *x*! А может, существует третья неучтенная переменная *z*, которая вызывает и *x*, и *y*. Может оказаться, что *x* и *y* вообще не обусловливают друг друга, а корреляция получилась случайно. Именно поэтому так важно измерять статистическую значимость.

А теперь у меня более насущный вопрос: умеют ли компьютеры отличать корреляцию от причинно-следственной связи? Ответ — нет, нет и еще раз нет! Компьютеры имеют представление о корреляции, но не о причинах и следствиях. Представим себе, что я загружаю в scikit-learn набор данных об объеме потребленной воды и о моих начислениях за водоснабжение. Компьютер или любая программа, включая scikit-learn, понятия не имеет о том, увеличивается ли счет, если интенсивнее потреблять воду, или потребление воды растет из-за того, что растут коммунальные платежи. Система искусственного интеллекта вполне может сделать вывод о том, что имеет место второй вариант, как бы нелепо это ни звучало. Именно поэтому многие проекты машинного обучения не обходятся без участия человека, который привносит в них здравый смысл.

С системами компьютерного зрения тоже случаются казусы. Чтобы предсказать категорию объекта, такие системы часто используют регрессию на основе числовых значений пикселов. Если я буду обучать систему компьютерного зрения распознавать коров, используя фотографии пасущихся коров на лугу, она вполне может скоррелировать целевые объекты с лугом, а не с коровами. И в итоге, если я покажу системе фотографию пустого луга, она пометит траву как коров! Это снова происходит потому, что компьютеры не ориентируются на причинно-следственную связь (форма коровы должна вызывать метку «корова»), а запутываются в корреляциях, которые нас не интересуют.

Стандартная ошибка оценки

Один из способов измерить общую ошибку линейной регрессии — это *SSE*, или *сумма квадратов ошибок*. Мы уже встречали эту величину, когда возводили в квадрат каждый остаток и суммировали их. Если $\hat{y}$ — это каждое значение, которое прогнозирует прямая регрессии, а y — каждое фактическое значение y из набора данных, то SSE вычисляется по такой формуле:

$$SSE = \sum\left(y - \hat{y}\right)^2$$

Однако все эти квадратичные значения трудно интерпретировать, поэтому лучше извлечь квадратный корень, чтобы вернуться к исходным единицам измерения. Кроме того, стоит усреднить все значения, и именно для этого служит *стандартная ошибка оценки* (S_e). Пример 5.19 показывает, как рассчитать стандартную ошибку S_e на Python, если n — это количество точек данных.

$$S_e = \sqrt{\frac{\sum\left(y - \hat{y}\right)^2}{n-2}}$$

Пример 5.19. Вычисление стандартной ошибки оценки

```
import pandas as pd
from math import sqrt

# Загружаем данные
points = list(pd.read_csv("https://bit.ly/2KF29Bd", delimiter=",").itertuples())

n = len(points)

# Прямая регрессии
m = 1.939
b = 4.733

# Вычисляем стандартную ошибку оценки
S_e = sqrt((sum((p.y - (m*p.x +b))**2 for p in points))/(n-2))

print(S_e) # 1.87406793500129
```

Почему здесь $n-2$, а не $n-1$, как это было в расчетах дисперсии в главе 3? Если не углубляться в математические доказательства, то это связано с тем, что в линейной регрессии две переменные, а не одна, поэтому нужно увеличить неопределенность еще на одну степень свободы.

Обратите внимание, что стандартная ошибка оценки удивительно похожа на стандартное отклонение, которое мы изучали в главе 3. Это не случайно: S_e — это и есть стандартное отклонение для линейной регрессии.

Интервалы прогнозирования

Как уже говорилось, данные, на основе которых строится линейная регрессия, — это выборка из генеральной совокупности. Поэтому регрессия хороша лишь настолько, насколько хороша выборка. Кроме того, для каждого значения *x* прогнозируемые значения *y* генеральной совокупности имеют нормальное распределение. Таким образом, каждое прогнозируемое значение *y* можно считать выборочной статистикой, подобно среднему. Фактически «среднее» перемещается вдоль прямой.

Помните, как в главе 2 мы говорили о дисперсии и стандартном отклонении в статистике? Эти понятия применимы и здесь. При линейной регрессии мы надеемся на то, что каждому *x* соответствует нормальное распределение *y*. В этом случае прямая регрессии служит смещающимся «средним» колоколообразной кривой, а разброс данных вокруг прямой отражает дисперсию и стандартное отклонение, как показано на рис. 5.16.

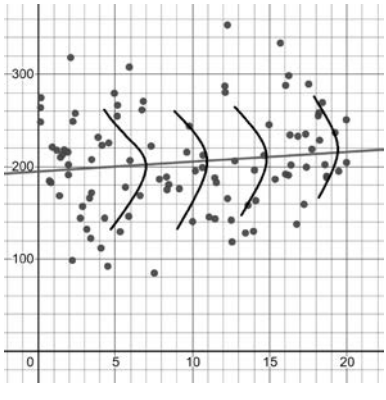


Рис. 5.16. Линейная регрессия предполагает, что нормальное распределение перемещается вдоль прямой

Если вдоль прямой регрессии перемещается нормальное распределение, это значит, что у нас есть не только одна переменная, но и вторая, которая тоже управляет распределением. Вокруг каждого прогнозируемого значения *y* существует доверительный интервал, который называется *интервалом прогнозирования*.

Давайте вернемся к примеру с ветеринарами, где оценивается возраст собаки и количество ее приемов у врача. Я хочу узнать интервал прогнозирования для количества приемов ветеринара на доверительном уровне 95 % для собаки в возрасте 8,5 года. На рис. 5.17 показано, как выглядит этот интервал. Мы на 95 % уверены, что собаку в возрасте 8,5 лет водили к ветеринару от 16,462 до 25,966 раза.

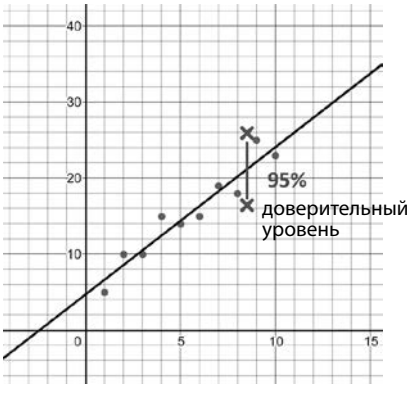


Рис. 5.17. Интервал прогнозирования для собаки в возрасте 8,5 года на доверительном уровне 95 %

Как получить этот результат? Нужно вычислить погрешность и отложить ее в обе стороны от предсказанного значения *y*. Погрешность вычисляется по жутковатой формуле, в которую входит критическое значение из распределения Стьюдента, а также стандартная ошибка оценки:

$$E = t_{0,025} \times S_e \times \sqrt{1 + \frac{1}{n} + \frac{n\left(x_0 + \overline{x}\right)^2}{n\left(\sum x^2\right) - \left(\sum x\right)^2}}$$

Здесь интересующее нас значение *x* обозначено x_0 и в данном случае равно 8,5. В примере 5.20 показано, как можно вычислить интервал прогнозирования на Python:

Пример 5.20. Интервал прогнозирования количества приемов у ветеринара для собаки в возрасте 8,5 года

```
import pandas as pd
from scipy.stats import t
from math import sqrt

# Загружаем данные
points = list(pd.read_csv("https://bit.ly/2KF29Bd", delimiter=",").itertuples())

n = len(points)

# Прямая линейной регрессии
m = 1.939
b = 4.733
```

```
# Вычисляем интервал прогнозирования для x = 8,5
x_0 = 8.5
x_mean = sum(p.x for p in points) / len(points)

t_value = t(n - 2).ppf(.975)

standard_error = sqrt(sum((p.y - (m * p.x + b)) ** 2 for p in points) / (n - 2))

margin_of_error = t_value * standard_error * \
                    sqrt(1 + (1 / n) + (n * (x_0 - x_mean) ** 2) / \
                    (n * sum(p.x ** 2 for p in points) - \
                    sum(p.x for p in points) ** 2))
predicted_y = m*x_0 + b

# Выводим интервал прогнозирования
print(predicted_y - margin_of_error, predicted_y + margin_of_error)
# 16.46251687560351 25.966483124396493
```

Ой, мама! Получилось очень много вычислений, и, к сожалению, SciPy и другие основные библиотеки для data science не выполняют их за нас. Но если вы неравнодушны к статистическому анализу, это очень полезная информация. Мы не только делаем прогноз на основе линейной регрессии (например: собака в возрасте 8,5 года побывает у ветеринара 21,2145 раза), но и можем заключить нечто гораздо менее очевидное: с вероятностью 95 % собака в возрасте 8,5 года побывает у ветеринара от 16,46 до 25,97 раза. Согласитесь, это здóрово? И это гораздо более надежное утверждение, потому что оно охватывает интервал, а не одно значение, и таким образом учитывает неопределенность.

ДОВЕРИТЕЛЬНЫЕ ИНТЕРВАЛЫ ДЛЯ ПАРАМЕТРОВ

Если задуматься, сама по себе прямая линейной регрессии — это выборочная статистика, и существует своя прямая линейной регрессии для всей генеральной совокупности, о которой мы пытаемся сделать вывод. Это значит, что у таких параметров, как m и b, есть свои собственные распределения, и можно построить доверительные интервалы для m и b по отдельности, чтобы охарактеризовать угловой коэффициент и пересечение с осью Y для совокупности. Эта тема выходит за рамки этой книги, но стоит отметить, что можно провести и такой анализ.

Нетрудно найти пособия о том, как выполнять эти расчеты с нуля, но, возможно, проще будет воспользоваться инструментами регрессии Excel или соответствующими библиотеками для Python.

Обучающая и тестовая выборки

К сожалению, практикующие специалисты по data science часто пренебрегают анализом, который я только что провел, когда вычислил коэффициент корреляции, статистическую значимость и коэффициент детерминации. Иногда они сталкиваются с таким объемом данных, что на анализ у них не хватает времени или технических возможностей. Например, изображение размером 128 × 128 пикселей — это не менее 16 384 переменных. Найдется ли у вас время, чтобы провести статистический анализ каждой из этих пиксельных переменных? Скорее всего, нет! Как ни печально, но это приводит к тому, что многие специалисты по data science вообще не изучают эти статистические показатели.

На одном малоизвестном форуме (http://disq.us/p/1jas3zg) я как-то увидел пост о том, что статистическая регрессия — это скальпель, а машинное обучение — бензопила. Когда вы имеете дело с огромным массивом данных и переменных, вы не сможете обработать все это скальпелем. Приходится прибегать к бензопиле, и хотя при этом теряется объяснимость и точность, решение, по крайней мере, можно масштабировать, чтобы делать более широкие прогнозы на большем количестве данных. При этом такие статистические проблемы, как смещение выборки и переобучение, никуда не деваются. Но есть несколько методов, с помощью которых можно быстро проверять решение.

Почему в scikit-learn нет доверительных интервалов и *p*-значений?

Библиотека scikit-learn не поддерживает доверительные интервалы и *p*-значения, потому что с этими двумя конструкциями связано много нерешенных проблем в случае многомерных данных. Это обстоятельство только подчеркивает разрыв между специалистами по статистике и по машинному обучению. Как сказал один из разработчиков scikit-learn Гейл Вароку (Gaël Varoquaux), «в общем случае, чтобы вычислялись корректные *p*-значения, данные должны удовлетворять допущениям, которые не свойственны реальным данным в машинном обучении (чтобы не было мультиколлинеарности, чтобы данных было достаточно по сравнению с размерностью)... *p*-значения — это такая метрика, которая должна быть хорошо проверена (на них опираются выводы в медицинских исследованиях). Внедрять их — значит навлекать на себя неприятности... Мы можем обеспечить *p*-значения только в очень узкой области (с небольшим количеством переменных)».

Если вы хотите углубиться в эту тему, обратите внимание на интересные обсуждения на GitHub:

https://github.com/scikit-learn/scikit-learn/issues/6773;
https://github.com/scikit-learn/scikit-learn/issues/16802.

Как уже упоминалось, библиотека statsmodel (https://oreil.ly/8oEHo) предоставляет полезные инструменты для статистического анализа. Однако имейте в виду, что по вышеупомянутым причинам ее решения, скорее всего, не будут масштабироваться на модели большей размерности.

Основной метод, с помощью которого специалисты по машинному обучению борются с переобучением, заключается в том, чтобы разделять набор данных на *обучающую и тестовую выборки*. Обычно при этом $\frac{1}{3}$ данных включается в тестовую выборку, а остальные $\frac{2}{3}$ — в обучающую, хотя бывают и другие пропорции. *Обучающая выборка* используется для того, чтобы подогнать линейную регрессию, а *тестовая* — чтобы оценить эффективность регрессии на данных, с которыми она раньше не сталкивалась. Этот прием обычно используется во всех видах машинного обучения с учителем, включая логистическую регрессию и нейронные сети. На рис. 5.18 наглядно показано, как можно выделить $\frac{2}{3}$ данных в обучающую выборку и $\frac{1}{3}$ — в тестовую.

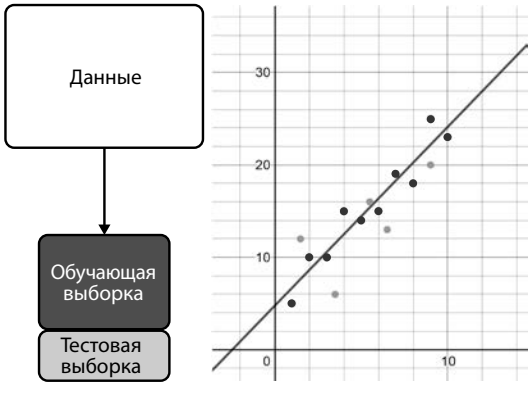


Рис. 5.18. Разделение данных на обучающую и тестовую выборки. Прямая регрессии подгоняется под обучающие данные (обозначены темным цветом) по методу наименьших квадратов, а затем проверяется на тестовых данных (обозначены светлым цветом), чтобы понять, насколько ошибочны предсказания на данных, которые не встречались ранее

Небольшой набор данных

Как мы узнаем позже, набор данных не обязательно разделять на обучающую и тестовую выборки в пропорции $\frac{2}{3}:\frac{1}{3}$. Если у вас такой небольшой набор данных, как в этом примере, то, может быть, лучше подойдет соотношение $\frac{9}{10}:\frac{1}{10}$ в сочетании с перекрестной валидацией или даже поэлементной валидацией. Подробнее см. во врезке «Обязательно ли разделять набор данных на трети?» на стр. 206.

В примере 5.21 показано, как с помощью scikit-learn разделить набор данных на обучающую и тестовую выборки в пропорции $\frac{2}{3}:\frac{1}{3}$.

Пример 5.21. Разделение набора данных на обучающую и тестовую выборки для линейной регрессии

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Загружаем данные
df = pd.read_csv("https://bit.ly/3cIH97A", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1]

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]

# Разделяем данные на обучающую и тестовую выборки;
# на тестовую выборку приходится ¹/₃ данных
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3)

model = LinearRegression()
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
print("r² = %.3f" % result)
```

Обучение — это подгонка регрессии

Обратите внимание, что подгонять регрессию — это то же самое, что обучать ее. Специалисты по машинному обучению предпочитают второй термин.

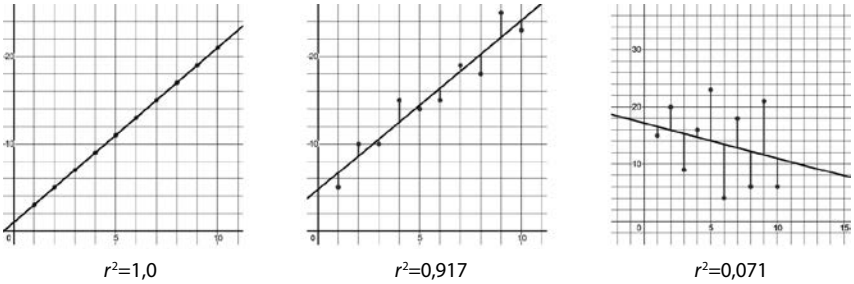
В этом коде функция `train_test_split()` берет набор данных (столбцы X и Y), перемешивает его, а затем возвращает обучающую и тестовую выборки в соответствии с заданной пропорцией (аргумент `test_size=1/3` задает размер тестовой выборки). Метод `fit()` класса `LinearRegression` подгоняет регрессию под обучающие выборки `X_train` и `Y_train`. Затем мы применяем метод `score()` к тестовым выборкам `X_test` и `Y_test`, чтобы оценить коэффициент детерминации r^2: он даст представление о том, хорошо ли регрессия работает на данных, с которыми она раньше не сталкивалась. Чем больше r^2 для тестовой выборки, тем лучше. Высокое значение, которое мы получили, свидетельствует о том, что регрессия эффективно работает на новых данных.

r^2 ДЛЯ ТЕСТОВОЙ ВЫБОРКИ

Обратите внимание, что r^2 здесь рассчитывается немного по-другому, потому что у нас есть уже обученная линейная регрессия. Чтобы вычислить r^2, мы сравниваем прогнозы на тестовой выборке с прямой регрессии, которая подогнана на обучающей выборке. Смысл остается прежним: чем ближе r^2 к 1, тем сильнее оказывается регрессионная корреляция даже на тестовой выборке, а если r^2 близок к 0, это значит, что регрессия плохо работает на тестовых данных. Этот показатель вычисляется по следующей формуле, где y_i — каждое фактическое значение y, $\widehat{y_i}$ — каждое предсказанное значение y, а $\overline{y}$ — среднее значение y для всех точек данных:

$$r^2 = 1 - \frac{\Sigma\left(y_i - \widehat{y_i}\right)^2}{\Sigma\left(y_i - \overline{y}\right)^2}$$

На рис. 5.19 показаны различные значения r^2 для нескольких линейных регрессий.



r^2=1,0
Идеально

r^2=0,917
Хорошо

r^2=0,071
Плохо

Рис. 5.19. r^2 для обученной линейной регрессии с различными тестовыми выборками

Можно также обучать и тестировать модель трижды, каждый раз назначая тестовой выборкой новую треть набора данных, а обучающей — оставшиеся две трети. Это называется *перекрестной валидацией* и зачастую считается эталонным методом валидации. На рис. 5.20 показано, как каждая треть набора данных попеременно служит тестовой выборкой.

Код в примере 5.22 демонстрирует трехкратную перекрестную валидацию. В конце выводятся метрики качества каждой из трех моделей (в данном случае это среднеквадратичные ошибки, или MSE), а также средняя MSE по всем моделям и ее стандартное отклонение, которое позволяет оценить, насколько единообразно модель ведет себя на разных выборках.

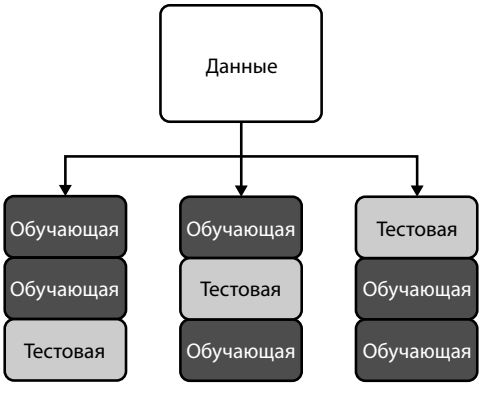


Рис. 5.20. Трехкратная перекрестная валидация

Пример 5.22. Трехкратная перекрестная валидация для линейной регрессии

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold, cross_val_score

df = pd.read_csv("https://bit.ly/3cIH97A", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1]

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]

# Простая линейная регрессия
kfold = KFold(n_splits=3, random_state=7, shuffle=True)
model = LinearRegression()
results = cross_val_score(model, X, Y, cv=kfold)
print(results)
print("MSE: mean = %.3f (stdev = %.3f)" % (results.mean(), results.std()))
```

ОБЯЗАТЕЛЬНО ЛИ РАЗДЕЛЯТЬ НАБОР ДАННЫХ НА ТРЕТИ?

Данные не обязательно разбивать на трети. Можно выполнять *k-кратную перекрестную валидацию*, когда в тестовую выборку входят $\frac{1}{k}$ данных. Обычно тестовая выборка составляет $\frac{1}{3}$, $\frac{1}{5}$ или $\frac{1}{10}$ часть набора данных, но чаще всего — $\frac{1}{3}$.

Как правило, *k* выбирают таким образом, чтобы тестовая выборка была достаточно большой в контексте поставленной задачи. Также само количество чередующихся тестовых выборок должно быть достаточным для

того, чтобы достоверно оценить, как модель работает на данных, которые ранее не встречались. Для небольших наборов данных можно использовать значения *k*, равные 3, 5 или 10. При *поэлементной перекрестной валидации* (LOOCV) тестовой выборкой будет поочередно каждая отдельная точка данных, и это может быть полезно, когда весь набор данных мал.

Если вас беспокоит дисперсия в модели, можно не просто разделять набор данных на обучающую и тестовую выборку или выполнять перекрестную валидацию, а применить *случайную валидацию*, которая позволяет сколько угодно раз перемешивать данные, разделять их на две выборки и агрегировать результаты тестирования. В примере 5.23 выполняется 10 итераций, на каждой из которых $\frac{1}{3}$ набора данных случайным образом отбирается в тестовую выборку, а остальные $\frac{2}{3}$ становятся обучающей выборкой. Затем результаты 10 тестирований усредняются и вычисляется их стандартное отклонение, по которому можно судить о том, насколько стабильна модель.

Пример 5.23. Случайная валидация для линейной регрессии

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score, ShuffleSplit

df = pd.read_csv("https://bit.ly/38XwbeB", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1]

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]

# Простая линейная регрессия
kfold = ShuffleSplit(n_splits=10, test_size=.33, random_state=7)
model = LinearRegression()
results = cross_val_score(model, X, Y, cv=kfold)

print(results)
print("mean = %.3f (stdev = %.3f)" % (results.mean(), results.std()))
```

В чем недостаток такого подхода? Он очень затратен с вычислительной точки зрения, потому что регрессию приходится обучать многократно.

Поэтому, когда вы ограничены по времени или ваши данные слишком объемны для статистического анализа, просто один раз разделяйте набор данных на обучающую и тестовую выборки. Это позволит оценить, насколько хорошо линейная регрессия работает на данных, с которыми она раньше не встречалась.

Обучающая и тестовая выборки не гарантируют успеха

Важно отметить, что ваша модель не будет хорошо работать просто из-за того, что вы применяете передовые методы машинного обучения и разделяете данные на обучающую и тестовую выборки. Вполне может получиться так, что вы переадаптируете свою модель и с помощью *p*-хакинга добьетесь хороших результатов в тестах, а потом обнаружите, что с реальными данными она работает плохо. Вот почему иногда необходимо предусмотреть еще одну выборку, которая называется *валидационной*, особенно когда вы сравниваете разные модели или конфигурации. Таким образом, если вы модифицируете обучающую выборку, чтобы достичь лучшей производительности на тестовых данных, это не приведет к тому, что в обучающие данные проникнет лишняя информация. Рассматривайте валидационную выборку как последний рубеж проверки того, не привел ли *p*-хакинг к переобучению модели под тестовую выборку.

Но даже в этом случае весь ваш набор данных (включая обучающую, тестовую и валидационную выборки) мог быть изначально смещенным, и никакие разделения не помогут это исправить. Эндрю Ын (Andrew Ng) назвал это большой проблемой машинного обучения в рамках интервью для DeepLearning.AI и Stanford HAI (https://oreil.ly/x23SJ). Он привел пример, который показывает, почему машинное обучение не смогло заменить врачей-рентгенологов.

Многомерная линейная регрессия

Эта глава была практически полностью посвящена линейной регрессии с одной входной и одной выходной переменной. Однако понятия, которые мы здесь изучили, в значительной степени применимы и к многомерной линейной регрессии. Для нее по-прежнему можно использовать такие метрики, как r^2, стандартная ошибка и доверительные интервалы, но чем больше переменных, тем это сложнее осуществить. Пример 5.24 демонстрирует линейную регрессию с двумя входными и одной выходной переменной с использованием библиотеки scikit-learn.

Пример 5.24. Линейная регрессия с двумя входными переменными

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Загружаем данные
df = pd.read_csv("https://bit.ly/2X1HWH7", delimiter=",")
```

```
# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1]

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]

# Обучаем модель
fit = LinearRegression().fit(X, Y)

# Выводим параметры модели
print(f"Угловые коэффициенты = {fit.coef_}")
print(f"Пересечение = {fit.intercept_}")
print(f"z = {fit.intercept_} + {fit.coef_[0]}x + {fit.coef_[1]}y")
```

Однако вы рискуете оказаться в шатком положении, когда модель настолько переполнится переменными, что начнет терять объяснимость, и именно тогда велик соблазн переключиться в привычный режим машинного обучения и рассматривать модель как черный ящик. Надеюсь, вы убедились, что статистические соображения никуда не делись, и чем больше переменных вы добавляете, тем разреженнее становятся данные. Но если сделать шаг назад, проанализировать взаимосвязи в каждой паре переменных с помощью корреляционной матрицы и попытаться понять, как переменные в каждой паре взаимодействуют друг с другом, это поможет создать эффективную модель машинного обучения.

Заключение

В этой главе мы рассмотрели много вопросов. Материал не ограничивался поверхностным представлением о линейной регрессии, когда разделение набора данных на обучающую и тестовую выборку становится единственным способом валидации. Я постарался показать вам как скальпель (статистику), так и бензопилу (машинное обучение), чтобы вы могли оценить, что лучше поможет решить конкретную проблему, с которой вы столкнулись. Существует много метрик и методов анализа, которые доступны только в линейной регрессии, и мы рассмотрели некоторые из них, чтобы понять, насколько надежна линейная регрессия для прогнозов. Возможно, иногда вам придется выполнять регрессию для грубого приближения, а иногда тщательно анализировать и систематизировать данные с помощью статистических инструментов. Какой подход использовать — зависит от ситуации. Если вы хотите больше узнать о том, какие статистические инструменты доступны в Python, ознакомьтесь с библиотекой statsmodel (https://oreil.ly/8oEHo).

В главе 6, которая посвящена логистической регрессии, мы вновь обратимся к r^2 и статистической значимости. Надеюсь, текущая глава убедила вас в том, что данные стоит анализировать осмысленно, а вложенные усилия могут стать залогом успешного проекта.

Упражнения для самопроверки

Набор данных из двух переменных x и y доступен по ссылке (https://bit.ly/3C8JzrM).

1. Выполните простую линейную регрессию, чтобы найти значения m и b, которые минимизируют потери (сумму квадратов остатков).
2. Рассчитайте коэффициент корреляции и статистическую значимость этих данных (на доверительном уровне 95 %). Значима ли корреляция?
3. Каков 95 %-ный интервал прогнозирования для прогнозируемого значения y при $x = 50$?
4. Начните регрессию заново и разделите данные на обучающую и тестовую выборки. Не стесняйтесь экспериментировать с перекрестной и случайной валидацией. Насколько хорошо и стабильно работает линейная регрессия на тестовых данных? Почему так?

Ответы см. в Приложении Б.

ГЛАВА 6

Логистическая регрессия и классификация

В этой главе речь пойдет о *логистической регрессии*, которая прогнозирует вероятность того или иного исхода по одной или нескольким независимым переменным. В свою очередь, результаты логистической регрессии можно использовать для *классификации*, то есть для того, чтобы предсказывать категории, а не числовые значения, как в случае с линейной регрессией.

Переменные не всегда должны быть *непрерывными* величинами и принимать значения из бесконечного множества вещественных десятичных чисел. Бывают ситуации, когда лучше, чтобы переменные были *дискретными*, то есть принимали в качестве значений целые числа или логические значения (1 или 0, «истина» или «ложь»). Логистическая регрессия обучается на переменной, которая является бинарной (1 или 0) или категориальной (целое число). Результат регрессии — непрерывная переменная в виде вероятности, но ее можно преобразовать в дискретную величину с помощью порогового значения.

Логистическую регрессию просто реализовать, и она достаточно устойчива к выбросам и другим дефектам данных. Многие задачи машинного обучения лучше всего решать с помощью логистической регрессии, которая оказывается практичнее и эффективнее, чем другие методы обучения с учителем.

Как и в главе 5, где шла речь о линейной регрессии, мы попытаемся пройти по тонкой грани между статистикой и машинным обучением, задействуя инструменты и методы анализа из обеих дисциплин. Логистическая регрессия объединит многие понятия, которые мы рассматривали в этой книге, — от теории вероятностей до линейной регрессии.

Что такое логистическая регрессия

Представьте, что на заводе произошла небольшая авария, и вы пытаетесь оценить последствия воздействия химических веществ. У вас есть 11 пациентов, которые в течение разного времени подвергались воздействию вредного химиката. (Пожалуйста, обратите внимание, что это вымышленные данные.) У одних пациентов проявились симптомы химического отравления (значение 1), а у других — нет (значение 0).

Давайте изобразим эти данные на рис. 6.1, где значение по оси *X* показывает, в течение скольких часов пациент подвергался воздействию, а по оси *Y* — проявились ли у него симптомы (1 или 0).

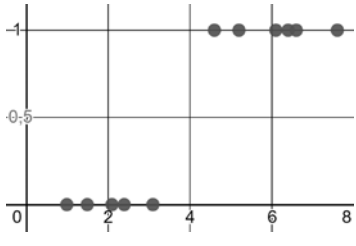


Рис. 6.1. Диаграмма наличия (1) или отсутствия (0) симптомов у пациентов в зависимости от продолжительности воздействия химиката

Как долго нужно подвергаться воздействию химических веществ, чтобы начали проявляться симптомы отравления? Легко заметить, что примерно на уровне четырех часов диаграмма резко переходит от пациентов без симптомов (0) к пациентам с симптомами (1). На рис. 6.2 представлены те же данные с прогностической кривой.

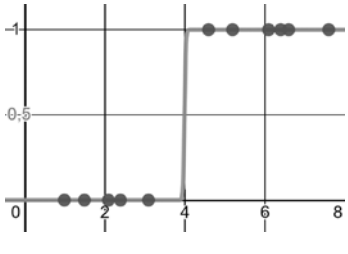


Рис. 6.2. Через четыре часа воздействия наблюдается отчетливый скачок, когда у пациентов начинают проявляться симптомы

Если бегло проанализировать эту выборку, можно сделать вывод: вероятность того, что симптомы проявятся у пациента, который подвергался воздействию

химиката менее четырех часов, составляет почти 0 %, но вероятность того, что они проявятся у пациента, который подвергался воздействию более четырех часов, равна практически 100 %. Между этими двумя группами наблюдается отчетливый скачок: симптомы начинают проявляться примерно через четыре часа.

Конечно, в реальном мире все не так однозначно. Допустим, вы собрали больше данных, и теперь в средней части интервала есть пациенты и с симптомами, и без, как показано на рис. 6.3.

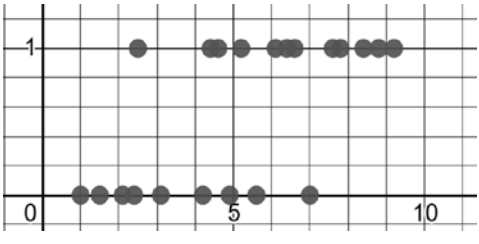


Рис. 6.3. В средней части диаграммы есть пациенты и с симптомами (1), и без симптомов (0)

Интерпретировать это можно так: чем дольше пациент подвергался воздействию химических веществ, тем выше вероятность того, что у него проявятся симптомы отравления. Представим это с помощью *логистической функции* — *S*-образной кривой, где выходная переменная находится в диапазоне от 0 до 1, как показано на рис. 6.4.

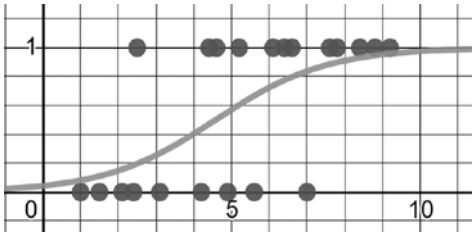


Рис. 6.4. Подгонка логистической функции к данным

Из-за того, что две группы точек перекрываются, теперь нет четкой временно́й границы, после которой у пациента проявляются симптомы, а наблюдается постепенный переход от вероятности 0 % к вероятности 100 % (то есть от 0 к 1). Этот пример демонстрирует, как *логистическая регрессия* порождает кривую, которая показывает вероятность того, что произошло некоторое событие (у пациента проявились симптомы химического отравления), в зависимости от значения независимой переменной (продолжительность воздействия).

Логистическую регрессию можно модифицировать, чтобы она не просто прогнозировала вероятность для тех или иных значений входных переменных, но и предсказывала, к какой из двух категорий относится пациент, — для этого нужно задать пороговое значение. Например, если я исследую нового пациента и узнаю́, что он подвергался опасному воздействию в течение 6 часов, я прогнозирую вероятность 71,1 % того, что у него проявятся симптомы (см. рис. 6.5). Если пороговая вероятность проявления симптомов составляет не менее 50 %, я прогнозирую, что они должны проявиться.

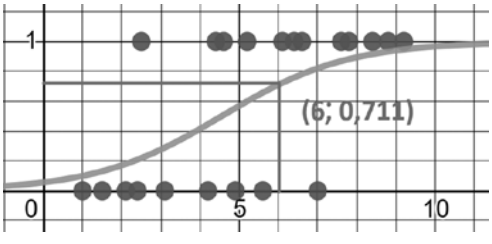


Рис. 6.5. Можно предположить, что у пациента, который подвергался воздействию химических веществ в течение 6 часов, симптомы отравления проявятся с вероятностью 71,1 %. Поскольку эта величина превышает пороговое значение в 50 %, мы прогнозируем, что симптомы должны проявиться

Как выполнять логистическую регрессию

Как же выполнить логистическую регрессию? Для начала рассмотрим логистическую функцию и изучим математический аппарат, на который она опирается.

Логистическая функция

Логистическая функция — это *S*-образная кривая (также известная как *сигмоида*), которая в зависимости от значений входных переменных принимает значения между 0 и 1. Поскольку выходная переменная лежит в интервале от 0 до 1, с ее помощью можно представлять вероятность.

Вот логистическая функция, которая выдает вероятность y для одного аргумента x:

$$y = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$

Обратите внимание, что в этой формуле фигурирует число e, которое мы рассматривали в главе 1. Переменная x — это независимая переменная, а β_0 и β_1 — коэффициенты регрессии, которые нужно найти.

Коэффициенты β_0 и β_1 находятся в показателе степени, который напоминает линейную функцию (как вы помните, она выглядит идентично: $y = mx + b$ или $y = \beta_0 + \beta_1 x$). Это не совпадение: логистическая регрессия на самом деле тесно связана с линейной регрессией, и мы поговорим об этом позже в этой главе. β_0 — это ордината пересечения вышеупомянутой линейной функции с осью Y (коэффициент b в простой линейной регрессии), а β_1 — ее угловой коэффициент (то же, что m в простой линейной регрессии). Эта линейная функция в показателе степени известна как логит-функция (log-odds или logit), хотя пока вам достаточно просто знать, что вся логистическая функция порождает эту S-образную кривую, которая отражает, как изменяется вероятность при движении вдоль оси X.

Объявить логистическую функцию на Python можно с помощью функции `exp()` из библиотеки `math`, которая задает степень числа e, как показано в примере 6.1.

Пример 6.1. Логистическая функция одного аргумента на Python

```
import math

def predict_probability(x, b0, b1):
    p = 1.0 / (1.0 + math.exp(-(b0 + b1 * x)))
    return p
```

Посмотрим, как выглядит график этой функции в случае, когда $\beta_0 = -2,823$ и $\beta_1 = 0,62$. В примере 6.2 показано, как построить этот график с помощью SymPy, а сам график изображен на рис. 6.6.

Пример 6.2. Построение графика логистической функции с помощью SymPy

```
from sympy import *
b0, b1, x = symbols('b0 b1 x')

p = 1.0 / (1.0 + exp(-(b0 + b1 * x)))

p = p.subs(b0,-2.823)
p = p.subs(b1, 0.620)
print(p)

plot(p)
```

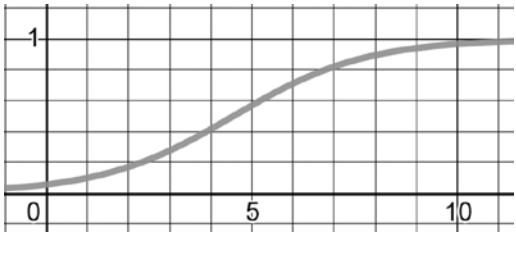


Рис. 6.6. График логистической функции

В некоторых учебниках логистическую функцию можно встретить в другой форме:

$$p = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$$

Не волнуйтесь: это та же самая функция, просто она алгебраически выражена иначе. Обратите внимание, что логистическую регрессию, как и линейную, можно распространить на две и более входных переменных ($x_1, x_2, ..., x_n$), как показано в следующей формуле. Для этого просто надо добавить дополнительные коэффициенты β_i:

$$p = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n)}}$$

Подгонка логистической кривой

Как подогнать логистическую кривую к заданной обучающей выборке? Начнем с того, что данные могут содержать любую комбинацию вещественных, целых и двоичных переменных, но выходная переменная должна быть только двоичной (0 или 1). Когда мы делаем прогнозы, значение логистической функции будет находиться между 0 и 1 и тем самым будет подобно вероятности.

Данные содержат значения входных и выходной переменных, но чтобы подогнать логистическую функцию, нам нужно найти коэффициенты β_0 и β_1. Возможно, вы вспомните, как в главе 5 мы использовали метод наименьших квадратов, однако здесь он не подходит. Вместо этого нужен *метод максимального правдоподобия*, который, как можно догадаться по названию, максимизирует правдоподобие того, что заданная логистическая кривая порождает наблюдаемые данные.

В отличие от линейной регрессии, не существует аналитического решения, с помощью которого вычисляются искомые коэффициенты. Однако можно применить градиентный спуск или поручить работу библиотеке. Давайте рассмотрим оба этих подхода, начав с библиотеки scikit-learn.

Использование библиотеки scikit-learn

Приятная особенность scikit-learn состоит в том, что модели машинного обучения часто поддерживают стандартизированный набор функций и API, так что во многих случаях можно копировать один и тот же код, чтобы повторно использовать его в разных моделях. В примере 6.3 вы видите логистическую регрессию, которая выполнена на данных о пациентах. Если сравнить этот пример с линейной регрессией из главы 5, то можно убедиться, что здесь используется практически тот же самый код, чтобы загружать и разделять данные, а также подгонять модель. Основное различие — в том, что для модели используется `LogisticRegression()`, а не `LinearRegression()`.

