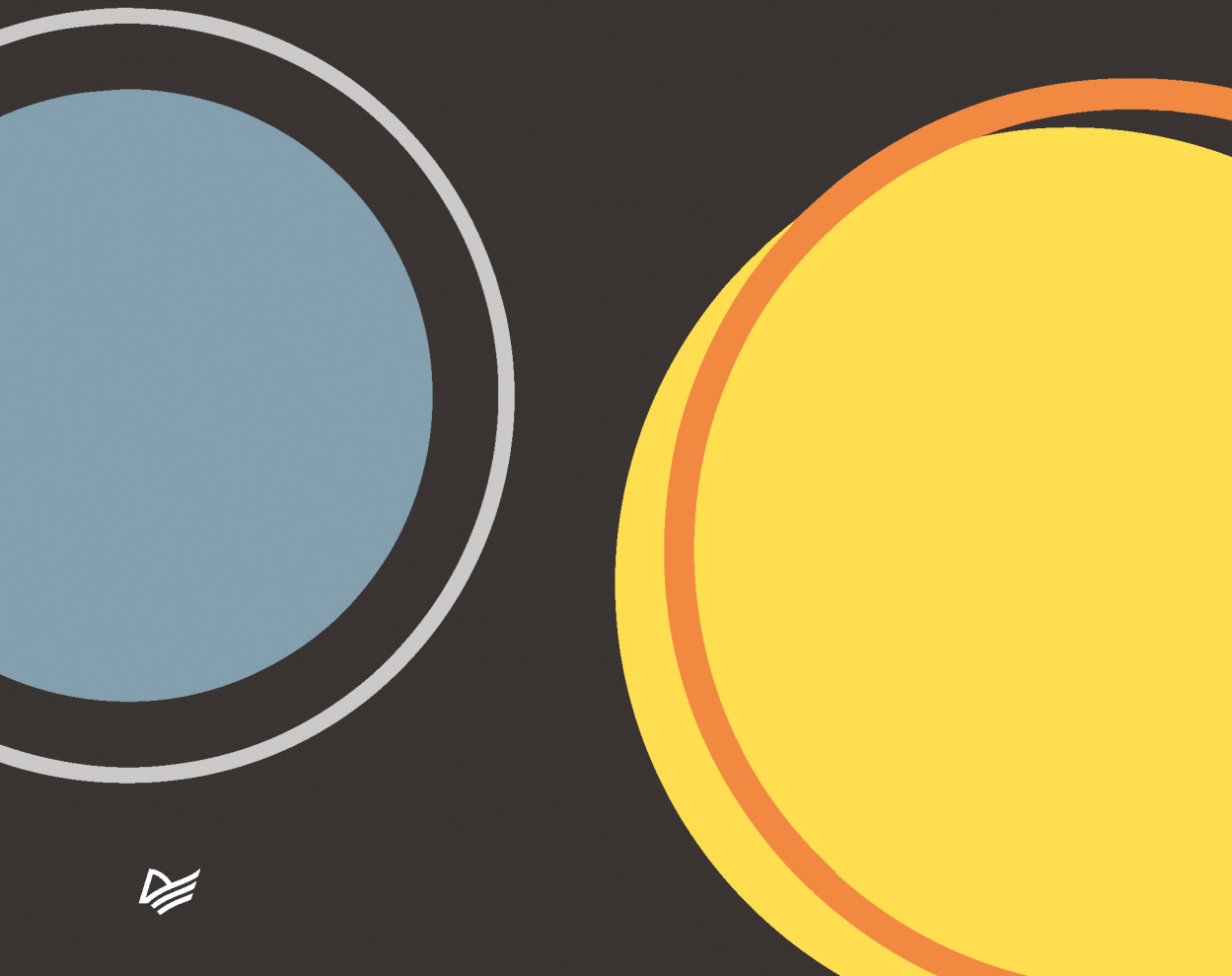
Андрей Бурков

МАШИННОЕ ОБУЧЕНИЕ

без лишних слов



The Hundred-Page Machine Learning Book

Andriy Burkov

Андрей Бурков

МАШИННОЕ ОБУЧЕНИЕ
без лишних слов

ПИТЕР®

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

2020

ББК 32.813
УДК 004.8
Б91

Бурков Андрей

Б91 Машинное обучение без лишних слов. — СПб.: Питер, 2020. — 192 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1560-0

Все, что вам действительно нужно знать о машинном обучении, может уместиться на паре сотен страниц.