Пример 6.3. Простая логистическая регрессия с помощью scikit-learn

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

# Импортируем данные
df = pd.read_csv("https://bit.ly/33ebs2R", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1]

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]

# Выполняем логистическую регрессию
# Отключаем штраф (penalty)
model = LogisticRegression(penalty=None)
model.fit(X, Y)

# Коэффициент β₁
print(model.coef_.flatten()[0]) # 0.6926893863085584

# Коэффициент β₀
print(model.intercept_.flatten()[0]) # -3.175805042563378
```

ПРОГНОЗИРОВАНИЕ

Чтобы прогнозировать значения *y* для конкретных значений *x*, применяйте к объекту `model` методы `predict()` и `predict_proba()` из scikit-learn. Они работают как для `LogisticRegression`, так и для любых других моделей классификации. Функция `predict()` прогнозирует конкретный класс (например, 1 или 0), а функция `predict_proba()` выводит значение вероятности для каждого класса.

Если запустить модель в scikit-learn, получится логистическая регрессия, в которой $\beta_0 = -3,175805042563378$, а $\beta_1 = 0,6926893863085584$. График логистической функции показан на рис. 6.7.

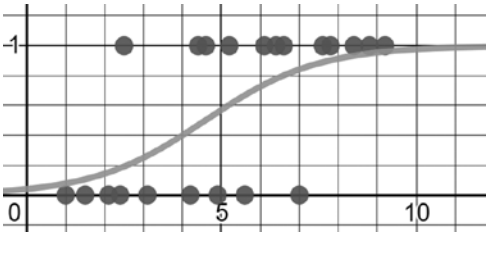


Рис. 6.7. График логистической функции

Здесь следует отметить несколько моментов. Когда я создавал модель `LogisticRegression()`, я аннулировал аргумент `penalty`, с помощью которого задается метод регуляризации, например `l1` или `l2`. Хотя эта тема выходит за рамки книги, я включил краткие сведения в следующую врезку «Параметры библиотеки scikit-learn», чтобы у вас под рукой были полезные ссылки.

Наконец, я применил метод `flatten()` к угловому коэффициенту и пересечению с осью *Y*: каждый из этих коэффициентов представлен как многомерная матрица, но содержит один элемент. Метод `flatten()` позволяет редуцировать матрицу в меньшую размерность, особенно когда количество элементов меньше, чем размерность. Например, здесь этот метод превращает двумерную матрицу в одномерный массив из одного числа, а индекс `[0]` извлекает это число в виде отдельного значения. После этого у нас появляются коэффициенты β_0 и β_1.

Параметры библиотеки scikit-learn

Библиотека scikit-learn предлагает множество настроек для моделей регрессии и классификации. К сожалению, в этой книге не хватит места, чтобы их рассмотреть, потому что книга не посвящена исключительно машинному обучению.

Однако scikit-learn сопровождается хорошей и подробной документацией, а сведения о логистической регрессии можно найти по адресу https://oreil.ly/eL8hZ.

Если многие термины вам незнакомы — например, регуляризация и штрафы l1 и l2, — вам помогут другие замечательные книги издательства O'Reilly. Один из наиболее полезных учебников, которые мне удалось найти, — «Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow»[1] Орельена Жерона (Aurelien Geron).

Методы максимального правдоподобия и градиентного спуска

На протяжении всей книги я стремлюсь дать представление о том, как применять те или иные методы с нуля, даже если библиотеки могут сделать это за нас. Подогнать логистическую регрессию можно несколькими способами, но все они обычно сводятся к оценке максимального правдоподобия (MLE). Этот метод максимизирует правдоподобие того, что заданная логистическая кривая порождает наблюдаемые данные. Это не то же самое, что метод наименьших квадратов, но здесь все равно можно применить градиентный спуск (в том числе стохастический).

[1] О. Жерон. «Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем». 2-е изд.

Я постараюсь упростить математический жаргон и свести к минимуму линейную алгебру. По сути, идея заключается в том, чтобы найти коэффициенты β_0 и β_1, которые максимально приближают логистическую кривую к этим точкам. Если вы помните из главы 2, чтобы найти вероятность пересечения нескольких событий, нужно перемножить их вероятности, и совместное правдоподобие вычисляется так же. В данном случае нужно вычислить правдоподобие того, что при данной кривой логистической регрессии будут наблюдаться все указанные точки.

Если применить принцип совместной вероятности, можно заметить, что для каждого пациента удается оценить правдоподобие того, что у него проявятся симптомы, *на основе подогнанной логистической функции*, как показано на рис. 6.8.

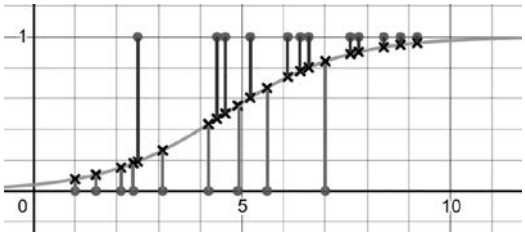


Рис. 6.8. Каждое входное значение соответствует правдоподобию на логистической кривой

Для каждой точки данных мы рассматриваем правдоподобие того, что она находится не на логистической кривой (то есть выше или ниже нее). Если точка находится ниже кривой, нужно вычесть соответствующую вероятность из 1, потому что мы хотим максимизировать правдоподобие и для тех точек, которые соответствуют пациентам без симптомов.

С учетом того, что $\beta_0 = -3{,}175805042563378$, а $\beta_1 = 0{,}6926893863085584$, в примере 6.4 показано, как вычислить совместное правдоподобие для этих данных на Python.

Пример 6.4. Совместное правдоподобие того, что для данной логистической регрессии будут наблюдаться все данные точки

```
import math
import pandas as pd

patient_data = pd.read_csv("https://bit.ly/33ebs2R", delimiter=",").itertuples()

b0 = -3.175805042563378
b1 = 0.6926893863085584

def logistic_function(x):
    p = 1.0 / (1.0 + math.exp(-(b0 + b1 * x)))
    return p
```

```
# Вычисляем совместное правдоподобие
joint_likelihood = 1.0

for p in patient_data:
    if p.y == 1.0:
        joint_likelihood *= logistic_function(p.x)
    elif p.y == 0.0:
        joint_likelihood *= (1.0 - logistic_function(p.x))

print(joint_likelihood) # 4.7911179961362736e-05
```

Чтобы упростить выражение внутри блока `if`, можно прибегнуть к математической хитрости. Как упоминалось в главе 1, любое число в степени 0 всегда равно 1. Посмотрите на эту формулу и обратите внимание на то, как ведут себя показатели степеней в случае, когда $y_i = 1$ (проявились симптомы) и когда $y_i = 0$ (нет симптомов):

$$\text{совместное правдоподобие} = \prod_{i=1}^{n} \left(\frac{1}{1+e^{-(\beta_0+\beta_1 x_i)}} \right)^{y_i} \times \left(1 - \frac{1}{1+e^{-(\beta_0+\beta_1 x_i)}} \right)^{1-y_i}$$

Эту формулу можно уместить в тело цикла `for`, как показано в примере 6.5.

Пример 6.5. Вычисление совместного правдоподобия без инструкции `if`

```
for p in patient_data:
    joint_likelihood *= logistic_function(p.x) ** p.y * \
                        (1.0 - logistic_function(p.x)) ** (1.0 - p.y)
```

Что здесь к чему? Обратите внимание, что это выражение состоит из двух частей: одна для случая, когда $y = 1$, а другая — когда $y = 0$. Если возвести любое число в степень 0, получится 1. Поэтому независимо от того, чему равно значение y (1 или 0), одна из частей выражения обратится в 1 и не будет оказывать никакого эффекта при умножении. Таким образом мы выразили инструкцию `if`, но сделали это полностью в математической форме. Это будет полезно в дальнейшем, потому что нельзя брать производные от выражений, в которых используется `if`.

Обратите внимание, что компьютеры могут не справиться с вычислениями, когда перемножают много маленьких десятичных дробей, — это называется *исчезновением порядка*. Это значит, что по мере того как десятичные числа при умножении становятся все меньше, компьютер рано или поздно теряет возможность отслеживать так много десятичных знаков. В результате микроскопическую, но все-таки ненулевую дробь компьютер ошибочно представляет как 0.

Чтобы обойти эту проблему, есть очередная математическая хитрость. Можно взять логарифм от каждой десятичной дроби, которая входит в произведение, и вместо того чтобы перемножать дроби, сложить их логарифмы. Это возможно благодаря свойствам сложения логарифмов, о которых шла речь в главе 1.

Сложение более вычислительно устойчиво, а полученную сумму можно преобразовать обратно в произведение правдоподобий: для этого надо возвести *e* в степень, равную этой сумме.

Давайте переделаем наш код так, чтобы не умножать дроби, а складывать их логарифмы (см. пример 6.6). Обратите внимание, что функция `log()` по умолчанию берет логарифм по основанию *e*. Хотя формально подойдет и любое другое основание, число e предпочтительнее, потому что функция e^x является производной от самой себя, что способствует более эффективным вычислениям.

Пример 6.6. Сложение логарифмов

```
# Вычисляем совместное правдоподобие
joint_likelihood = 0.0

for p in patient_data:
    joint_likelihood += math.log(logistic_function(p.x) ** p.y * \
                                 (1.0 - logistic_function(p.x)) ** (1.0 - p.y))

joint_likelihood = math.exp(joint_likelihood)
```

Эта формула выражает предыдущий код на Python математически:

$$\text{совместное правдоподобие} = \sum_{i=1}^{n} \ln\left(\left(\frac{1}{1+e^{-(\beta_0+\beta_1 x_i)}}\right)^{y_i} \times \left(1-\frac{1}{1+e^{-(\beta_0+\beta_1 x_i)}}\right)^{1-y_i}\right)$$

Вы горите желанием вычислять частные производные по β_0 и β_1 в этом выражении? Думаю, вряд ли; это была бы лишняя морока. Если уж на то пошло, даже просто записать эту функцию на SymPy — это уже целая эпопея! Посмотрите на выражение в примере 6.7.

Пример 6.7. Совместное правдоподобие для логистической регрессии на SymPy

```
joint_likelihood = Sum(log((1.0 / (1.0 + exp(-(b + m * x(i)))))**y(i) * \
        (1.0 - (1.0 / (1.0 + exp(-(b + m * x(i))))))**(1-y(i))), (i, 0, n))
```

Поэтому давайте просто поручим SymPy вычислить частные производные по β_0 и β_1. После этого их можно сразу откомпилировать и применить для градиентного спуска, как показано в примере 6.8.

Пример 6.8. Градиентный спуск для логистической регрессии

```
from sympy import *
import pandas as pd

points = list(pd.read_csv("https://tinyurl.com/y2cocoo7").itertuples())

b1, b0, i, n = symbols('b1 b0 i n')
x, y = symbols('x y', cls=Function)
```

```
joint_likelihood = Sum(log((1.0 / (1.0 + exp(-(b0 + b1 * x(i))))) ** y(i) \
    * (1.0 - (1.0 / (1.0 + exp(-(b0 + b1 * x(i)))))) ** (1 - y(i))), (i, 0, n))

# Частная производная по β₀ с подстановкой точек
d_b1 = diff(joint_likelihood, b1) \
                    .subs(n, len(points) - 1).doit() \
                    .replace(x, lambda i: points[i].x) \
                    .replace(y, lambda i: points[i].y)

# Частная производная по β₁ с подстановкой точек
d_b0 = diff(joint_likelihood, b0) \
                    .subs(n, len(points) - 1).doit() \
                    .replace(x, lambda i: points[i].x) \
                    .replace(y, lambda i: points[i].y)

# Компилируем с помощью lambdify, чтобы ускорить вычисления
d_b1 = lambdify([b1, b0], d_b1)
d_b0 = lambdify([b1, b0], d_b0)

# Градиентный спуск
b1 = 0.01
b0 = 0.01
L = .01

for j in range(10_000):
    b1 += d_b1(b1, b0) * L
    b0 += d_b0(b1, b0) * L

print(b1, b0)
# 0.6926693075370812 -3.175751550409821
```

После того как мы вычислили частные производные по β_0 и β_1, мы подставляем в них значения x и y, а также количество точек данных n. Затем мы вызываем функцию `lambdify()`, чтобы откомпилировать обе производные и тем самым сделать вычисления эффективнее (эта функция использует NumPy «под капотом»). После этого мы выполняем градиентный спуск, как в главе 5, но поскольку здесь мы пытаемся максимизировать, а не минимизировать целевую функцию, мы добавляем каждую поправку к β_0 и β_1, а не вычитаем, как в методе наименьших квадратов.

Как видно из примера 6.8, мы получили $\beta_0 = -3{,}17575$ и $\beta_1 = 0{,}692667$. Эти результаты весьма близки к тем коэффициентам, которые мы раньше вычислили с помощью scikit-learn.

По примеру того, что мы узнали из главы 5, можно также выполнить стохастический градиентный спуск и на каждой итерации обрабатывать выборку только из одной или из нескольких записей. Это позволит увеличить скорость и производительность вычислений, а также предотвратить переобучение. Повторять здесь все подробности вряд ли имеет смысл, поэтому перейдем к следующей теме.

Многомерная логистическая регрессия

Давайте рассмотрим пример логистической регрессии с несколькими входными переменными. В таблице 6.1 приведены некоторые записи из вымышленного набора данных о занятости сотрудников и текучести кадров (полный набор находится по адресу https://bit.ly/3aqsOMO).

Таблица 6.1. Выборка данных о занятости сотрудников

Пол sex	Возраст age	Повышения promotions	Стаж работы years_employed	Уволен did_quit
1	32	3	7	0
1	34	2	5	0
1	29	2	5	1
0	42	4	10	0
1	43	4	10	0

Всего в наборе данных 54 записи. Допустим, мы хотим с его помощью прогнозировать, собираются ли увольняться другие сотрудники, и допустим, что здесь уместна логистическая регрессия. (На самом деле и то и другое — не очень хорошие идеи, и чуть позже я объясню, почему.) Вспомните, что регрессия допускает более одной входной переменной, как показано в этой формуле:

$$y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n)}}$$

Я создам β-коэффициенты для каждой из переменных: пол (`sex`), возраст (`age`), количество повышений по службе (`promotions`) и стаж работы в годах (`years_employed`). Выходная переменная `did_quit` (уволен ли сотрудник) является бинарной, и именно она будет отвечать за результат логистической регрессии, который мы прогнозируем. Поскольку мы имеем дело с несколькими измерениями, будет непросто наглядно изобразить изогнутую гиперплоскость, которая здесь служит логистической кривой. Поэтому мы воздержимся от визуализации.

Давайте сделаем задачу увлекательнее: мы будем использовать scikit-learn, но создадим интерактивную оболочку, с помощью которой сможем тестировать сотрудников. В примере 6.9 показан соответствующий код, и, когда мы его запустим, он построит модель логистической регрессии, а затем позволит нам вводить данные о новых сотрудниках, чтобы получить прогноз о том, уволятся они или нет. Что здесь может пойти не так? Конечно же, ничего! Мы всего лишь делаем прогнозы относительно личных качеств людей и принимаем решения на основе этих прогнозов. Я уверен, что все пойдет как по маслу. (Если что, это была ирония.)

Пример 6.9. Многомерная логистическая регрессия на данных о сотрудниках

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

employee_data = pd.read_csv("https://tinyurl.com/y6r7qjrp")

# Столбцы с независимыми переменными
inputs = employee_data.iloc[:, :-1].to_numpy()

# Столбец с зависимой переменной did_quit (уволен ли сотрудник)
output = employee_data.iloc[:, -1].to_numpy()

# Логистическая регрессия
fit = LogisticRegression(penalty=None).fit(inputs, output)

# Выводим коэффициенты
print(f"УГЛОВЫЕ КОЭФФИЦИЕНТЫ: {fit.coef_.flatten()}")
print(f"ПЕРЕСЕЧЕНИЕ: {fit.intercept_.flatten()}")

# Функция для интерактивного прогнозирования об увольнении новых сотрудников
def predict_employee_will_stay(sex, age, promotions, years_employed):
    prediction = fit.predict([[sex, age, promotions, years_employed]])
    probabilities = fit.predict_proba([[sex, age, promotions, years_employed]])
    if prediction == [[1]]:
        return f"УВОЛИТСЯ: {probabilities}"
    else:
        return f"ОСТАНЕТСЯ: {probabilities}"

# Проверяем прогнозирование
while True:
    n = input("Прогнозируем, останется ли сотрудник или уйдет.\n\
Введите данные (целые неотрицательные числа) через запятую:\n\
пол, возраст, количество повышений, стаж работы (лет): ")
    (sex, age, promotions, years_employed) = n.split(",")
    print(predict_employee_will_stay(int(sex), int(age), int(promotions),
        int(years_employed)))
```

На рис. 6.9 показано, как программа прогнозирует, уволится ли сотрудник. Пол сотрудника — «1», возраст — 34 года, он получил одно повышение и работает в компании уже 5 лет. Неудивительно, что прогноз будет такой: «УВОЛИТСЯ».

Можно заметить, что функция `predict_proba()` выводит два значения: первое — вероятность того, что сотрудник останется (то есть вероятность нуля, или значения «ложь»), а второе — вероятность того, что сотрудник уволится (то есть вероятность единицы, или значения «истина»).

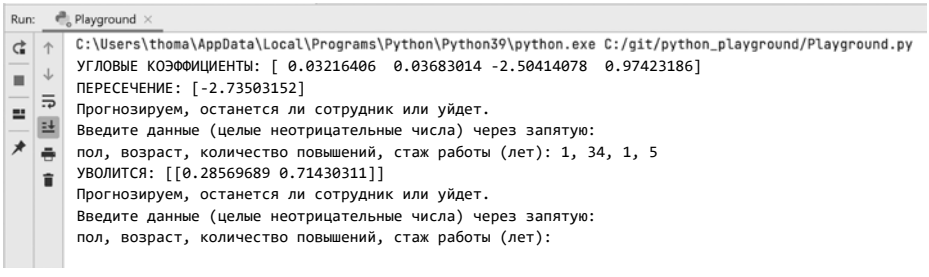


Рис. 6.9. Прогнозирование того, уволится ли 34-летний сотрудник, который получил одно повышение и проработал 5 лет

Обратите внимание, что коэффициенты для пола, возраста, количества повышений и стажа работы отображаются именно в таком порядке. По весу коэффициентов видно, что пол и возраст играют очень малую роль в прогнозе (их вес близок к 0). Однако коэффициенты для количества повышений и стажа имеют существенные веса: `2,504` и `0,97`. Вот в чем секрет этого искусственного набора данных: я сконструировал его так, что сотрудник увольняется, если не получает повышения примерно раз в два года. Естественно, логистическая регрессия уловила эту закономерность, и вы можете проверить ее и на других сотрудниках. Однако если выйти за пределы интервала данных, на которых обучалась модель, то предсказания, скорее всего, станут ненадежными. (Например, трудно предугадать, как поведет себя регрессия, если ввести данные о 70-летнем сотруднике, который не получал повышения в течение 3 лет: ведь у модели нет данных в этом возрастном диапазоне.)

Конечно, в реальности все не так однозначно. Может оказаться, что сотрудник, который работает в компании уже 8 лет и ни разу не получал повышения, вполне доволен своей позицией и не собирается уходить в ближайшее время. В этом случае такие переменные, как возраст, тоже могут сыграть свою роль и получить вес. Конечно, на практике имеет смысл озаботиться и другими релевантными переменными, которые в этом примере не учитывались. Далее во врезке эта тема раскрывается подробнее.

Классифицировать людей надо с умом!

Быстрый и верный способ создать себе проблемы — это собирать данные о людях и бездумно делать прогнозы, опираясь на эти данные. В этом случае могут всплыть не только вопросы конфиденциальности, но и юридические и репутационные затруднения, если модель окажется дискриминационной. Машинное обучение может придать вес таким входным переменным, как раса и пол, в результате чего пострадают представители

> определенных демографических групп: например, их не возьмут на работу или откажут им в кредите. Еще более вопиющие случаи происходят, когда системы видеонаблюдения ошибочно распознают людей как нарушителей или когда искусственный интеллект отказывает в условно-досрочном освобождении. Отметим также, что такие, казалось бы, безобидные переменные, как время поездки на работу, тоже нередко коррелируют с дискриминационными переменными.
>
> Случаи дискриминации в области машинного обучения упоминаются в ряде статей, например:
>
> Катианна Куач, «Школьницу не пустили на каток после того, как искусственный интеллект ошибочно распознал ее как дебошира» (https://oreil.ly/boUcW), The Register, 16.07.2021 (Katyanna Quach, «Teen turned away from roller rink after AI wrongly identifies her as banned troublemaker»);
>
> Кашмир Хилл, «Дефектный алгоритм принял добропорядочного гражданина за преступника» (https://oreil.ly/dOJyI), New York Times, 24.06.2020 (Kashmir Hill, «Wrongfully Accused by an Algorithm»).
>
> Поскольку законодательство о конфиденциальности данных продолжает развиваться, стоит проявлять осторожность и аккуратнее обращаться с персональной информацией. Задумывайтесь о том, какие решения будут принимать автоматизированные системы и как они могут навредить людям. Иногда лучше просто отказаться от помощи компьютеров и продолжать решать ту или иную проблему вручную.

Наконец, в примере с занятостью сотрудников стоило бы выяснить, откуда взялись эти данные. Да, этот набор данных я создал искусственно, но в реальных задачах всегда имеет смысл уточнить, как конкретно получились такие данные. Какой период времени охватывает выборка? Насколько давно мог уволиться сотрудник, который попал в выборку? Как определяется, что сотрудник остался? Таковыми считаются все действующие сотрудники на данный момент времени? Откуда мы знаем, что они не собираются увольняться? (Если они планируют уходить, но выборка этого не учитывает, это порождает ложноотрицательные точки данных.) Специалисты по data science легко впадают в заблуждение, когда анализируют только то, о чем говорят данные, но не задаются вопросом, откуда эти данные взялись и какие допущения в них заложены.

Лучший способ узнать ответы на эти вопросы — понять, ради чего мы делаем прогнозы. Мы хотим решить, когда стоит повышать сотрудников по службе, чтобы удержать их? Может ли это создать круговое смещение, из-за которого сотрудники с определенным набором качеств получат преимущества? Не усугубится ли это смещение, когда эти повышения войдут в новые обучающие данные?

Все это важные и даже, возможно, неудобные вопросы, из-за которых в коллективе могут возникнуть нежелательные трения. Если ваша команда или руководство не приветствует такую скрупулезность, подумайте о том, чтобы переключиться на другую роль, где любопытство помогает, а не мешает.

Логит-функция

На этом месте пришло время углубиться в логистическую регрессию и разобраться, как она устроена с математической точки зрения. Этот может показаться непростым, поэтому не торопитесь. Если вы почувствуете, что теряете нить, отложите чтение и вернитесь к этому разделу позже.

С начала XX века математиков интересовало, как отмасштабировать область значений линейной функции так, чтобы они лежали в интервале от 0 до 1, а значит, подходили для того, чтобы прогнозировать вероятность. В логистической регрессии для этого можно использовать функцию логарифма шансов, которая также называется логит-функцией или просто логитом.

Помните, я уже упоминал, что показатель степени $\beta_0 + \beta_1 x$ — это линейная функция? Взгляните на логистическую функцию еще раз:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

Линейная функция, которая стоит в показателе степени числа *e*, — это и есть *логарифм шансов* для интересующего нас события. Возможно, вы удивитесь: «Подождите, я не вижу тут ни логарифма, ни шансов. Это просто линейная функция!» Немного терпения, и я покажу вам все математические подробности.

В качестве примера возьмем предыдущую логистическую регрессию, в которой $\beta_0 = -3{,}175805042563378$, а $\beta_1 = 0{,}6926893863085584$. Какова вероятность того, что симптомы проявятся у пациента, который подвергался воздействию химических веществ в течение шести часов, т. е. при $x = 6$? Мы уже знаем, как это вычислить: нужно просто подставить эти коэффициенты в логистическую функцию:

$$p = \frac{1}{1 + e^{-(-3{,}175805042563378 + 0{,}6926893863085584 \times 6)}} = 0{,}727173943653684$$

Таким образом мы получили вероятность 0,72717. Но давайте посмотрим на нее с точки зрения шансов. Вспомните, как в главе 2 мы научились переводить вероятность в шансы:

$$\text{шансы} = \frac{p}{1-p} = \frac{0,72717}{1-0,72717} = 2,6653390566575306$$

Таким образом, если пациент подвергался воздействию в течение 6 часов, то вероятность того, что у него проявятся симптомы, в 2,66534 раза выше вероятности того, что они не проявятся.

Если взять от функции шансов натуральный логарифм (по основанию e), то получится функция, которая так и называется — *логарифм шансов*, или *логит-функция*:

$$\text{логит} = \ln\left(\frac{p}{1-p}\right)$$

$$\text{логит} = \ln\left(\frac{0,72717}{1-0,72717}\right) = 0,9803312752879725$$

Значение логит-функции для 6 часов составляет 0,9803313. Что это значит и почему нам это важно? Когда мы оперируем логарифмами шансов, становится легче сравнивать одни шансы с другими. Если логит больше 0, то шансы говорят в пользу того, что событие произойдет, а если он меньше 0, — в пользу того, что событие не произойдет. На числовой прямой логарифм шансов −1,05 находится на том же расстоянии от 0, что и 1,05. Однако если рассматривать шансы без логарифмов, эти значения равны 0,3499 и 2,858 соответственно, что не так удобно интерпретировать. В этом и заключается удобство логитов.

Шансы и логарифмы

Логарифмы и шансы интересным образом связаны друг с другом. Шансы на неудачу события находятся только в интервале от 0 до 1, а шансы на успех занимают всю числовую прямую от 1 до положительной бесконечности. Такая асимметрия выглядит неаккуратно. Однако логарифмирование масштабирует шансы так, что они становятся полностью линейными и симметричными. Если логит равен 0, это означает, что шансы на успех и на неудачу одинаковы. Логит, равный −1,05, линейно находится на том же расстоянии от 0, что и 1,05, поэтому сравнивать шансы гораздо проще.

У Джоша Стармера (Josh Starmer) есть отличное видео (https://oreil.ly/V0H8w) об этой взаимосвязи между шансами и логарифмами.

Помните, я говорил, что линейная функция $\beta_0 + \beta_1 x$ в формуле логистической регрессии — это и есть логарифм шансов? Убедитесь в этом:

$$\text{логит} = \beta_0 + \beta_1 x =$$

$$= -3,175805042563378 + 0,6926893863085584 \times 6 = 0,9803312752879725$$

Это то же самое значение 0,9803313, что и в предыдущем расчете, где мы вычисляли шансы логистической регрессии при $x = 6$, а затем логарифмировали их! Как одно с другим связано? Почему обе формулы дают одинаковый результат? Получается, что если p — это вероятность из логистической регрессии, а x — входная переменная, то верно равенство:

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x$$

Давайте построим график логит-функции вместе с логистической регрессией, как показано на рис. 6.10.

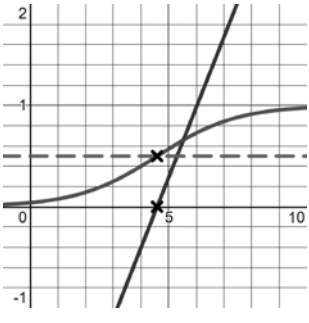


Рис. 6.10. Прямая логарифма шансов преобразуется в логистическую функцию, которая возвращает вероятность

Каждая логистическая регрессия на самом деле опирается на линейную функцию, и эта функция — логарифм шансов. Обратите внимание на рис. 6.10, что когда логит равен 0, то вероятность на логистической кривой равна 0,5. Это логично, потому что когда шансы составляют один к одному, то вероятность успеха — 0,5, о чем и свидетельствует логистическая регрессия, а логарифм шансов обращается в 0 — то есть его прямая пересекает ось *X*.

Есть еще одно преимущество в том, чтобы рассматривать логистическую регрессию с точки зрения шансов: можно сравнивать друг с другом эффекты при разных значениях *x*. Допустим, мы хотим выяснить, насколько различаются шансы людей, которые подвергались воздействию химиката в течение 6 и 8 часов. Можно взять по отдельности шансы для $x = 6$ и $x = 8$, а потом разделить одни шансы на другие и получить *отношение шансов*. Не путайте эту величину с обычными шансами, которые тоже являются отношением, но не отношением шансов.

Сначала найдем вероятности проявления симптомов для 6 и 8 часов (обозначим их как p_6 и p_8 соответственно):

$$p = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$

$$p_6 = \frac{1}{1+e^{-(-3{,}175805042563378+0{,}6926893863085584\times 6)}} = 0{,}727173943653684$$

$$p_8 = \frac{1}{1+e^{-(-3{,}175805042563378+0{,}6926893863085584\times 8)}} = 0{,}914174871957759$$

Теперь преобразуем их в шансы, которые обозначим как o_x:

$$o = \frac{p}{1-p}$$

$$o_6 = \frac{0{,}727173943653684}{1-0{,}727173943653684} = 2{,}6653390566575306$$

$$o_8 = \frac{0{,}914174871957759}{1-0{,}914174871957759} = 10{,}651599278801253$$

И наконец, составим из этих двух показателей отношение шансов — пропорцию, где в числителе стоят шансы для восьми часов, а в знаменателе — для шести. В результате получится примерно 3,996: это значит, что если подвергаться воздействию химиката на 2 часа дольше, то вероятность появления симптомов увеличивается почти в 4 раза:

$$\text{отношение шансов} = \frac{10{,}651599278801253}{2{,}6653390566575306} = 3{,}9963393220819325$$

Можно убедиться, что отношение шансов 3,996 сохраняется на любом интервале продолжительностью 2 часа, например между 2 и 4 часами, между 4 и 6 часами, между 8 и 10 часами и т. д. При двухчасовом интервале отношение шансов остается неизменным, а при другой продолжительности интервала оно будет отличаться.

R^2

В главе 5 мы рассмотрели несколько метрик качества для линейной регрессии, и сейчас попытаемся сделать то же самое для логистической регрессии. Нас по-прежнему заботят многие из тех же проблем, что и в линейной регрессии, в том числе переобучение и дисперсия. К счастью, можно позаимствовать некоторые метрики из линейной регрессии и адаптировать их к логистической. Начнем с R^2.

Для логистической регрессии, как и для линейной, существует показатель R^2.[1] Если вы помните из главы 5, он характеризует, насколько хорошо та или иная независимая переменная объясняет зависимую. Применительно к нашей задаче воздействия химических веществ это означает, что мы хотим оценить, в какой степени продолжительность воздействия объясняет проявление симптомов.

Не существует единого мнения о том, как лучше вычислять R^2 для логистической регрессии, но есть популярная метрика, которая известна как псевдо-R^2 Макфаддена и очень похожа на коэффициент детерминации, который вычисляется в линейной регрессии. Мы будем использовать эту метрику в последующих примерах, а вот формула, по которой она рассчитывается:

$$R^2 = \frac{\ln(\text{правдоподобие}) - \ln(\text{правдоподобие с подгонкой})}{\ln(\text{правдоподобие})} =$$

$$= 1 - \frac{\ln(\text{правдоподобие с подгонкой})}{\ln(\text{правдоподобие})}$$

Для начала разберемся, что такое «ln(правдоподобие)» и «ln(правдоподобие с подгонкой)» и как их вычислить, чтобы получить R^2.

Здесь не получится использовать остатки, как в линейной регрессии, но можно спроецировать полученные результаты на логистическую кривую, как показано на рис. 6.11, и найти соответствующие им правдоподобия от 0 до 1.

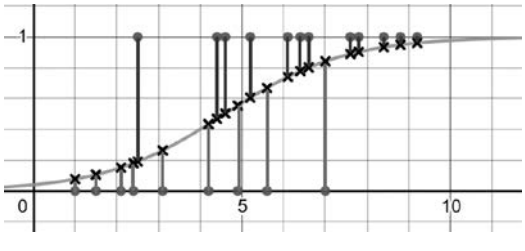


Рис. 6.11. Проецирование полученных значений на логистическую кривую

Затем можно взять натуральный логарифм от каждого из этих правдоподобий и сложить их вместе. Это и будет «ln(правдоподобие с подгонкой)» из предыдущей формулы (см. пример 6.10). Как и при расчете максимального правдоподобия, мы вычитаем правдоподобия «ложных» результатов из 1.

---

[1] Формально в логистической регрессии нельзя вычислить такой же коэффициент детерминации, как в линейной, потому что выходная переменная здесь принимает не числовые, а категориальные значения. Однако разработано несколько альтернативных метрик, которые называются *псевдо-R^2* и оценивают качество подгонки, подобно R^2 в линейной регрессии. Далее в книге рассматривается одна из этих метрик — псевдо-R^2 Макфаддена. — *Примеч. науч. ред.*

Пример 6.10. Вычисление логарифмического правдоподобия с подгонкой

```
from math import log, exp
import pandas as pd

patient_data = pd.read_csv("https://bit.ly/33ebs2R", delimiter=",").itertuples()

b0 = -3.175805042563378
b1 = 0.6926893863085584

def logistic_function(x):
    p = 1.0 / (1.0 + exp(-(b0 + b1 * x)))
    return p

# Сумма логарифмических правдоподобий
log_likelihood_fit = 0.0
for p in patient_data:
    if p.y == 1.0:
        log_likelihood_fit += log(logistic_function(p.x))
    elif p.y == 0.0:
        log_likelihood_fit += log(1.0 - logistic_function(p.x))

print(log_likelihood_fit) # -9.946161678665305
```

Используя специальные приемы двоичного умножения и списковое включение Python, можно объединить цикл `for` и инструкцию `if` в одну строку, которая возвращает значение `log_likelihood_fit`. Аналогично тому, как мы поступали с формулой максимального правдоподобия, мы используем двоичное вычитание между положительными и отрицательными случаями, чтобы математически исключить либо первый, либо второй вариант. На каждой итерации одно из слагаемых (но не оба) умножается на 0, а значит, в итоговую сумму входит либо положительный, либо отрицательный случай, но не оба сразу (см. пример 6.11).

Пример 6.11. Однострочная инструкция для вычисления логарифмического правдоподобия с подгонкой

```
log_likelihood_fit = sum(log(logistic_function(p.x)) * p.y +
                         log(1.0 - logistic_function(p.x)) * (1.0 - p.y)
                         for p in patient_data)
```

Вот как выглядит ln(правдоподобие с подгонкой) в математической форме. Обратите внимание, что $f(x_i)$ — это логистическая функция для заданного аргумента x_i:

$$\ln(\textit{правдоподобие с подгонкой}) = \sum_{i=1}^{n}\left(\ln\left(f\left(x_i\right)\right)\times y_i + \ln\left(1-f\left(x_i\right)\right)\times\left(1-y_i\right)\right)$$

Как было рассчитано в примерах 6.10 и 6.11, логарифмическое правдоподобие с подгонкой равно −9,9461. Чтобы вычислить R^2, необходима еще одна величина:

ln(правдоподобие), которое рассчитывается без каких-либо входных переменных и просто равно количеству положительных случаев, деленному на количество всех случаев (фактически это соответствует просто пересечению с осью). Обратите внимание, что количество случаев, в которых проявляются симптомы, можно подсчитать, если просто сложить все значения y (то есть получить сумму $\sum y_i$), потому что в сумму войдут только значения 1, а не 0. Вот итоговая формула:

$$\ln(\textit{правдоподобие}) = \sum_{i=1}^{n}\left(\ln\left(\frac{\sum y_i}{n}\right)\times y_i + \ln\left(1-\frac{\sum y_i}{n}\right)\times(1-y_i)\right)$$

В примере 6.12 представлен расширенный эквивалент этой формулы на Python.

Пример 6.12. Логарифмическое правдоподобие для выборки пациентов

```
import pandas as pd
from math import log, exp

patient_data = list(pd.read_csv("https://bit.ly/33ebs2R", delimiter=",") \
        .itertuples())

likelihood = sum(p.y for p in patient_data) / len(patient_data)

log_likelihood = 0.0

for p in patient_data:
    if p.y == 1.0:
        log_likelihood += log(likelihood)
    elif p.y == 0.0:
        log_likelihood += log(1.0 - likelihood)

print(log_likelihood) # -14.341070198709906
```

Чтобы привести эту логику в соответствие с формулой, можно преобразовать цикл `for` и инструкцию `if` в одну строку, используя двоичное умножение для положительных и отрицательных случаев (пример 6.13).

Пример 6.13. Однострочная инструкция для вычисления логарифмического правдоподобия

```
log_likelihood = sum(log(likelihood)*p.y + log(1.0 - likelihood)*(1.0 - p.y) \
        for p in patient_data)
```

Наконец, подставим эти значения в исходную формулу, чтобы вычислить R^2:

$$R^2 = 1 - \frac{\ln(\text{правдоподобие с подгонкой})}{\ln(\text{правдоподобие})} =$$

$$= 1 - \frac{-9,946161678665305}{-14,341070198709906} = 0,30645610537768353$$

В примере 6.14 приведен код на Python, который вычисляет R^2 от начала до конца.

Пример 6.14. Вычисление R^2 для логистической регрессии

```
import pandas as pd
from math import log, exp

patient_data = list(pd.read_csv("https://bit.ly/33ebs2R", delimiter=",") \
                                .itertuples())

# Коэффициенты подогнанной логистической регрессии
b0 = -3.175805042563378
b1 = 0.6926893863085584

def logistic_function(x):
    p = 1.0 / (1.0 + exp(-(b0 + b1 * x)))
    return p

# Логарифмическое правдоподобие с подгонкой
log_likelihood_fit = sum(log(logistic_function(p.x)) * p.y +
                         log(1.0 - logistic_function(p.x)) * (1.0 - p.y)
                         for p in patient_data)

# Логарифмическое правдоподобие без подгонки
likelihood = sum(p.y for p in patient_data) / len(patient_data)

log_likelihood = sum(log(likelihood) * p.y + log(1.0 - likelihood) * (1.0 - p.y)
\
        for p in patient_data)

# Вычисляем R²
r2 = 1 - (log_likelihood_fit / log_likelihood)

print(r2) # 0.3064561053776833
```

Итак, после того как мы получили $R^2 = 0{,}306456$, можно ли сказать, что продолжительность воздействия химических веществ объясняет проявление симптомов у пострадавших? Как мы выяснили в главе 5, где обсуждалась линейная регрессия, при слабой подгонке R^2 будет ближе к 0, а при сильной — ближе к 1. Таким образом, можно сделать вывод, что продолжительность воздействия плохо прогнозирует симптомы, потому что R^2 составляет всего 0,306456. Должны существовать другие переменные, помимо продолжительности воздействия, которые лучше прогнозируют, проявятся ли симптомы. Это вполне логично, потому что на большей части интервала наблюдаемых данных встречаются пациенты как с симптомами, так и без симптомов, как показано на рис. 6.12.

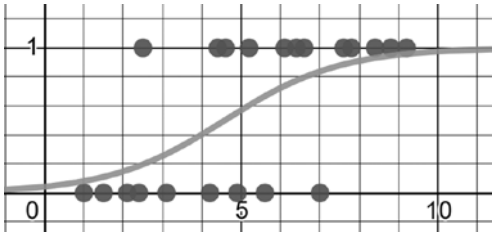


Рис. 6.12. Данные показывают посредственный $R^2 = 0,306456$, потому что в центральной части кривой большой разброс

Но если бы данные были четко разделены и исходы 1 и 0 не перекрывали друг друга, как показано на рис. 6.13, то мы бы получили идеальное значение R^2, равное 1.

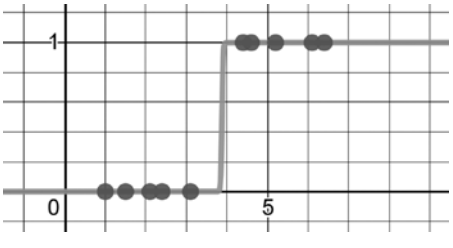


Рис. 6.13. Эта логистическая регрессия имеет идеальное значение $R^2 = 1$, потому что результаты, предсказанные в зависимости от продолжительности воздействия, четко отделены друг от друга

p-значения

Как и в случае с линейной регрессией, наша работа не заканчивается на том, что мы получили R^2. Нужно выяснить, насколько вероятно, что эти данные возникли случайно, а не в результате объективной взаимосвязи. Это значит, что надо вычислить *p*-значение.

Для этого нам понадобится изучить новое распределение вероятностей, которое называется *распределением хи-квадрат* и обозначается χ^2. Это непрерывное распределение используется в различных областях статистики, включая логистическую регрессию.

Если взять каждое значение стандартного нормального распределения (со средним 0 и стандартным отклонением 1) и возвести его в квадрат, то получится распределение χ^2 с одной степенью свободы. Для наших целей количество степеней свободы зависит от количества параметров *n* в логистической регрессии и равно $n-1$. Примеры распределений с разным количеством степеней свободы представлены на рис. 6.14.

Рис. 6.14. Распределение χ^2 с различным количеством степеней свободы

Поскольку в нашей регрессии два параметра (продолжительность воздействия и наличие симптомов), количество степеней свободы будет равно $2-1=1$.