Начнем с простой истины: машины не учатся. Типичное машинное обучение заключается в поиске математической формулы, которая при применении к набору входных данных (называемых обучающими данными) даст желаемые результаты.

Андрей Бурков постарался дать все необходимое, чтобы каждый мог стать отличным современным аналитиком или специалистом по машинному обучению. То, что удалось вместить в пару сотен страниц, в других книгах растянуто на тысячи. Типичные книги по машинному обучению консервативны и академичны, здесь же упор сделан на алгоритмах и методах, которые пригодятся в повседневной работе.

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

ББК 32.813
УДК 004.8

ISBN 978-1999579500 англ.
ISBN 978-5-4461-1560-0

Оглавление

Моим родителям Татьяне и Валерию и моей семье:
дочерям Катрин и Еве и брату Дмитрию

Все модели ошибочны, но некоторые полезны.

Джордж Бокс

Письмо это вышло более длинным только потому, что мне некогда было написать его короче.

Блез Паскаль

Предисловие к русскому изданию

Машинное обучение, наверное, самая горячая и быстроразвивающаяся дисциплина в современной информатике, если не в современной науке вообще. Каждый день появляются новые модели и выходят новые статьи, раз в пару месяцев происходит очередной прорыв, который попадает в новости и открывает новые возможности, а раз в год-два происходит переворот в целой отрасли. Вот уже десять лет, после революции глубокого обучения, мы живем на новой (третьей) волне хайпа искусственного интеллекта, и пока ничто не предвещает, что она скоро закончится.

Неудивительно, что сейчас машинное обучение привлекает множество людей, которые никогда раньше им не занимались. Кто-то узнал о заработках в индустрии и хочет «разрабатывать искусственный интеллект за 300 К/сек», кто-то хочет узнать, не пора ли «перевести свой бизнес с big data на machine learning», а кто-то приходит в AI с глубокими идеями о том, как сделать этичным общий искусственный интеллект, который не поработит и не убьет людей, а будет помогать им (в целом это вполне серьезный разговор, но любому профессионалу очевидно, что до практики или содержательных исследований этого еще очень, очень, очень далеко).

Поэтому в наше время действительно очень полезно иметь краткое введение в машинное обучение, на которое всегда можно давать ссылку и после которого можно быть уверенным, что человек говорит на одном с тобой языке. Попытку дать именно такое введение я вижу в этой книге, и мне кажется, что эта попытка получилась очень удачной. Книга действительно представляет читателю широкий спектр основных понятий и методов машинного обучения, которые здесь изложены корректно, хоть и по понятным причинам очень кратко. Но если освоить эту книгу, дальше самообразование может пойти куда проще и быстрее, ведь вы уже сможете читать более специальные источники. Кроме того, вам будет куда понятнее, что именно делает код библиотек машинного обучения — для специалиста в этом не должно оставаться никакой магии.

Не стоит обольщаться: царского пути нет ни в геометрии, ни в машинном обучении, ни вообще где бы то ни было. На свете нет и не может быть волшебного способа «обучиться разрабатывать искусственный интеллект за 30 дней без sms и регистрации». И эта книга тоже, конечно, такого способа не дает. С одной стороны, вам потребуется некоторая математическая квалификация, чтобы понять изложенное здесь (хотя глава 2 начинается буквально с того, что такое «множество», ее, конечно, следует рассматривать скорее как напоминание для тех, кто когда-то это уже изучал). С другой стороны, эта книга — только самое начало пути в интересный и разнообразный мир машинного обучения; прочитав ее, вы не станете профессионалом — вы сделаете первый маленький шаг.

Но если книгу прочитать вдумчиво и действительно освоить то, о чем здесь говорится, этот шаг может превратиться в большой скачок. Чего я и желаю всем читателям: разбирайтесь, познавайте, интересуйтесь новым и не бойтесь трудностей. Удачи!

Сергей Николенко,
автор книги «Глубокое обучение. Погружение в мир нейронных сетей»,
сотрудник лаборатории математической логики Санкт-Петербургского
отделения Математического института РАН,
директор по научным исследованиям (Chief Research Officer)
платформы Neuromation

Отличное введение в машинное обучение от специалиста мирового уровня.

Каролис Урбонас (Karolis Urbonas),
руководитель отдела анализа данных в Amazon

Хотела бы я встретить такую книгу, когда обучалась статистике в аспирантуре и пыталась освоить машинное обучение.

Чао Хан (Chao Han),
вице-президент, руководитель исследований и разработок в Lucidworks

Книга Андрея, фантастическим образом устраняя все лишнее, идет на полной скорости прямо к цели с самой первой страницы.

Суджит Варахеди (Sujeet Varakhedi),
технический руководитель в eBay

Прекрасная книга для инженеров, желающих начать использовать машинное обучение в своей повседневной работе и не затратить на это слишком много времени.

Дипак Агарвал (Deepak Agarwal),
вице-президент по развитию искусственного интеллекта в LinkedIn

Предисловие из оригинального издания

В последние двадцать лет мы наблюдаем взрывной рост объемов доступных данных и, как следствие, интереса к статистическим приложениям и машинному обучению. Последствия оказались далекоидущими. Десять лет назад, когда я смог привлечь полный класс студентов MBA к изучению моего нового факультативного курса по статистике, коллеги были удивлены, потому что наш факультет не мог добиться такого результата для большинства факультативов. Сегодня мы предлагаем курс магистратуры по бизнес-аналитике, который является крупнейшей специализированной магистерской программой в университете и по популярности конкурирует с нашими программами MBA. Количество предлагаемых нами курсов значительно возросло, однако наши студенты все еще жалуются на нехватку мест. Наш опыт не уникален, поскольку программы по науке о данных и машинному обучению развиваются с необычайной скоростью, так как спрос на специалистов в этой области непрерывно растет.

Такая популярность обусловлена простым, но неоспоримым фактом. Развитие машинного обучения привело к новым открытиям во многих областях, таких как социальные науки, бизнес, биология и медицина, и это далеко не полный список. В результате возник огромный спрос на людей с определенным набором навыков. Тем не менее обучение студентов этим навыкам оказалось сложной задачей, потому что большая часть ранней литературы по этим методам была нацелена на людей, занимающихся академическими исследованиями, и сосредоточена на статистических и теоретических свойствах алгоритмов обучения или полученных моделей. Почти полностью отсутствовали материалы, ориентированные на исследователей и практиков, нуждавшихся в помощи при реализации определенного метода для решения практических задач. Таким людям важнее знать и понимать спектр методов, применимых к каждой задаче, их сильные и слабые стороны и лежащие в их основе предположения, а теоретические свойства или подробная информация об алгоритмах подбора имеют для них второстепенное значение. Работая над книгой «An Introduction to Statistical Learning with R»[1] (ISLR), мы преследовали цель

[1] *Джеймс Г., Уиттон Д., Хасти Т., Тибширани Р.* Введение в статистическое обучение с примерами на языке R. ДМК-Пресс, 2016. — *Примеч. пер.*

создать источник информации для такой группы людей. Энтузиазм, с которым была встречена эта книга, демонстрирует спрос, существующий в обществе.

Книга «Машинное обучение без лишних слов» следует аналогичной парадигме. Так же как в ISLR, в ней отсутствуют теоретические выводы, и основное внимание уделяется описанию ключевых деталей реализации различных подходов. Это компактное руководство «как анализировать данные», и я убежден, что оно станет ценным источником информации для научных сотрудников и практиков. Книга достаточно коротка, чтобы ее можно было прочитать за один присест. Тем не менее, несмотря на небольшой объем, она охватывает все основные подходы машинного обучения, от классической линейной и логистической регрессии до современного метода опорных векторов, глубокого обучения, бустинга и случайных лесов. При этом описание различных подходов не страдает от недостатка деталей, и заинтересованный читатель сможет получить дополнительную информацию о любом конкретном методе в «Википедии». Книга не предполагает наличия математической или статистической подготовки высокого уровня или даже опыта программирования, поэтому доступна почти каждому, кто решит потратить время на изучение этих методов. С другой стороны, эту книгу обязательно должны прочитать все начинающие обучение в этой области, и она послужит им полезным справочником в будущем. Наконец, книга иллюстрирует некоторые алгоритмы, используя код на Python, одном из самых популярных языков программирования для машинного обучения. Я настоятельно рекомендую книгу «Машинное обучение без лишних слов» как для начинающих, желающих узнать больше о машинном обучении, так и для опытных практиков, стремящихся расширить свой кругозор.