Нам понадобится логарифмическое правдоподобие с подгонкой и без, которое мы вычисляли в предыдущем разделе, посвященном R^2. Вот формула, по которой рассчитывается нужное значение χ^2:

$$\chi^2 = 2\times\left(\ln\left(\text{правдоподобие с подгонкой}\right)-\ln\left(\text{правдоподобие}\right)\right)$$

Затем возьмем это значение и найдем соответствующую ему вероятность из распределения χ^2. Это и будет *p*-значение:

$$\text{p-значение} = \text{PDF}\left(\chi^2\right) = \text{PDF}\left(2\times\left(\ln\left(\text{правдоподобие с подгонкой}\right)-\ln\left(\text{правдоподобие}\right)\right)\right)$$

В примере 6.15 показано, как вычислить *p*-значение для нашей логистической регрессии. Чтобы работать с распределением хи-квадрат, мы используем класс `chi2` из библиотеки SciPy.

Пример 6.15. Вычисление *p*-значения для логистической регрессии

```
import pandas as pd
from math import log, exp
from scipy.stats import chi2

patient_data=list(pd.read_csv("https://bit.ly/33ebs2R", delimiter=",").
itertuples())
```

```
# Коэффициенты подогнанной логистической регрессии
b0 = -3.175805042563378
b1 = 0.6926893863085584

def logistic_function(x):
    p = 1.0 / (1.0 + exp(-(b0 + b1 * x)))
    return p

# Логарифмическое правдоподобие с подгонкой
log_likelihood_fit = sum(log(logistic_function(p.x)) * p.y +
                         log(1.0 - logistic_function(p.x)) * (1.0 - p.y)
                         for p in patient_data)

# Логарифмическое правдоподобие без подгонки
likelihood = sum(p.y for p in patient_data) / len(patient_data)

log_likelihood = sum(log(likelihood) * p.y +
                     log(1.0 - likelihood) * (1.0 - p.y) \
                     for p in patient_data)

# Вычисляем p-значение
chi2_input = 2 * (log_likelihood_fit - log_likelihood)
p_value = chi2.pdf(chi2_input, 1) # одна степень свободы (n - 1)
print(p_value) # 0.0016604875719245015
```

Итак, *p*-значение нашей логистической регрессии равно 0,00166, и если задан уровень значимости 0,05, то можно сказать, что взаимосвязь между продолжительностью воздействия химикатов и проявлением симптомов отравления статистически значима и не является случайностью.

Обучающая и тестовая выборки

Как уже упоминалось в главе 5, которая была посвящена линейной регрессии, для валидации алгоритмов машинного обучения можно разбивать набор данных на обучающую и тестовую выборки. Этот способ оценить эффективность логистической регрессии больше отвечает духу машинного обучения. Хотя полагаться на такие традиционные статистические показатели, как R^2 и *p*-значения, — хорошая идея, она становится менее практичной, когда вы имеете дело с большим количеством переменных. Вот тут-то снова имеет смысл разделить набор данных на две выборки. На рис. 6.15 показана трехкратная перекрестная валидация, где на каждой итерации тестовой выборкой становится новая треть набора данных.

В примере 6.16 выполняется логистическая регрессия на наборе данных о занятости сотрудников, но данные разбиваются на трети, и каждая треть поочередно используется в качестве тестовой выборки. Наконец, чтобы обобщить три значения точности, мы вычисляем их среднее и стандартное отклонение.

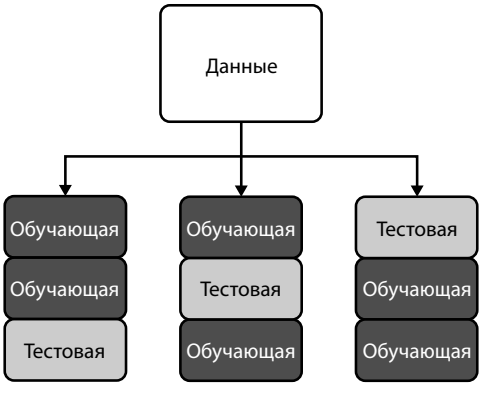


Рис. 6.15. Трехкратная перекрестная валидация:
каждая треть набора данных попеременно становится тестовой выборкой

Пример 6.16. Логистическая регрессия с трехкратной валидацией

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, cross_val_score

# Загружаем данные
df = pd.read_csv("https://tinyurl.com/y6r7qjrp", delimiter=",")

X = df.values[:, :-1]
Y = df.values[:, -1]

# random_state — затравочное значение для генератора случайных чисел
kfold = KFold(n_splits=3, random_state=7, shuffle=True)
model = LogisticRegression(penalty=None)
results = cross_val_score(model, X, Y, cv=kfold)

print(f"Средняя точность: {results.mean():.3f} (ст. откл.= {results.
std():.3f})")
```

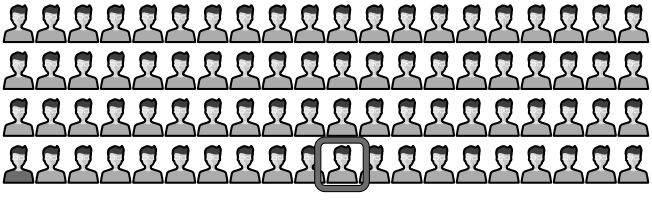
Также можно использовать случайную валидацию, поэлементную перекрестную валидацию и все остальные варианты валидации, которые мы рассматривали в главе 5. Не будем повторяться, а лучше обсудим, почему точность[1] — плохая метрика для задач классификации.

[1] В русскоязычной литературе по data science нет консенсуса о том, как переводить английские названия метрик бинарной классификации. В частности, и *accuracy*, и *precision* в разных источниках (а иногда в одном и том же источнике) переводятся как *точность*. Другой встречающийся вариант: *precision* – точность, *accuracy* – доля верных результатов. В этой книге для *accuracy* сохранен перевод *точность* (как наиболее устоявшийся вариант), а *precision* переводится как *прецизионность* (в соответствии со стандартами ГОСТ Р ИСО 5725-1-2002 и РМГ 29-2013). — *Примеч. науч. ред.*

Матрица ошибок

Предположим, модель пришла к выводу, что люди с именем Майкл склонны увольняться из компании. На самом деле использовать имена и фамилии в качестве входных переменных — сомнительный подход, потому что трудно представить себе, чтобы имя сотрудника влияло на то, уволится он или нет. Однако, чтобы упростить пример, давайте продолжим рассматривать такую модель. Она прогнозирует, что любой человек по имени Майкл уволится с работы.

Вот тут-то как раз точность (доля верных прогнозов) и не работает. В компании сто сотрудников, из которых одного зовут Майкл, а другого — Сэм. Модель ошибочно предсказывает, что Майкл уволится, но в итоге увольняется Сэм. Какова точность модели? Она равна 98 %, потому что на сто сотрудников приходится только два неверных прогноза, как показано на рис. 6.16.



Фактически:

Увольняется этот сотрудник. Прогноз оказался ошибочным, но точность модели все равно составляет 98 %!

Прогноз:

Этого сотрудника зовут Майкл. Этот сотрудник уволится

Рис. 6.16. Модель прогнозирует, что сотрудник по имени Майкл уволится, но на самом деле увольняется другой сотрудник, а точность модели при этом равна 98 %

В задачах классификации метрика точности особенно сбивает с толку в случае дисбаланса данных, когда интересующие нас события (например, увольнения) происходят редко. Если поставщик, консультант или специалист по data science пытается продать вам систему классификации и утверждает, что она обладает высокой точностью, попросите показать матрицу ошибок.

Матрица ошибок — это таблица, в которой прогнозы сопоставляются с фактическими результатами и подсчитываются результаты четырех типов: истинно положительные, истинно отрицательные, ложноположительные (ошибки первого рода) и ложноотрицательные (ошибки второго рода) результаты. На рис. 6.17 показана матрица ошибок для нашего примера.

		Фактическое событие	
		Сотрудник остался (положительный результат)[1]	Сотрудник остался (отрицательный результат)
Прогнозируемое событие	Сотрудник уволится (положительный результат)	$TP = 0$ истинно положительные (true positive)	$FP = 1$ ложноположительные (false positive)
	Сотрудник останется (отрицательный результат)	$FN = 1$ ложноотрицательные (false negative)	$TN = 98$ истинно отрицательные (true negative)

Рис. 6.17. Простая матрица ошибок

Как правило, нас интересует, чтобы значения на главной диагонали (которая идет из левого верхнего угла матрицы в правый нижний) были выше, потому что они отражают правильную классификацию. Нам важно знать, сколько сотрудников, чье увольнение предсказала модель, на самом деле уволились (истинно положительные результаты). Но не менее важно знать, сколько сотрудников, которых модель пометила как остающихся, действительно остались (истинно отрицательные результаты).

Остальные ячейки отражают неверные прогнозы. Когда сотрудник, чье увольнение спрогнозировала модель, в итоге остался — это ложноположительный результат, а когда уволился сотрудник, которого модель сочла остающимся, — это ложноотрицательный результат.

Нужно вместо одной общей метрики точности рассмотреть более узкие показатели, которые ориентированы на различные части матрицы ошибок. Взгляните на рис. 6.18, где перечислены несколько полезных метрик.

Из матрицы ошибок можно вывести всевозможные полезные метрики, помимо точности. Можно легко заметить, что прецизионность (какая доля спрогнозированных положительных результатов спрогнозирована правильно) и чувствительность (какая доля фактических положительных результатов была верно спрогнозирована) равны 0, а это значит, что наша модель машинного обучения совершенно не оправдала себя на положительных прогнозах.

[1] В задачах бинарной классификации *положительным* результатом называется тот, для которого значение целевого признака принимается за 1 (в данном случае это увольнение сотрудника). Это не всегда то, что понимают под положительным результатом в повседневной речи (то есть благоприятный или желательный исход). — *Примеч. науч. ред.*

		Фактическое событие		
		Сотрудник уволился	Сотрудник остался	
Прогнозируемое событие	Сотрудник уволится	$TP = 0$	$FP = 1$	**Прецизионность** $\frac{TP}{TP+FP} = \frac{0}{0+1} = 0$
	Сотрудник останется	$FN = 1$	$TN = 98$	**Отрицательная прогностическая ценность** $\frac{TN}{TN+FN} = \frac{98}{98+1} = 0,99$
		Полнота (Чувствительность) $\frac{TP}{TP+FN} = \frac{0}{0+1} = 0$	**Специфичность** $\frac{TN}{TN+FP} = \frac{98}{98+1} = 0,99$	**Точность** $\frac{TP+TN}{TP+TN+FP+FN} = \frac{98}{0+98+1+1} = 0,98$
				F_1-мера $\frac{2 \times Прецизионность \times Полнота}{Прецизионность + Полнота} = не\ определено$

Рис. 6.18. Полезные метрики для матрицы ошибок

В примере 6.17 показано, как использовать API матрицы ошибок из библиотеки scikit-learn для логистической регрессии с разбиением на обучающую и тестовую выборки. Обратите внимание, что матрица ошибок здесь применяется только к тестовой выборке.

Пример 6.17. Матрица ошибок для тестовой выборки с помощью scikit-learn

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

# Загружаем данные
df = pd.read_csv("https://bit.ly/3cManTi", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1]

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]

model = LogisticRegression(solver="liblinear")

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.33,
    random_state=10)
```

```
model.fit(X_train, Y_train)
prediction = model.predict(X_test)

""" Матрица ошибок определяет количество результатов в каждой категории:
[[истинно_положительные ложноотрицательные]
 [ложноположительные истинно_отрицательные]]

[[6 3]
 [4 5]]

Главная диагональ отражает правильные прогнозы,
поэтому желательно, чтобы значения на ней были больше
"""
matrix = confusion_matrix(y_true=Y_test, y_pred=prediction)
print(matrix)
```

Формула Байеса и классификация

Помните ли вы формулу Байеса из главы 2? С ее помощью можно привлекать дополнительную информацию извне, чтобы подтвердить выводы, которые сделаны на основе матрицы ошибок. На рис. 6.19 показана матрица ошибок для 1 000 пациентов, которые обследовались на предмет некоторого заболевания.

		Пациент в группе риска?[1]	
		Да	Нет
Результат обследования	Положительный (выявлено заболевание)	198	50
	Отрицательный (не выявлено заболевание)	2	750

Рис. 6.19. Матрица ошибок для медицинского обследования, выявляющего заболевание

Данные свидетельствуют о том, что у 99 % пациентов, входящих в группу риска, диагностировано заболевание (чувствительность). Используя матрицу ошибок, можно увидеть, что это подтверждается математически:

$$\text{чувствительность} = \frac{TP}{TP + FN} = \frac{198}{198 + 2} = 0,99$$

Но что, если перевернуть условие? Какая доля тех, кто получил положительный результат, входит в группу риска (прецизионность)? Хотя мы переворачиваем

[1] В этом примере результат обследования рассматривается как прогноз того, что у пациента есть заболевание, а в группу риска входят все пациенты, у которых, предположительно, фактически есть заболевание. — *Примеч. науч. ред.*

условную вероятность, не нужно использовать формулу Байеса, потому что в матрице ошибок уже есть все необходимые данные:

$$\text{прецизионность} = \frac{198}{198+50} = 0,798$$

Вроде бы 79,8 % — не так уж и мало: такова доля пациентов с выявленным заболеванием, которые, предположительно, действительно больны. Но подумайте вот о чем: на какие допущения об исходных данных мы опираемся? Репрезентативна ли эта выборка по отношению к генеральной совокупности?

Допустим, из другого исследования известно, что этим заболеванием страдает 1 % всего населения. Это позволяет воспользоваться формулой Байеса. Если объединить эту долю с показателями нашей матрицы ошибок, можно обнаружить кое-что важное:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

$$P(\text{в группе риска} \mid \text{положительный результат}) =$$

$$= \frac{P(\text{положительный результат} \mid \text{в групе риска}) \times P(\text{в группе риска})}{P(\text{положительный результат})}$$

$$P(\text{в группе риска} \mid \text{положительный результат}) = \frac{0,99 \times 0,01}{0,248} = 0,399$$

Если учесть, что в группе риска находится только 1 % всего населения, а в нашей выборке таких 20 %, то вероятность попасть в группу риска при положительном результате обследования составит 3,99 %! Как же она снизилась с 99 %? Этот пример показывает, как легко обмануться, если полагаться на вероятности, которые высоки только в конкретной выборке — например, среди 1 000 пациентов, которых обследовала фармацевтическая компания. Так что, если это обследование на самом деле выявляет истинно положительные результаты с вероятностью всего 3,39 %, — скорее всего, его не стоит использовать.

ROC-кривая и показатель AUC

Оценивая различные конфигурации машинного обучения, можно получить десятки, сотни или тысячи матриц ошибок. Изучать их может быть утомительно, поэтому можно обобщить их все с помощью *ROC-кривой*[2], как показано на рис. 6.20.

[2] ROC — это аббревиатура от receiver operating characteristic (рабочая характеристика приемника). — *Примеч. перев.*

Она показывает, как связаны чувствительность и специфичность модели при разной пороговой вероятности положительного результата и позволяет найти приемлемый баланс между долей истинно положительных и ложноположительных результатов.

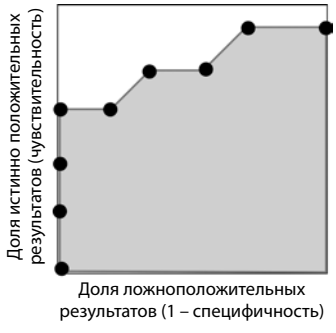


Рис. 6.20. ROC-кривая

Также можно сравнивать различные модели машинного обучения, если построить для каждой из них отдельную ROC-кривую. Например, если на рис. 6.21 верхняя кривая представляет логистическую регрессию, а нижняя — дерево решений (метод машинного обучения, который не рассматривается в этой книге), то можно наглядно сравнить их эффективность. *AUC* (area under the curve, площадь под кривой) — подходящая метрика для того, чтобы выбрать одну из двух моделей. Площадь под верхней кривой (логистическая регрессия) больше, а значит, эта модель лучше[1].

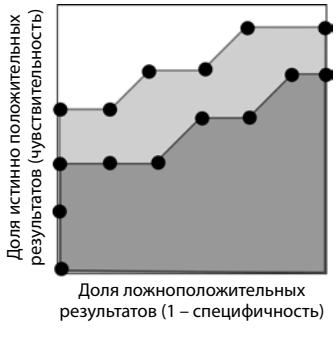


Рис. 6.21. ROC-кривые и соответствующие площади AUC для сравнения двух моделей

[1] Чем ближе метрика AUC к 1, тем лучше модель. Если AUC близка к 0,5, это значит, что модель неудачна: качество ее прогнозов не лучше, чем при случайном гадании. Когда AUC близка к 0, это значит, что она хорошо прогнозирует «на 180°», принимая положительные результаты за отрицательные и наоборот. Если в такой модели поменять местами положительные и отрицательные прогнозы, то ее AUC будет близка к 1, и качество окажется высоким. — *Примеч. науч. ред.*

Чтобы использовать AUC как метрику качества модели, в API scikit-learn измените параметр `scoring` на `roc_auc`, как показано в примере 6.18 для перекрестной проверки.

Пример 6.18. Использование AUC в качестве метрики качества модели

```
# Поместите сюда логистическую регрессию из примера 6.16

results = cross_val_score(model, X, Y, cv=kfold, scoring="roc_auc")
print(f"AUC = {results.mean():.3f} (ст. откл.= {results.std():.3f})")
# AUC = 0.814 (ст. откл.= 0.056)
```

Дисбаланс классов

Перед тем как завершить эту главу, необходимо затронуть еще одну тему. Как уже говорилось ранее, когда мы обсуждали матрицы ошибок, одной из проблем машинного обучения является *дисбаланс классов*, который возникает, когда в разных классах результатов представлены непропорционально разные объемы данных. К сожалению, это характерно для многих задач, которые представляют интерес: например, прогноза заболеваемости, борьбы с утечками информации, обнаружения мошенничества и т. д. Проблема дисбаланса классов до сих пор остается открытой, и у нее нет универсального решения. Однако существуют несколько методов, которые можно попробовать.

Для начала можно принять очевидные меры: например, собрать больше данных или испытать разные модели, а также использовать матрицы ошибок[2]. Все это поможет отслеживать некачественные прогнозы и заблаговременно выявлять ошибки.

Еще один распространенный прием — дублировать выборки в миноритарном классе до тех пор, пока он не сравняется с мажоритарным. Кроме того, при разбиении данных на обучающую и тестовую выборки можно передать методу `train_test_split` параметр `stratify` со столбцом значений зависимой переменной, и он попытается сохранить в каждой выборке такую же пропорцию классов, как в этом столбце (см. пример 6.19).

Пример 6.19. Параметр `stratify` в scikit-learn для балансировки классов в выборках

```
X, Y = …
X_train, X_test, Y_train, Y_test = \
        train_test_split(X, Y, test_size=.33, stratify=Y)
```

[2] Для сильно несбалансированных классов метрика ROC-AUC подходит плохо, зато можно применять PR-AUC. Это похожая метрика, которая характеризует площадь под кривой PR (Precision — Recall, или прецизионность — полнота). — *Примеч. науч. ред.*

Существует также семейство алгоритмов под названием SMOTE, которые генерируют искусственные выборки миноритарного класса. Однако, чтобы бороться с дисбалансом, лучше всего использовать модели обнаружения аномалий, которые специально разработаны для того, чтобы искать редкие события. Правда, такие модели ищут выбросы и не обязательно относятся к классификации, потому что представляют собой алгоритмы обучения без учителя. Все эти методы выходят за рамки этой книги, но их стоит упомянуть, потому что в некоторых случаях они могут обеспечить лучшее решение для конкретной задачи.

Заключение

Логистическая регрессия — одна из самых распространенных моделей для прогнозирования вероятностей и классификации данных. Логистическая регрессия может прогнозировать не только бинарный результат («Да/Нет»), но и множественные классы. Достаточно построить отдельные логистические регрессии, которые моделируют, принадлежит ли точка данных к той или иной категории, и классифицировать точку в соответствии с той моделью, которая дает наибольшую вероятность. Впрочем, библиотека scikit-learn по большей части может сделать эту работу за вас и определить, когда данные делятся более чем на два класса.

В этой главе мы рассмотрели не только то, как подогнать логистическую регрессию с помощью градиентного спуска и библиотеки scikit-learn, но и различные подходы к валидации — с точки зрения статистики и с точки зрения машинного обучения. В области статистики мы изучили R^2 и *p*-значения, а в области машинного обучения — разбиение на обучающую и тестовую выборки, матрицы ошибок и ROC-кривые.

Если вы хотите узнать больше о логистической регрессии, то, вероятно, лучше всего начать с YouTube-канала StatQuest Джоша Стармера (Josh Starmer). Я весьма признателен Джошу за то, что он помог мне написать некоторые части этой главы, особенно про то, как вычислять R^2 и *p*-значения для логистической регрессии. Эти видеоролики стоит посмотреть хотя бы ради потрясающих вступительных песен (https://oreil.ly/tueJJ)!

Как и в других разделах data science, вам придется иметь дело с обеими дисциплинами — статистикой и машинным обучением. Существует много книг и других ресурсов о логистической регрессии с точки зрения машинного обучения, но постарайтесь найти и ресурсы из области статистики. У каждой дисциплины есть свои преимущества и недостатки, и вы преуспеете только в том случае, если будете хорошо ориентироваться в обеих!

Упражнения для самопроверки

По ссылке (https://bit.ly/3imidqa) представлен набор данных с тремя входными переменными `RED`, `GREEN` и `BLUE`, которые задают цвет фона в формате RGB, а также выходной переменной `LIGHT_OR_DARK_FONT_IND`, которая прогнозирует, какой шрифт лучше подойдет для этого фона — светлый (0) или темный (1).

1. Выполните логистическую регрессию на этих данных, используя трехкратную перекрестную валидацию и точность как метрику качества модели.
2. Постройте матрицу ошибок, которая сравнивает прогнозы и фактические данные.
3. Выберите несколько разных цветов фона и проверьте, правильно ли логистическая регрессия выбирает светлый (0) или темный (1) шрифт для каждого из них. (Чтобы выбирать цвета, можно использовать онлайн-палитру, например (https://bit.ly/3FHywrZ).
4. Сделайте вывод на основе предыдущих упражнений: эффективна ли логистическая регрессия для того, чтобы прогнозировать тип шрифта для заданного цвета фона?

Ответы см. в Приложении Б.

ГЛАВА 7

Нейронные сети

В последнее десятилетие переживают ренессанс такие методы регрессии и классификации, как *нейронные сети*. В самом простом понимании нейронная сеть — это многослойная регрессия, в которой между входными и выходными переменными находятся слои весовых коэффициентов, смещений и нелинейных функций. *Глубокое обучение* — это популярная разновидность нейронных сетей, в которой используется несколько скрытых (или промежуточных) уровней узлов с весовыми коэффициентами и смещениями. Каждый узел похож на линейную функцию, которая затем поступает на вход нелинейной функции (так называемой *функции активации*). Как и в случае с линейной регрессией, которую мы изучали в главе 5, чтобы найти оптимальные значения весовых коэффициентов и смещений и тем самым минимизировать остатки, используются методы оптимизации, например стохастический градиентный спуск.

Нейронные сети предлагают впечатляющие решения проблем, с которыми компьютеры раньше не справлялись. Выполняя широкий спектр задач — от идентификации объектов на изображениях до обработки слов в аудиозаписи, — нейронные сети породили инструменты, которые меняют нашу повседневную жизнь. К ним относятся виртуальные помощники и поисковые системы, а также средства обработки фотографий в современных смартфонах.

Учитывая шумиху в прессе и громкие заявления, которыми пестрят заголовки новостей о нейронных сетях, может показаться удивительным, что они существуют еще с 1950-х годов. В 2010-х годах они внезапно стали чрезвычайно популярными из-за того, что данные начали становиться гораздо доступнее, а вычислительные мощности значительно выросли. Пожалуй, наибольший вклад в возрождение нейросетей внес проект ImageNet, который развивался в 2011–2015 годах и позволил повысить производительность классификации с тысячей категорий на материале 1,4 млн изображений, достигнув точности 96,4 %.

Однако глубокое обучение, как и любая другая разновидность машинного обучения, решает только узкоспециализированные задачи. Даже в проектах по созданию беспилотных автомобилей не используется одно только глубокое обучение, а в основном применяются системы правил, которые закодированы вручную, и сверточные нейронные сети, которые выполняют функцию «разметчика», идентифицируя объекты на дороге. Позже в этой главе мы подробнее поговорим о том, где на самом деле используются нейронные сети. Но сначала мы разработаем простую нейронную сеть на NumPy, а затем применим scikit-learn в качестве библиотечной реализации.

Когда использовать нейронные сети и глубокое обучение

Нейронные сети и глубокое обучение можно использовать для классификации и регрессии, но как отделить область их применения от задач линейной регрессии, логистической регрессии и других методов машинного обучения? Возможно, вы слышали пословицу: «Когда у вас нет ничего, кроме молотка, все вокруг начинает казаться гвоздями»[1]. У каждого семейства алгоритмов есть свои преимущества и недостатки, которые зависят от конкретной ситуации. Линейная и логистическая регрессия, а также градиентный бустинг (который не рассматривается в этой книге) отлично справляются с прогнозами на структурированных данных — то есть данных, которые легко представить в виде таблицы со строками и столбцами. Но такие задачи восприятия, как классификация изображений, гораздо менее структурированы: чтобы распознать формы и паттерны, мы пытаемся найти нечеткие корреляции между группами пикселей, а не между строками данных в таблице. Когда программа предсказывает следующие четыре или пять слов в набираемом предложении или расшифровывает текст по аудиозаписи, — это тоже задачи восприятия и примеры того, как нейронные сети используются для обработки естественного языка.

В этой главе мы сосредоточимся на простых нейронных сетях с одним скрытым слоем.

---

[1] Эта пословица восходит к цитате из книги американского психолога Абрахама Маслоу «Психология науки» (1966): «Думаю, что весьма соблазнительно, располагая из инструментов лишь одним молотком, относиться ко всему вокруг как к гвоздям». Впрочем, похожие по смыслу фразы можно встретить и в более ранних источниках. — *Примеч. науч. ред.*

РАЗНОВИДНОСТИ НЕЙРОННЫХ СЕТЕЙ

Одна из разновидностей нейросетей — сверточные нейронные сети, которые часто применяются для распознавания образов. Другая разновидность — LSTM-сети — позволяет прогнозировать временные ряды. А рекуррентные нейронные сети широко используются в приложениях, которые преобразовывают текст в речь.

Нейронная сеть — из пушки по воробьям?

Вероятно, в примере, который последует далее, нейронные сети будут излишеством; логистическая регрессия была бы практичнее. Можно даже вычислить результат по формуле (https://oreil.ly/M4W8i). Однако я всегда был сторонником того, чтобы объяснять сложные методы на примере простых задач. Это помогает понять сильные и слабые стороны метода, не отвлекаясь на большие массивы данных. Так что имейте это в виду и старайтесь не использовать нейронные сети там, где лучше подойдут более простые модели. Однако в этой главе мы нарушим это правило ради того, чтобы лучше усвоить основные принципы нейронных сетей.

Простая нейронная сеть

Вот простой пример, который поможет получить представление о нейронных сетях. Для произвольного цвета фона нужно спрогнозировать, какой шрифт к нему лучше подойдет — светлый (1) или темный (0). На рис. 7.1 приведено шесть примеров с разными цветами фона. Верхний ряд лучше всего выглядит со светлым шрифтом, а нижний — с темным.

Рис. 7.1. Темный шрифт лучше смотрится на светлом фоне, а светлый — на темном

Один из распространенных способов представить цвета в компьютере — значения RGB: это тройка чисел, которые соответствуют красному (R, red), зеленому (G, green) и синему (B, blue) компонентам цвета. Каждое число находится

в диапазоне от 0 до 255, и итоговое значение показывает, как эти три цвета смешиваются, чтобы создать желаемый цвет. Например, если представить значение RGB в формате *(красный, зеленый, синий)*, то оранжевому цвету будет соответствовать значение RGB (255, 140, 0), а розовому — (255, 192, 203). Черный цвет обозначается как (0, 0, 0), а белый — (255, 255, 255).

С точки зрения машинного обучения и регрессии цвет фона задается тремя входными переменными: `red` (красный), `green` (зеленый) и `blue` (синий). Задача состоит в том, чтобы подогнать функцию под эти аргументы и вывести, какой шрифт лучше использовать для того или иного цвета фона — светлый (1) или темный (0).

Цвета в формате RGB

В интернете есть сотни цветовых палитр, где можно выбирать цвета, чтобы поэкспериментировать со значениями RGB. W3 Schools предлагает одну из таких палитр (https://oreil.ly/T57gu).

Обратите внимание, что наш пример не так уж далек от типичной задачи нейронных сетей — распознавания изображений, потому что каждый пиксель часто моделируется как три числовых значения в формате RGB. В данном случае мы просто рассматриваем только один «пиксель», который задает цвет фона.

Давайте начнем с общей картины и на время отложим все детали реализации. Мы будем рассматривать эту тему по принципу матрешки: сперва обсудим самый внешний уровень, а затем будем постепенно погружаться в подробности. Поэтому процесс, который принимает входные значения и выдает выходные, мы пока обозначим просто как «загадочная математика». Этот процесс обрабатывает три числовые входные переменные *R* (красный), *G* (зеленый) и *B* (синий) и возвращает прогноз в интервале от 0 до 1, как показано на рис. 7.2.

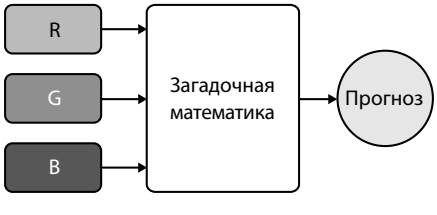


Рис. 7.2. На основе трех чисел, которые составляют значение RGB, нейронная сеть прогнозирует светлый или темный шрифт

Итоговый прогноз представляет собой вероятность. Вывод вероятностей — самый распространенный способ классификации с помощью нейронных сетей. Если заменить *R*, *G* и *B* на соответствующие числовые значения, мы увидим, что вероятность меньше 0,5 означает темный шрифт, а больше или равная 0,5 — светлый, как показано на рис. 7.3.

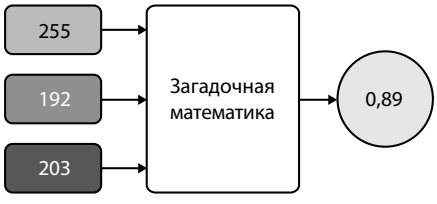


Рис. 7.3. Если задать розовый цвет фона (255, 192, 203), то загадочная математика рекомендует светлый шрифт, потому что итоговая вероятность 0,89 больше, чем 0,5

Что же происходит внутри этого черного ящика с загадочной математикой? Давайте посмотрим на рис. 7.4.

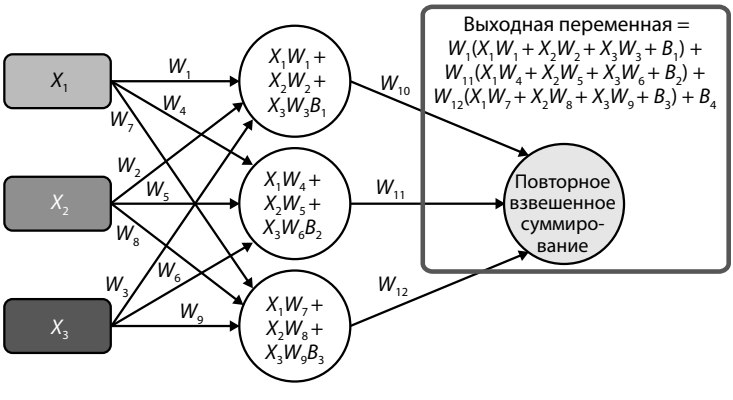


Рис. 7.4. Скрытый слой нейронной сети накладывает весовые коэффициенты и смещения на каждую входную переменную, а выходной слой накладывает другие весовые коэффициенты и смещения на вывод скрытого слоя

В этой нейронной сети не хватает еще одного компонента — функции активации, но о ней чуть позже. Давайте сначала разберемся, что здесь происходит. Первый слой слева — это просто три входные переменные, которые в данном случае представляют собой значения красной, зеленой и синей составляющих цвета. В скрытом (среднем) слое, который находится

между входами и выходом, расположены три *узла*, или весовые функции со смещением. По сути каждый узел — это линейная функция, которая умножает входные переменные X_i на весовые коэффициенты W_j и суммирует произведения друг с другом и пересечением B_k. Между каждым входным узлом и каждым скрытым узлом определен весовой коэффициент W_j, а между каждым скрытым узлом и выходным узлом — еще один весовой коэффициент. К каждому скрытому и выходному узлу добавляется смещение B_k.

Обратите внимание, что выходной узел повторяет ту же операцию: он принимает взвешенные суммы из скрытых слоев и передает их на вход в последний слой, где применяется другой набор весовых коэффициентов и смещений.

Короче говоря, это такая же регрессия, как линейная или логистическая, но в ней нужно вычислить гораздо больше параметров. Весовые коэффициенты и смещения играют ту же роль, что параметры m и b (или β_1 и β_0) в линейной регрессии. Как и в линейной регрессии, мы используем стохастический градиентный спуск и минимизируем потери, но нам понадобится дополнительный инструмент, который называется обратным распространением и позволяет отвязать друг от друга весовые коэффициенты W_j и смещения B_k и вычислить частные производные по ним с помощью цепного правила. Об этом мы поговорим позже в этой главе, а пока давайте предположим, что весовые коэффициенты и смещения уже оптимизированы. Сначала обсудим функции активации.

Функции активации

Функция активации — это обычно нелинейная функция, которая преобразует или уплотняет взвешеную сумму в узле, помогая нейронной сети эффективно разделять данные, чтобы их можно было классифицировать. Давайте посмотрим на рис. 7.5. Если бы не было функций активации, то скрытые слои функционировали бы неэффективно, и нейросеть была бы не лучше линейной регрессии.

Функция активации ReLU (линейный выпрямитель) обнуляет все отрицательные выходные данные скрытых узлов. Если взвешенная сумма входных переменных со смещением оказывается отрицательной, она преобразуется в 0, иначе выходное значение остается без изменений. На рис. 7.6 приведен график функции активации ReLU, построенный с помощью библиотеки SymPy (см. пример 7.1).

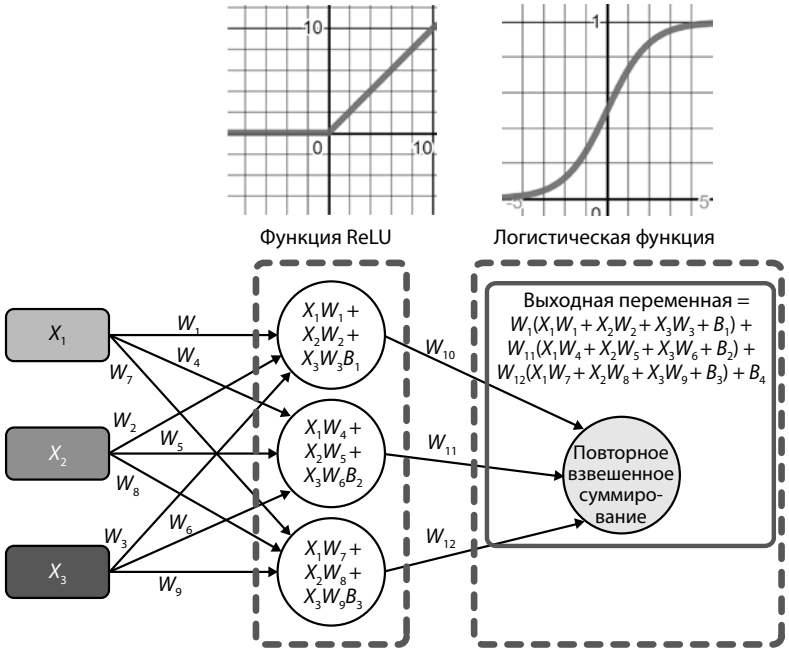


Рис. 7.5. Применение функций активации

Пример 7.1. График функции ReLU

```
from sympy import *

# Строим график функции ReLU
x = symbols('x')
relu = Max(0, x)
plot(relu)
```

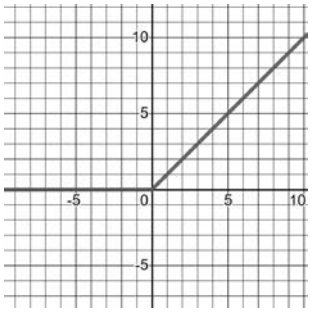


Рис. 7.6. График функции ReLU

Название ReLU происходит из электротехники, где оно расшифровывается как «rectified linear unit» (линейный выпрямитель). Однако в контексте машинного обучения это просто особый способ сказать «обратить отрицательные значения в 0». Функция ReLU стала популярна для скрытых слоев в нейронных сетях и глубоком обучении благодаря тому, что она работает быстро и помогает бороться с проблемой исчезающего градиента (https://oreil.ly/QGlM7). Исчезающие градиенты возникают, когда угловые коэффициенты частных производных становятся настолько малыми, что преждевременно достигают нуля, и обучение впадает в ступор.

Выходной слой выполняет важную работу: он агрегирует множество математических выводов из скрытых слоев нейронной сети и преобразует их в интерпретируемый результат — например, вырабатывает прогнозы классификации. В нашем примере в выходном слое нейронной сети используется *логистическая функция активации*, которая представляет собой простую сигмоиду. Если вы читали главу 6, логистическая функция должна быть вам знакома, и вы легко убедитесь, что логистическая регрессия действует как слой в нашей нейронной сети. Выходной узел вычисляет взвешенную сумму всех значений, которые поступают из скрытого слоя, и применяет к ней смещение. После этого он пропускает полученное значение через логистическую функцию, чтобы на выходе получилось число от 0 до 1. Как и в логистической регрессии в главе 6, это число — вероятность того, что к цвету фона, который введен в нейронную сеть, подходит светлый шрифт. Если эта вероятность больше или равна 0,5, то нейросеть рекомендует светлый шрифт, а если меньше, то темный.

На рис. 7.7 приведен график логистической функции, построенный с помощью библиотеки SymPy (см. пример 7.2).

Пример 7.2. График логистической функции активации

```
from sympy import *

# Строим график логистической функции активации
x = symbols('x')
logistic = 1 / (1 + exp(-x))
plot(logistic)
```

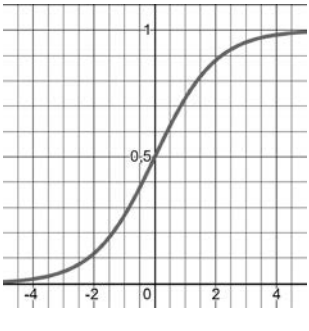


Рис. 7.7. График логистической функции активации

Обратите внимание, что после того как значение узла (взвешенная сумма со смещением) проходит через функцию активации, его можно называть *активированным выходом*: это значит, что он отфильтрован через эту функцию. Когда активированный выход покидает скрытый слой, сигнал готов к подаче на следующий слой. Функция активации могла усилить сигнал, ослабить его или оставить как есть. С этим поведением связана метафора, которая уподобляет нейронные сети биологической нервной системе[1].

Заметив, что перечисленные функции довольно просты, вы можете спросить, существуют ли другие функции активации. Наиболее распространенные из них приведены в таблице 7.1.