Гарет Джеймс (Gareth James),
профессор в области теории и методов анализа данных в Южно-Калифорнийском университете, соавтор (вместе с Уиттоном, Хасти и Тибширани) книги-бестселлера «An Introduction to Statistical Learning, with Applications in R»

Вступление

Начнем с простой истины: машины не учатся. Типичное машинное обучение заключается в поиске математической формулы, которая при применении к набору входных данных (называемых обучающими данными) дает желаемые результаты. Эта математическая формула также генерирует правильные выходные данные для большинства других входных данных (отличных от обучающих) при условии, что эти входные данные поступают из того же или подобного статистического распределения, из которого были получены обучающие данные.

Почему это не является обучением? Потому что стоит слегка исказить входные данные, и результат, скорее всего, получится полностью неправильным. Обучение у животных — это нечто иное. Если вы научились играть в видеоигру при прямой ориентации экрана, вы все равно сможете играть в нее, даже если кто-то слегка повернет экран. Алгоритм машинного обучения, обучавшийся при прямой ориентации экрана и не обученный распознаванию поворота, не сможет играть в игру на повернутом экране.

Но почему тогда используется название «машинное обучение»? Причина, как это часто бывает, заключается в маркетинге: Артур Сэмюэл (Arthur Samuel), американский пионер в области компьютерных игр и искусственного интеллекта, придумал этот термин в 1959 году, когда работал в IBM. Подобно тому как в 2010-х годах IBM пыталась продвигать термин «когнитивные вычисления», чтобы выделиться среди конкурентов, в 1960-х годах IBM использовала новый крутой термин «машинное обучение», чтобы привлечь клиентов и талантливых сотрудников.

Как видите, подобно тому как искусственный интеллект не является интеллектом, машинное обучение тоже не является обучением. Тем не менее термин «машинное обучение» получил широкое распространение и под ним часто подразумевается теория и практика создания машин, способных выполнять различные полезные действия без явного программирования. Слово «обучение» в данном случае используется лишь как аналогия с обучением в животном мире, а не буквально.

Кому адресована эта книга

Эта книга содержит только те сведения о машинном обучении, которые появились после 1960-х годов и доказали свою практическую ценность. Новичок в машинном обучении найдет в этой книге достаточно подробностей, чтобы обеспечить себе такой уровень понимания, который позволит начать задавать правильные вопросы.

Практики с опытом могут использовать эту книгу как набор рекомендаций для дальнейшего самосовершенствования. Книга также пригодится при мозговом штурме в начале проекта, когда требуется ответить на вопрос, является ли данная техническая или бизнес-задача «машинно-обучаемой», и, если да, какие методы способны помочь решить ее.

Как пользоваться этой книгой

Если вы собираетесь приступить к изучению темы машинного обучения, желательно, чтобы вы прочитали эту книгу от начала и до конца. (Здесь всего чуть больше ста страниц, и вы без труда одолеете их.) Если вас заинтересует какая-то конкретная тема из описываемых в книге и вы захотите узнать больше, в большинстве разделов вы найдете QR-код.

Отсканировав такой QR-код с помощью телефона, вы получите ссылку на страницу в сопутствующем вики-справочнике книги на сайте theMLbook.com, где найдете рекомендуемые материалы для чтения, видеоролики, вопросы и ответы, фрагменты кода, учебные пособия и многое другое. Вики-справочник постоянно пополняется публикациями самого автора книги, а также добровольцев со всего мира, то есть эта книга, как хорошее вино, со временем становится только лучше.