Таблица 7.1. Распространенные функции активации

Название	Типовой слой	Описание	Примечания
Линейная	Выходной	Оставляет значения как есть	Обычно не используется
Логистическая	Выходной	Сигмоида	Сжимает значения в интервал от 0 до 1, часто применяется в бинарной классификации
Гиперболический тангенс	Скрытый	Подобен сигмоиде; возвращает значения в диапазоне от −1 до 1	Помогает «центрировать» данные, приближая среднее к 0
ReLU (линейный выпрямитель)	Скрытый	Обращает отрицательные значения в 0	Популярная функция активации: работает быстрее сигмоиды и гиперболического тангенса, снижает проблему исчезающего градиента и потребляет меньше вычислительных ресурсов
ReLU с «утечкой»	Скрытый	Умножает отрицательные значения на малый коэффициент (например, 0,01)	Сомнительный вариант ReLU, который не обнуляет, а уменьшает отрицательные значения
Softmax (многопеременная логистическая функция)	Выходной	Обеспечивает, чтобы сумма всех выходных узлов была равна 1	Используется для многомерной классификации и масштабирует результаты так, чтобы они в сумме давали 1

[1] Нейронные сети называются так потому, что опираются на математические модели, которые изначально имитировали поведение биологических нейронов (нервных клеток). Впрочем, эти модели никогда не претендовали на то, чтобы воспроизводить природные системы с исчерпывающей точностью. Современные нейросети обогатились многими качествами, которые полезны для прикладных задач, однако еще меньше соответствуют биологической метафоре. Некоторые авторы впадают в заблуждение, когда по поведению компьютерных нейронных сетей судят о том, как устроен головной мозг, и наоборот. — *Примеч. науч. ред.*

Это неполный список функций активации, и теоретически в нейронной сети любая функция может быть функцией активации.

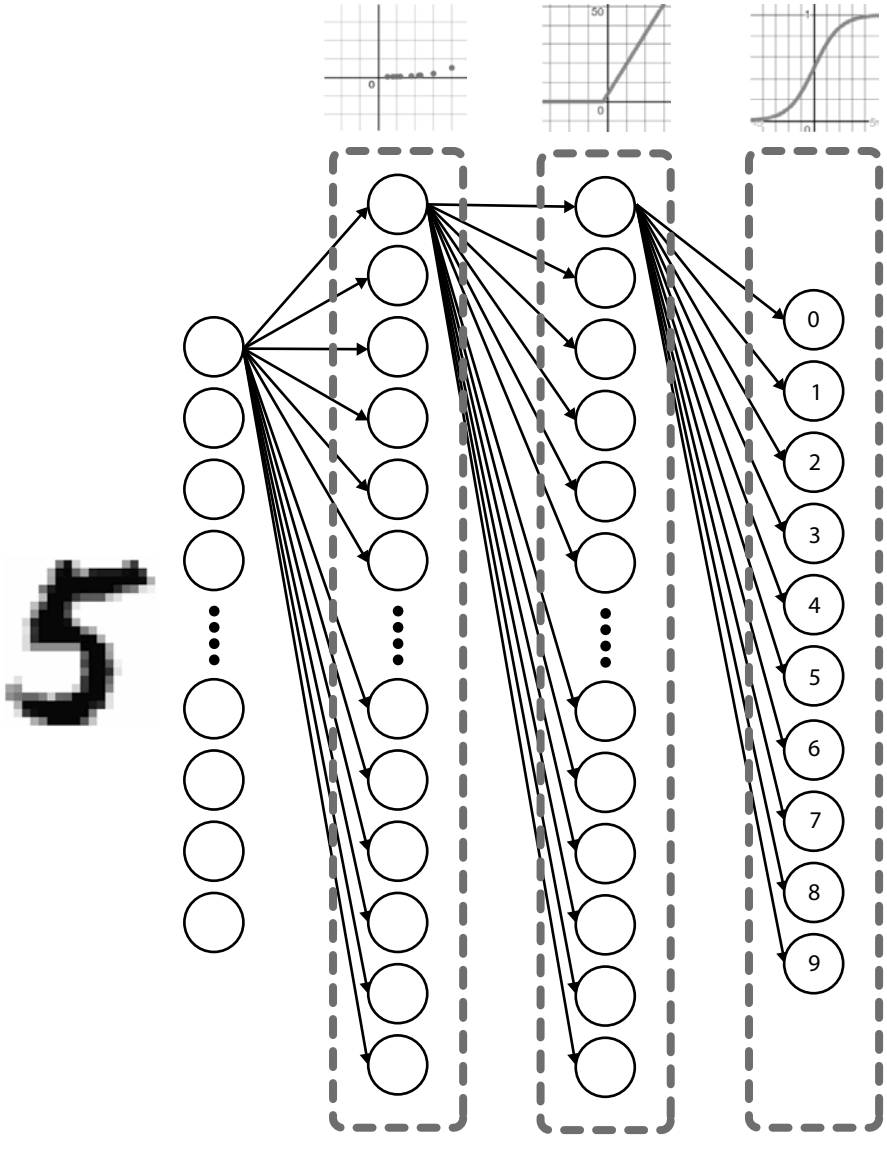


Рис. 7.8. Нейронная сеть, которая принимает каждый пиксель в качестве входного сигнала и прогнозирует, какую цифру содержит изображение

Хотя создается впечатление, что наша нейронная сеть различает два класса (светлый или темный шрифт), на самом деле она моделирует один класс: должен ли шрифт быть светлым (1) или нет (0). Если нужно поддерживать несколько классов, для каждого из них можно добавить соответствующий выходной узел. Например, если вы пытаетесь распознавать рукописные цифры от 0 до 9, то в нейронной сети понадобится 10 выходных узлов, каждый из которых представляет вероятность того, что на изображении находится то или иное число. Когда у вас несколько классов, в качестве выходной функции активации можно также попробовать Softmax (многопеременную логистическую функцию). На рис. 7.8 показан пример, где оцифрованное изображение рукописной цифры попиксельно разбивается на отдельные входы нейронной сети и проходит через два промежуточных слоя, а затем через выходной слой, который состоит из 10 узлов — по одному на вероятность для каждого из 10 классов (цифр от 0 до 9).

В Приложении А приведен пример того, как в нейронной сети используется набор данных MNIST.

Как выбрать функцию активации?

Если вы не знаете, какую функцию активации использовать, то на сегодняшний день лучший вариант — ReLU для промежуточных (скрытых) слоев и логистическая функция (сигмоида) для выходного слоя. Если на выходе у вас несколько классов, используйте Softmax на выходном слое.

Прямое распространение

Давайте соберем воедино все, что мы уже научились делать с помощью scikit-learn. Обратите внимание, что мы пока не оптимизировали параметры (весовые коэффициенты и смещения). Для начала мы инициализируем их случайными значениями.

В примере 7.3 показан код на Python, который создает простую нейронную сеть прямого распространения, причем она пока не оптимизирована. *Прямое распространение* означает, что мы просто вводим цвет в нейронную сеть и смотрим, что она выведет. Весовые коэффициенты и смещения задаются случайным образом. Мы будем оптимизировать их позже в этой главе, так что пока от нейросети не стоит ожидать больших успехов.

Пример 7.3. Простая нейронная сеть прямого распространения со случайными весовыми коэффициентами и смещениями

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
all_data = pd.read_csv("https://tinyurl.com/y2qmhfsr")

# Извлекаем входные столбцы, масштабируем данные делением на 255
all_inputs = (all_data.iloc[:, 0:3].values / 255.0)
all_outputs = all_data.iloc[:, -1].values

# Разделяем данные на обучающую и тестовую выборки
X_train, X_test, Y_train, Y_test = train_test_split(all_inputs, all_outputs,
    test_size=1/3)
n = X_train.shape[0] # количество записей в обучающей выборке

# Конструируем нейронную сеть,
# инициализируем весовые коэффициенты и смещения случайными числами
w_hidden = np.random.rand(3, 3)
w_output = np.random.rand(1, 3)

b_hidden = np.random.rand(3, 1)
b_output = np.random.rand(1, 1)

# Функции активации
relu = lambda x: np.maximum(x, 0)
logistic = lambda x: 1 / (1 + np.exp(-x))

# Пропускаем входные данные через нейронную сеть,
# чтобы на выходе получить прогнозы
def forward_prop(X):
    Z1 = w_hidden @ X + b_hidden
    A1 = relu(Z1)
    Z2 = w_output @ A1 + b_output
    A2 = logistic(Z2)
    return Z1, A1, Z2, A2

# Вычисляем точность классификации
test_predictions = forward_prop(X_test.transpose())[3] # берем только выходной
слой (A2)
test_comparisons = np.equal((test_predictions >= .5).flatten().astype(int), Y_
test)
accuracy = sum(test_comparisons.astype(int) / X_test.shape[0])
print("Точность: ", accuracy)
```

Здесь стоит отметить несколько моментов. Набор данных, который содержит входные значения RGB, а также выходные значения (1 — светлый шрифт и 0 — темный), находится в файле CSV по адресу https://oreil.ly/1TZIK. Программа уменьшает входные столбцы R, G и B в 255 раз, чтобы их значения были между 0 и 1. Если сжать пространство чисел, это поможет в дальнейшем обучать нейросеть.

Обратите внимание, что мы также выделили $\frac{2}{3}$ данных в обучающую выборку и $\frac{1}{3}$ — в тестовую с помощью библиотеки scikit-learn (мы изучали этот прием в главе 5). Число `n` — это просто количество записей в обучающей выборке.

Теперь обратите внимание на строки кода, которые показаны в примере 7.4.

Пример 7.4. Матрицы весовых коэффициентов и векторы смещения, заданные с помощью NumPy

```
# Конструируем нейронную сеть,
# инициализируем весовые коэффициенты и смещения случайными числами
w_hidden = np.random.rand(3, 3)
w_output = np.random.rand(1, 3)

b_hidden = np.random.rand(3, 1)
b_output = np.random.rand(1, 1)
```

Здесь объявляются весовые коэффициенты и смещения для скрытого и выходного слоев нейронной сети. Возможно, это пока не очевидно, но умножение матриц радикально упростит наш код благодаря линейной алгебре и библиотеке NumPy.

Весовые коэффициенты и смещения инициализируются как случайные значения в интервале от 0 до 1. Для начала рассмотрим матрицы весовых коэффициентов. Запустив код, я получил такие матрицы:

$$W_{скрытый} = \begin{bmatrix} 0,034535 & 0,5185636 & 0,81485028 \\ 0,3329199 & 0,53873853 & 0,96359003 \\ 0,19808306 & 0,45422182 & 0,36618893 \end{bmatrix}$$

$$W_{выходной} = \begin{bmatrix} 0,82652072 & 0,30781539 & 0,93095565 \end{bmatrix}$$

Обратите внимание, что $W_{скрытый}$ — это весовые коэффициенты в скрытом слое. Первая строка матрицы соответствует первому узлу с коэффициентами W_1, W_2 и W_3, вторая строка — второму узлу с коэффициентами W_4, W_5 и W_6, а третья строка — третьему узлу с коэффициентами W_7, W_8 и W_9.

В выходном слое только один узел, а значит, его матрица состоит всего из одной строки с весовыми коэффициентами W_{10}, W_{11} и W_{12}.

Видите, в чем тут закономерность? Каждый узел представлен в виде строки в матрице. Если в слое три узла, то в его матрице три строки. Если узел один, то строка одна. В каждом столбце содержатся весовые коэффициенты для каждого из узлов предыдущего слоя, входящих по отношению к данному узлу.

Давайте посмотрим и на смещения. Поскольку на каждый узел приходится по одному смещению, в матрицах $B_{скрытый}$ и $B_{выходной}$ будет всего по одному столбцу. В первой матрице будет три строки смещений для скрытого слоя, а во второй — одна строка смещений для выходного слоя.

$$B_{скрытый} = \begin{bmatrix} 0,41379442 \\ 0,81666079 \\ 0,07511252 \end{bmatrix}$$

$$B_{выходной} = [0,58018555]$$

Давайте сопоставим эти матрицы с визуализацией нашей нейронной сети, как показано на рис. 7.9.

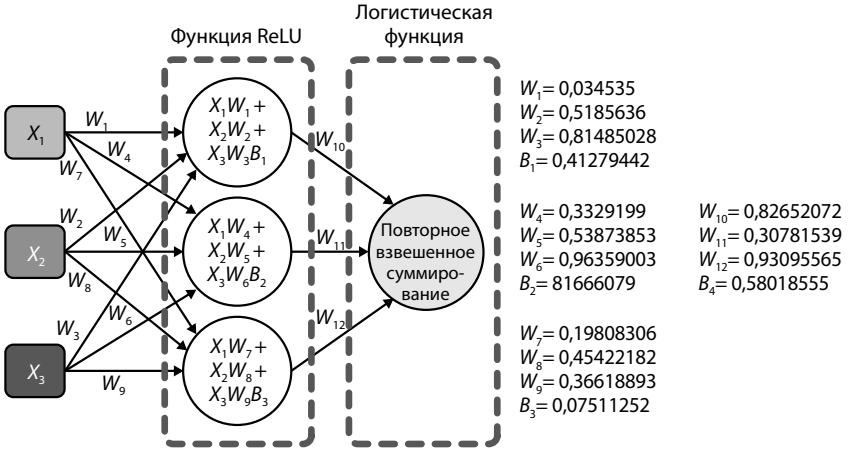


Рис. 7.9. Визуализация нейронной сети со значениями весовых коэффициентов и смещений

Итак, помимо того, что матрицы выглядят предельно компактно, в чем преимущество того, чтобы представлять весовые коэффициенты и смещения в матричной форме? Обратите внимание на следующие строки кода в примере 7.5.

Пример 7.5. Функции активации и прямого распространения для нейронной сети

```
# Функции активации
relu = lambda x: np.maximum(x, 0)
logistic = lambda x: 1 / (1 + np.exp(-x))

# Пропускаем входные данные через нейронную сеть,
# чтобы на выходе получить прогнозы
def forward_prop(X):
    Z1 = w_hidden @ X + b_hidden
    A1 = relu(Z1)
    Z2 = w_output @ A1 + b_output
    A2 = logistic(Z2)
    return Z1, A1, Z2, A2
```

Этот код особенно интересен, потому что он лаконично реализует всю нашу нейронную сеть с помощью умножения матриц на другие матрицы и векторы — об этих операциях рассказывалось в главе 4. Всего в нескольких строках кода цвет фона в виде значения RGB из трех входов проходит через весовые коэффициенты, смещения и функции активации.

Сначала здесь объявляются функции активации `relu()` и `logistic()`, которые просто принимают заданное входное значение и возвращают соответствующее выходное. Функция `forward_prop()` реализует всю нейронную сеть для заданного входа *X*, который содержит цвет фона в виде значений `R`, `G` и `B`. Нейронная сеть возвращает выходные значения (матрицы, векторы или числа) для четырех этапов: `Z1`, `A1`, `Z2` и `A2`. Цифры «1» и «2» показывают, что операции относятся к слоям 1 и 2 соответственно. «Z» означает неактивированный выход слоя, а «A» — активированный.

Скрытый слой представлен вектором `Z1` и матрицей `A1`. `Z1` — это весовые коэффициенты и смещения, которые применяются к `X`. Затем `A1` принимает выход из `Z1` и пропускает его через функцию активации ReLU. `Z2` принимает выход из `A1` и применяет весовые коэффициенты и смещения выходного слоя. В свою очередь, этот выход пропускается через функцию активации (логистическую кривую) и превращается в `A2`. Последний этап `A2` — это прогнозируемая вероятность на выходном слое, которая возвращает значения между 0 и 1. Мы называем его `A2`, потому что это активированный выход слоя 2.

Давайте разберемся во всем этом подробнее, начиная с вектора `Z1`:

$$Z_1 = W_{скрытый} + B_{скрытый}$$

Сначала мы умножаем матрицу $W_{скрытый}$ на вектор входного цвета `X`, который содержит красный, зеленый и синий компоненты цвета. Каждая строка $W_{скрытый}$ представляет собой набор весовых коэффициентов для узла, и мы скалярно умножаем каждую строку на `X`. Затем к этому результату добавляются смещения, как показано на рис. 7.10.

$$Z_1 = W_{скрытый} + B_{скрытый}$$

$$Z_1 = \begin{bmatrix} 0,034535 & 0,5185636 & 0,81485028 \\ 0,3329199 & +0,53873853 & +0,96359003 \\ 0,19808306 & 0,45422182 & 0,36618893 \end{bmatrix} \begin{bmatrix} 0,82652072 \\ 0,30781539 \\ 0,93095565 \end{bmatrix} + \begin{bmatrix} 0,41379442 \\ 0,81666079 \\ 0,07511252 \end{bmatrix}$$

$$Z_1 = \begin{bmatrix} 0,946755221909086 \\ 1,33805678888247 \\ 0,644441873391768 \end{bmatrix} + \begin{bmatrix} 0,41379442 \\ 0,81666079 \\ 0,07511252 \end{bmatrix}$$

$$Z_1 = \begin{bmatrix} 1,36054964190909 \\ 2,15471757888247 \\ 0,719554393391768 \end{bmatrix}$$

Рис. 7.10. Весовые коэффициенты и смещения скрытого слоя применяются ко входу *X* с помощью матрично-векторного умножения, а также векторного сложения

Вектор Z_1 — это необработанный выход скрытого слоя, который нужно провести через функцию активации, чтобы превратить в A_1. Это делается довольно просто: нужно применить функцию ReLU к каждой координате вектора Z_1, и получится A_1. Поскольку все координаты положительные, вектор останется без изменений.

$$A_1 = ReLU(Z_1) = \begin{bmatrix} ReLU(1,36054964190909) \\ ReLU(2,15471757888247) \\ ReLU(0,719554393391768) \end{bmatrix} = \begin{bmatrix} 1,36054964190909 \\ 2,15471757888247 \\ 0,719554393391768 \end{bmatrix}$$

Теперь возьмем выход скрытого слоя A_1 и пропустим его через последний слой, чтобы получить Z_2, а затем A_2. A_1 станет входом выходного слоя.

$$Z_2 = W_{выходной} A_1 + B_{выходной} =$$

$$= \begin{bmatrix} 0,82652072 & 0,3078159 & 0,93095565 \end{bmatrix} \begin{bmatrix} 1,36054964190909 \\ 2,15471757888247 \\ 0,719554393391768 \end{bmatrix} + \begin{bmatrix} 0,58018555 \end{bmatrix} =$$

$$= \begin{bmatrix} 2,45765202842636 \end{bmatrix} + \begin{bmatrix} 0,58018555 \end{bmatrix} = \begin{bmatrix} 3,03783757842636 \end{bmatrix}$$

Наконец, пропустим этот одномерный вектор Z_2 через функцию активации, чтобы получить A_2. В результате получится прогнозируемое значение, равное примерно 0,95425:

$$A_2 = logistic(Z_2) = logistic(\begin{bmatrix} 3,0378364795204 \end{bmatrix}) = 0,954254478103241$$

В этом примере задействована вся нейронная сеть, хотя мы ее еще не обучали. Тем не менее уделите время тому, чтобы убедиться, что она принимает все входные значения, применяет к ним весовые коэффициенты, смещения и нелинейные функции и вырабатывает одно значение, которое и служит прогнозом.

Повторимся, `A2` — это конечный результат, который прогнозирует, какой шрифт подходит к этому цвету фона — светлый (1) или темный (0). Хотя весовые коэффициенты и смещения еще не оптимизированы, давайте рассчитаем точность, как показано в примере 7.6. Возьмем тестовую выборку `X_test`, транспонируем ее и пропустим через функцию `forward_prop()`, однако из полученной матрицы извлечем только вектор `A2` с прогнозами для каждого цвета из тестовой выборки. Затем сравним прогнозы с фактическими данными и вычислим долю правильных прогнозов.

Пример 7.6. Вычисление точности нейронной сети

```
# Вычисляем точность классификации
test_predictions = forward_prop(X_test.transpose())[3]
# берем только выходной слой (A2)
```

```
test_comparisons = np.equal((test_predictions >= .5).flatten().astype(int), Y_test)
accuracy = sum(test_comparisons.astype(int) / X_test.shape[0])
print("Точность: ", accuracy)
```

Когда я запускал полный код из примера 7.3, точность оказывалась в интервале от 55 до 67 %. Помните, что весовые коэффициенты и смещения генерируются случайным образом, поэтому от запуска к запуску результаты могут различаться. Хотя с учетом случайных параметров точность может показаться высокой, не забывайте, что на выходе получается бинарный прогноз: светлый или темный шрифт. Даже если случайно подбрасывать монету, это вполне может привести к аналогичному результату, так что пусть такая точность вас не удивляет.

Не забывайте проверять данные на дисбаланс!

Как уже говорилось в главе 6, не забывайте анализировать данные на предмет дисбаланса классов. В нашем примере набор данных с цветами фона немного несбалансирован: у 512 записей выходное значение равно 0, а у 833 — 1. Это может сместить точность, и, возможно, поэтому, несмотря на случайные весовые коэффициенты и смещения, она получается выше 50 %. Если данные чрезвычайно несбалансированы (например, 99 % данных относятся к одному и тому же классу), не забывайте строить матрицы ошибок, чтобы отслеживать ложноположительные и ложноотрицательные прогнозы.

Хорошо ли вам удалось усвоить весь предыдущий материал? Возможно, прежде чем двигаться дальше, стоит повторить все, что вы узнали до этого. Нам осталось сделать последний шаг: оптимизировать весовые коэффициенты и смещения. Приготовьте кофе покрепче, потому что впереди вас ждет математика, которая будет самой сложной из всего, чем мы занимаемся в этой книге!

Обратное распространение

Прежде чем оптимизировать нейронную сеть методом стохастического градиентного спуска, нам предстоит выяснить, как регулировать каждый весовой коэффициент и каждое смещение с учетом того, что все они совместно участвуют в выработке выходной переменной, на основе которой затем вычисляются остатки. Как найти производную от функции потерь по каждой переменной весового коэффициента W_i и смещения B_j? В этом нам поможет цепное правило (правило дифференцирования сложной функции), которое рассматривалось в главе 1.

Как вычислять производные по весовым коэффициентам и смещениям

У нас еще не все готово для того, чтобы обучать нейронную сеть с помощью стохастического градиентного спуска. Нужно получить частные производные по весовым коэффициентам W_i и смещениям B_j, и здесь пригодится цепное правило.

Хотя стохастический градиентный спуск в целом выполняется так же, как в главе 5, в нейронных сетях его применение сопряжено с некоторыми сложностями. Узлы каждого слоя, кроме выходного, передают свои весовые коэффициенты и смещения в следующий слой, который затем применяет другой набор весовых коэффициентов и смещений. Таким образом, получается структура с несколькими уровнями вложенности, которую нужно распутать, начиная с выходного слоя.

В процессе градиентного спуска необходимо выяснить, какие весовые коэффициенты и смещения и на сколько следует скорректировать, чтобы уменьшить общую функцию потерь. Потери одного прогноза будут равны квадрату разности выхода нейронной сети A_2 и фактического значения Y:

$$C = \left(A_2 - Y\right)^2$$

Но давайте отступим на один слой назад. Активированный выход A_2 — это просто значение Z_2, к которому применена функция активации:

$$A_2 = sigmoid\left(Z_2\right)$$

В свою очередь, Z_2 получается в результате того, что выходные весовые коэффициенты и смещения применяются к активированному выходу A_1, который поступает из скрытого слоя[1]:

$$Z_2 = W_2 A_1 + B_2$$

A_1 получается из вектора Z_1, который проходит через функцию активации ReLU:

$$A_1 = ReLU\left(Z_1\right)$$

Наконец, Z_1 — это входные значения X, к которым применяются весовые коэффициенты и смещения скрытого слоя:

$$Z_1 = W_1 X + B_1$$

Нужно найти такие весовые коэффициенты и смещения (то есть матрицы и векторы W_1, B_1, W_2 и B_2), которые минимизируют потери. Чтобы изменить

[1] С этого момента автор обозначает буквами и не отдельные числа, как раньше, а матрицы и векторы весовых коэффициентов и смещений. А именно: $W_1 = W_{скрытый}$ = `w_hidden`, $B_1 = B_{скрытый}$ = `b_hidden`, $W_2 = W_{выходной}$ = `w_output`, $B_2 = W_{выходной}$ = `b_output`. Соответственно, производные вычисляются по этим матричным и векторным переменным. — *Примеч. науч. ред.*

весовые коэффициенты и смещения, которые в наибольшей степени помогают уменьшить потери, можно регулировать угловые коэффициенты по этим переменным. Однако каждая небольшая коррекция веса или смещения будет распространяться по всему пути к функции потерь на внешнем слое. Здесь и пригодится цепное правило, с помощью которого мы разберемся, как именно параметры влияют на потери.

Давайте сперва найдем зависимость между весовым коэффициентом W_2 выходного слоя и функцией потерь C. Если изменить W_2, от этого изменится неактивированный выход Z_2. В свою очередь, это повлияет на активированный выход A_2 и в итоге на функцию потерь C. По цепному правилу производную от C по W_2 можно представить так:

$$\frac{dC}{dW_2} = \frac{dZ_2}{dW_2}\frac{dA_2}{dZ_2}\frac{dC}{dA_2}$$

Если перемножить эти три градиента, мы узнаем, в какой степени меняется значение функции потерь C при изменении W_2.

Теперь вычислим эти три производные. Давайте воспользуемся SymPy, чтобы найти производную функции потерь по A_2 (см. пример 7.7).

$$\frac{dC}{dA_2} = 2A_2 - 2y$$

Пример 7.7. Вычисление производной от функции потерь по A_2

```
from sympy import *

A2, Y = symbols('A2 Y')
C = (A2 - Y)**2
dC_dA2 = diff(C, A2)
print(dC_dA2) # 2*A2 - 2*Y
```

Далее найдем производную от A_2 по Z_2 (см. пример 7.8). Не забывайте, что A_2 — это выход функции активации, в данном случае логистической функции. Таким образом, мы просто берем производную сигмоиды.

$$\frac{dA_2}{dZ_2} = \frac{e^{-Z_2}}{\left(1+e^{-Z_2}\right)^2}$$

Пример 7.8. Вычисление производной от A_2 по Z_2

```
from sympy import *

Z2 = symbols('Z2')

logistic = lambda x: 1 / (1 + exp(-x))
```

```
A2 = logistic(Z2)
dA2_dZ2 = diff(A2, Z2)
print(dA2_dZ2) # exp(-Z2)/(1 + exp(-Z2))**2
```

Производная от Z_2 по W_2 будет равна A_1, потому что Z_2 — просто линейная функция, и ее производная возвращает угловой коэффициент (пример 7.9).

$$\frac{dZ_2}{dW_2} = A_1$$

Пример 7.9. Вычисление производной от Z_2 по W_2

```
from sympy import *

A1, W2, B2 = symbols('A1, W2, B2')

Z2 = A1*W2 + B2
dZ2_dW2 = diff(Z2, W2)
print(dZ2_dW2) # A1
```

Если свести все это воедино, то получится производная, которая позволяет определить, насколько изменение весового коэффициента W_2 влияет на функцию потерь C:

$$\frac{dC}{dW_2} = \frac{dZ_2}{dW_2}\frac{dA_2}{dZ_2}\frac{dC}{dA_2} = A_1\left(\frac{e^{-Z_2}}{\left(1+e^{-Z_2}\right)^2}\right)\left(2A_2 - 2y\right)$$

Если подать на вход нейронной сети вектор X с тремя компонентами R, G и B, мы получим значения A_1, A_2, Z_2 и y.

Не заблудитесь в формулах!

На этом этапе легко запутаться в формулах и перестать понимать, чего мы пытаемся добиться в первую очередь: найти производную функции потерь по весовому коэффициенту (W_2) в выходном слое. Если вы обнаружите, что запутались и забыли, ради чего затеяли все эти вычисления, — сделайте паузу, прогуляйтесь, выпейте кофе и напомните себе, в чем состояла исходная задача. Если вдруг это не удается, начните все заново и повторно доберитесь до того места, где заблудились.

Однако производная от функции потерь по W_2 — это лишь один компонент нейронной сети. В примере 7.10 показано, как с помощью SymPy найти остальные частные производные, которые понадобятся, чтобы применить цепное правило.

Пример 7.10. Вычисление всех частных производных, которые понадобятся, чтобы построить нейронную сеть

```
from sympy import *

W1, W2, B1, B2, A1, A2, Z1, Z2, X, Y = \
    symbols('W1 W2 B1 B2 A1 A2 Z1 Z2 X Y')
```

```
# Производная от функции потерь по A2
C = (A2 - Y)**2
dC_dA2 = diff(C, A2)
print("dC_dA2 = ", dC_dA2) # 2*A2 - 2*Y

# Производная от A2 по Z2
logistic = lambda x: 1 / (1 + exp(-x))
_A2 = logistic(Z2)
dA2_dZ2 = diff(_A2, Z2)
print("dA2_dZ2 = ", dA2_dZ2) # exp(-Z2)/(1 + exp(-Z2))**2

# Производная от Z2 по A1
_Z2 = A1*W2 + B2
dZ2_dA1 = diff(_Z2, A1)
print("dZ2_dA1 = ", dZ2_dA1) # W2

# Производная от Z2 по W2
dZ2_dW2 = diff(_Z2, W2)
print("dZ2_dW2 = ", dZ2_dW2) # A1

# Производная от Z2 по B2
dZ2_dB2 = diff(_Z2, B2)
print("dZ2_dB2 = ", dZ2_dB2) # 1

# Производная от A1 по Z1
relu = lambda x: Max(x, 0)
_A1 = relu(Z1)
d_relu = lambda x: x > 0
    # Угловой коэффициент равен 1, если x > 0, иначе 0
dA1_dZ1 = d_relu(Z1)
print("dA1_dZ1 = ", dA1_dZ1) # Z1 > 0

# Производная от Z1 по W1
_Z1 = X*W1 + B1
dZ1_dW1 = diff(_Z1, W1)
print("dZ1_dW1 = ", dZ1_dW1) # X

# Производная от Z1 по B1
dZ1_dB1 = diff(_Z1, B1)
print("dZ1_dB1 = ", dZ1_dB1) # 1
```

Обратите внимание, что производная от функции ReLU вычислена вручную, а не с помощью функции `diff()` из библиотеки SymPy. Это связано с тем, что производные SymPy работают с гладкими кривыми, а не с такими угловатыми, как ReLU. Но это легко обойти: можно просто объявить угловой коэффициент равным 1 для положительных чисел и 0 — для остальных. Это логично, потому что в области отрицательных чисел функция ReLU представляет собой горизонтальную линию с угловым коэффициентом 0, а в области положительных чисел — тождественную функцию с угловым коэффициентом 1.

Эти частные производные можно объединять в цепочки, чтобы найти новые частные производные по весовым коэффициентам и смещениям. Давайте выпишем все четыре частные производные от функции потерь по весовым коэффициентам W_1 и W_2 и смещениям B_1 и B_2. Ранее мы уже рассмотрели производную $\frac{dC}{dW_2}$. Продемонстрируем ее вместе с тремя другими нужными производными, которые получены по цепному правилу:

$$\frac{dC}{dW_2} = \frac{dZ_2}{dW_2}\frac{dA_2}{dZ_2}\frac{dC}{dA_2} = A_1\left(\frac{e^{-Z_2}}{\left(1+e^{-Z_2}\right)^2}\right)\left(2A_2 - 2y\right)$$

$$\frac{dC}{dB_2} = \frac{dZ_2}{dB_2}\frac{dA_2}{dZ_2}\frac{dC}{dA_2} = 1\times\left(\frac{e^{-Z_2}}{\left(1+e^{-Z_2}\right)^2}\right)\left(2A_2 - 2y\right)$$

$$\frac{dC}{dW_1} = \frac{dC}{dA_2}\frac{dA_2}{dZ_2}\frac{dZ_2}{dA_1}\frac{dA_1}{dZ_1}\frac{dZ_1}{dW_1} = \left(2A_2 - 2y\right)\left(\frac{e^{-Z_2}}{\left(1+e^{-Z_2}\right)^2}\right)W_2 Z_1 X\left(\text{при } Z_1 > 0\right)$$

$$\frac{dC}{dB_1} = \frac{dC}{dA_2}\frac{dA_2}{dZ_2}\frac{dZ_2}{dA_1}\frac{dA_1}{dZ_1}\frac{dZ_1}{dB_1} = \left(2A_2 - 2y\right)\left(\frac{e^{-Z_2}}{\left(1+e^{-Z_2}\right)^2}\right)W_2 Z_1 \times 1\left(\text{при } Z_1 > 0\right)$$

Далее с помощью этих градиентов мы будем вычислять угловые коэффициенты функции потерь C по W_1, B_1, W_2 и B_2.

АВТОМАТИЧЕСКОЕ ДИФФЕРЕНЦИРОВАНИЕ

Как видите, вычислять производные даже с помощью цепного правила и символьных библиотек вроде SymPy все еще довольно трудоемко. Поэтому появляются библиотеки дифференциального программирования, такие как JAX (https://oreil.ly/N96Pk), которую разработал Google. Она практически идентична NumPy, но позволяет вычислять производные по параметрам, которые упакованы в виде матриц.

Если вы хотите узнать больше об автоматическом дифференцировании, то видео по этой ссылке (https://youtu.be/wG_nF1awSSY) отлично объясняет его суть.

Стохастический градиентный спуск

Теперь все готово для того, чтобы выполнить стохастический градиентный спуск, применяя цепное правило. Чтобы не усложнять задачу, мы на каждой

итерации будем использовать обучающую выборку только из одной записи. На практике в нейронных сетях и глубоком обучении широко применяется пакетный и мини-пакетный градиентный спуск, но здесь и без того хватает непростой линейной алгебры и математического анализа, так что мы обойдемся одной выборкой за итерацию.

Давайте посмотрим на пример 7.11, где приведена полная реализация нашей нейронной сети с обратным распространением и стохастическим градиентным спуском.

Пример 7.11. Реализация нейронной сети с помощью стохастического градиентного спуска

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

all_data = pd.read_csv("https://tinyurl.com/y2qmhfsr")

# Скорость обучения определяет, насколько быстро мы приближаемся к решению.
# Если скорость слишком низкая, нейросеть будет работать чересчур долго.
# Если скорость слишком высокая, нейросеть может промахнуться и не найти решения.
L = 0.05

# Извлекаем входные столбцы, масштабируем данные делением на 255
all_inputs = (all_data.iloc[:, 0:3].values / 255.0)
all_outputs = all_data.iloc[:, -1].values

# Разделяем данные на обучающую и тестовую выборки
X_train, X_test, Y_train, Y_test = train_test_split(all_inputs, all_outputs,
test_size=1 / 3)
n = X_train.shape[0]

# Конструируем нейронную сеть,
# инициализируем весовые коэффициенты и смещения случайными числами
w_hidden = np.random.rand(3, 3)
w_output = np.random.rand(1, 3)

b_hidden = np.random.rand(3, 1)
b_output = np.random.rand(1, 1)

# Функции активации
relu = lambda x: np.maximum(x, 0)
logistic = lambda x: 1 / (1 + np.exp(-x))

def forward_prop(X):
    """Прогоняет входные переменные через нейронную сеть,
       чтобы на выходе получить прогноз
    """
    Z1 = w_hidden @ X + b_hidden
    A1 = relu(Z1)
    Z2 = w_output @ A1 + b_output
```

```
    A2 = logistic(Z2)
    return Z1, A1, Z2, A2

# Производные функций активации
d_relu = lambda x: x > 0
d_logistic = lambda x: np.exp(-x) / (1 + np.exp(-x)) ** 2

def backward_prop(Z1, A1, Z2, A2, X, Y):
    """Возвращает угловые коэффициенты по весовым коэффициентам
       и смещениям, используя цепное правило
    """
    dC_dA2 = 2 * A2 - 2 * Y
    dA2_dZ2 = d_logistic(Z2)
    dZ2_dA1 = w_output
    dZ2_dW2 = A1
    dZ2_dB2 = 1
    dA1_dZ1 = d_relu(Z1)
    dZ1_dW1 = X
    dZ1_dB1 = 1

    dC_dW2 = dC_dA2 @ dA2_dZ2 @ dZ2_dW2.T

    dC_dB2 = dC_dA2 @ dA2_dZ2 * dZ2_dB2

    dC_dA1 = dC_dA2 @ dA2_dZ2 @ dZ2_dA1

    dC_dW1 = dC_dA1 @ dA1_dZ1 @ dZ1_dW1.T

    dC_dB1 = dC_dA1 @ dA1_dZ1 * dZ1_dB1

    return dC_dW1, dC_dB1, dC_dW2, dC_dB2

# Выполняем градиентный спуск
for i in range(100_000):
    # Случайно выбираем одну запись из обучающей выборки
    idx = np.random.choice(n, 1, replace=False)
    X_sample = X_train[idx].transpose()
    Y_sample = Y_train[idx]

    # Прогоняем случайно выбранные обучающие данные через нейронную сеть
    Z1, A1, Z2, A2 = forward_prop(X_sample)

    # Обратное распространение ошибки;
    # получаем угловые коэффициенты для весовых коэффициентов и смещений
    dW1, dB1, dW2, dB2 = backward_prop(Z1, A1, Z2, A2, X_sample, Y_sample)

    # Обновляем значения весовых коэффициентов и смещений
    w_hidden -= L * dW1
    b_hidden -= L * dB1
    w_output -= L * dW2
    b_output -= L * dB2
```

```
# Вычисляем точность классификации
test_predictions = forward_prop(X_test.transpose())[3] # берем только выходной
слой (A2)
test_comparisons = np.equal((test_predictions >= .5).flatten().astype(int), Y_test)
accuracy = sum(test_comparisons.astype(int) / X_test.shape[0])
print("Точность: ", accuracy)
```

Этот код устроен непросто, но он опирается на все, что мы изучили в этой главе. В нем выполняется 100 000 итераций стохастического градиентного спуска. Разделив данные на обучающую и тестовую выборки в пропорции $\frac{2}{3}:\frac{1}{3}$, я получаю на тестовой выборке точность примерно 97–99 % в зависимости от того, что вырабатывает генератор случайных чисел. Это значит, что после обучения нейронная сеть правильно спрогнозирует светлый или темный шрифт для 97–99 % тестовых данных.

Ключевой компонент здесь — функция `backward_prop()` с цепным правилом, которая принимает на вход ошибку в выходном узле (квадрат остатка), затем распределяет ее и распространяет обратно на весовые коэффициенты и смещения выходного и скрытого узлов, чтобы получить угловые коэффициенты по каждому весу и смещению. Затем в цикле `for` мы умножаем эти угловые коэффициенты на скорость обучения `L`, как это делалось в главах 5 и 6, и вычитаем полученную поправку из соответствующих весовых коэффициентов и смещений, чтобы скорректировать их. Чтобы распределить ошибку в обратном направлении на основе угловых коэффициентов, мы умножаем матрицы на векторы, а чтобы размеры строк и столбцов соответствовали друг другу, по необходимости транспонируем матрицы и векторы.

Если вы хотите, чтобы нейронная сеть была более интерактивной, посмотрите фрагмент кода в примере 7.12, который позволяет вводить различные цвета фона в формате «R, G, B» и смотреть, какой цвет шрифта спрогнозирует нейросеть — светлый или темный. Добавьте этот фрагмент в конец кода из примера 7.11 и попробуйте сами!

Пример 7.12. Добавление интерактивной оболочки к нейронной сети

```
# Взаимодействие с нейросетью и ее тестирование на новых цветах фона
def predict_probability(r, g, b):
    X = np.array([[r, g, b]]).transpose() / 255
    Z1, A1, Z2, A2 = forward_prop(X)
    return A2

def predict_font_shade(r, g, b):
    output_values = predict_probability(r, g, b)
    if output_values > .5:
        return "ТЕМНЫЙ ШРИФТ"
    else:
        return "СВЕТЛЫЙ ШРИФТ"
```

```
while True:
    print("Прогнозирование оттенка шрифта")
    col_input = input("Введите цвет фона в формате: {красный}, {зеленый},
{синий}: ")
    (r, g, b) = col_input.split(",")
    print(predict_font_shade(int(r), int(g), int(b)))
```

Чтобы создать собственную нейронную сеть с нуля, нужно много работы и математических вычислений, но это позволяет понять истинную природу нейронных сетей. Если вдаваться в подробности слоев, математического анализа и линейной алгебры, можно получить более полное представление о том, что делают на внутреннем уровне такие библиотеки глубокого обучения, как PyTorch и TensorFlow.

Прочитав эту главу, вы наверняка поняли, что в нейронной сети очень много компонентов, которые взаимодействуют друг с другом. Может оказаться полезным расставить точки останова в разных частях кода, чтобы проследить, что делает каждая матричная операция. Кроме того, можно перенести код в интерактивную тетрадь Jupyter Notebook и благодаря этому получить более наглядное представление о каждом шаге.

3Blue1Brown об обратном распространении

На YouTube-канале 3Blue1Brown есть несколько классических видеороликов об обратном распространении (https://youtu.be/Ilg3gGewQ5U) и вычислениях, которые лежат в основе нейронных сетей (https://youtu.be/tIeHLnjs5U8).

Scikit-learn для нейронных сетей

Хотя в scikit-learn доступны некоторые возможности нейронных сетей, их функциональность весьма ограничена. Если вы всерьез решили заняться глубоким обучением, то вам, вероятно, стоит изучить PyTorch или TensorFlow и обзавестись мощным графическим процессором (это отличный предлог, чтобы приобрести игровой компьютер, о котором вы мечтали!). Я слышал, что все реальные пацаны сейчас используют PyTorch. Однако в scikit-learn есть несколько удобных моделей, в том числе `MLPClassifier` — так называемый *классификатор с многослойными персептронами*[1]. Это нейронная сеть, которая предназначена для классификации и по умолчанию использует логистическую функцию активации на выходном слое.

[1] Формально говоря, узлы классификаторов такого рода некорректно называть *многослойными перспептронами* (multi-layer perceptron, MLP), учитывая математический смысл этого понятия. Однако этот обманчивый термин закрепился в широком употреблении; так принято называть полносвязные нейронные сети прямого распространения (FCN). — *Примеч. науч. ред.*

В примере 7.13 наше приложение для классификации цвета фона реализовано с помощью scikit-learn. Аргумент `activation` задает функцию активации для скрытого слоя[1].

Пример 7.13. Нейросетевой классификатор на основе scikit-learn

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

# Загружаем данные
df = pd.read_csv("https://bit.ly/3GsNzGt", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
# Обратите внимание, что здесь нужно линейное масштабирование
X = (df.values[:, :-1] / 255.0)

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]

# Разделяем набор данных на обучающую и тестовую выборки
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3)

nn = MLPClassifier(solver="sgd",
                   hidden_layer_sizes=(3, ),
                   activation="relu",
                   max_iter=100_000,
                   learning_rate_init=.05)

nn.fit(X_train, Y_train)

# Выводим весовые коэффициенты и смещения
print(nn.coefs_ )
print(nn.intercepts_)

print(f"Средняя точность на обучающей выборке: {nn.score(X_train, Y_train)}")
print(f"Средняя точность на тестовой выборке: {nn.score(X_test, Y_test)}")
```

Запуская этот код, я получаю на тестовых данных точность более 99 %.

Пример использования scikit-learn с базой данных MNIST

В Приложении А можно увидеть пример нейронной сети на основе scikit-learn, которая распознает рукописные цифры из базы данных MNIST.

[1] Параметр `activation` по умолчанию принимает значение `relu`, так что в данном случае его можно было не задавать в явном виде. — *Примеч. науч. ред.*

Ограничения нейронных сетей и глубокого обучения

При всех своих достоинствах нейронные сети плохо справляются с некоторыми типами задач. Широкие возможности работы со слоями, узлами и функциями активации позволяют гибко подгонять нейронную сеть к данным нелинейным образом — но, возможно, чересчур гибко. Почему? Нейросети могут чрезмерно подстраиваться под данные. Эндрю Ын (Andrew Ng), первопроходец в области глубокого обучения и бывший глава проекта Google Brain, упомянул об этой проблеме в интервью в 2021 году. В журнале IEEE Spectrum цитируется его ответ на вопрос, почему машинное обучение до сих пор не заменило врачей-рентгенологов (https://oreil.ly/ljXsz):

> Оказалось, что когда мы собираем данные по Стэнфордской больнице, а затем обучаем и тестируем модель на данных из той же больницы, то действительно можем публиковать результаты, которые показывают, что алгоритмы способны выявлять определенные заболевания не хуже врачей-рентгенологов.
>
> Однако если перенести ту же модель, ту же самую систему искусственного интеллекта в менее современную больницу на соседней улице, со старым оборудованием, где изображения обрабатываются по немного другому протоколу, то из-за дрейфа данных искусственный интеллект покажет куда менее впечатляющие результаты. Между тем, если любой врач-рентгенолог прогуляется по улице до старой больницы, он и там по-прежнему отлично справится со своей работой.
>
> Поэтому даже если в какой-то момент времени на конкретном наборе данных удается продемонстрировать, что искусственный интеллект работает, клиническая реальность такова, что этим моделям еще есть куда развиваться, прежде чем их можно будет применять в реальной жизни.

Другими словами, машинное обучение переобучилось на обучающей и тестовой выборках из Стэнфордской больницы, а в других больницах с другим оборудованием модели оказались гораздо менее эффективными.

Те же проблемы возникают с автономными транспортными средствами и беспилотными автомобилями. Недостаточно просто обучить нейронную сеть на одном дорожном знаке! Ее нужно обучить на бесчисленных комбинациях условий вокруг этого знака: в хорошую погоду и в дождь, ночью и днем, когда знак испорчен граффити или загорожен деревом, когда его дизайн соответствует иностранным правилам дорожного движения и т. д. Задумайтесь о том, как в реальном трафике нейронная сеть будет различать всевозможные транспортные средства и пешеходов, а также пешеходов в карнавальных костюмах, ростовых кукол и бесчисленное количество пограничных случаев. Сколько бы весовых коэффициентов и смещений ни добавлять в модель, она все равно не сможет эффективно учитывать все возможные объекты и события, которые встречаются на дороге.

Именно поэтому автономные транспортные средства не полагаются на одни только нейронные сети. Вместо этого различные программные и сенсорные модули выполняют разные функции. Например, один модуль может использовать нейронную сеть, чтобы построить прямоугольную область вокруг объекта. Затем другой модуль с помощью другой нейронной сети классифицирует объект в этой области — например, пешехода. После этого традиционная программная логика, основанная на правилах, попытается спрогнозировать траекторию движения пешехода, а жестко закодированые алгоритмы выберут реакцию системы, исходя из различных условий. В этом примере машинное обучение сводится к тому, чтобы размечать объекты, а не определять тактику и маневры автомобиля. Кроме того, базовые датчики, такие как радар, просто остановят автомобиль, если прямо по курсу обнаружится неизвестный объект, — это еще одна часть технологического стека, в которой не используется машинное или глубокое обучение.

Это может показаться странным, если учитывать многочисленные заголовки в СМИ о том, что нейронные сети и глубокое обучение побеждают человека в таких играх, как шахматы и го (https://oreil.ly/9zFxM), или даже превосходят пилотов в симуляторах боевых вылетов (https://oreil.ly/hbdYI). Однако, когда мы имеем дело с подобными средами обучения с подкреплением, важно помнить, что симуляции — это замкнутые виртуальные реальности, где можно генерировать бесконечное количество размеченных данных и обучаться на ограниченных искусственных выборках. Однако реальный мир — это не симуляция, где можно специально генерировать любые данные, которые нам заблагорассудится. (Также эта книга — не философский трактат, поэтому мы пропустим дискуссию о том, живем ли мы в симуляции. Прости, Илон Маск!) В реальном мире собирать данные затратно и трудоемко. Кроме того, реальный мир крайне непредсказуем и наполнен редкими событиями. Все эти факторы заставляют специалистов по машинному обучению прибегать к ручной разметке изображений дорожных объектов (https://oreil.ly/mhjvz) и других данных. Стартапам в области беспилотных автомобилей часто приходится совмещать эту разметку с симуляцией, потому что, чтобы собрать необходимые обучающие данные, нужно покрыть астрономические расстояния и рассмотреть бесчисленное количество пограничных сценариев, а это нельзя осуществить, если просто поручить автопарку наездить миллионы километров.

Вот почему исследователи в области искусственного интеллекта любят привлекать настольные и видеоигры, которые позволяют легко и просто сгенерировать сколько угодно размеченных данных. Известный инженер из Google Франсуа Шолле (Fran ois Chollet), который разработал библиотеку Keras для TensorFlow (а также написал отличную книгу «Deep Learning with Python»[1]), поделился некоторыми соображениями на этот счет в статье на сайте *Verge* (https://oreil.ly/4PDLf).

Дело в том, что, если вы выберете ту или иную метрику, вы будете «прокачивать» ее всеми доступными способами. Например, если выбрать в качестве

[1] Ф. Шолле. «Глубокое обучение на Python». 2-е межд. изд.— СПб, издательство «Питер».

меры интеллекта уровень игры в шахматы (как мы начали делать в 1970-х годах и продолжали вплоть до 1990-х), то в конце концов вы получите систему, которая умеет играть в шахматы, и ничего более. Нет никаких причин полагать, что она будет хороша для чего-то еще. Все сведется к поиску по дереву и минимаксу, и это совершенно ничего не говорит о человеческом интеллекте. В наше время вы попадаете в такую же ментальную ловушку, когда рассматриваете мастерство в видеоиграх вроде Dota или StarCraft как показатель общего интеллекта...

Если я зададусь целью пройти Warcraft III на сверхчеловеческом уровне с помощью глубокого обучения, — можете быть уверены, что у меня это получится, если только удастся привлечь талантливых технических специалистов и задействовать достаточные вычислительные мощности (которые для такой задачи составляют порядка десятков миллионов долларов). Но когда я этого добьюсь, узнаю ли я что-то новое об интеллекте? Поможет ли это решить проблему обобщения в области искусственного интеллекта? Скорее нет, чем да. Возможно я научусь лучше масштабировать глубокое обучение. Так что я не готов рассматривать эту задачу как научное исследование, потому что она не приносит новых знаний, которых у нас не было раньше. Такое «исследование» не отвечает ни на один открытый вопрос. Если бы вопрос звучал так: «Можно ли пройти такую-то игру на сверхчеловеческом уровне?», то ответ был бы однозначным: «Безусловно, можно, если только удастся сгенерировать достаточно большую выборку обучающих ситуаций и скормить их достаточно выразительной модели глубокого обучения». Это и так уже давно известно.

Иными словами, мы должны быть осторожны, чтобы не смешивать производительность алгоритма в игре с более широкими возможностями, которые еще предстоит исследовать. Машинное обучение, нейронные сети и глубокое обучение решают узкоспециализированные четко очерченные задачи. Они не умеют рассуждать на отвлеченные темы, самостоятельно выбирать себе задания или осмыслять объекты, которых раньше не видели. Как и любая компьютерная программа, они делают только то, на что их запрограммировали разработчики.

Решайте каждую задачу с помощью того инструмента, который для нее подходит. Не стоит с особым предубеждением относиться к нейронным сетям или к любому другому инструменту в вашем распоряжении. Нейронная сеть может оказаться не самым лучшим способом решить задачу, которая перед вами стоит. Важно всегда исходить из того, какую проблему вы пытаетесь решить, а не ставить основной целью задействовать тот или иной инструмент. Глубокое обучение стоит применять обоснованно и обдуманно. Безусловно, для него существуют подходящие сценарии использования, но в большинстве текущих задач вы, скорее всего, добьетесь большего успеха с более простыми и смещенными моделями, такими как линейная регрессия, логистическая регрессия или традиционные системы, основанные на правилах. Но если вам понадобилось классифицировать объекты на изображениях и у вас есть бюджет и человеческие ресурсы, чтобы составить соответствующий набор данных, то глубокое обучение — то, что вам нужно.

ГРЯДЕТ ЛИ ЗИМА ИСКУССТВЕННОГО ИНТЕЛЛЕКТА?

Полезны ли нейронные сети и глубокое обучение? Конечно, полезны! И их определенно стоит изучать. Однако вы наверняка видели, как СМИ, политики и знаменитости превозносят глубокое обучение как некий универсальный искусственный интеллект, который может сравниться с человеческим, если не превзойти его, и даже вот-вот захватит мир. Я бывал на выступлениях лидеров мнений в области IT, которые убеждали программистов, что через несколько лет те останутся без работы из-за машинного обучения, а код будет писать искусственный интеллект.

Спустя почти 10 лет эти предсказания все еще не сбылись, потому что они просто не соответствуют действительности. Искусственный интеллект порождает гораздо больше проблем, чем прорывов, а мне все еще приходится самому водить машину и писать код. Нейронные сети отдаленно напоминают человеческий мозг, но ни в коем случае не воспроизводят его. Их возможности даже близко не похожи на то, что вы видели в таких фильмах, как «*Терминатор*», «*Мир Дикого Запада*» или «*Военные игры*». Вместо этого нейронные сети и глубокое обучение работают над узкоспециализированными задачами — например, распознают фотографии собак и кошек после обучения на тысячах изображений. Как уже говорилось, они не умеют рассуждать, самостоятельно ставить себе задачи, учитывать неопределенность или распознавать объекты, которых они раньше не видели. Нейронные сети и глубокое обучение делают только то, на что их запрограммировали.

Такое несоответствие может привести к завышенным инвестициям и ожиданиям, и образуется пузырь, который грозит лопнуть. Тогда начнется очередная «зима искусственного интеллекта», когда из-за разочарования прекратится финансирование исследований в этой области. В Северной Америке, Европе и Японии «зимы искусственного интеллекта» случались неоднократно, начиная с 1960-х годов. Вполне вероятно, что очередная зима уже не за горами, но это не значит, что нейронные сети и глубокое обучение станут невостребованными. Они продолжат применяться в тех областях, где хорошо себя зарекомендовали: например, компьютерное зрение, обработка звука или распознавание естественного языка. Возможно, вы даже найдете новые способы их применения! Для каждой задачи следует выбирать тот инструмент, который лучше для нее подходит, будь то линейная регрессия, логистическая регрессия, традиционная програмная логика на основе правил или нейронная сеть. Умение подбирать правильные инструменты для правильных задач творит чудеса!

Заключение

Нейронные сети и глубокое обучение полезны во многих прикладных задачах, и в этой главе мы только поверхностно их затронули. Эти технологии находят широкое применение в разных областях — от распознавания изображений до обработки естественного языка.

Мы научились с нуля строить простую нейронную сеть с одним скрытым слоем, которая прогнозирует, какой шрифт следует использовать на фоне того или иного цвета — светлый или темный. Мы также применили некоторые продвинутые понятия математического анализа, чтобы вычислить частные производные вложенных функций, которые использовались в стохастическом градиентном спуске, чтобы обучить нейронную сеть. Мы также коснулись таких библиотек, как scikit-learn. Хотя TensorFlow, PyTorch и более мощные инструменты выходят за рамки этой книги, существуют отличные ресурсы, которые помогут вам расширить свои знания.

На YouTube-канале 3Blue1Brown есть великолепный плейлист по нейронным сетям и обратному распространению (https://oreil.ly/VjwBr), и его стоит посмотреть несколько раз. Не менее полезен и плейлист на канале StatQuest Джоша Стармера (Josh Starmer) по нейронным сетям (https://oreil.ly/YWnF2) — особенно в том, как визуализировать нейронные сети в форме манипуляций с топологическими многообразиями. Еще одно замечательное видео о теории многообразий и нейронных сетях можно найти на канале Art of the Problem (https://youtu.be/e5xKayCBOeU). Наконец, когда вы будете готовы глубже изучать материал, ознакомьтесь с уже упомянутыми книгами Орельена Жерона «Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow» и Франсуа Шолле «Deep Learning with Python».

Если вы дошли до конца этой главы и чувствуете, что усвоили бóльшую часть материала, примите мои поздравления! Вы не просто плодотворно познакомились с теорией вероятностей, статистикой, математическим анализом и линейной алгеброй, но и применили эти дисциплины на практике — например, в линейной регрессии, логистической регрессии и нейронных сетях. В следующей главе мы поговорим о том, как вам действовать дальше и как начать новый этап профессионального роста.

Упражнения для самопроверки

Примените нейронную сеть к набору данных о занятости сотрудников, с которым мы работали в главе 6. Данные можно импортировать отсюда (https://tinyurl.com/y6r7qjrp). Попробуйте построить нейронную сеть так, чтобы она давала прогнозы на этом наборе данных. Чтобы оценить ее эффективность, используйте метрику точности и матрицы ошибок. Хорошая ли это модель для этой задачи? Почему да или почему нет?

Хотя нейронную сеть можно построить с нуля, для экономии времени воспользуйтесь scikit-learn, PyTorch или другой библиотекой глубокого обучения.

Ответы см. в Приложении Б.

ГЛАВА 8

Советы по дальнейшей карьере

Приближаясь к концу этой книги, неплохо было бы подумать о том, куда двигаться дальше. Вы изучили и осмыслили широкий спектр дисциплин прикладной математики: математический анализ, теорию вероятностей, математическую статистику и линейную алгебру. Затем вы применили эту теорию к практическим методам, включая линейную регрессию, логистическую регрессию и нейронные сети. В этой главе мы поговорим о том, как использовать эти навыки в дальнейшем, обживая незнакомый, захватывающий и удивительно разнообразный мир, в котором развивается карьера специалиста по data science. Я подчеркну, что важно иметь ориентиры и осязаемую цель, к которой вы стремитесь, а не заучивать инструменты и методы, не представляя себе реальных задач.

Поскольку мы отвлекаемся от фундаментальных понятий и прикладных методов, эта глава будет отличаться от остальной книги. Вероятно, вы ожидаете советов о том, как предметно и результативно применить полученные навыки математического моделирования в своей профессиональной деятельности. Однако чтобы добиться успеха как специалист по data science, вам придется освоить еще несколько технических областей, таких как SQL и программирование, а также гибкие навыки (soft skills), чтобы развить профессиональную интуицию. Гибкие навыки особенно важны для того, чтобы не сбиться с пути в быстро меняющемся мире data science и чтобы «невидимая рука рынка» не застала вас врасплох.

Я не претендую на то, чтобы за вас решать, в чем состоят ваши профессиональные цели или чего вы надеетесь достичь, прочитав эту книгу. Однако, раз уж вы ее читаете, я позволю себе сделать несколько закономерных предположений. Я предполагаю, что вас интересует карьера в области data science, или, возможно, вы уже работаете в сфере анализа данных и хотите формализовать свою математическую подготовку. Возможно, у вас есть опыт программирования, и вы рассчитываете получить представление об искусственном интеллекте и машинном обучении. А может быть, вы руководите тем или иным проектом и чувствуете, что вам необходимо понимать возможности команды специалистов по data science

или искусственному интеллекту, чтобы правильно оценить масштабы проекта. Наконец, не исключено, что вы просто любознательный специалист, которому интересно, как математика может пригодиться на практике, а не только на экзаменах в университете.

Я постараюсь учесть все эти мотивы и дать несколько советов общего характера по профессиональному развитию, которые, надеюсь, будут полезны большинству читателей. Начнем с того, что же такое data science. Мы изучили математические основы этой дисциплины, а теперь поговорим о ней в контексте карьерного роста и перспектив ее развития.

Объективен ли автор?

Трудно не показаться субъективным в таких темах, как эта, где я даю советы по профессиональному росту, опираясь на собственный опыт (и персональный опыт других людей), а не на репрезентативные, тщательно организованные опросы и исследования, которые я же сам пропагандирую в главе 3. Однако в мою пользу можно заметить, что я проработал более десяти лет в компаниях из списка Fortune 500 и наблюдал, как они менялись благодаря развитию data science. Я выступал на многих технических конференциях по всему миру и слышал от бесчисленного количества собратьев по цеху: «У нас тоже такое случалось!» Я постоянно читаю множество блогов и авторитетных изданий от «Wall Street Journal» до «Forbes» и подозреваю, что научился распознавать, когда расхожие стереотипы не соответствуют реальности. Особенно пристально я слежу за признанными лидерами и «серыми кардиналами» в различных отраслях и наблюдаю, как они формируют и развивают рынки с помощью data science и искусственного интеллекта. В настоящее время я преподаю и консультирую представителей бизнеса по вопросам безопасности в сфере искусственного интеллекта в Университете Южной Калифорнии, на кафедре авиационной безопасности.

Я привел свой послужной список только для того, чтобы отметить, что, хотя я не проводил официальных опросов и исследований и не исключено, что мои выводы опираются на разрозненные частные случаи, тем не менее во всех моих источниках прослеживаются одни и те же устойчивые нарративы. Вики Бойкис (Vicki Boykis), прозорливый инженер по машинному обучению в компании Tumblr, в своем блоге поделилась похожими выводами (https://oreil.ly/vm8Vp), и я настоятельно рекомендую прочесть эту статью. Безусловно, вы имеете полное право воспринимать мои соображения с долей скепсиса, однако обращайте пристальное внимание на то, что происходит в вашем профессиональном окружении, и следите за тем, на какие предпосылки опирается ваше руководство и коллеги.

Так что же такое data science?

Data science анализирует данные, чтобы получать результаты, которые можно непосредственно применить на практике. В действительности эта дисциплина объединяет в себе различные области, связанные с данными: статистику, анализ и визуализацию данных, машинное обучение, исследование операций, разработку программного обеспечения — и это далеко не полный список. Практически любую деятельность, которая связана с данными, можно назвать «data science». Отсутствие четкого определения стало проблемой для всей индустрии. В конце концов, если у предмета нет однозначного определения, его можно трактовать по-разному, подобно произведению абстрактного искусства. Именно поэтому компаниям непросто составлять объявления о вакансиях специалистов по data science, потому что формулировки таких вакансий варьируют в самом широком диапазоне (https://oreil.ly/NHnbu). На рис. 8.1 перечислены различные дисциплины и инструменты, которые можно отнести к data science.

Именно поэтому я говорю своим клиентам, что лучше всего считать, что *data science* — это разработка программного обеспечения на основе статистики, машинного обучения и оптимизации. Если убрать что-то одно из этого (разработку программного обеспечения, статистику, машинное обучение или оптимизацию), специалист по data science рискует не справиться с задачей. Большинство организаций жалуются на то, что им трудно четко обозначить, какие навыки требуются от эффективного специалиста по data science, но надеюсь, что это определение поможет внести ясность. Хотя кому-то может показаться, что разработка программного обеспечения — это спорное требование, я считаю, что оно крайне необходимо, если учитывать, в каком направлении движется индустрия в целом. Мы еще вернемся к этому вопросу.

Но для начала, чтобы разобраться, что такое data science, давайте проследим историю этого понятия.

Как мы к этому пришли? И как получилось, что предмет, у которого нет четкого определения, превратился в такую непреодолимую силу в деловом мире? А главное, как определение data science (или его отсутствие) влияет на вашу карьеру? Все важные вопросы такого рода мы рассмотрим в этой главе.

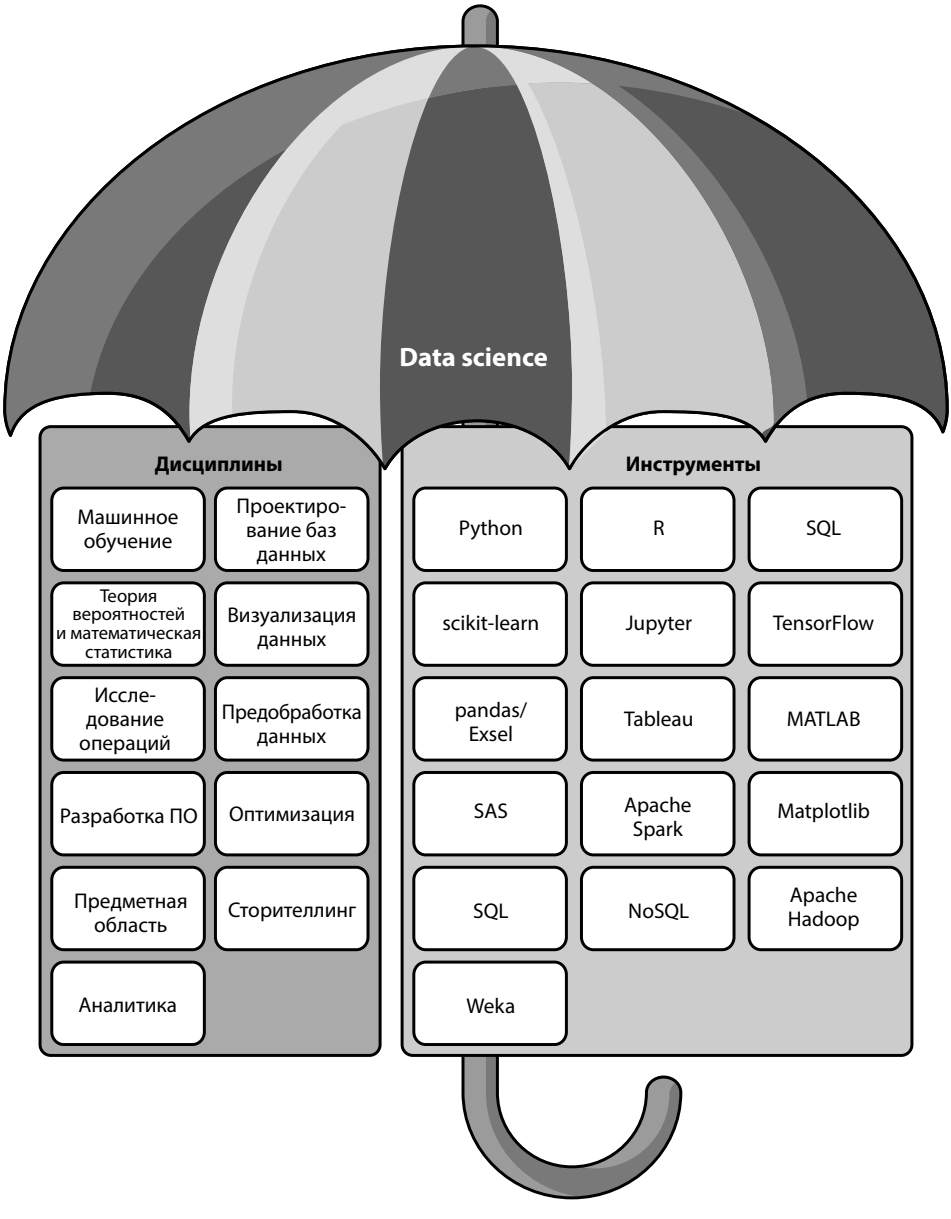


Рис. 8.1. «Зонтик» data science

Краткая история data science

Data science уходит корнями в раннюю историю статистики, которую принято исчислять с XVII или даже с VIII века (https://oreil.ly/tYPB5). Однако для краткости мы начнем сразу с 1990-х годов, когда аналитики, специалисты по статистике, исследователи, «кванты» (финансовые аналитики) и инженеры по обработке данных зачастую выполняли различные задачи. Их набор инструментов в основном состоял из электронных таблиц, R, MATLAB, SAS и SQL.

После 2000 года ситуация начала стремительно меняться. Интернет и подключенные к нему устройства стали генерировать огромные объемы данных. С появлением проекта Hadoop компания Google развила аналитику и сбор данных до невообразимых высот. Ближе к 2010 году руководители Google стали прогнозировать, что в следующем десятилетии статистика станет весьма заманчивой профессией (https://oreil.ly/AZgfM), и эти заявления оказались пророческими.

В 2012 году журнал «Harvard Business Review» широко популяризовал понятие data science, объявив ее «самой привлекательной профессией XXI века» (https://oreil.ly/XYbrf). После выхода статьи в «Harvard Business Review» как целые компании, так и отдельные сотрудники бросились восполнять пробел в области data science. Консультанты по управлению начали обучать руководителей компаний из списка Fortune 500 тому, как внедрить data science в свои организации. Специалисты по SQL, аналитики, исследователи, кванты, статистики, инженеры, физики и представители многих других профессий стали называть себя «специалистами по data science». Технологические компании тоже почувствовали, что традиционные названия должностей вроде «аналитик», «статистик» или «исследователь» звучат устаревшими, и переименовали их в «специалист по data science».

Естественно, компании из списка Fortune 500 под давлением своего высшего руководства стремились «запрыгнуть на подножку трамвая data science». Изначально это оправдывалось тем, что компьютерные системы собирают очень много данных, поэтому большие данные становятся трендом, и нужны специалисты по data science, чтобы их осмыслять. Примерно в это же время к самым разнообразным продуктам, процессам и технологиям стали приставлять словосочетание «на основе данных» или «управляемый данными». Корпоративный мир считал, что данные, в отличие от людей, объективны и беспристрастны.

Данные не бывают объективными и беспристрастными!

Многие специалисты, в том числе менеджеры, по сей день заблуждаются, считая, что данные объективны и не смещены. Надеемся, что, прочитав эту книгу, вы убедитесь, что это далеко не так. Если вам захочется освежить в памяти, в чем тут дело, — обратитесь к главе 3.

Руководители и кадровые отделы компаний оказались неспособны конкурировать за дипломированных специалистов в области глубокого обучения, которых поглотили крупнейшие корпорации (Facebook, Amazon, Apple, Netflix и Google). Однако, поскольку менее могущественным компаниям по-прежнему было важно поставить галочку в графе «data science», они предприняли интересный ход: переименовали существующих аналитиков, SQL-разработчиков и операторов Excel в «специалистов по data science». Кэсси Козырькова (Cassie Kozyrkov), которая работала старшим научным консультантом по принятию решений в Google, описала этот секрет Полишинеля в блоге в 2018 году (https://oreil.ly/qNl53).

> В каждой компании, где я занимала должность специалиста по data science, я до этого уже занималась той же самой работой под другим названием, пока озабоченные ребрендингом деятели из отдела кадров не переименовали должности в базе сотрудников. Мои обязанности не изменились ни на йоту. Я не одна такая: в моем кругу общения полно бывших специалистов по статистике, инженеров по поддержке принятия решений, финансовых аналитиков, профессоров математики, специалистов по большим данным, экспертов по бизнес-аналитике, ведущих аналитиков, ученых-исследователей, программистов, операторов Excel, дипломированных специалистов в узких областях — и все они сегодня гордо называются «специалистами по data science».

Формально data science не исключает ни одну из этих профессий, потому что все они так или иначе используют данные, чтобы делать полезные выводы. Разумеется, научное сообщество не торопилось провозглашать data science (в буквальном переводе «науку о данных») настоящей наукой. В конце концов, знаете ли вы хоть одну науку, которая не имеет дела с данными? В 2011 году Пит Уорден (Pete Warden), который на момент написания этой книги работает руководителем направления TensorFlow в Google, написал интересную статью в защиту сферы data science (https://oreil.ly/HXgvI). Он также удачно сформулировал аргументы тех, кто возражает по поводу отсутствия четкого определения.

> [Что касается отсутствия определения data science, то это], пожалуй, самое значительное возражение, и на него сложнее всего ответить. Не существует общепринятого понимания, что входит, а что не входит в предметную область data science. Может быть, это просто временный ребрендинг статистики? Мне так не кажется, но у меня тоже нет исчерпывающего определения. Я считаю, что современное обилие данных породило некую новую сущность, и если оглянуться вокруг, то можно увидеть людей с похожими навыками, которые не вписываются в традиционные категории. Как правило, деятельность этих людей не ограничивается рамками узких специализаций, которые доминируют в корпоративном и научном мире. Они занимаются всем сразу: ищут данные, всесторонне их обрабатывают, визуализируют и представляют выводы в форме истории. Кроме того, они обычно сначала выясняют, о чем могут рассказать данные, а потом

выбирают интересные темы, которые стоит изучить глубже. Это отличается от традиционного подхода ученых, которые сперва выбирают проблему, а затем ищут данные, чтобы пролить на нее свет.

Как ни странно, Пит тоже не смог придумать определение data science, зато внятно обосновал, почему это понятие несовершенно, но полезно. Он также обратил внимание на то, что в исследованиях все чаще отказываются от научного метода в пользу таких некогда порицаемых практик, как дата-майнинг, о котором мы говорили в главе 3.

Всего через несколько лет после статьи в Harvard Business Review в data science произошел интересный поворот. Когда примерно в 2014 году машинное и глубокое обучение стало доминировать в заголовках газет, данные преподносились как сырье для создания искусственного интеллекта. Это естественным образом расширило границы data science и привело к слиянию с искусственным интеллектом и машинным обучением. В частности, проект ImageNet оживил всеобщий интерес к искусственному интеллекту и стимулировал ренессанс машинного и глубокого обучения. Такие компании, как Waymo и Tesla, обещали, что благодаря достижениям в области глубокого обучения беспилотные автомобили появятся всего через несколько лет, что еще сильнее подогревало ажиотаж в СМИ и привлекало слушателей на курсы по повышению квалификации.

Этот всплеск интереса к нейронным сетям и глубокому обучению привел к интересному побочному эффекту. Методы регрессии, такие как деревья решений, метод опорных векторов и логистическая регрессия, которые десятилетиями не выходили за пределы академических кругов и специализированных статистических профессий, «упали на хвост» глубокому обучению и стали достоянием общественности. В то же время такие библиотеки, как scikit-learn, значительно снизили барьер входа в новую область. Это обернулось определенными издержками: в сфере data science появились работники, которые не понимали, как функционируют те или иные библиотеки или модели, но все равно использовали их.

Поскольку дисциплина data science развивалась быстрее, чем для нее успевали придумать определение, специалистам по data science стали приписывать самые разные должностные обязанности. Я знаком с несколькими людьми, которые занимали такую должность в компаниях из списка Fortune 500. Некоторые из них прекрасно разбираются в программировании и, возможно, раньше даже были разработчиками, но не имеют ни малейшего представления о том, что такое статистическая значимость. Другие не вылезают из Excel и почти не знают SQL, не говоря уже о Python или R. Я встречал «специалистов по data science», которые самостоятельно освоили несколько функций из scikit-learn и быстро оказывались в затруднительном положении, потому что больше ничего не знали.

Какие же выводы для вас из этого следуют? Как добиться успеха в такой нечетко определенной и переменчивой области? Все сводится к тому, какого рода задачи вас интересуют, и не спешите полагаться на то, как работодатели называют ту или

иную должность. Чтобы заниматься data science, необязательно именоваться «специалистом по data science». Существует множество сфер деятельности, в которых можно успешно работать с теми знаниями, которыми вы сейчас обладаете. Можно быть аналитиком, исследователем, инженером машинного обучения, советником, консультантом или представителем многих других должностей, в названиях которых не обязательно упоминается data science.

Но сначала давайте обсудим, в каких направлениях можно обучаться дальше, чтобы найти работу своей мечты в сфере data science.

Как найти работу своей мечты

Чтобы успешно работать, специалисту по data science недостаточно просто разбираться в статистике и машинном обучении. В большинстве случаев не стоит ожидать, что в вашем распоряжении будут готовые данные, которые сразу можно использовать для машинного обучения и других проектов. Вместо этого вам придется искать источники данных, разрабатывать скрипты и программы, перелопачивать документы и таблицы Excel и даже создавать собственные базы данных. А когда вы будете писать код, 95 % ваших усилий будут связаны не с машинным обучением или статистическим моделированием, а с созданием, перемещением и преобразованием данных, чтобы их можно было использовать.

Кроме того, вам стоит представлять себе, как устроена ваша организация и куда она движется. Определяя круг ваших обязанностей, руководители могут опираться на те или иные допущения, и важно их выявить, чтобы понимать, чего от вас ожидают. Хотя по вопросам бизнеса вы полагаетесь на своих клиентов и руководство, вы должны обеспечивать технические знания и предлагать решения, которые реально осуществимы. Давайте поговорим о том, какие технические и социальные навыки вам, скорее всего, понадобятся.

Язык SQL

SQL (structured query language, структурированный язык запросов) — это язык запросов, который позволяет получать, передавать и записывать табличные данные. Цифровые данные чаще всего хранятся в *реляционных базах данных*, которые состоят из таблиц, связанных друг с другом. Язык SQL поддерживают все системы управления реляционными базами данных, такие как MySQL, Microsoft SQL Server, Oracle, SQLite и PostgreSQL. Как вы наверняка заметили, SQL и реляционные базы данных так тесно связаны друг с другом, что аббревиатура SQL часто используется в названии соответствующих продуктов.

Пример 8.1 — это простой запрос SQL, который извлекает поля `CUSTOMER_ID` и `NAME` (идентификатор и имя клиента) из таблицы `CUSTOMER` («Клиенты») для записей, в которых поле `STATE` имеет значение `'TX'` (клиенты из штата Техас).

Пример 8.1. Простой запрос SQL

```
SELECT CUSTOMER_ID, NAME
FROM CUSTOMER
WHERE STATE = 'TX'
```

Проще говоря, трудно добиться успеха в data science, если не знать SQL. Все компании используют хранилища данных, и чаще всего данные из них извлекаются с помощью SQL. Вам должны быть хорошо знакомы ключевые слова `SELECT`, `WHERE`, `GROUP BY`, `ORDER BY`, `CASE`, `INNER JOIN` и `LEFT JOIN`. Чтобы извлечь максимальную пользу из данных, стоит еще разбираться в подзапросах, производных таблицах, обобщенных табличных выражениях и оконных функциях.

НЕСКРОМНЫЙ СОВЕТ. АВТОР НАПИСАЛ КНИГУ ПО SQL!

По заказу издательства O'Reilly я написал книгу «Getting Started with SQL» (https://oreil.ly/K2Na9). В ней чуть больше ста страниц, так что всю книгу можно прочитать за день. В ней рассматриваются основные понятия SQL, включая соединение и агрегирование, а также рассказывается, как создать собственную базу данных. Используется СУБД SQLite, которую можно настроить менее чем за минуту.

O'Reilly выпустило и другие замечательные книги по SQL, в том числе «Learning SQL»[1] Алана Больe (Alan Beaulieu) и «SQL. Pocket Guide»[2] Элис Жао (Alice Zhao). После того как вы быстро прочтете мой стостраничный опус, обратите внимание и на эти книги.

SQL также необходим для того, чтобы легко обращаться к базам данных из Python и других языков программирования. Если вы отправляете из Python запросы SQL к базе данных, можно возвращать данные в формате датафеймов pandas, коллекций Python и других структур.

В примере 8.2 показан простой запрос SQL, который выполняется на Python с помощью библиотеки SQLAlchemy. Он возвращает записи в виде именованных кортежей. Чтобы этот код работал, не забудьте скачать файл базы данных SQLite (https://bit.ly/3F8heTS) и поместить его в свой проект Python, а также запустить команду `pip install sqlalchemy`.

---

[1] А. Больe. «Изучаем SQL. Генерация, выборка и обработка данных». 3-е изд.

[2] Э. Жао. «SQL. Pocket guide». 4-е изд.— СПб., издательство «Питер».

Пример 8.2. Выполнение запроса SQL на Python с помощью SQLAlchemy

```
from sqlalchemy import create_engine, text

engine = create_engine("sqlite:///thunderbird_manufacturing.db")[1]
conn = engine.connect()

stmt = text("SELECT * FROM CUSTOMER")
results = conn.execute(stmt)

for customer in results:
    print(customer)
```

А что насчет pandas и NoSQL?

Меня часто спрашивают об «альтернативах» SQL, таких как NoSQL или pandas. На самом деле это не альтернативы, а инструменты другого назначения, которые тоже занимают свое место в технологическом стеке data science. Возьмем, например, pandas. В примере 8.3 создается запрос SQL, который извлекает все записи из таблицы `CUSTOMER` и помещает их в датафрейм pandas.

Пример 8.3. Импорт запроса SQL в датафрейм pandas