QR-код
с адресом
статьи в вики

Отсканируйте QR-код слева, чтобы попасть в вики-справочник для книги. В некоторых разделах нет QR-кода, но для многих из них тоже есть страницы в вики. Вы сможете найти их, выполнив поиск по названию раздела в поисковой системе вики.

Теперь устраивайтесь поудобнее. Приятного чтения!

Андрей Бурков

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

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

1 Введение

1.1. Что такое машинное обучение

Машинное обучение — это раздел информатики, посвященный созданию алгоритмов, опирающихся на набор данных о каком-либо явлении. Эти данные могут быть получены из естественной среды, созданы вручную или сгенерированы другим алгоритмом.

Машинное обучение также можно определить как процесс решения практической задачи путем 1) формирования набора данных и 2) алгоритмического построения статистической модели на его основе. Предполагается, что эта статистическая модель будет каким-то образом использоваться для решения практической задачи.

Чтобы сэкономить на нажатиях клавиш, я буду использовать термины «обучение» и «машинное обучение» как взаимозаменяемые.

1.2. Типы обучения

Обучение может быть с учителем, без учителя и с подкреплением.

1.2.1. Обучение с учителем

В **обучении с учителем набор данных** организован как коллекция **размеченных образцов** $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Каждый элемент $\mathbf{x}_i$ из N называется **вектором признаков**. Вектор признаков — это вектор, в котором каждое измерение $j = 1, ..., D$ содержит значение, описывающее некоторую характеристику образца. Это значение называется **признаком** и обозначается как $x^{(j)}$. Например, если каждый образец $\mathbf{x}$ в нашей

коллекции представляет человека, тогда первый признак, $x^{(1)}$, мог бы описывать его рост в сантиметрах, второй признак, $x^{(2)}$, мог бы описывать его вес в килограммах, $x^{(3)}$ — пол, и т. д. Для всех данных в наборе данных признак в позиции j в векторе признаков всегда описывает одну и ту же характеристику. Это означает, что если $x_i^{(2)}$ описывает вес некоторого образца $\mathbf{x}_i$ в килограммах, тогда $x_k^{(2)}$ также будет описывать вес в килограммах каждого образца x_k, $k = 1, ..., N$. **Метка** y_i может быть элементом конечного множества **классов** $\{1, 2, ..., C\}$, вещественным числом или более сложной структурой, такой как вектор, матрица, дерево или граф. В этой книге, если явно не оговаривается иное, под y_i будет подразумеваться элемент конечного множества классов или вещественное число[1]. Класс можно представить как категорию, которой принадлежит образец. Например, если роль данных играют электронные письма и вы решаете задачу определения спама, тогда вы могли бы определить два класса: {*спам*, *не_спам*}.

Цель **алгоритма обучения с учителем** — на основе набора данных создать **модель**, которая принимает вектор признаков **x** на входе и возвращает информацию, которая позволяет определить метку для этого вектора признаков. Например, модель, созданная с использованием набора данных людей, могла бы принимать вектор признаков, описывающих человека, и возвращать вероятность, что этот человек болен раком.

1.2.2. Обучение без учителя

В **обучении без учителя** набор данных представлен коллекцией **неразмеченных образцов** $\{\mathbf{x}_i\}_{i=1}^{N}$. И снова, **x** — это вектор признаков, а цель **алгоритма обучения без учителя** — создать **модель**, которая принимает вектор признаков **x** на входе и преобразует его в другой вектор или в значение, которое можно использовать для решения практической задачи. Например, в задачах **кластеризации** модель возвращает идентификатор кластера для каждого вектора признаков в наборе данных. В задачах **уменьшения размерности** модель возвращает вектор признаков, который имеет меньше элементов, чем входной вектор **x**. В задачах **выявления аномалий** возвращается действительное число, которое указывает, насколько **x** отличается от «типичного» образца в наборе данных.

1.2.3. Обучение с частичным привлечением учителя

В обучении с частичным привлечением учителя (semi-supervised learning) набор данных содержит как размеченные, так и неразмеченные образцы. Обычно неразмеченных образцов намного больше, чем размеченных. **Алгоритм обучения**

[1] Вещественное число — это величина, которую можно представить как расстояние на числовой прямой. Примеры вещественных чисел: 0, –256.34, 1000, 1000.2.