```
from sqlalchemy import create_engine, text
import pandas as pd

engine = create_engine('sqlite:///thunderbird_manufacturing.db')
conn = engine.connect()

df = pd.read_sql("SELECT * FROM CUSTOMER", conn)
print(df) # выводит результаты запроса SQL в виде датафрейма
```

SQL здесь понадобился для того, чтобы преодолеть разрыв между реляционной базой данных и средой Python и загрузить данные в датафрейм pandas. А если в задаче фигурируют сложные вычисления, с которыми умеет справляться SQL, эффективнее будет выполнить их на сервере базы данных с помощью SQL, а не через pandas на локальном компьютере. Проще говоря, pandas и SQL могут работать вместе и не конкурируют друг с другом.

То же самое можно сказать и о системах класса NoSQL, к которым относятся такие платформы, как Couchbase и MongoDB. Хотя некоторые читатели со мной не согласятся и наверняка приведут веские аргументы, я считаю, что сравнивать

[1] Путь к базе данных зависит от того, где вы сохранили файл thunderbird_manufacturing.db на локальном компьютере и каким образом подключили его к проекту. Например, если вы работаете в Windows и сохранили файл в каталоге D:\SQL, то эта строка кода будет выглядеть так: engine = create_engine(«sqlite:///D:\\SQL\\thunderbird_manufacturing.db»). — *Примеч. науч. ред.*

SQL и NoSQL — это все равно что сравнивать яблоки с апельсинами. Безусловно, системы той и другой категории хранят данные и позволяют обращаться к ним с запросами, но я не думаю, что это противопоставляет их друг другу. Они устроены по-разному и ориентированы на разные сценарии использования. NoSQL расшифровывается как «не только SQL» (not only SQL) и лучше подходит для того, чтобы хранить неструктурированные данные, такие как изображения или текстовые статьи в свободной форме. В свою очередь, SQL лучше приспособлен для структурированных данных. SQL более строго поддерживает целостность данных, чем NoSQL, но расходует больше вычислительных ресурсов и хуже масштабируется.

SQL — ВСЕОБЩИЙ ЯЗЫК МИРА ДАННЫХ

В 2015 году многие предсказывали, что технологии NoSQL и распределенной обработки данных, такие как Apache Spark, вытеснят SQL и реляционные базы данных. В действительности получилось наоборот: SQL оказался настолько важен для пользователей данных, что по их многочисленным заявкам в эти платформы были добавлены компоненты SQL. Например, с такими компонентами работают Presto (https://oreil.ly/Qf6c1), BigQuery (https://oreil.ly/iCEWW) и Apache Spark SQL (https://oreil.ly/IIPft). Большинство задач по работе с данными не относятся к большим данным, и в этой области ничто не может превзойти SQL по эффективности запросов. Поэтому SQL продолжает процветать и остается «всеобщим языком» в мире данных.

При этом пропаганда платформ NoSQL и больших данных может быть эффектом синдрома «серебряной пули». Хади Харири (Hadi Hariri) из JetBrains выступил с докладом на эту тему в 2015 году (https://oreil.ly/hPEIF), и это выступление стоит посмотреть.

Программирование

Многие специалисты по data science не умеют программировать, по крайней мере на уровне профессиональных разработчиков. Однако уметь писать код становится все важнее, и этот навык дает преимущество при трудоустройстве. Изучайте объектно-ориентированное программирование, функциональное программирование, модульное тестирование, системы контроля версий (например, Git и GitHub), анализ алгоритмов с помощью нотации «О-большое», криптографию, а также другие разделы computer science и возможности языков, с которыми вы сталкиваетесь.

И вот зачем это нужно. Допустим, вы создали перспективную регрессионную модель (например, логистическую регрессию или нейронную сеть) на основе выборочных данных, которые вам предоставили. Вы просите штатных программистов из вашего отдела IT подключить ее к существующему программному обеспечению.

Они воспринимают вашу идею без особого энтузиазма. «Нам понадобится переписать это на Java, а не на Python», — нехотя говорит один из них. «А где у вас модульные тесты? — спрашивает другой. — У вас не определено ни одного класса или типа? Нам придется переделать код, чтобы он стал объектно-ориентированным». Вдобавок ко всему программисты не понимают математического смысла вашей модели и справедливо опасаются, что она может неправильно вести себя на данных, с которыми раньше не сталкивалась. Поскольку вы не обеспечили модульных тестов, которые не так просто подготовить в случае машинного обучения, они не знают, как проверить качество вашей модели. К тому же они беспокоятся о том, как придется управлять двумя версиями кода — на Python и на Java.

Вы начинаете чувствовать себя не в своей тарелке и говорите: «Я не понимаю, почему нельзя просто подключить скрипт на Python». Один из программистов делает задумчивую паузу и отвечает: «Мы могли бы создать веб-службу на Flask, и тогда код не понадобится переписывать на Java. Однако другие проблемы никуда не денутся. Нам придется беспокоиться о масштабируемости и большом трафике, который может обрушиться на веб-службу. Впрочем, погодите: не исключено, что мы сможем развернуть решение в облаке Microsoft Azure в виде масштабируемого набора виртуальных машин, но тогда нам все равно понадобится разрабатывать серверную часть архитектуры. В любом случае, как ни крути, систему придется перепроектировать».

Именно поэтому многие специалисты по data science держат всю свою работу только на собственном компьютере. На самом деле внедрять машинное обучение в массовую эксплуатацию стало настолько проблематично, что в последние годы растет спрос на специалистов в этой области. Существует огромный разрыв между теми, кто занимается data science, и теми, кто разрабатывает программное обеспечение, поэтому неудивительно, что первым приходится так или иначе становиться вторыми.

Такая разносторонняя специализация может показаться чрезмерной, потому что data science и так охватывает весьма широкую область, со множеством дисциплин и требований. Однако речь не о том, что вам нужно срочно учить Java. Вы можете успешно писать программы на Python (или на любом другом языке, который годится для промышленной разработки), но вам не помешает хорошо в нем разбираться. Освойте объектно-ориентированное программирование,

структуры данных, функциональное программирование, конкурентность и другие паттерны разработки. Если говорить о Python, то вот две хорошие книги: Лусиану Рамальо (Luciano Ramalho), «Fluent Python»[1] и Эл Свейгарт (Al Sweigart), «Beyond the Basic Stuff with Python»[2].

ЯЗЫК GO И DATA SCIENCE

В 2016 году Дэниел Уайтнек (Daniel Whitenack) написал для O'Reilly статью «Data science gophers» («Go-разработчики в data science») (https://oreil.ly/j4z4F), в которой рассказывал о достоинствах языка программирования Go для data science. Примечательно, что Дэниел осветил проблемы внедрения моделей data science в массовую эксплуатацию задолго до того, как эти проблемы стали предметом всеобщего обсуждения.

Наконец, познакомьтесь с прикладными областями, без которых не обойтись в практических задачах: API баз данных, веб-службы (https://oreil.ly/gN9e7), JSON (https://oreil.ly/N8uef), регулярные выражения (https://oreil.ly/IyD2P), веб-скрейпинг (https://oreil.ly/9oWWb), безопасность и криптография (https://oreil.ly/oxliO), облачные вычисления (Amazon Web Services, Microsoft Azure) и все остальное, что поможет вам выжить в профессиональном мире.

Как уже говорилось, вам не обязательно осваивать именно Python. Вы можете овладеть другим языком программирования, но желательно, чтобы он был широко распространенным и подходил для задач data science. На момент написания этой книги к таким языкам относятся, например, Python, R, Java, C# и C++. А на устройствах Apple и Android доминируют Swift и Kotlin — тоже замечательные языки с широкой поддержкой. Хотя многие из перечисленных языков преимущественно ориентированы не на data science, вам стоило бы выучить хотя бы еще один язык, помимо Python, чтобы опираться на более широкий кругозор.

А ЧТО НАСЧЕТ БЛОКНОТОВ JUPYTER?

Специалистов по data science часто укоряют за то, что они пишут плохой код. Это может происходить по многим причинам, и одна из них — стиль работы, который поощряют блокноты (они же тетрадки) Jupyter Notebook.

[1] Л. Рамальо. «Python — к вершинам мастерства: Лаконичное и эффективное программирование». 2-е изд.

[2] Э. Свейгарт. «Python. Чистый код для продолжающих».— СПб., издательство «Питер».

Должно быть, вы задавались вопросом, почему я не использовал (или не рекомендовал) блокноты Jupyter в этой книге. Jupyter Notebook — это популярная платформа для того, чтобы писать код для data science. Она позволяет сосредоточить в одном месте текстовые заметки, фрагменты выполняемого кода и результаты вывода консоли и диаграмм. Блокноты могут служить полезным инструментом и обеспечивают удобный способ рассказать историю с помощью данных.

Тем не менее, чтобы успешно заниматься data science, совершенно не обязательно использовать блокноты, если только этого не требует ваш работодатель. Все, о чем мы говорили в этой книге, было реализовано на чистом Python без Jupyter. Я сделал это специально, потому что не хотел обременять читателей тем, чтобы устанавливать дополнительные инструменты. Возможно, кому-то мое мнение покажется ересью, но я убежден, что можно сделать неплохую карьеру, не притрагиваясь к подобным блокнотам.

Дело в том, что они развивают вредные привычки программирования. Блокноты нацелены на такой стиль работы, который подчеркивает линейность, а не модульность, а это значит, что с ними труднее писать код, который можно использовать повторно; а ведь повторное использование — это, пожалуй, самая фундаментальная цель программирования. Кроме того, ячейки блокнотов с фрагментами кода можно запускать в произвольном порядке или многократно перезапускать. Если вовремя не уследить, можно породить несогласованные состояния и дефекты, которые в лучшем случае явно нарушат работу кода, а в худшем — приведут к незаметным ошибкам в важных расчетах. Если вы начинающий программист, такой способ изучения Python может оказаться для вас особенно мучительным, потому что эти технические ловушки неочевидны для новичков. Также может оказаться, что с помощью блокнотов Jupyter вы обнаружите интересные результаты и обнародуете их — а в результате выяснится, что они возникли из-за ошибки и не соответствуют действительности.

Я не призываю вас избегать блокнотов. Конечно, пользуйтесь ими на здоровье, если они доставляют удовольствие вам и вашему предприятию! Однако я убежден, что на них не стоит слепо полагаться. Джоэл Грус (Joel Grus), автор книги «Data Science. From Scratch», убедительно раскрыл эту тему на JupyterCon в докладе, который можно посмотреть по ссылке (https://oreil.ly/V00bQ).

Эффект привязки и первые языки программирования

Технические специалисты склонны эмоционально привязываться к технологиям и платформам, особенно к языкам программирования. Пожалуйста, не заводите эту вредную привычку! Такая предвзятость непродуктивна и расходится с реальностью: ведь разные языки программирования подходят для разных задач и сценариев использования. Еще один фактор заключается в том, что одни языки приживаются, а другие нет, и причины этого часто не имеют никакого отношения к достоинствам самих языков. Если крупная компания не финансирует развитие языка, то шансы выжить у него невелики.

Мы говорили о разных типах когнитивных смещений в главе 3. Еще одно смещение — эффект привязки (https://oreil.ly/sXNh0), который заключается в том, что мы можем чрезмерно прикипеть душой к первому, что мы изучаем, — например, к языку программирования. Если вы чувствуете, что пора выучить новый язык, будьте непредвзяты и дайте ему шанс! Ни один язык не идеален, и важно лишь, чтобы тот или иной язык позволял эффективно решить вашу задачу.

Однако будьте осторожны, если жизнеспособность языка вызывает сомнения: например, если его существование поддерживается искусственно, он не обновляется или его не сопровождает никакая крупная корпорация. В качестве примеров можно привести VBA от Microsoft (https://oreil.ly/B8c5A), Ceylon от Red Hat (https://oreil.ly/LJdw4) и Haskell (https://oreil.ly/ASnnN).

БИБЛИОТЕКИ JAVA ДЛЯ DATA SCIENCE

Хотя язык Java не так популярен в области data science, как Python, в Java есть несколько соответствующих библиотек, которые активно поддерживаются. ND4J (https://github.com/deeplearning4j/nd4j) — это аналог NumPy для виртуальной машины Java, а SMILE (https://haifengl.github.io) — аналог scikit-learn. TableSaw (https://github.com/jtablesaw/tablesaw) — это эквивалент pandas для Java.

Apache Spark (https://spark.apache.org) был изначально разработан на платформе Java, в том числе на языке Scala. Интересно, что Spark в свое время был флагманом инициативы, которая пыталась превратить Scala в основной язык для data science, хотя он не прижился в той степени, на которую, вероятно, рассчитывало сообщество Scala. Именно поэтому разработчики Spark приложили много усилий, чтобы добавить совместимость с Python, SQL и R, а не только с Java и Scala.

Визуализация данных

Еще один технический навык, которым вам стоит владеть в той или иной степени, — визуализация данных. Научитесь строить графики, схемы и диаграммы, которые не только расскажут историю руководителям, но и помогут вам самим изучать данные. Данные можно легко обобщить с помощью команд SQL, но иногда гистограмма или диаграмма рассеяния поможет получить более полное представление о данных за меньшее время.

Какие инструменты нужны, чтобы визуализировать данные? Ответить на этот вопрос не так просто, потому что пространство выбора чрезвычайно широко и разнообразно. Если вы работаете в традиционной офисной среде, то, скорее всего, предпочтете Excel и PowerPoint, и знаете что? Они вполне хороши! Я не использую их для всего подряд, но с большинством задач они прекрасно справляются. Нужна диаграмма рассеяния для небольшого набора данных? Или гистограмма? Нет проблем! Ее можно построить всего за несколько минут, если скопировать и вставить данные в таблицу Excel. Этот способ отлично подходит, чтобы единоразово визуализировать данные, и нет ничего постыдного в том, чтобы использовать Excel, если он работает.

Однако бывают ситуации, когда нужно запрограммировать построение графиков, чтобы код можно было повторно использовать и (или) интегрировать с другим кодом на Python. Лучшим решением для этого уже давно считается библиотека matplotlib (https://matplotlib.org), и без нее сложно обойтись, если вы в основном работаете с Python. Библиотека Seaborn (https://seaborn.pydata.org) предоставляет обертку поверх matplotlib, чтобы ее было проще использовать для стандартных графиков. Библиотека SymPy, к которой мы часто обращались в этой книге, использует matplotlib в качестве внутреннего интерфейса. Однако некоторые считают matplotlib уже не столько зрелой, сколько «перезрелой» библиотекой, которая все больше превращается в устаревшую. Зато активно развиваются такие приятные в использовании библиотеки, как Plotly (https://plotly.com/python), которая основана на библиотеке JavaScript D3.js (https://d3js.org). Лично я успешно работаю с Manim (https://www.manim.community). Она создает визуализации в стиле 3Blue1Brown, которые выглядят выразительно и производят на клиентов тот самый вау-эффект, а ее API удивительно просто использовать, учитывая всю мощь анимации, которую предлагает библиотека. Однако она появилась относительно недавно и еще не достигла зрелости, а значит, не исключено, что ее обновления будут нарушать совместимость с кодом, который вы писали ранее.

Вы точно не прогадаете, если изучите все эти решения и остановитесь на том, которое лучше всего подойдет именно вам, если у вашего работодателя или клиента нет конкретных предпочтений.

Существуют и коммерческие лицензируемые системы, такие как Tableau (https://www.tableau.com/products/desktop), которые тоже хороши в определенной степени. Разработчики Tableau стремились создать собственную платформу, которая специализируется на визуализации и предлагает интерфейс с перетаскиванием, который доступен для нетехнических специалистов. У Tableau даже есть документ под названием «Как сделать из каждого сотрудника вашей организации специалиста по data science» (https://oreil.ly/kncmP), который, впрочем, не проясняет уже обсуждавшийся вопрос о том, кого же считать специалистом по data science. Я вижу проблемы Tableau в том, что она хорошо справляется только с визуализацией и требует дорогостоящей лицензии. Хотя с помощью TabPy (https://tableau.github.io/TabPy/docs/about.html) можно в некоторой степени интегрировать Python в Tableau, лучшее решение может заключаться в том, чтобы использовать вышеупомянутые библиотеки с открытым исходным кодом, если только ваш работодатель не настаивает на Tableau.

Лицензируемое ПО как элемент корпоративной политики

Представьте, что вы разработали приложение на Python или Java, которое запрашивает определенные данные от пользователей, получает и обрабатывает информацию из разных источников, выполняет несколько тонко отрегулированных алгоритмов, а затем выводит визуализацию и таблицу с результатами. После нескольких месяцев напряженной работы вы демонстрируете это приложение на совещании, но тут один из менеджеров поднимает руку и спрашивает: «А почему нельзя было сделать то же самое в Tableau?»

Некоторым руководителям тяжело принять тот факт, что они потратили тысячи долларов на лицензии корпоративного программного обеспечения, а вы приходите и показываете более эффективное (хотя и более сложное в использовании) решение с открытым исходным кодом, которое не надо лицензировать. В свою защиту вы можете сделать упор на том, что Tableau не поддерживает нужные алгоритмы или комплексные технологические процессы, которые вам пришлось разработать. В конце концов, Tableau — это просто программа для визуализации, а не платформа, которая позволяет писать код с нуля, чтобы создать специализированное решение.

Руководителям часто внушают, что Tableau, Alteryx или другой коммерческий инструмент может сделать все. В конце концов, они потратили на него кучу денег и, вероятно, находятся под впечатлением от рекламных презентаций поставщиков. Естественно, они хотят окупить инвестиции и добиться того, чтобы лицензией пользовались как можно больше сотрудников. Вероятно, они потратили еще одну кучу денег, чтобы обучить сотрудников работать с программой, и хотят, чтобы другие могли сопровождать вашу работу.

Отнеситесь к этому с пониманием. Если руководители хотят, чтобы вы использовали инструмент, за который они заплатили, то изучите, насколько это осуществимо. Если у инструмента есть критические ограничения или если он категорически неудобен в ваших конкретных задачах, найдите способ дипломатично сообщить об этом.

Предметная область

Давайте сравним две отрасли: стриминговую платформу (например, Netflix) и военно-промышленную корпорацию (например, Lockheed Martin). Есть ли у них что-то общее? Едва ли! Обе компании опираются на высокие технологии, но одна из них просто показывает фильмы по интернету, а другая производит военные самолеты.

Когда я консультирую по вопросам искусственного интеллекта в контексте безопасности, то в качестве одного из первых соображений обращаю внимание на то, что эти две отрасли различаются совершенно разной терпимостью к рискам. Стриминговая компания может нахваливать собственную систему искусственного интеллекта, которая рекомендует фильмы клиентам, но что будет, если она даст плохую рекомендацию? В худшем случае вы получите слегка разочарованного зрителя, который потратил два часа, чтобы посмотреть фильм, который ему не понравился.

А как насчет оборонного предприятия? Если истребитель оснащен искусственным интеллектом, который автоматически стреляет по целям, насколько катастрофичной будет ошибка? Речь идет о человеческих жизнях, а не о рекомендациях фильмов!

Допустимый уровень риска в этих двух отраслях существенно различается. Поэтому неудивительно, что аэрокосмическая оборонная корпорация будет гораздо консервативнее относиться к внедрению любой экспериментальной системы. Это означает бюрократию и рабочие группы по безопасности, которые оценивают каждый проект и блокируют его, если считают риски неприемлемыми, — и это вполне оправданный подход. Однако интересно, что по мере того как искусственный интеллект завоевывал успех в стартапах Кремниевой долины (в основном в малорискованных приложениях, таких как рекомендации фильмов), даже руководители оборонной промышленности начали бояться пропустить все самое интересное. Вероятно, это связано с тем, что не все в полной мере представляют себе, чем различается допустимый уровень риска в этих двух областях.

Конечно, риски сильно разнятся — от раздраженных пользователей в одной сфере до гибели людей в другой. Банки могут с помощью искусственного интеллекта решать, кому одобрить кредит, но при этом возникает риск, что система будет

дискриминировать определенные слои населения. Судебные органы экспериментировали с искусственным интеллектом в области условно-досрочного освобождения и надзора, но столкнулись с теми же проблемами дискриминации. Социальные сети пытаются благодаря искусственному интеллекту отличать приемлемые посты от неприемлемых (https://oreil.ly/VoK95), но при этом они разочаровывают пользователей, когда блокируют безобидный контент (ложноположительные результаты), а также настораживают регуляторов, когда не блокируют опасный контент (ложноотрицательные результаты).

Это говорит о том, что вам стоит понимать предметную область, в которой вы работаете. Если вы хотите интенсивно заниматься машинным обучением, то, скорее всего, вам имеет смысл трудиться в низкорисковых отраслях, где ложноположительные и ложноотрицательные результаты не представляют опасности и никого не огорчают. Но если это вас не привлекает и вы хотите работать над более амбициозными проектами, такими как беспилотные автомобили (https://oreil.ly/sOYs6), авиация и медицина, то будьте готовы к тому, что модели машинного обучения, которые вы создаете, будут часто забраковываться.

Не удивляйтесь, если в этих высокорисковых отраслях потребуется специализированное образование и другие официальные заслуги. Даже если у вас есть ученая степень в нужной области, ложноположительные и ложноотрицательные результаты не пропадут сами собой. Если вы не готовы достигать высот в рамках такой узкой специализации, то вам, скорее всего, будет лучше освоить другие инструменты, помимо машинного обучения, — например, разработку программного обеспечения, оптимизацию, статистику и системы бизнес-логики.

Эффективное обучение

В 2008 году стендап-комик Брайан Риган (Brian Regan) рассказывал, насколько он нелюбопытен, сравнивая себя с теми, кто читает газеты. Он отметил, что история на первой полосе газеты всегда обрывается на самом интересном месте, и заявил, что у него не возникает желания переходить на другую страницу, чтобы узнать, чем все закончится. «После девятилетнего судебного разбирательства присяжные наконец вынесли вердикт: обвиняемый... смотрите продолжение на странице 22 в третьей колонке. Что ж, не очень-то и хотелось», — пренебрежительно заметил комик. Затем он для пущего контраста изобразил того, кто перелистывает страницы, восклицая: «Я хочу знать! Я хочу получать знания!»

По-видимому, Риган имел в виду самоиронию, но, возможно, в чем-то он был прав. Изучать предмет ради него самого — это не самая плодотворная мотивация, а оставаться равнодушным — не всегда плохо. Если вы откроете учебник по математическому анализу, не имея цели его изучать, то, скорее всего, в итоге окажетесь разочарованы. Предполагаю, что вы ориентируетесь на какой-то проект

или другую цель, а если вы считаете, что тема неинтересна в контексте этого проекта, то зачем ее изучать? Лично я почувствовал огромное облегчение, когда перестал интересоваться темами, которые казались мне не заслуживающими внимания. Что еще более удивительно, я стал работать гораздо продуктивнее.

Это не значит, что вам не стоит проявлять любопытство. Однако в мире очень много информации, и вы, безусловно, выиграете, если научитесь расставлять приоритеты в том, чем занимать свою голову. Спрашивайте себя, какая польза от тех или иных сведений, и, если не удается сформулировать однозначный ответ, позвольте себе обойтись без них! Если все вокруг галдят об обработке естественного языка, — из этого не следует, что вам стоит присоединяться к общему гвалту! Большинству компаний не нужна обработка естественного языка, и не будет ничего плохого, если вы сочтете, что эта тема не стоит ваших усилий и времени.

Неважно, идет ли речь о ваших проектах на работе или о самообразовании, — хорошо, когда у вас есть осязаемые ориентиры, к которым можно стремиться. Только *вы сами* решаете, что вам стоит изучать, и ничто не мешает отстраниться от модного повального увлечения, сосредоточившись на том, что вам самим кажется интересным и актуальным.

Практик или консультант

Возможно, это досужее обобщение, но в разных предметных областях я наблюдаю экспертов двух типов: практики и консультанты. Чтобы найти свое место, решите, кем из них вы хотите стать, и соответствующим образом развивайте профессиональные навыки.

В мире data science и аналитической работы практикующие специалисты пишут код, разрабатывают модели, исследуют данные и пытаются непосредственно создать реальную ценность. В свою очередь, консультанты объясняют руководителям, состоятельны ли их цели, помогают разрабатывать стратегию и задают направление. Иногда практики становятся консультантами, но нередки и такие консультанты, которые никогда не были практиками. У каждой роли есть свои плюсы и минусы.

Практику может нравиться программировать, анализировать данные и выполнять другую работу с осязаемыми результатами. Преимущество практикующего специалиста в том, что он развивает профессиональные навыки. Однако, если слишком глубоко погрузиться в код, вычисления и данные, легко потерять из виду общую картину и утратить связь с остальной частью компании и отраслью в целом. Я часто слышу от руководителей жалобы на то, что их специалист по data science предпочитает работать над задачами, которые ему кажутся

интересными, но не приносят выгоды для компании. Я также слышал жалобы от практикующих специалистов, которые хотели бы продвигаться по карьерной лестнице, но чувствуют, что корпоративное окружение их стесняет и не дает простора для развития.

Считается, что работать консультантом в некотором смысле легче, чем практиком. Консультанты дают советы и снабжают руководителей информацией, а также помогают задать стратегическое направление развития бизнеса. Как правило, они не пишут код и не исследуют данные, но помогают руководству находить тех, кто этим занимается. Карьерные риски здесь совсем другие, потому что консультантам не приходится волноваться о том, чтобы укладываться в дедлайн спринта, отлаживать код или исправлять ошибочные модели, как это делают практики. Но для консультантов важно оставаться компетентными, авторитетными и современными.

Чтобы эффективно выполнять роль консультанта, вам нужно быть *действительно* сведущим и знать то, чего не знают другие. Вы должны владеть критически важной и актуальной информацией, которая точно соответствует потребностям вашего клиента. Чтобы ваши советы не оказались устаревшими, вам понадобится каждый день читать, еще раз читать и снова читать, а также искать и обобщать информацию, которую другие упускают из виду. Недостаточно просто разбираться в машинном обучении, статистике и глубоком обучении. Вам придется ориентироваться в отрасли своего клиента, а также в других отраслях и отслеживать, кто добивается успеха, а кто нет. Вы также должны научиться находить правильные решения для правильно поставленных задач в условиях, когда многие деятели бизнеса ищут «серебряную пулю». И чтобы осуществлять все это, вам нужно быть хорошим собеседником и уметь делиться информацией так, чтобы помочь своему клиенту, а не просто продемонстрировать, насколько вы сведущи.

Самый большой риск для консультанта — предоставить информацию, которая в итоге окажется неверной. Некоторые консультанты наловчились перекладывать вину на внешние факторы, например: «Никто в отрасли не ожидал такого развития событий» или «Это событие за пределами шести сигм» (то есть у нежелательного события был один из полумиллиарда шансов произойти, но оно все равно произошло). Еще один риск — не обладать необходимыми навыками практикующего специалиста и действовать в отрыве от технической стороны бизнеса. Вот почему стоит регулярно самостоятельно практиковаться в программировании и моделировании или, по крайней мере, не переставать читать техническую литературу.

В результате хороший консультант старается стать связующим звеном между клиентом и его конечной целью, часто заполняя существующий пробел в знаниях. Речь идет не о том, чтобы отрабатывать как можно больше часов или создавать видимость кипучей деятельности, а о том, чтобы на самом деле разбираться, что беспокоит вашего клиента, и помогать ему разрешать проблемы.

УСПЕХ НЕ ВСЕГДА ИЗМЕРЯЕТСЯ ПРИБЫЛЬЮ

Выясните, что ваш клиент понимает под успехом. Компании внедряют искусственный интеллект, машинное обучение и data science, чтобы стать успешными, верно? Но что такое успех?

Определяется ли он рентабельностью предприятия? Не всегда. В нашей крайне спекулятивной экономике успех может означать очередной раунд венчурного финансирования, рост клиентской базы или выручки, а может, высокую рыночную стоимость компании, даже если она фактически теряет миллионы или миллиарды долларов. Ни один из этих показателей никак не связан с рентабельностью.

Почему так происходит? Венчурный капитал терпимо относится к долгосрочным стратегиям и отодвигает рентабельность на второй план, надеясь, что ее удастся достигнуть на более поздних стадиях проекта. Однако не исключено, что именно это приводит к образованию финансовых пузырей, подобных тому, который мы наблюдали во время бума доткомов в 2000 году.

В конце концов, чтобы компания была успешной в долгосрочной перспективе, ей необходимо стать рентабельной, но не все ставят перед собой такую цель. Многие основатели и инвесторы просто хотят заработать на росте и вывести деньги до того, как лопнет пузырь, — зачастую при публичной продаже компании через IPO.

Что это может означать для вас? Независимо от того, практик вы или консультант, работаете ли вы в стартапе или в компании из списка Fortune 500, выясните, что движет вашим клиентом или работодателем. Стремятся ли они к более высокой капитализации, операционной прибыли, внутренней стоимости или воспринимаемой ценности? От этого напрямую зависит, над чем вы будете работать и что хотят услышать ваши клиенты, и на основе этого вы сможете судить, способны ли вы им помочь.

Если вы хотите узнать больше о венчурном капитале, спекулятивной оценке компании и культуре стартапов, то вам прекрасно подойдет книга «The Cult of We» (https://www.cultofwe.com) авторов «Wall Street Journal» Элиота Брауна (Eliot Brown) и Морин Фаррелл (Maureen Farrell).

Когда проекты планируются в зависимости от инструментов, а не от решаемых задач, велика вероятность того, что они не преуспеют. Это значит, что вы как консультант должны оттачивать свое умение слушать и выявлять вопросы, которые клиенты затрудняются задать, не говоря уже о том, чтобы получить на

них ответы. Если крупная сеть ресторанов быстрого питания наняла вас, чтобы вы помогли разработать «стратегию искусственного интеллекта», и вы видите, что их отдел кадров спешно ищет талантливых специалистов по глубокому обучению, вам стоит спросить: «Какие проблемы вы пытаетесь решить с помощью глубокого обучения?» Если вам не удается получить четкий ответ, предложите руководству компании сделать шаг назад и оценить, с какими реальными проблемами они сталкиваются в своей отрасли. Они не могут наладить эффективный график работы персонала? Тогда им нужно не глубокое обучение, а линейное программирование! Возможно, некоторым читателям этот пример покажется элементарным, но многие нынешние руководители с трудом отличают одно от другого. Я не раз встречал поставщиков и консультантов, которые называли свои решения в области линейного программирования искусственным интеллектом, а затем смешивали их по смыслу с глубоким обучением.

На что стоит обратить внимание, устраиваясь на работу в области data science

Чтобы понять, как устроен рынок вакансий в области data science, можно провести параллели с одним классическим произведением американского телевидения.

В 2010 году в США вышел сериал «Давай еще, Тэд» («Better Off Ted»). Его эпизод «Бармаглот» («Jabberwocky») (сезон 1, эпизод 12) отразил характерное явление из мира корпоративных «модных словечек». В сериале Тед, главный герой, придумал для своей корпорации фиктивный проект «Бармаглот», чтобы скрыть просчеты рекламной кампании. Это привело к комическому эффекту: руководитель Тэда, генеральный директор, а в конце концов и вся корпорация оказались вовлечены в «работу» над «Бармаглотом», даже не зная, что это такое. Ближе к финалу тысячи сотрудников делают вид, будто занимаются этим проектом, и никому не приходит в голову уточнить, над чем же они работают на самом деле, — ведь никто не хочет признавать, что он не в курсе дел и не осведомлен о чем-то важном.

Эффектом Бармаглота часто называют ситуацию, когда отрасль или компания продвигает «модные словечки» или проекты, даже если никто не может толком объяснить, в чем их смысл. Компании могут раз за разом страдать от такого эффекта, позволяя терминам распространяться без четких определений, а сотрудникам — массово увлекаться этими терминами. В качестве примеров можно привести блокчейн, искусственный интеллект, data science, большие данные, биткойн, интернет вещей, квантовые компьютеры, NFT, «управляемые данными», облачные вычисления и «цифровую трансформацию». Даже реальные, масштабные и вполне осязаемые проекты могут превратиться в загадочные «модные словечки», которые понимают немногие, но о которых судачат все кому не лень.

Чтобы не допустить «эффекта Бармаглота», вам стоит катализировать продуктивный диалог. Интересуйтесь методами и средствами того или иного проекта или инициативы, а не только результатами и количественными показателями. Когда речь идет о должности — не нанимает ли вас компания, чтобы работать над «Бармаглотом»? Или вам все-таки предстоит заниматься практическими и конкретными проектами? Не стала ли компания жертвой «модных словечек», не привлекает ли она вас просто потому, что боится упустить какие-то новые веяния, в которых не до конца разбирается? Или у работодателей действительно есть определенные и объективные потребности, ради которых они вас нанимают? От того, насколько хорошо вы это выясните, может зависеть ваша дальнейшая карьера — будет ли она плавно идти в гору или застрянет в неловкой позиции.

С учетом этих соображений давайте рассмотрим несколько моментов, на которые стоит обратить внимание, если вы устраиваетесь на работу в области data science. Начнем с того, как очерчивается круг ваших обязанностей.

Должностные обязанности

Допустим, вас приглашают на работу в качестве специалиста по data science. Собеседование прошло отлично. Вы задали вопросы о должностных обязанностях и получили исчерпывающие ответы. Вам предложили работу, и теперь самое важное — четко представлять себе, какими проектами предстоит заниматься.

Желательно, чтобы круг ваших обязанностей был четко определен и чтобы ваша деятельность ориентировалась на осязаемые цели. Хорошо, когда не приходится гадать, над чем вам нужно работать. Еще лучше, если у вас будет руководство с целостным ви́дением, которое понимает, что нужно бизнесу. Вы сможете добиваться четко поставленных целей и будете знать своего клиента.

И наоборот, если вас взяли на эту должность, потому что подразделение хочет быть «ориентированным на данные» или иметь конкурентное преимущество в области «data science», — это тревожный сигнал. Не исключено, что от вас будут ожидать, чтобы вы сами искали проблемы, которыми будете заниматься, и делились первыми попавшимися решениями. Если вы попросите стратегического совета, вам скажут, что нужно применить «машинное обучение» к бизнесу. Разумеется, когда у вас нет ничего, кроме молотка, все вокруг начинает казаться гвоздями. Команды, которые занимаются data science, чувствуют, будто их вынуждают обеспечивать решение (например, машинное обучение) еще до того, как появляется цель или проблема, которую нужно решить. А когда обнаруживается настоящая проблема, оказывается непросто получить поддержку заинтересованных сторон и согласовать ресурсы, так что остается только переключаться с одной второстепенной задачи на другую.

Проблема в том, что вас наняли на должность, ориентируясь на модные словечки, а не на функциональные требования. Если должностные обязанности плохо

определены, это обычно приводит к другим проблемам, о которых пойдет речь далее. Давайте перейдем к организационной направленности.

Организационная направленность и заинтересованность

Еще один фактор, на который стоит обратить внимание, — это то, насколько компания согласована по конкретным целям и все ли участники проектов в них заинтересованы.

После бума data science многие компании пересмотрели свою структуру, создав центральную команду по data science. По задумке руководителей эта команда должна быть в курсе всех событий, консультировать и помогать другим отделам ориентироваться на данные и внедрять такие инновационные методы, как машинное обучение. На эту команду также могут возложить ответственность за то, чтобы сводить в единую систему разрозненные данные разных подразделений. Хотя на бумаге это выглядит многообещающе, в реальности во многих компаниях такой подход приводит к проблемам.

Почему это происходит? Руководство создает команду специалистов по data science, но не ставит перед ней четкой цели. Поэтому команда только и занимается тем, что ищет проблемы, которые может решить, и у нее нет полномочий, чтобы решать действительно актуальные проблемы. Как я уже отмечал, именно поэтому команды специалистов по data science пользуются дурной славой из-за того, что предлагают решение (например, машинное обучение) еще до того, как определились с целью. Особенно плохо они проявляют себя в качестве движущей силы, которая борется с разрозненностью данных, потому что эта задача — совершенно не по их профилю.

Устранять разрозненность данных — это работа для IT-специалистов!

Это большое заблуждение — поручать специалистам по data science устранять разрозненность данных («силос данных») в компании. Зачастую данные оказываются разрозненными потому, что в компании не налажена инфраструктура хранилища данных, и подразделения хранят данные в самодельных электронных таблицах и секретных файлах, а не в централизованной и поддерживаемой базе данных.

Если вы считаете, что силос данных — это проблема, то вам понадобятся серверы, облачные службы, сертифицированные администраторы баз данных, протоколы безопасности и рабочая группа специалистов по IT, чтобы собрать все это воедино. Это не то, что стоит поручать команде по data science, потому что у нее, скорее всего, нет необходимых навыков, бюджета и организационных полномочий, чтобы справиться с этой задачей (за исключением разве что очень маленьких организаций).

Как только проблема обнаружена, оказывается непросто получить поддержку заинтересованных сторон и распределить ресурсы. Если появляется возможность это осуществить, необходимо сильное руководство, чтобы сделать следующее.

- Задать четкую цель и составить план действий.
- Получить бюджет на то, чтобы собрать данные и поддерживать инфраструктуру.
- Наладить доступ к данным и согласовать владение данными.
- Привлечь к участию в проекте заинтересованные стороны и экспертов в предметной области.

Добиться, чтобы представители руководства, чье участие критично для проекта, находили время для важных совещаний и другой необходимой деятельности.

Выполнить эти требования *после* того, как компания наняла команду специалистов по data science, гораздо сложнее, чем до того, потому что функции этой команды очерчиваются и финансируются по факту. Если высшее руководство не выделило ресурсы и не заручилось поддержкой всех необходимых сторон, то проект по внедрению data science не будет успешным. Именно поэтому публикуется столько статей, которые обвиняют компании в том, что те не готовы к data science, — например, в «Harvard Business Review» (https://oreil.ly/IlicW) или «MIT Sloan Management Review» (https://oreil.ly/U9C9F).

Лучше работать в команде по data science, которая организационно относится к тому же подразделению, что и ее заказчик. Тогда информация, финансы и коммуникация циркулируют свободнее и слаженнее. В этом случае в коллективе меньше напряженности, потому что все участники одного проекта находятся в одной команде и гораздо меньше подвержены внутрикорпоративной конкуренции.

Доступ к данным — вопрос корпоративной политики

Ни для кого не секрет, что организации ревностно охраняют свои данные, но это связано не только с соображениями безопасности или недоверия. Данные — важный актив в политических играх внутри компании, и многие сотрудники неохотно делятся данными даже со своими ближайшими коллегами. Даже подразделения одной и той же организации скрывают свои данные друг от друга по одной простой причине: они не хотят, чтобы другие выполняли их работу, а тем более выполняли ее неправильно. Они рассуждают так: чтобы интерпретировать данные, могут потребоваться *наши* профессиональные навыки, которые подразумевают знания в *нашей* предметной области. В конце концов, наши данные — это наше дело! И если вы просите доступ к *нашим* данным, значит, вы хотите влезть в *нашу* сферу деятельности.

Кроме того, специалисты по data science иногда переоценивают свою способность интерпретировать чужие наборы данных и ошибочно считают, что их знаний хватит, чтобы использовать эти данные. Чтобы преодолеть это затруднение, нужно установить доверительные отношения со всеми привлеченными экспертами в предметной области, договориться о том, как будут передаваться знания, и, если понадобится, предоставить им значительную роль в проекте.

Необходимые ресурсы

Еще один риск, которого стоит остерегаться, заключается в том, что у вас не будет нужных ресурсов, чтобы справляться со своими задачами. Неприятно, когда вас нанимают на работу, но не снабжают всем необходимым, чтобы ее выполнять. Конечно, находчивость и профессиональная смекалка — бесценные качества. Но даже самый смекалистый разработчик или специалист по data science может быстро «упереться в потолок». Иногда для работы нужны вещи, которые стоят денег, а работодатель не хочет их выделять.

Допустим, вам нужна база данных, чтобы делать прогнозы. Соединение с базой данных стороннего производителя работает кое-как, с частыми обрывами и простоями. Последнее, что вы хотите услышать в такой ситуации, — это «сделайте так, чтобы все работало», потому что это не задача специалиста по data science. Допустим, вы приходите к выводу, что можно было бы реплицировать базу данных локально, но для этого нужно сохранять 40 Гб данных в день, а значит, требуется сервер или облачный ресурс. Так вы и «упираетесь в потолок»: специалист по data science превращается в отдел IT, но без соответствующего бюджета!

В таких ситуациях приходится изобретать решения, чтобы сократить расходы без ущерба для проекта. Может быть, хранить только самые свежие данные и удалять остальные? А может, создать какой-нибудь сценарий на Python, который будет обрабатывать ошибки и восстанавливать соединение, когда связь обрывается, и при этом разбивать данные на пакеты так, чтобы загрузка возобновлялась начиная с последнего успешно переданного пакета?

Если вам кажется, что это слишком редкая проблема и слишком специфическое решение, — имейте в виду, что мне приходилось этим заниматься, и мои меры сработали! Приятно придумывать обходные решения и оптимизировать процесс без дополнительных затрат. Но тем не менее для многих проектов по работе с данными могут понадобиться конвейеры данных, серверы, кластеры, рабочие станции на базе GPU и другие вычислительные ресурсы, которых не может обеспечить обычный настольный компьютер. Другими словами, все это стоит денег, а у вашей организации может не оказаться нужного бюджета.

А как же математическое моделирование?

Если вы задаетесь вопросом, как так получилось, что вас наняли заниматься регрессией, статистикой, машинным обучением и прочей прикладной математикой, а вы оказались «айтишником под прикрытием», — то знайте, что в нынешнем корпоративном мире это не редкость.

Впрочем, вы работаете с данными, а это неявно подразумевает, что вам приходится касаться задач из области IT. Главное — убедиться, что ваши навыки не расходятся с должностными обязанностями и требуемыми результатами. Об этом мы будем говорить на протяжении всей оставшейся части этой главы.

Разумные цели

Это важный момент, который стоит проконтролировать. В атмосфере, полной шумихи и громких обещаний, легко столкнуться с неадекватными целями.

Бывают случаи, когда руководитель нанимает специалиста по data science и ожидает, что он гладко впишется в рабочую среду и будет приносить компании громадную пользу. Безусловно, так и будет, когда в компании выполняется много ручной работы и есть большой потенциал для автоматизации. Например, если компания обсчитывает все в электронных таблицах, а прогнозы делаются на основе чистых догадок, то специалист по data science сможет отлично проявить себя, если упорядочит процессы в единой базе данных и улучшит прогнозы даже с помощью простых регрессионных моделей.

Однако если компания нанимает специалиста по data science, чтобы внедрить в свои программные продукты машинное обучение, которое будет распознавать объекты на изображениях, все становится сложнее. Опытный специалист объяснит руководству, что это начинание обойдется по меньшей мере в сотни тысяч долларов! Понадобится бюджет не только на то, чтобы раздобыть обучающие изображения, но и на то, чтобы оплачивать ручной труд операторов, которые будут размечать объекты на них (https://oreil.ly/ov7S5). И все это только на этапе сбора данных!

Вовсе не удивительно, когда специалист по data science тратит первые 18 месяцев своей работы на то, чтобы объяснить руководству, что он до сих пор не закрыл задачу, потому что он все еще пытается собрать и подготовить данные, что отнимает 95 % усилий, связанных с машинным обучением. Руководство может испытать разочарование, потому что оно доверилось расхожему представлению

о том, будто машинное обучение и искуственный интеллект избавят от ручной работы, — а потом обнаружило, что променяло одну ручную работу на другую — подготовку размеченных данных.

Поэтому держитесь подальше от среды, в которой ставятся неадекватные цели, и находите дипломатичные способы согласовывать ожидания руководства с реальностью, особенно когда другие обещают кнопку «Сделать все и сразу». Даже в авторитетных деловых журналах и высказываниях дорогостоящих консультантов по управлению часто встречаются утверждения о том, что сверхинтеллектуальный искусственный интеллект уже не за горами. Менеджеры, у которых нет технической подготовки, могут стать жертвами этого раскрученного нарратива.

Кому это выгодно?

Этот вопрос стоит задать, если вы пытаетесь понять, почему люди поступают тем или иным образом, когда «эффект Бармаглота» в самом разгаре. Если СМИ продвигают истории об искусственном интеллекте, кто от этого выигрывает? Независимо от того, кто окажется основным бенефициаром, имейте в виду, что СМИ тоже извлекают выгоду из кликов и доходов от рекламы. Высокооплачиваемые консалтинговые компании начисляют больше рабочих часов за «стратегию ИИ». Производитель микросхем может рекламировать глубокое обучение, чтобы продавать больше видеокарт, а облачные платформы зарабатывают на дисковом пространстве и процессорном времени для проектов машинного обучения.

Что объединяет всех этих бенефициаров? Дело не только в том, что тема искусственного интеллекта помогает им продавать свои продукты, но и в том, что у них нет долгосрочной заинтересованности в том, чтобы их клиенты добивались успеха. Они продают не результат проекта, а единицы продукции, подобно тому, как продавали лопаты во времена золотой лихорадки.

Однако я не утверждаю, будто СМИ и поставщики ведут себя неэтично. Их сотрудникам нужно зарабатывать деньги для своей компании и обеспечивать свои семьи. Обещания, с помощью которых они продвигают свои продукты, могут быть вполне обоснованными и выполнимыми. Однако слово не воробей; стоит учитывать, что если компания что-то пообещала, то потом трудно дать обратный ход, даже если обещание оказалось неосуществимым. Многие компании скорее сменят вектор своих усилий, чем признают, что их посулы не оправдались. Так что помните об этой тенденции и всегда задавайте вопрос: «Кому это выгодно?»

Конкуренция с существующими системами

Это предостережение можно было бы отнести к теме разумных целей, но я считаю, что такая ситуация настолько распространена, что ее стоит рассмотреть отдельно. Возникает поначалу незаметная, но пагубная проблема, если ваши трудовые функции конкурируют с существующей системой, которая вполне справляется со своими задачами. Такие случаи могут возникать в рабочей среде, где у сотрудников мало реальной работы, но им нужно выглядеть постоянно занятыми.

Допустим, несколько лет назад ваш работодатель закупил систему, которая прогнозирует объем продаж. Теперь руководитель поручает вам усовершенствовать эту систему, повысив точность на 1 %.

Вы заметили, в чем здесь статистическая проблема? Если вы читали главу 3, то понимаете, что 1 % вряд ли будет статистически значимым, и случайность может легко обеспечить этот 1 % без каких-либо усилий с вашей стороны. И наоборот, она может изменить расклад в другую сторону, и невидимая рука рынка, которую вы никак не контролируете, сведет на нет эффект от ваших разработок. Если квартал продаж был неудачным, и вмешались не зависящие от вас факторы (например, на рынок вашей компании вышли конкуренты), выручка может снизиться на 3 %, а не повыситься на 1 %, который вам пришлось «нацедить» с помощью *p*-хакинга.

Главная проблема здесь, помимо дублирования работы, заключается в том, что результат находится вне вашего контроля. Эта ситуация может оказаться весьма неприятной. Одно дело, если существующая система, с которой вы конкурируете, неисправна, нефункциональна или предусматривает ручную работу без всякой автоматизации. Но конкурировать с системой, которая нормально работает, — значит копать себе яму. Если есть возможность, не приближайтесь к таким проектам на пушечный выстрел.

«ЧТО Я ЗДЕСЬ ДЕЛАЮ?»

Может ли случиться так, что специалиста по data science нанимают на работу, которая не приносит никакой пользы, несмотря на добросовестный труд и старания? К сожалению, да: факторы, которые не зависят от сотрудника, могут перечеркнуть самую лучшую работу, и стоит следить, чтобы не попасть в такое положение.

Комедия Майка Джаджа (Mike Judge) «Офисное пространство» 1999 года стала культовой для многих офисных работников в США. В фильме главный герой Питер Гиббонс, сотрудник отдела IT, повреждается умом из-за того, что должен отчитываться перед восемью разными менеджерами. Когда консультанты по оптимизации кадров спрашивают, чем он занима-

ется в течение рабочего дня, он честно отвечает, что примерно 15 минут он занимается «настоящей, реальной работой». Я не буду спойлерить для тех, кто не смотрел фильм, но, как и в любой хорошей комедии, развязка сюжета окажется весьма неожиданной.

Развивая предыдущий пример, можно сказать, что дублировать систему, которая работает, — это то, что покойный антрополог Дэвид Грэбер (David Graeber) назвал бы *бредовой работой* (bullshit job). Согласно Грэберу, это оплачиваемая работа, которая настолько бессмысленна, напрасна или попросту вредна, что даже сам работник не верит, что от нее есть какой-то прок, но вынужден это скрывать. В своей книге «Бредовая работа» и одноименной вирусной статье 2013 года (https://strikemag.org/bullshit-jobs/) Грэбер замечает, что такая работа становится настолько обыденной, что наносит психологическую травму рабочей силе и экономике.

Хотя работы Грэбера изобилуют смещением из-за самоотбора и субъективными выводами, а отсутствие эмпирических доказательств дает повод для критики, трудно утверждать, будто такой «бредовой» работы вовсе не существует. В защиту Грэбера можно сказать, что эмпирически это явление трудно измерить, и мало кто из работников будет честно высказываться на эту тему, чтобы не поставить под угрозу свою карьеру.

Защищена ли работа в области data science от таких проблем? Кэсси Козырькова, бывший старший научный консультант по принятию решений в Google, делится характерным случаем из жизни (https://oreil.ly/fwPKn), который помогает ответить на этот вопрос.

Несколько лет назад один знакомый технический директор, который работал в сфере высоких технологий, сетовал на своих «бесполезных» специалистов по data science. Я ему сказала: «Мне кажется, ты нанимаешь таких специалистов так же, как наркобарон покупает тигра для своего садового участка. Ты не знаешь, зачем тебе тигр, но у всех остальных наркобаронов он уже есть».

Вот тебе и здрасьте. Может быть, руководство нанимает специалистов по data science, чтобы повысить авторитет компании и ее корпоративную репутацию? Если вы оказались на работе, которая не позволяет вам создавать реальную ценность, подумайте, как вы можете повлиять на то, чтобы ситуация изменилась к лучшему. Можете ли вы создать благоприятные условия, а не ждать, пока их обеспечат другие? Можете ли вы взять на себя ответственность за то, чтобы реализовать намеченные инициативы и таким образом продвинуть свою карьеру? Если это невозможно, будьте готовы искать другие варианты трудоустройства.

Должность не соответствует вашим ожиданиям

Что делать, если вы приступили к работе и обнаружили, что она не соответствует вашим ожиданиям? Например, вам пообещали, что ваша деятельность будет связана со статистикой и машинным обучением, но вместо этого вам приходится больше заниматься задачами из области IT, потому что работа с данными в организации просто недостаточно развита для машинного обучения.

Возможно, вам удастся превратить недостатки в достоинства. Не исключено, что вас даже устроит превращение из специалиста по data science в специалиста по IT, и, может быть, в процессе этого превращения вы овладеете навыками программирования и работы с базами данных. Вы даже можете стать незаменимым экспертом по SQL или техническим гуру, и это положительно скажется на вашем профессиональном развитии. Приводя в порядок управление данными и рабочие процессы в компании, вы тем самым подготавливаете ее к более сложным задачам в будущем. Пока ваша трудовая деятельность идет гладко, вы можете выделять время на то, чтобы учиться и профессионально развиваться в том, что вам интересно.

Однако если вы рассчитывали заниматься статистическим анализом и машинным обучением, а вместо этого отлаживаете неисправные электронные таблицы, базы Microsoft Access и макросы VBA, вы можете испытать разочарование. В такой ситуации можно попробовать инициировать перемены. Предлагайте идеи о том, как модернизировать инструментарий, перейти на Python и внедрить современную платформу баз данных — например, MySQL или даже SQLite. Если вы сможете этого добиться, то по крайней мере создадите задел, который поможет внедрять инновации, и существенно приблизитесь к тому, чтобы применять идеи из этой книги. Это также принесет пользу компании, поскольку инструменты станут более гибкими и будут лучше поддерживаться, а найти хороших специалистов по современному Python проще, чем по таким устаревшим технологиям, как Microsoft Access или VBA.

ЧТО ТАКОЕ ТЕНЕВЫЕ IT?

Под *теневыми IT* (https://oreil.ly/9ZDb8) обычно понимают деятельность сотрудников, которые разрабатывают цифровые системы в обход своего отдела IT. К этим системам могут относиться базы данных, скрипты и процессы, а также программное обеспечение, которое сотрудники приобрели или разработали сами без участия отдела IT.

Раньше теневые IT в компаниях не одобрялись, потому что их работа не регламентирована и происходит без ведома отдела IT. Конечно, когда бухгалтерия, отдел маркетинга или другие «неайтишные» подразделения налаживают свое доморощенное IT, это может привести к скрытым

издержкам компании в виде неэффективных рабочих процессов и проблем с безопасностью. Могут возникнуть и неприятные трения, когда отделы IT и не IT вступают в конфликт, и каждая сторона обвиняет другую в том, что она выходит за рамки своих полномочий или просто искусственно создает себе объем работ, чтобы не увольнять ненужных сотрудников.

Однако одно из преимуществ развития data science заключается в том, что благодаря ему теневые IT стали восприниматься как необходимый фактор инноваций. Сотрудники, которые не специализируются на компьютерных технологиях, тоже могут создавать прототипы и экспериментировать с наборами данных, сценариями на Python и регрессионными моделями. В свою очередь, отдел IT может перенять эти разработки и поддерживать их официально по мере развития. Это также позволяет бизнесу более гибко подстраиваться к рыночной ситуации. Чтобы изменить бизнес-логику, достаточно быстро поправить сценарий на Python или «самопальную» базу данных, а не подавать заявку в службу поддержки IT. Правда, при этом изменение не подвергается тщательному тестированию и внедряется в обход бюрократических процедур — но это может оказаться разумным решением в пользу оперативности.

В целом, если вы окажетесь в роли теневого IT-специалиста (а это вполне вероятно), убедитесь, что вы вполне понимаете риски и наладили хорошие отношения с отделом IT. Если вы добьетесь успеха, такая работа может взбодрить компанию и принести ей пользу. Если вы предчувствуете потенциальный конфликт с отделом IT, откровенно расскажите об этом своему руководству. Если вы объясните, что ваша работа — это «прототипирование» и «исследование» (особенно если это правда), то начальство может согласиться, что она не относится к компетенции отдела IT. Однако никогда не действуйте без поддержки руководства: пусть оно само занимается вопросами взаимодействия между подразделениями.

А существует ли работа вашей мечты?

Хотя вы всегда можете отказаться от работы, которая не соответствует вашим ожиданиям, не забудьте оценить, насколько они реалистичны. Может быть, современные технологии, на которые вы ориентируетесь, — слишком современные?

Возьмем, например, обработку естественного языка. Допустим, вы хотите разрабатывать чат-боты на основе глубокого обучения. Однако для этой работы не так уж много реальных вакансий, потому что у большинства компаний нет

практической потребности в чат-ботах. Дело в том, что на момент написания этой книги они еще недостаточно развиты. Хотя у таких компаний, как OpenAI, есть любопытные опытные разработки вроде GPT-3 (https://openai.com/blog/gpt-3-apps), они в основном представляют собой именно предварительные опытные разработки[1]. В конечном счете GPT-3 — это вероятностный распознаватель шаблонов, который соединяет слова в цепочки и не обладает человеческим здравым смыслом. Это подтверждается исследованиями, в том числе работами Гэри Маркуса (Gary Marcus) из Нью-Йоркского университета (https://oreil.ly/fxakC).

Это означает, что разработка чат-ботов для массового применения остается открытой проблемой, и подавляющее большинство компаний еще не воспринимает их как фактор добавленной ценности. Если вы действительно стремитесь заниматься обработкой естественного языка, но не видите перспектив карьерного роста, то, может быть, вам будет лучше всего работать в научной сфере и проводить исследования. Хотя существуют компании вроде Alphabet, которые занимаются НИОКР, многие из их сотрудников пришли из академических кругов.

Поэтому, когда вы осматриваетесь на рынке труда, будьте реалистичны в своих ожиданиях. Если они превосходят то, что может предоставить рынок, подумайте о научной карьере. Вам также следует рассмотреть это направление, если для работы вашей мечты требуется ученая степень или специализированное образование, без которого не принимают на работу.

Куда же податься?

Теперь, когда мы рассмотрели спектр возможностей в области data science, куда стоит двигаться дальше? И что ждет data science в будущем?

Для начала подумайте о том, какие осложнения связаны с должностью специалиста по data science. От такого сотрудника часто неявно ожидают широкой квалификации без четко очерченных границ — в основном из-за того, что у этой должности нет стандартизированного определения с ограниченной областью

[1] Англоязычное издание книги вышло в июне 2022 года. В ноябре 2022 года компания OpenAI выпустила в массовое обращение чат-бот ChatGPT на основе GPT-3.5. На момент выхода русского перевода широко используется модель GPT-4 — в частности, в продуктах Microsoft, в системе управления знаниями Morgan Stanley и в проектах правительства Исландии.

Хотя модели GPT показывают впечатляющие результаты, они также вызывают все больше опасений по части дискриминации, политических и религиозных тем, рисков в сфере юриспруденции и медицины, эффекта привязки и других когнитивных смещений. На момент выхода русского перевода большие языковые модели (LLM) для обработки естественного языка стремительно развиваются. Ситуация в индустрии меняется с каждым днем, и многие преимущества и недостатки чат-ботов еще только предстоит обнаружить. — *Примеч. науч. ред.*

ответственности. Если мы и научились чему-то за последние 10 лет, наблюдая за тем, как развивается data science, так это тому, что от определения многое зависит. Современный специалист по data science все больше становится похож на программиста, который владеет статистикой, оптимизацией и машинным обучением. У сотрудника, который занимается data science, может даже не быть такого словосочетания в названии должности. Несмотря на то, что требования к специалисту стали гораздо шире, чем тогда, когда data science была объявлена «самой привлекательной работой XXI века», необходимо обладать соответствующими навыками, чтобы устроиться на хорошую позицию.

Другой вариант — перейти на более узкую специализацию, что в последние годы происходит все чаще. Вновь становятся популярными такие должности, как инженер по компьютерному зрению, инженер данных, аналитик данных, исследователь, аналитик по исследованию операций и консультант. Мы все реже встречаемся с должностью «специалист по data science», и, скорее всего, эта тенденция сохранится в ближайшие 10 лет, в первую очередь из-за специализации. Нет ничего плохого в том, чтобы следовать этой тенденции.

Важно отметить, что рынок труда сильно изменился, и именно поэтому вам пригодятся конкурентные преимущества, о которых шла речь в этой главе. Если в 2014 году специалисты по data science считались «звездами» и получали шестизначные зарплаты (в долларах), то сегодня на аналогичную вакансию в любой компании могут поступать сотни или тысячи заявок, притом что годовая зарплата будет всего лишь пятизначной. Многочисленные университетские программы и учебные курсы по data science привели к огромному буму предложения специалистов в этой области, и среди соискателей возникает острая конкуренция. Именно поэтому не стоит сбрасывать со счетов такие вакансии, как «аналитик», «специалист по исследованию операций» и «разработчик программного обеспечения». Пожалуй, лучше всех об этом сказала Вики Бойкис (Vicki Boykis), старший инженер по машинному обучению в Tumblr, в своей статье «Data science is different now» (https://oreil.ly/vm8Vp):

> Помните, что конечная цель состоит в том, чтобы обскакать толпы студентов, которые получили диплом в области data science, закончили интенсивные курсы и просмотрели терабайты обучающих видеороликов.
>
> Вам стоит ухватиться за удачную возможность и получить должность, которая хоть как-то связана с данными, а затем продвигаться к работе своей мечты, впитывая при этом как можно больше знаний об индустрии высоких технологий в целом.
>
> Не впадайте в аналитический паралич. Выберите небольшой участок деятельности и начните с него. Делайте что-то небольшое. Научитесь чему-то небольшому, разработайте что-то небольшое. Доносите это до других людей. Имейте в виду, что ваша первая должность в области data science, скорее всего, будет называться не «специалист по data science».

Заключение

Эта глава отличается от остальных глав книги, но она важна, если вы хотите ориентироваться на рынке трудоустройства в области data science и эффективно применять полученные знания. Вряд ли вам будет приятно изучать статистические инструменты и машинное обучение только для того, чтобы обнаружить, что большинство вакансий побуждают вас переключиться на другую работу. Если так случится, рассматривайте эту ситуацию как возможность продолжить обучение и приобрести новые умения. Когда вы объедините свои фундаментальные математические знания с навыками программирования и разработки программного обеспечения, вы станете бесценным специалистом, потому что сможете преодолеть разрыв между IT и data science.

Обращайте внимание не на рекламную шумиху, а на практические решения и не замыкайтесь на чисто технических аспектах, чтобы не пострадать от «невидимой руки рынка». Старайтесь понимать мотивы руководства, а также людей в целом. Интересуйтесь, *почему* тот или иной метод или инструмент решает проблему, а не только *как* он работает с технической точки зрения.

Учитесь не ради того, чтобы учиться, а ради того, чтобы наращивать свои возможности и подбирать правильные инструменты, которые позволяют решать правильно поставленные задачи. Один из самых эффективных способов обучения — взяться за проблему, которая вам интересна (а не зацикливаться на определенном инструменте). Потянув за эту ниточку, вы обнаружите еще одну интересную тему, а потом еще одну, и еще. Не упускайте из виду поставленную цель, продолжайте углубляться в нужные темы и вовремя отделывайтесь от ненужных. Такой подход оправдывается с лихвой и позволяет получить удивительно много знаний и опыта за короткое время.

ПРИЛОЖЕНИЕ А

Дополнительные материалы

Как использовать верстку LaTeX для выражений SymPy

По мере того как вы осваиваетесь с математическими обозначениями, вам может быть полезно отображать выражения SymPy в виде полноценных формул.

Самый быстрый способ сделать это — вызвать для выражения функцию `latex()` из библиотеки SymPy, а затем скопировать результат в программу просмотра формул LaTeX.

В примере A1 мы берем простое выражение и преобразуем его в строку в формате LaTeX. Конечно, выражения с производными, интегралами и другими операциями, которые доступны в SymPy, тоже можно представить в LaTeX. Но пусть этот пример останется простым.

Пример A1. Использование SymPy для преобразования выражения в формат LaTeX

```
from sympy import *

x,y = symbols('x y')

z = x**2 / sqrt(2*y**3 - 1)

print(latex(z))
# выводит: \frac{x^{2}}{\sqrt{2 y^{3} - 1}}
```

Строка `\frac{x^{2}}{\sqrt{2 y^{3} - 1}}` представлена в формате исходного кода LaTeX, и существует множество инструментов и форматов документов, которые можно настроить, чтобы они поддерживали этот формат. Но чтобы

просто отобразить формулу, воспользуйтесь редактором формул LaTeX. Вот два онлайн-редактора, которыми я пользуюсь:

Lagrida LaTeX Equation Editor (https://latexeditor.lagrida.com);
CodeCogs Equation Editor (https://latex.codecogs.com).

На рис. А1 показано, как отображается математическое выражение в онлайн-редакторе Lagrida LaTeX.

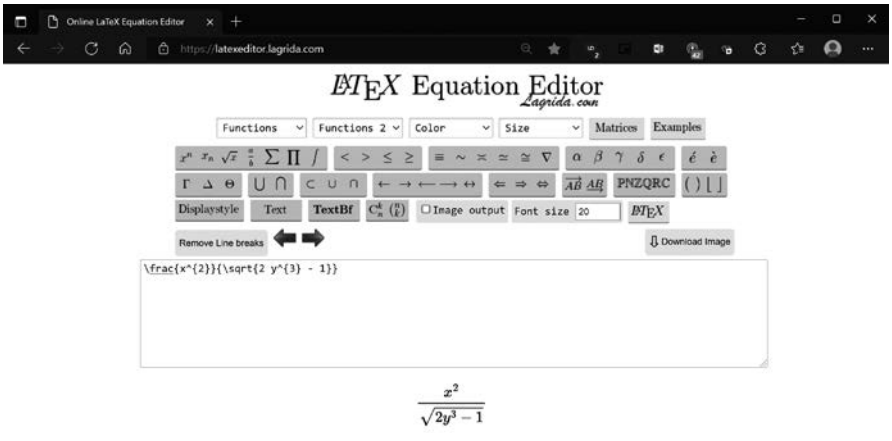


Рис. А1. Вывод формулы в формате LaTeX в онлайн-редакторе формул

Если вы хотите обойтись без копирования и вставки, можно добавить строку в формате LaTeX непосредственно в качестве аргумента к адресу URL редактора CodeCogs LaTeX, как показано в примере А2. В этом случае математическое выражение отобразится в вашем браузере.

Пример А2. Отображение формулы, записанной в формате LaTeX, с помощью CodeCogs

```
import webbrowser
from sympy import *

x,y = symbols('x y')

z = x**2 / sqrt(2*y**3 - 1)

webbrowser.open("https://latex.codecogs.com/png.image?\dpi{200}" + latex(z))
```

Если вы используете Jupyter, в нем можно отображать формулы LaTeX с помощью специальных плагинов (https://oreil.ly/mWYf7).

Биномиальное распределение с нуля

Если вы хотите реализовать биномиальное распределение с нуля, то все необходимое для этого можно найти в примере А3.

Пример А3. Построение биномиального распределения с нуля

```
# Факториал — это произведение чисел от 1 до n
# Пример: 5! = 5 × 4 × 3 × 2 × 1
def factorial(n: int):
    f = 1
    for i in range(n):
        f *= (i + 1)
    return f

# Генерирует коэффициенты, которые необходимы для биномиального распределения
def binomial_coefficient(n: int, k: int):
    return factorial(n) / (factorial(k) * factorial(n - k))

# Биномиальное распределение показывает вероятность наступления k успехов
из n попыток,
# в каждой из которых вероятность успеха равна p
def binomial_distribution(k: int, n: int, p: float):
    return binomial_coefficient(n, k) * (p ** k) * (1.0 - p) ** (n - k)

# 10 попыток, в каждой из которых вероятность успеха составляет 90 %
n = 10
p = 0.9

for k in range(n + 1):
    probability = binomial_distribution(k, n, p)
    print(f"{k} - {probability}")
```

С помощью функций `factorial()` и `binomial_coefficient()` можно построить биномиальную функцию распределения с нуля. Факториал — это произведение последовательных целых чисел от 1 до n. Например, факториал 5 (обозначается 5!) будет равен $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$.

Функция `binomial_coefficient()` возвращает количество *сочетаний* из *n* по *k*, то есть количество всех подмножеств размера *k* в множестве размера *n* без учета порядка. Если $k = 2$, а $n = 3$, это значит, что нужно подсчитать количество неупорядоченных пар в множестве из трех элементов. Например, из множества $\{1,2,3\}$ можно составить пары $\{1,3\}$, $\{1,2\}$ и $\{2,3\}$. Поскольку здесь 3 сочетания, то биномиальный коэффициент будет равен 3. Разумеется, функция `binomial_coefficient()` позволяет не возиться со всеми этими перестановками, а использовать факториалы и арифметические операции.

В реализации функции `binomial_distribution()` обратите внимание на то, что биномиальный коэффициент умножается на вероятность того, что успех (вероятность которого равна `p`) произойдет `k` раз (множитель `p ** k`), и на вероятность того, что неудача (вероятность которой равна `1.0 - p`) произойдет `n - k` раз (множитель `(1.0 - p) ** (n - k)`). В результате получается вероятность того, что успех произойдет ровно `k` раз из `n` попыток.

Бета-распределение с нуля

Если вам интересно, как реализовать бета-распределение с нуля, вам пригодится функция `factorial()`, которую мы использовали для биномиального распределения, а также функция `approximate_integral()`, которая упоминается в главе 2.

Аналогично тому, как мы делали это в главе 1, мы укладываем прямоугольники под кривой на интервале, который нас интересует (см. рис. А2).

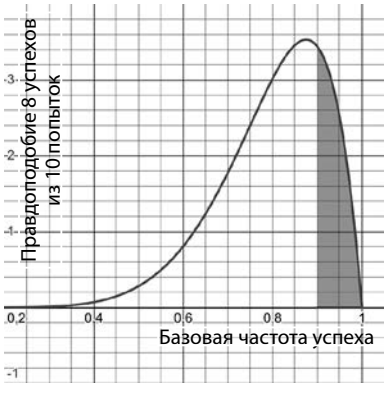


Рис. А2. Искомая вероятность — это закрашенная часть площади под кривой

Чем больше прямоугольников мы укладываем, тем точнее результат. Давайте реализуем функцию `beta_distribution()`, которая строит бета-распределение с нуля, и интегрируем 1 000 прямоугольников в интервале от 0,9 до 1, как показано в примере А4.

Пример А4. Бета-распределение с нуля

```
# Факториал — это произведение чисел от 1 до n
# Пример: 5! = 5 × 4 × 3 × 2 × 1
def factorial(n: int):
     f = 1
     for i in range(n):
         f *= (i + 1)
     return f
```

```
def approximate_integral(a, b, n, f):
    delta_x = (b - a) / n
    total_sum = 0

    for i in range(1, n + 1):
        midpoint = 0.5 * (2 * a + delta_x * (2 * i - 1))
        total_sum += f(midpoint)

    return total_sum * delta_x

def beta_distribution(x: float, alpha: float, beta: float) -> float:
    if x < 0.0 or x > 1.0:
        raise ValueError("x должен быть между 0.0 и 1.0")

    numerator = x ** (alpha - 1.0) * (1.0 - x) ** (beta - 1.0)
    denominator = (1.0 * factorial(alpha - 1) * factorial(beta - 1)) / \
        (1.0 * factorial(alpha + beta - 1))

    return numerator / denominator

greater_than_90 = approximate_integral(a=.90, b=1.0, n=1000,
    f=lambda x: beta_distribution(x, 8, 2))
less_than_90 = 1.0 - greater_than_90

print(f"Вероятность больше 90 %: {greater_than_90}")
print(f"Вероятность меньше 90 %: {less_than_90}")
```

Вы наверняка заметили, что в функции `beta_distribution()` мы указываем заданное правдоподобие `x` (левую границу каждого прямоугольника), значение `alpha` (количество успехов) и значение `beta` (количество неудач). Функция возвращает площадь прямоугольника, который соответствует координате `x`. Но, опять же, чтобы получить вероятность вероятности `x`, нужно найти площадь для интервала значений `x`.

К счастью, у нас есть готовая функция `approximate_integral()` из главы 2. С ее помощью можно вычислить вероятность того, что вероятность успеха больше 90 %, а также меньше 90 %, как показано в последних нескольких строках кода.

Вывод формулы Байеса

Если вы хотите разобраться, почему формула Байеса работает, а не просто поверить мне на слово, давайте проведем мысленный эксперимент. Допустим, мы изучаем генеральную совокупность из 100 000 человек. Умножим это значение на известные вероятности, чтобы получить количество людей, которые пьют кофе, и количество людей, которые больны раком:

$$N = 100\,000;$$

$$P(\text{Кофе}) = 0,65;$$

$$P(\text{Рак}) = 0,005;$$

$$\text{Пьют кофе} = 65\,000;$$

$$\text{Больны раком} = 500.$$

Итак, у нас есть 65 000 любителей кофе и 500 больных раком. Сколько из этих 500 больных пьют кофе? Поскольку дана условная вероятность $P(\text{Кофе} \mid \text{Рак})$, ее можно умножить на это количество больных, что даст нам 425 больных раком, которые пьют кофе:

$$P(\text{Кофе} \mid \text{Рак}) = 0,85;$$

$$\text{Любители кофе, которые больны раком} = 500 \times 0,85 = 425.$$

А какова доля любителей кофе среди тех, кто болен раком? Что на что нужно разделить, чтобы получить эту величину? Мы уже знаем, сколько людей пьют кофе *и* больны раком. Поэтому можно разделить это число на общее количество любителей кофе:

$$P(\text{Рак} \mid \text{Кофе}) = \frac{\text{Любители кофе, которые больны раком}}{\text{Любители кофе}} = \frac{425}{65000} = 0,006538$$

Подождите минутку, неужели мы только что перевернули нашу условную вероятность? Именно так! Мы начали с вероятности $P(\text{Кофе} \mid \text{Рак})$, а закончили вероятностью $P(\text{Рак} \mid \text{Кофе})$. Мы взяли два подмножества исходной генеральной совокупности (65 000 любителей кофе и 500 больных раком), применили совместную вероятность с использованием заданной условной вероятности и получили, что 425 человек из всей совокупности одновременно пьют кофе и больны раком. Затем мы разделили это число на количество любителей кофе, чтобы получить вероятность заболеть раком при условии, что человек пьет кофе.

Но где же здесь формула Байеса? Давайте вернемся к вероятности $P(\text{Рак} \mid \text{Кофе})$ и подставим в ее формулу все выражения, которые мы вычислили ранее:

$$P(\text{Рак} \mid \text{Кофе}) = \frac{100\,000 \times P(\text{Рак}) \times P(\text{Кофе} \mid \text{Рак})}{100\,000 \times P(\text{Кофе})}$$

Обратите внимание, что размер генеральной совокупности $N = 100\,000$ есть и в числителе, и в знаменателе, поэтому он сокращается. Теперь формула выглядит знакомо?

$$P(\text{Рак} \mid \text{Кофе}) = \frac{P(\text{Рак}) \times P(\text{Кофе} \mid \text{Рак})}{P(\text{Кофе})}$$

Это и есть формула Байеса!

$$P(A \mid B) = \frac{P(B) \times P(B \mid A)}{P(A)}$$

Если вы запутались в формуле Байеса или не понимаете, как она выводится, попробуйте брать из фиксированной генеральной совокупности разные подмножества в соответствии с заданными вероятностями. Отталкиваясь от них, вы сможете проследить, как перевернуть условную вероятность.

Как построить функцию распределения (CDF) и обратную к ней функцию с нуля

Безусловно, чтобы вычислить площадь под кривой нормального распределения, можно использовать метод укладки прямоугольников, который мы изучали в главе 1 и применяли к бета-распределению ранее в этом приложении. Этот метод не требует функции распределения (CDF), а просто укладывает прямоугольники под функцией плотности вероятности (PDF). Используя его с 1 000 прямоугольников, можно найти вероятность того, что золотистый ретривер весит от 61 до 62 фунтов, как показано в примере A5.

Пример A5. Площадь под кривой нормального распределения на Python

```
import math

def normal_pdf(x: float, mean: float, std_dev: float) -> float:
    return (1.0 / (2.0 * math.pi * std_dev ** 2) ** 0.5) * \
            math.exp(-1.0 * ((x - mean) ** 2 / (2.0 * std_dev ** 2)))

def approximate_integral(a, b, n, f):
    delta_x = (b - a) / n
    total_sum = 0

    for i in range(1, n + 1):
        midpoint = 0.5 * (2 * a + delta_x * (2 * i - 1))
        total_sum += f(midpoint)

return total_sum * delta_x

p_between_61_and_62 = approximate_integral(a=61, b=62, n=7,
  f=lambda x: normal_pdf(x,64.43,2.99))

print(p_between_61_and_62) # 0.0825344984983386
```

Таким образом, мы получим вероятность примерно 8,25 % того, что золотистый ретривер весит от 61 до 62 фунтов. Но если мы хотим использовать функцию распределения, которая уже проинтегрирована и не требует возиться с прямоугольниками, ее можно определить с нуля, как показано в примере А6.

Пример А6. Использование функции распределения (CDF) на Python

```
import math

def normal_cdf(x: float, mean: float, std_dev: float) -> float:
    return (1 + math.erf((x - mean) / math.sqrt(2) / std_dev)) / 2

mean = 64.43
std_dev = 2.99

x = normal_cdf(66, mean, std_dev) - normal_cdf(62, mean, std_dev)

print(x) # 0.49204501470628936
```

Функция `math.erf()` известна как функция ошибок и часто используется, чтобы вычислять функции распределений. Наконец, чтобы вычислить PPF (функцию, обратную CDF) с нуля, нужно применить функцию `erfinv()`, которая обратна `erf()`. В примере А7 генерируется 1 000 случайных значений массы тела золотистого ретривера с помощью PPF, созданной с нуля.

Пример А7. Случайно сгенерированная выборка масс тела золотистых ретриверов

```
import random
from scipy.special import erfinv

def inv_normal_cdf(p: float, mean: float, std_dev: float): # функция PPF
    return mean + (std_dev * (2.0 ** 0.5) * erfinv((2.0 * p) - 1.0))

mean = 64.43
std_dev = 2.99

for i in range(0,1000):
    random_p = random.uniform(0.0, 1.0)
    print(inv_normal_cdf(random_p, mean, std_dev))
```

Как применять число *e*, чтобы прогнозировать вероятность события во времени

Рассмотрим еще один вариант использования числа *e*, который может вам пригодиться. Допустим, вы производите баллоны с пропаном. Очевидно, вы не хотите, чтобы баллоны протекали, потому что это может вызвать детонацию

и пожар, особенно вблизи открытого огня и искр. Испытывая новую конструкцию баллона, ваш инженер сообщает, что вероятность протечки в течение года составляет 5 %.

Вы понимаете, что это уже неприемлемо высокое число, но вы также хотите оценить, как эта вероятность растет со временем. Можно задать вопрос: «Какова вероятность того, что протечка произойдет в течение 2 лет? 5 лет? 10 лет?» Правда ли, что, чем больше времени проходит, тем выше вероятность того, что баллон протечет? Число e снова поможет найти ответ!

$$P_{утечки} = 1 - e^{-\lambda T}$$

Эта функция описывает вероятность события с течением времени, в данном случае — вероятность протечки баллона через количество лет T. Здесь λ — это частота отказов в каждую единицу времени (каждый год).

Если построить график этой функции, где по оси X откладывается время T, по оси Y — вероятность протечки, а $\lambda = 0,05$, то мы получим рис. А3.

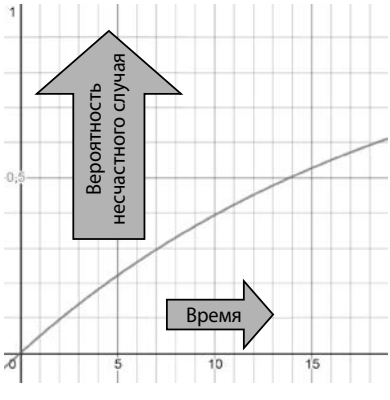


Рис. А3. Прогноз вероятности протечки с течением времени

А вот так выглядит эта функция на Python для $\lambda = 0,05$ и $T = 5$ лет.

Пример А8. Прогнозирование вероятности протечки с течением времени