с частичным привлечением учителя преследует ту же цель, что и алгоритм обучения с учителем. В данном случае предполагается, что использование множества неразмеченных данных поможет алгоритму обучения найти (можно также сказать «произвести» или «вычислить») лучшую модель.

Предположение о выигрыше от добавления большего количества неразмеченных данных может показаться нелогичным. На первый взгляд кажется, что мы, наоборот, добавляем больше неопределенности. Однако добавляя неразмеченные данные, вы вносите больше информации о задаче: бóльшая выборка лучше отражает распределение вероятностей в данных, откуда взяты размеченные образцы. Теоретически, алгоритм обучения должен уметь воспользоваться этой дополнительной информацией.

1.2.4. Обучение с подкреплением

Обучение с подкреплением — это раздел машинного обучения, где предполагается, что машина «живет» в определенном окружении и способна воспринимать **состояние** этого окружения как вектор характеристик. Машина может выполнять некоторые **действия** в каждом состоянии. Разные действия приносят разные **вознаграждения**, а также могут перевести машину в другое состояние окружения. Цель алгоритма обучения с подкреплением — выучить **линию поведения**.

Линия поведения — это стратегия (похожая на модель в обучении с учителем), которая принимает вектор признаков, описывающий состояние, и возвращает оптимальное действие для выполнения в этом состоянии. Действие является оптимальным, если приводит к максимальному **ожидаемому среднему вознаграждению**.

Обучение с подкреплением решает особый класс задач, когда решения принимаются последовательно, а цель является долгосрочной, например игра в видеоигру, роботизация производства, управление ресурсами или логистика. В этой книге основное внимание будет уделяться принятию единовременных решений, когда исходные данные не зависят друг от друга, и решений, принятых в прошлом. Обучение с подкреплением я оставлю за рамками этой книги.

1.3. Как работает обучение с учителем

В этом разделе я кратко объясню, как работает обучение с учителем, чтобы дать вам общую картину процесса, прежде чем углубиться в детали. В качестве примера

я выбрал обучение с учителем, так как этот тип машинного обучения чаще других используется на практике.

Процесс обучения с учителем начинается со сбора данных. Данные для этого вида обучения — это коллекция пар (вход, выход). Входными данными может быть все что угодно, например электронные письма, изображения или замеры, полученные от датчика. Выходными данными обычно являются действительные числа или метки (например, «спам», «не_спам», «кошка», «собака», «мышь» и т. д.). В некоторых случаях выходные данные могут быть представлены векторами (например, четырьмя координатами углов прямоугольника вокруг человека на картинке), последовательностями (например, [«прилагательное», «прилагательное», «существительное»] для входа «большая красивая машина») или иметь какую-то другую структуру.

Допустим, вы решили использовать обучение с учителем для решения задачи определения спама. Вы собираете данные, например, 10 000 электронных писем, каждое из которых снабжается меткой «спам» или «не_спам» (вы можете добавить эти метки вручную или отдать эту работу на аутсорсинг). Затем вы должны преобразовать каждое электронное письмо в вектор признаков.

На основе своего опыта специалист по анализу данных решает, как преобразовать сущность реального мира, такую как электронное письмо, в вектор признаков. Часто для преобразования текста в вектор признаков используется метод, называемый **мешком слов**. Его суть заключается в том, чтобы взять словарь (допустим, что он содержит 20 000 слов, отсортированных по алфавиту) и условиться, что в векторе признаков:

- первый признак равен 1, если электронное письмо содержит слово «а» (союз), и 0 в другом случае;
- второй признак равен 1, если электронное письмо содержит слово «аарон» (имя), и 0 в другом случае;
- ...;
- признак в позиции 20 000 равен 1, если электронное письмо содержит слово «ящур» (болезнь), и 0 в другом случае.

Вы повторяете описанную процедуру для каждого электронного письма в вашей коллекции и получаете 10 000 векторов признаков (каждый вектор имеет размерность 20 000) и меток («спам»/«не_спам»).