```
from math import exp

# Вероятность протечки за 1 год
p_leak = .05

# Количество лет
t = 5
```

```
# Вероятность протечки за 5 лет
# 0.22119921692859512
p_leak_5_years = 1.0 - exp(-p_leak * t)

print(f"Вероятность протечки за 5 лет: {p_leak_5_years}")
```

Вероятность того, что баллон выйдет из строя через 2 года, составляет около 9,5 %, через 5 лет — около 22,1 %, а через 10 лет — около 39,3 %. Чем больше времени проходит, тем вероятнее, что баллон даст течь. Эту формулу можно обобщить, чтобы прогнозировать события с заданной вероятностью за определенный период времени и смотреть, как эта вероятность изменяется со временем[1].

Поиск восхождением к вершине и линейная регрессия

Если вам кажется непосильной задачей строить модели машинного обучения по формулам из математического анализа, можно попробовать более грубый метод. Давайте рассмотрим алгоритм *поиска восхождением к вершине*, в котором мы случайным образом регулируем коэффициенты m и b, добавляя случайные значения на каждой итерации. Эти значения могут быть положительными или отрицательными (тогда операция сложения фактически превратится в вычитание), и мы будем сохранять только те поправки, которые улучшают сумму квадратов.

Но имеет ли смысл просто генерировать любые случайные числа в качестве поправок? Нам хотелось бы делать шаги поменьше, но в отдельных случаях стоит допустить и большие перемещения. Таким образом, в основном мы станем добавлять маленькие поправки, но иногда будем делать большие скачки, если нужно. Лучший инструмент для этого — стандартное нормальное распределение со средним 0 и стандартным отклонением 1. Вспомните главу 3: стандартное нормальное распределение имеет высокую плотность значений вблизи 0, и чем дальше значение от 0 (как в отрицательном, так и в положительном направлении), тем менее оно вероятно, как показано на рис. А4.

Возвращаясь к линейной регрессии, зададим для m и b значения 0 или другие начальные значения. Затем в течение 150 000 итераций в цикле `for` мы будем случайным образом регулировать m и b, добавляя поправки, которые отобраны из стандартного нормального распределения. Если случайная поправка улучшает (то есть уменьшает) сумму квадратов, мы ее оставляем, а если сумма квадратов увеличивается, мы отменяем поправку. Пример А9 демонстрирует эту процедуру.

[1] Соответствующее распределение вероятностей называется *экспоненциальным*, или *показательным*. — *Примеч. науч. ред.*

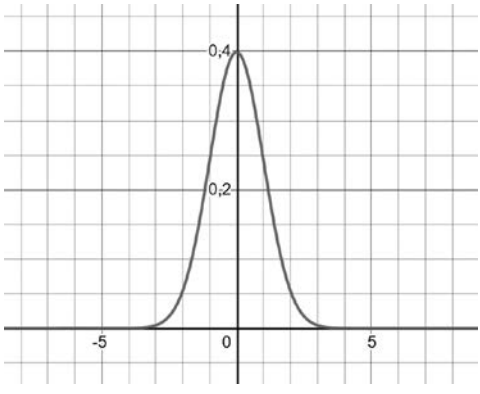


Рис. А4. Большинство значений в стандартном нормальном распределении малы и близки к 0, а большие значения встречаются реже и находятся на хвостах распределения

Пример А9. Поиск восхождением к вершине для линейной регрессии

```
from numpy.random import normal
import pandas as pd

# загружаем данные из файла CSV
points = [p for p in pd.read_csv("https://bit.ly/2KF29Bd").itertuples()]

# Строим начальную модель
m = 0.0
b = 0.0

# Количество итераций
iterations = 150000

# Количество точек
n = float(len(points))

# Инициализируем модель с очень большим значением потерь,
# которое заведомо изменится
best_loss = 10000000000000.0

for i in range(iterations):

    # Корректируем коэффициенты m и b случайным образом
    m_adjust = normal(0,1)
    b_adjust = normal(0,1)

    m += m_adjust
    b += b_adjust

    # Вычисляем функцию потерь, которая представляет собой сумму квадратов
```

```
    new_loss = 0.0
    for p in points:
        new_loss += (p.y - (m * p.x + b)) ** 2

    # Если потери уменьшились, сохраняем новые значения m и b,
    # иначе возвращаемся к предыдущим
    if new_loss < best_loss:
        print(f"y = {m}x + {b}")
        best_loss = new_loss
    else:
        m -= m_adjust
        b -= b_adjust

print(f"y = {m}x + {b}")
```

Вы увидите на экране ход работы алгоритма, но в конечном итоге получите подогнанную функцию, которая равна примерно `y = 1,9395722046562853x + + 4,731834051245578`. Чтобы проверить этот результат, можно выполнить линейную регрессию с помщью Excel или Desmos. Например, Desmos выдал мне `y = 1,93939x + 4,73333`. Это очень близко к нашему результату!

Как узнать, сколько итераций нужно? Опытным путем я обнаружил, что после определенного количества итераций решение уже не сильно улучшается и сходится в малой окрестности оптимальных значений m и b, которые минимизируют сумму квадратов. Во многих библиотеках и алгоритмах машинного обучения есть параметр для количества итераций, и здесь имеется в виду именно он. Итераций должно хватать, чтобы алгоритм сходился приблизительно к верному ответу, но их не должно быть чересчур много, чтобы впустую не тратить вычислительные ресурсы, когда уже найдено приемлемое решение.

У вас может возникнуть вопрос, почему я присвоил переменной `best_loss` очень большое число. Это сделано для того, чтобы инициализировать функцию потерь таким значением, которое точно будет замещено после начала поиска. Затем на каждой итерации уже замещенное значение будет сравниваться с новым результатом, чтобы проверить, привели ли поправки к улучшению. Вместо очень большого числа можно было бы использовать положительную бесконечность: `float('inf')`.

Поиск восхождением к вершине и логистическая регрессия

Подобно предыдущему примеру с линейной регрессией, алгоритм поиска восхождением к вершине можно применить к логистической регрессии. Опять же, используйте этот прием, если вам кажется, что математический анализ и частные производные — это слишком сложно.

Принцип восхождения остается прежним: мы наращиваем m и b случайными значениями из нормального распределения. Однако у нас другая целевая функция — оценка максимального правдоподобия, которую мы рассматривали в главе 6. Поэтому мы принимаем только те случайные поправки, которые увеличивают эту оценку, и после достаточного количества итераций алгоритм должен сойтись на подходящей логистической регрессии.

Все это демонстрируется в примере A10.

Пример A10. Поиск восхождением к вершине для простой логистической регрессии

```
import math
import random

import numpy as np
import pandas as pd

# График на Desmos: https://www.desmos.com/calculator/6cb10atg3l
points = [p for p in pd.read_csv("https://tinyurl.com/y2cocoo7").itertuples()]

best_likelihood = -10_000_000
b0 = .01
b1 = .01

# Вычисляем максимальное правдоподобие
def predict_probability(x):
    p = 1.0 / (1.0001 + math.exp(-(b0 + b1 * x)))
    return p

for i in range(1_000_000):

    # Случайно выбираем коэффициент β₀ или β₁ и регулируем его случайным образом
    random_b = random.choice(range(2))

    random_adjust = np.random.normal()

    if random_b == 0:
        b0 += random_adjust
    elif random_b == 1:
        b1 += random_adjust

    # Вычисляем суммарное правдоподобие
    true_estimates = sum(math.log(predict_probability(p.x)) \
        for p in points if p.y == 1.0)
    false_estimates = sum(math.log(1.0 - predict_probability(p.x)) \
        for p in points if p.y == 0.0)

    total_likelihood = true_estimates + false_estimates
```

```
    # Если правдоподобие улучшилось, сохраняем поправки,
    # иначе возвращаемся к предыдущим значениям коэффициентов
    if best_likelihood < total_likelihood:
        best_likelihood = total_likelihood
    elif random_b == 0:
        b0 -= random_adjust
    elif random_b == 1:
        b1 -= random_adjust

print(f"1.0 / (1 + exp(-({b0} + {b1}*x))")
print(f"Наилучшее правдоподобие: {math.exp(best_likelihood)}")
```

Подробнее об оценке максимального правдоподобия, логистической функции и о том, зачем мы используем функцию `log()`, читайте в главе 6.

Краткое введение в линейное программирование

Каждому специалисту по data science стоит овладеть *линейным программированием* — методом решения систем неравенств, при котором неравенства преобразуются в равенства с помощью так называемых *переменных рассогласования*. Когда переменные в системе линейного программирования являются дискретными целыми или бинарными числами (0 или 1), это называется *целочисленным программированием*. Когда используются и непрерывные, и дискретные переменные, это называется *смешанным целочисленным программированием*.

Линейное программирование и его разновидности больше ориентированы на алгоритмы, чем на данные, однако с их помощью можно решать широкий спектр классических задач искусственного интеллекта. Хотя называть системы линейного программирования искусственным интеллектом — сомнительная практика, она стала обычной для многих производителей и компаний, потому что это позволяет им выглядеть солиднее.

Реальные задачи линейного программирования лучше всего решать с помощью многочисленных доступных библиотек, однако в конце этого раздела будут ссылки на материалы о том, как делать это с нуля. Здесь мы задействуем библиотеку PuLP (https://pypi.org/project/PuLP), хотя Pyomo (https://www.pyomo.org) тоже хорошо подошла бы. Кроме того, для наглядности мы будем использовать визуализацию, хотя случаи с более чем тремя измерениями трудно изобразить таким образом.

Рассмотрим пример. Допустим, вы производите две линейки продуктов — iPac и iPac Ultra. Каждый iPac приносит прибыль 200 долларов, а iPac Ultra — 300 долларов.

Однако сборочная линия может работать только 20 часов в сутки, причем на производство iPac уходит 1 час, а на производство iPac Ultra — 3 часа.

В день вам поставляют только 45 комплектов для сборки, причем, чтобы произвести iPac, требуется 6 комплектов, а чтобы произвести iPac Ultra — 2 комплекта.

Если предположить, что вся произведенная продукция будет продана, сколько iPac и iPac Ultra нужно продать, чтобы получить максимальную прибыль?

Для начала рассмотрим первое ограничение и разложим его на составные части.

Сборочная линия может работать только 20 часов в сутки, причем на производство iPac уходит 1 час, а на производство iPac Ultra — 3 часа.

Это можно выразить в виде неравенства, где x — количество единиц iPac, а y — количество единиц iPac Ultra. Оба значения должны быть положительными, и на рис. A5 показан соответствующий график.

$$x + 3y \le 20 \left(x \ge 0, y \ge 0 \right)$$

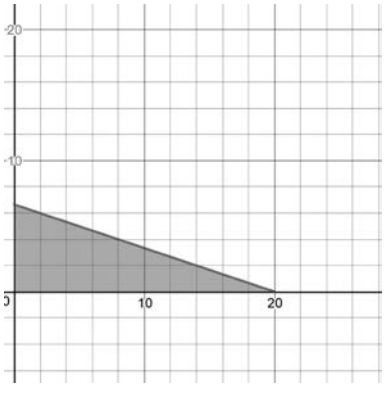


Рис. A5. График первого ограничения

Теперь давайте рассмотрим второе ограничение.

В день вам поставляют только 45 комплектов для сборки, причем, чтобы произвести iPac, требуется 6 комплектов, а чтобы произвести iPac Ultra — 2 комплекта.

Для этого ограничения тоже можно построить модель в форме уравнения и ее график, который добавлен на рис. A6.

$$6x + 2y \leq 45 \left(x \geq 0, y \geq 0 \right)$$

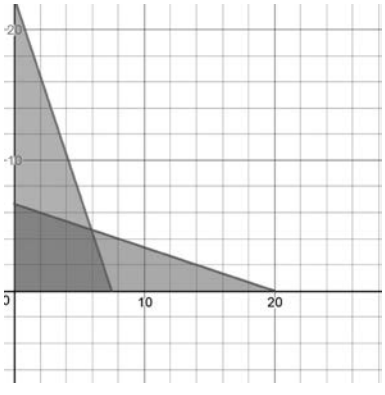


Рис. А6. График первого и второго ограничений

На рис. А6 видно, что области этих двух ограничений пересекаются. Наше решение находится где-то в пределах этого пересечения, и мы будем называть его *областью допустимых решений*. Наконец, с учетом того, какую прибыль приносит iPac и iPac Ultra, мы максимизируем общую прибыль *Z*, которая выражается так:

$$Z = 200x + 300y$$

Если представить эту функцию в виде прямой, то увеличивать *Z* можно до тех пор, пока прямая не выйдет за пределы области допустимых решений. Затем мы зафиксируем соответствующие значения *x* и *y*, как показано на рис. А7.

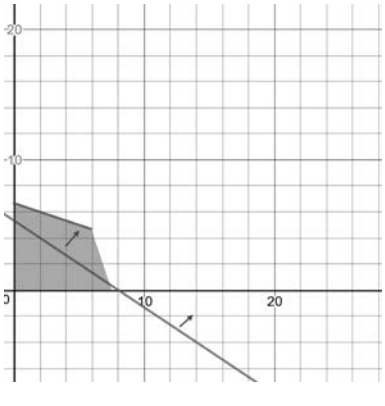


Рис. А7. Перемещение целевой прямой до тех пор, пока она не выйдет из области допустимых решений

График целевой функции в Desmos

Если вы хотите посмотреть на целевую функцию в более интерактивном и анимированном виде, то вот ее график в Desmos (https://oreil.ly/RQMBT).

Когда прибыль увеличится настолько, что эта прямая станет касаться области допустимых решений в одной точке, вы попадете в вершину этой области. Координаты *x* и *y* этой вершины максимизируют прибыль, как показано на рис. A8.

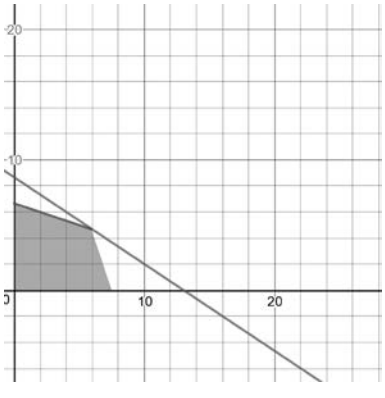


Рис. A8. Максимизированная целевая функция для системы линейного программирования

Хотя можно использовать NumPy и много матричных операций, чтобы решить эту задачу численно, проще будет обойтись средствами PuLP, как показано в примере A11. Обратите внимание, что `LpVariable` определяет неизвестные, относительно которых мы решаем систему уравнений. `LpProblem` — это система линейного программирования, в которую добавляются ограничения и целевые функции с помощью операторов Python. Затем для `LpProblem` вызывается метод `solve()`, чтобы выразить неизвестные.

Пример A11. Решение системы линейного программирования с помощью библиотеки PuLP для Python

```
# График: https://www.desmos.com/calculator/iildqi2vt7

from pulp import *

# Объявляем неизвестные
x = LpVariable("x", 0) # x ≥ 0
y = LpVariable("y", 0) # y ≥ 0

# Определяем проблему
prob = LpProblem("factory_problem", LpMaximize)
```

```
# Определяем ограничения
prob += x + 3*y <= 20
prob += 6*x +2*y <= 45

# Определяем целевую функцию, которую нужно максимизировать
prob += 200*x + 300*y

# Решаем проблему
status = prob.solve()
print(LpStatus[status])

# Выводим результаты: x = 5.9375, y = 4.6875
print(value(x))
print(value(y))
```

Вы можете спросить: а как же производить дробные единицы продукции (5,9375 и 4,6875)? Дело в том, что системы линейного программирования работают гораздо эффективнее, если допускать непрерывные значения переменных, которые потом в случае чего можно просто округлить. Но некоторые типы задач категорически требуют, чтобы переменные были целыми или бинарными числами.

Чтобы неизвестные *x* и *y* рассматривались как целые числа, передайте аргумент категории `cat=LpInteger`, как показано в примере А12.

Пример А12. Принудительное приведение неизвестных к целым числам

```
# Объявляем неизвестные
x = LpVariable("x", 0, cat=LpInteger) # x ≥ 0
y = LpVariable("y", 0, cat=LpInteger) # y ≥ 0
```

Графически это означает, что область допустимых решений состоит из дискретных точек, а не представляет собой непрерывный участок плоскости. Решение не обязательно будет попадать в вершину; оно может попасть в точку, которая находится ближе всего к вершине, как показано на рис. А9.

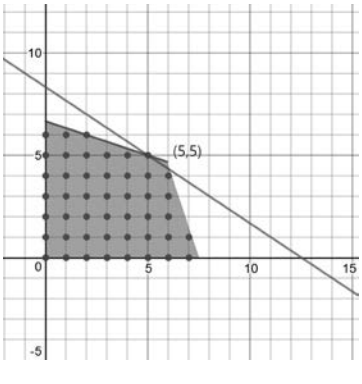
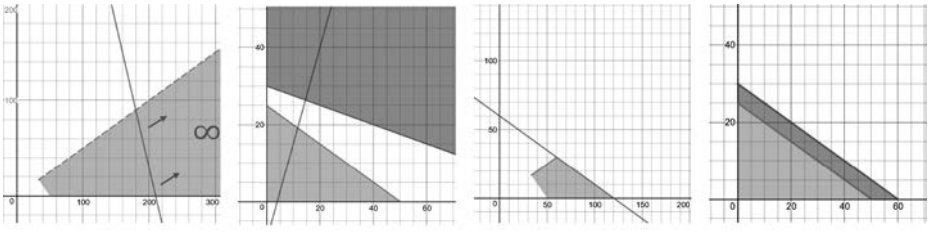


Рис. А9. Система дискретного линейного программирования

В линейном программировании встречается несколько особых случаев, которые показаны на рис. A10. Иногда решений может быть много, а иногда может не быть вообще.



Неограниченная область

Целевую функцию можно максимизировать или минимизировать неограниченно, обычно из-за того, что ограничений слишком мало или они дефектны

Нет области допустимых решений

Не существует области, в которой пересекались бы все области ограничений

Много решений

Целевая прямая параллельна стороне области допустимых решений, и все решения, лежащие на этой стороне, одинаково оптимальны

Избыточность

Два ограничения задают границу области допустимых решений в одном и том же направлении, и их можно объединить

Рис. A10. Особые случаи линейного программирования

Это было всего лишь краткое введение в линейное программирование, и в этой книге, к сожалению, не хватит места, чтобы уделить этой теме должное внимание. Линейное программирование помогает решать самые разные задачи, в том числе составлять расписания в условиях ограниченных ресурсов (например, ограниченного количества сотрудников, мощностей сервера или помещений), решать судоку и оптимизировать финансовые портфели.

Если вы хотите узнать больше, могу порекомендовать несколько хороших видеороликов на YouTube, в том числе на каналах PatrickJMT (https://oreil.ly/lqeeR) и Джошуа Эммануэля (Joshua Emmanuel, https://oreil.ly/jAHWc). А если вам интересно глубоко окунуться в дискретную оптимизацию, то профессор Паскаль Ван Хентенрик (Pascal Van Hentenryck) оказал вам огромную услугу, разместив обучающий курс на Coursera (https://oreil.ly/aVGxY).

Классификатор MNIST на основе scikit-learn

В примере A13 показано, как распознавать рукописные цифры с помощью нейронной сети из библиотеки scikit-learn.

Пример A13. Нейронная сеть — классификатор рукописных цифр на scikit-learn

```
import numpy as np
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

# Загружаем данные
df = pd.read_csv("https://bit.ly/3ilJc2C", compression="zip", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
# Обратите внимание, что здесь нужно линейное масштабирование
X = (df.values[:, :-1] / 255.0)

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]

# Выводим количество элементов в каждом классе,
# чтобы убедиться, что классы сбалансированы
print(df.groupby(["class"]).agg({"class" : [np.size]}))

# Разделяем набор данных на обучающую и тестовую выборки.
# Параметр stratify нужен для того, чтобы
# каждый класс был пропорционально представлен в обеих выборках
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
    test_size=.33, random_state=10, stratify=Y)

nn = MLPClassifier(solver="sgd",
                   hidden_layer_sizes=(100, ),
                   activation="logistic",
                   max_iter=480,
                   learning_rate_init=.1)

nn.fit(X_train, Y_train)

print(f"Точность на обучающей выборке: {nn.score(X_train, Y_train)}")
print(f"Точность на тестовой выборке: {nn.score(X_test, Y_test)}")

# Строим тепловую карту
import matplotlib.pyplot as plt
fig, axes = plt.subplots(4, 4)

# Используем глобальные минимальный и максимальный коэффициенты,
# чтобы все весовые коэффициенты отображались в одном масштабе
vmin, vmax = nn.coefs_[0].min(), nn.coefs_[0].max()
for coef, ax in zip(nn.coefs_[0].T, axes.ravel()):
    ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray, vmin=.5 * vmin, vmax=.5 * vmax)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```

ПРИЛОЖЕНИЕ Б

Ответы на упражнения для самопроверки

Глава 1

1. Число 62,6738 рациональное, потому что в нем конечное количество десятичных знаков, а значит, его можно выразить в виде дроби $\frac{626738}{10000}$.
2. $10^7 \times 10^{-5} = 10^{7+(-5)} = 10^2 = 100.$
3. $81^{1/2} = \sqrt{81} = 9.$
4. $25^{3/2} = \left(25^{1/2}\right)^3 = 5^3 = 125.$
5. Итоговая сумма составит 1161,47 доллара. Код на Python выглядит так:

```
from math import exp

p = 1000
r = .05
t = 3
n = 12

a = p * (1 + (r/n))**(n * t)

print(a) # выводит 1161.4722313334678
```

6. Итоговая сумма составит 1161,47 доллара. Код на Python выглядит так:

```
from math import exp
```

```
p = 1000 # начальный капитал
r = .05 # процентная ставка, годовых
t = 3.0 # время, количество лет

a = p * exp(r*t)

print(a) # выводит 1161.834242728283
```

7. Производная равна 6*x*, а уклон при $x = 3$ будет равен 18. Код с использованием SymPy выглядит так:

```
from sympy import *

# Объявляем символ x для SymPy
x = symbols('x')

# Объявляем функцию через обычный синтаксис Python
f = 3*x**2 + 1

# Вычисляем производную функции
dx_f = diff(f)
print(dx_f) # выводит 6*x
print(dx_f.subs(x,3)) # 18
```

8. Площадь под графиком функции на отрезке от 0 до 2 равна 10. Код с использованием SymPy выглядит так:

```
from sympy import *

# Объявляем символ x для SymPy
x = symbols('x')

# Объявляем функцию через обычный синтаксис Python
f = 3*x**2 + 1

# Вычисляем интеграл от функции по x
# на отрезке от 0 до 2
area = integrate(f, (x, 0, 2))

print(area) # выводит 10
```

Глава 2

1. $0,3 \times 0,4 = 0,12$; см. раздел «Вероятность пересечения событий» на с. 59.
2. $(1-0,3)+0,4-((1-0,3)\times 0,4)=0,82$; см. раздел «Вероятность объединения событий» на с. 60 и помните, что нас интересует, чтобы дождя не было, поэтому вероятность дождя нужно вычесть из 1.

3. $0,3 \times 0,2 = 0,06$; см. раздел «Условная вероятность и формула Байеса» на с. 62.

4. Следующий код на Python вычисляет результат 0,822, суммируя вероятности того, что 50 или более пассажиров не придут:

```
from scipy.stats import binom

n = 137
p = .40
p_50_or_more_noshows = 0.0
for x in range(50,138):
    p_50_or_more_noshows += binom.pmf(x, n, p)

print(p_50_or_more_noshows) # 0.822095588147425
```

5. Используя бета-распределение, как показано далее в коде на SciPy, вычислите площадь до граничного значения 0,5 и вычтите ее из 1. Результат составляет около 0,98, поэтому вероятность того, что эта монета — шулерская, очень высока.

```
from scipy.stats import beta

heads = 8
tails = 2

p = 1.0 - beta.cdf(.5, heads, tails)

print(p) # 0.98046875
```

Глава 3

1. Выборочное среднее равно 1,752, а стандартное отклонение — примерно 0,02135. Код на Python выглядит так:

```
from math import sqrt

sample = [1.78, 1.75, 1.72, 1.74, 1.77]

def mean(values):
    return sum(values) / len(values)

def variance_sample(values):
    var = sum((v - mean(sample)) ** 2 for v in values) / (len(values) - 1)
    return var

def std_dev_sample(values):
    return sqrt(variance_sample(values))
```

```
mean_sample = mean(sample)
std_dev = std_dev_sample(sample)

print("Выборочное среднее:", mean_sample) # 1.752
print("Стандартное отклонение:", std_dev) # 0.023874672772626667
```

2. Используйте CDF, чтобы получить площадь под кривой между 30 и 20 месяцами, которой соответствует вероятность около 0,06. Код на Python выглядит так:

```
from scipy.stats import norm

mean = 42
std_dev = 8

x = norm.cdf(30, mean, std_dev) - norm.cdf(20, mean, std_dev)

print(x) # 0.0638274380338035
```

3. С вероятностью 99 % средний диаметр филамента находится в диапазоне от 1,70 до 1,73. Код на Python выглядит так:

```
from math import sqrt
from scipy.stats import norm

def critical_z_value(p, mean=0.0, std=1.0):
    norm_dist = norm(loc=mean, scale=std)
    left_area = (1.0 - p) / 2.0
    right_area = 1.0 - ((1.0 - p) / 2.0)
    return norm_dist.ppf(left_area), norm_dist.ppf(right_area)

def ci_large_sample(p, sample_mean, sample_std, n):
    # В выборке должно быть больше 30 элементов

    lower, upper = critical_z_value(p)
    lower_ci = lower * (sample_std / sqrt(n))
    upper_ci = upper * (sample_std / sqrt(n))
    return sample_mean + lower_ci, sample_mean + upper_ci

print(ci_large_sample(p=.99, sample_mean=1.715588,
    sample_std=0.029252, n=34))
# (1.7026658973748656, 1.7285101026251342)
```

4. Маркетинговая кампания сработала с *p*-значением 0,01888. Код на Python выглядит так:

```
from scipy.stats import norm

mean = 10345
std_dev = 552

p1 = 1.0 - norm.cdf(11641, mean, std_dev)
```

```
# Воспользуемся симметрией
p2 = p1

# p-значение обоих хвостов
# Можно было также просто удвоить p-значение одного хвоста

p_value = p1 + p2

print("Двустороннее p-значение:", p_value)
if p_value <= .05:
    print("Двусторонний тест пройден")
else:
     print("Двусторонний тест не пройден")

# Двустороннее p-значение: 0.01888333596496139
# Двусторонний тест пройден
```

Глава 4

1. Вектор перейдет в точку $\begin{bmatrix} 2 \\ 3 \end{bmatrix}$. Код на Python выглядит так:

```
from numpy import array

v = array([1,2])

i_hat = array([2, 0])
j_hat = array([0, 1.5])

basis = array([i_hat, j_hat]).transpose()

# преобразуем вектор v в w
w = basis.dot(v)

print(w) # [2. 3.]
```

2. Вектор перейдет в точку $\begin{bmatrix} 0 \\ -3 \end{bmatrix}$. Код на Python выглядит так:

```
from numpy import array

v = array([1,2])

i_hat = array([-2, 1])
j_hat = array([1, -2])

basis = array([i_hat, j_hat]).transpose()

# преобразуем вектор v в w
w = basis.dot(v)

print(w) # [ 0 -3]
```

3. Определитель равен 2. Код на Python выглядит так:

```
import numpy as np
from numpy.linalg import det

i_hat = np.array([1, 0])
j_hat = np.array([2, 2])

basis = np.array([i_hat,j_hat]).transpose()

determinant = det(basis)

print(determinant) # 2.0
```

4. Да, потому что матричное умножение позволяет объединить несколько матриц в одну, которая представляет композицию преобразований.

5. $x = 19{,}8$, $y = -5{,}4$, $z = -6$. Код на Python выглядит так:

```
from numpy import array
from numpy.linalg import inv

A = array([
    [3, 1, 0],
    [2, 4, 1],
    [3, 1, 8]
])

B = array([
    54,
    12,
    6
])

X = inv(A).dot(B)

print(X) # [19.8 -5.4 -6. ]
```

6. Да, матрица линейно зависима, потому что ее определитель равен 0. Обратите внимание, что в NumPy может проявиться погрешность вычислений с плавающей точкой[1]:

```
from numpy.linalg import det
from numpy import array

i_hat = array([2, 6])
j_hat = array([1, 3])
```

[1] В зависимости от версий Python и NumPy на вашем компьютере этот код может вывести другое малое число или даже точно `0.0`. — *Примеч. науч. ред.*

```
basis = array([i_hat, j_hat]).transpose()
print(basis)

determinant = det(basis)

print(determinant) # -3.330669073875464e-16
```

Чтобы избавиться от погрешности, можно использовать SymPy, которая выводит точно 0:

```
from sympy import *

basis = Matrix([
    [2,1],
    [6,3]
])

determinant = det(basis)

print(determinant) # 0
```

Глава 5

1. Для линейной регрессии есть много методов и инструментов, о которых мы узнали в главе 5. Здесь представлено решение с помощью scikit-learn. Угловой коэффициент m равен 1,75919315, а ордината пересечения b — 4,69359655.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Загружаем данные
df = pd.read_csv("https://bit.ly/3C8JzrM", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1]

# Извлекаем столбец с выходными значениями (все строки, только последний
столбец)
Y = df.values[:, -1]

# Подгоняем прямую к точкам
fit = LinearRegression().fit(X, Y)

# m = 1.75919315, b = 4.69359655
m = fit.coef_.flatten()
```

```
b = fit.intercept_.flatten()
print(f"m = {m}")
print(f"b = {b}")

# Выводим график
plt.plot(X, Y, 'o') # диаграмма рассеяния
plt.plot(X, m*X+b) # прямая регрессии
plt.show()
```

2. Коэффициент корреляции довольно высок: 0,92421. Тестовая статистика равна 23,8355 при критическом интервале от −1,9844 до 1,9844. Эта корреляция определенно состоятельна и статистически значима. Код выглядит так:

```
import pandas as pd

# Загружаем данные в датафрейм pandas
df = pd.read_csv("https://bit.ly/3C8JzrM", delimiter=",")

# Выводим коэффициенты корреляции между переменными
correlations = df.corr(method='pearson')
print(correlations)

# ВЫВОД:
#         x       y
# x 1.00000 0.92421
# y 0.92421 1.00000

# Тест на статистическую значимость
from scipy.stats import t
from math import sqrt

# Размер выборки
n = df.shape[0]
print(n)
lower_cv = t(n - 1).ppf(.025)
upper_cv = t(n - 1).ppf(.975)

# Вычисляем коэффициент корреляции
r = correlations["y"]["x"]

# Выполняем тест
test_value = r / sqrt((1 - r ** 2) / (n - 2))

print(f"Тестовое значение (t-статистика): {test_value}")
print(f"Критический интервал: {lower_cv}, {upper_cv}")

if test_value < lower_cv or test_value > upper_cv:
    print("Корреляция обоснована, отвергаем H₀")
else:
    print("Корреляция не обоснована, нельзя отвергнуть H₀")
```

```
# Вычисляем p-значение
if test_value > 0:
    p_value = 1.0 - t(n-1).cdf(test_value)
else:
    p_value = t(n-1).cdf(test_value)

# Двусторонний тест, поэтому умножаем на 2
p_value = p_value * 2
print(f"p-значение: {p_value}")

"""
Тестовое значение: 23.835515323677328
Критический интервал: -1.9844674544266925, 1.984467454426692
Корреляция обоснована, отвергаем H₀
p-значение: 0.0 (пренебрежимо мало)
"""
```

3. При $x = 50$ интервал прогнозирования составляет от 50,79 до 134,51. Код выглядит так:

```
import pandas as pd
from scipy.stats import t
from math import sqrt

# Загружаем данные
points = list(pd.read_csv("https://bit.ly/3C8JzrM", delimiter=",") \
    .itertuples())

n = len(points)

# Прямая линейной регрессии
m = 1.75919315
b = 4.69359655

# Вычисляем интервал прогнозирования для x = 50
x_0 = 50
x_mean = sum(p.x for p in points) / len(points)

t_value = t(n - 2).ppf(.975)

standard_error = sqrt(sum((p.y - (m * p.x + b)) ** 2 for p in points) / \
    (n - 2))

margin_of_error = t_value * standard_error * \
        sqrt(1 + (1 / n) + (n * (x_0 - x_mean) ** 2) / \
            (n * sum(p.x ** 2 for p in points) - \
    sum(p.x for p in points) ** 2))

predicted_y = m*x_0 + b
```

```
# Выводим интервал прогнозирования
print(predicted_y - margin_of_error, predicted_y + margin_of_error)
# 50.792086501055955 134.51442159894404
```

4. Тестовые выборки показывают удовлетворительные результаты, если разделить набор данных на трети и обработать с помощью трехкратной перекрестной валидации. Для трех наборов данных среднее значение MSE составляет около 0,83, а стандартное отклонение — 0,04.

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold, cross_val_score

df = pd.read_csv("https://bit.ly/3C8JzrM", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1]

# Извлекаем столбец с выходными значениями (все строки, только последний
столбец)
Y = df.values[:, -1]

# Простая линейная регрессия
kfold = KFold(n_splits=3, random_state=7, shuffle=True)
model = LinearRegression()
results = cross_val_score(model, X, Y, cv=kfold)
print(results)
print("MSE: среднее = %.3f (ст. откл. = %.3f)" % (results.mean(), results.
std()))
"""
[0.86119665 0.78237719 0.85733887]
MSE: среднее = 0.834 (ст. откл. = 0.036)
"""
```

Глава 6

1. Регрессия, построенная с помощью scikit-learn, обеспечивает крайне высокую точность. Запуская код ниже, я получаю среднюю точность на тестовых выборках не менее 99,9 %.

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold, cross_val_score
```

```
# Загружаем данные
df = pd.read_csv("https://bit.ly/3imidqa", delimiter=",")

X = df.values[:, :-1]
Y = df.values[:, -1]

kfold = KFold(n_splits=3, shuffle=True)
model = LogisticRegression(penalty=None)
results = cross_val_score(model, X, Y, cv=kfold)

print(f"Точность = {results.mean():.3f} (ст. откл.= {results.std():.3f})")
```

2. Матрица ошибок показывает очень большое количество истинно положительных и истинно отрицательных результатов и очень мало ложноположительных и ложноотрицательных. Чтобы убедиться в этом, запустите код:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

# Загружаем данные
df = pd.read_csv("https://bit.ly/3imidqa", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1]

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]

model = LogisticRegression(solver="liblinear")

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.33)
model.fit(X_train, Y_train)
prediction = model.predict(X_test)

""" Матрица ошибок определяет количество результатов в каждой категории:
[[истинно_положительные ложноотрицательные]
 [ложноположительные истинно_отрицательные]]

Главная диагональ отражает правильные прогнозы,
поэтому желательно, чтобы значения на ней были больше
"""
matrix = confusion_matrix(y_true=Y_test, y_pred=prediction)
print(matrix)
```

3. Вот интерактивная оболочка, с помощью которой можно тестировать цвета, введенные пользователем. Попробуйте протестировать модель для черного (0, 0, 0) и белого (255, 255, 255) фона, чтобы проверить, правильно ли она прогнозирует темный и светлый шрифт.

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
import numpy as np
from sklearn.model_selection import train_test_split

# Загружаем данные
df = pd.read_csv("https://bit.ly/3imidqa", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего столбца)
X = df.values[:, :-1]

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]

model = LogisticRegression(solver="liblinear")

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.33)
model.fit(X_train, Y_train)
prediction = model.predict(X_test)

# Проверяем прогнозирование
while True:
    n = input("Введите цвет фона в формате: {красный}, {зеленый}, {синий}: ")
    (r, g, b) = n.split(",")
    x = model.predict(np.array([[int(r), int(g), int(b)]]))
    if model.predict(np.array([[int(r), int(g), int(b)]]))[0] == 0.0:
        print("СВЕТЛЫЙ ШРИФТ")
    else:
        print("ТЕМНЫЙ ШРИФТ")
```

4. Да, логистическая регрессия очень эффективно предсказывает светлый или темный шрифт для заданного цвета фона. Точность модели чрезвычайно высока, а в матрице ошибок высокие показатели на главной диагонали и низкие — в остальных ячейках.

Глава 7

Очевидно, что здесь можно много экспериментировать и пробовать различные скрытые слои, функции активации, разные размеры тестовых выборок и т. д. Я попробовал использовать один скрытый слой с тремя узлами с функцией активации ReLU, и мне не удалось получить хороших прогнозов на тестовых данных. Точность оказывалась стабильно низкой, а матрицы ошибок — неудовлетворительными, и никакие изменения конфигурации не приводили к лучшим результатам.

Почему нейронная сеть не справилась с задачей? Во-первых, возможно, тестовая выборка была слишком мала для нейронной сети (которая очень требовательна

к объемам данных), а во-вторых, задачи такого рода гораздо проще и эффективнее решаются с помощью других моделей, например логистической регрессии. Это не значит, что в принципе невозможно найти конфигурацию нейросети, которая хорошо сработает, но вам стоит следить за тем, чтобы не скатиться в *p*-хакинг и чтобы ради хорошего результата не переобучить модель на том небольшом объеме обучающих и тестовых данных, которые у вас есть.

Вот код на scikit-learn, который я использовал:

```
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

# Загружаем данные
df = pd.read_csv("https://tinyurl.com/y6r7qjrp", delimiter=",")

# Извлекаем входные переменные (все строки и все столбцы, кроме последнего
столбца)
X = df.values[:, :-1]

# Извлекаем столбец с выходной переменной (все строки, только последний столбец)
Y = df.values[:, -1]

# Разделяем данные на обучающую и тестовую выборки
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3)

nn = MLPClassifier(solver="sgd",
                   hidden_layer_sizes=(3, ),
                   activation="relu",
                   max_iter=100_000,
                   learning_rate_init=.05)

nn.fit(X_train, Y_train)

print(f"Средняя точность на обучающей выборке: {nn.score(X_train, Y_train)}")
print(f"Средняя точность на тестовой выборке: {nn.score(X_test, Y_test)}")

print("Матрица ошибок:")
matrix = confusion_matrix(y_true=Y_test, y_pred=nn.predict(X_test))
print(matrix)
```

Об авторе

Томас Нилд — основатель компании Nield Consulting Group, а также преподаватель O'Reilly Media и Университета Южной Калифорнии. Ему нравится излагать технические темы доступно и интересно для тех, кто не знаком с предметом или побаивается его. Томас регулярно ведет занятия по анализу данных, машинному обучению, математической оптимизации и практическому искусственному интеллекту. Он написал книги «Getting Started with SQL» (O'Reilly) и «Learning RxJava» (Packt), а также основал компанию Yawman Flight, которая разрабатывает универсальные портативные устройства управления для авиасимуляторов и беспилотных летательных аппаратов.

Иллюстрация на обложке

Животные на обложке книги — мышевидные грызуны, известные также как полосатые мыши или крысы (*Rhabdomys pumilio*). Эти грызуны обитают в южной части Африки — в саваннах, пустынях, сельскохозяйственных угодьях, кустарниковых степях и даже в городах. Они называются так потому, что имеют на спине четыре отчетливые продольные темные полосы. Уже при рождении эти полосы видны как пигментированные участки на безволосой коже детеныша.

Окраска меха полосатой мыши варьируется от темно-коричневой до серовато-белой, с более светлыми боками и брюшком. Животное вырастает до 18–21 сантиметра в длину (не считая хвоста примерно такой же длины, как тело) и весит 30–55 граммов. Мышь наиболее активна днем и всеядна; она питается семенами, растениями и насекомыми. В летние месяцы она, как правило, ест больше растений и семян, а также запасается жиром, чтобы продержаться в период скудного питания.

За полосатыми мышами легко наблюдать, учитывая их широкий ареал обитания. Было отмечено, что они перемежают одиночный и социальный образ жизни. В сезон размножения они держатся обособленно (возможно, чтобы избежать чрезмерной репродуктивной конкуренции), а самки придерживаются территории своих нор. Однако в остальное время мыши собираются в группы, чтобы добывать пищу, спасаться от хищников и прижиматься друг к другу, чтобы согреться.

Многие из животных, которые изображены на обложках книг издательства O'Reilly, находятся под угрозой исчезновения; все они важны для нашего мира.

Иллюстрацию для обложки нарисовала Карен Монтгомери (Karen Montgomery) на основе старинной гравюры из Музея естественной истории.

Томас Нилд

Математика для Data Science. Управляем данными с помощью линейной алгебры, теории вероятностей и статистики

Перевел с английского А. Гаврилов

Изготовлено в России. Изготовитель: ТОО «Спринт Бук».
Место нахождения и фактический адрес: 010000, Казахстан, город Астана, район Алматы,
Проспект Ракымжан Кошкарбаев, дом 10/1, н.п. 18

Дата изготовления: 12.2024.

Наименование: книжная продукция.

Срок годности: не ограничен.

Подписано в печать 18.10.24. Формат 70×100/16. Бумага офсетная.
Усл. п. л. 28,380. Тираж 700. Заказ 0000.