Теперь у вас есть машиночитаемые входные данные, но выходные метки по-прежнему имеют вид простого текста. Некоторые алгоритмы обучения требуют преобразования меток в числа. Например, некоторые алгоритмы требуют исполь-

зовать числа 0 (для представления метки «не_спам») и 1 (для представления метки «спам»). Модель машинного обучения, которую я использую для иллюстрации обучения с учителем, называется **методом опорных векторов (Support Vector Machine, SVM)**. Она требует, чтобы положительная метка (в данном случае «спам») имела числовое значение +1 (один), а отрицательная метка («не_спам») имела значение –1 (минус один).

Теперь у вас есть **набор данных** и **алгоритм обучения** и вы готовы применить алгоритм обучения к набору данных, чтобы получить **модель**.

SVM рассматривает каждый вектор признаков как точку в многомерном пространстве (в данном случае пространство имеет 20 000 измерений). Алгоритм помещает все векторы признаков на воображаемый 20 000-мерный график и рисует воображаемую 19 999-мерную линию (*гиперплоскость*), которая отделяет данные с положительными метками от данных с отрицательными метками. Граница, разделяющая данные разных классов, в машинном обучении называется **границей принятия решения**.

Уравнение гиперплоскости задается двумя **параметрами**: вещественным вектором **w** той же размерности, что и входной вектор признаков **x**, и действительным числом b, например:

$$\mathbf{wx} - b = 0,$$

где выражение **wx** означает $w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + ... + w^{(D)}x^{(D)}$, а D — число измерений в векторе признаков **x**.

(Сейчас некоторые уравнения могут показаться вам сложными, но в главе 2 мы рассмотрим необходимые математические и статистические понятия. А пока просто попробуйте понять происходящее в меру своих знаний. Многое прояснится, когда вы прочитаете следующую главу.)

Теперь прогнозируемую метку для некоторого входного вектора признаков **x** можно выразить так:

$$y = \text{sign}(\mathbf{wx} - b),$$

где sign — это математический оператор, принимающий произвольное значение и возвращающий +1, если входное значение является положительным числом, и –1, если входное значение является отрицательным числом.

Цель алгоритма обучения, в данном случае SVM, используя набор данных, найти оптимальные значения $\mathbf{w}^*$ и b^* для параметров **w** и b. После того как

алгоритм обучения найдет эти оптимальные значения, модель $f(\mathbf{x})$ будет определяться как:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^*\mathbf{x} - b^*).$$

Чтобы с помощью модели SVM предсказать, является ли электронное письмо спамом или нет, вы должны взять текст письма, преобразовать его в вектор признаков, затем умножить этот вектор на $\mathbf{w}^*$, вычесть b^* и взять знак результата. Это даст нам прогноз (+1 означает «спам», а –1 означает «не_спам»).

Но как машина находит $\mathbf{w}^*$ и b^*? Она решает задачу оптимизации. Машины хорошо справляются с оптимизацией функций в условиях ограничений.

Итак, какие ограничения должны удовлетворяться здесь? Прежде всего, модель должна правильно предсказывать метки имеющихся 10 000 данных. Напомню, что каждый образец $i = 1, ..., 10\,000$ задается парой $(\mathbf{x}_i, y_i)$, где $\mathbf{x}_i$ — вектор признаков i-го образца, а y_i — его метка, которая принимает значение –1 или +1. Ограничения выглядят следующим образом:

$$\mathbf{w}\mathbf{x}_i - b \geq +1, \text{ если } y_i = +1,$$
$$\mathbf{w}\mathbf{x}_i - b \leq -1, \text{ если } y_i = -1.$$

Желательно также, чтобы гиперплоскость отделяла положительные данные от отрицательных с максимальным **зазором**. **Зазор** — это расстояние между ближайшими образцами двух классов, отделяемых границей принятия решения. Большой зазор способствует лучшему **обобщению**, то есть тому, насколько хорошо модель будет классифицировать новые данные. Для максимизации зазора нужно минимизировать евклидову норму $\mathbf{w}$, которая обозначается как $\|\mathbf{w}\|$ и определяется выражением $\sqrt{\sum_{j=1}^{D}(w^{(j)})^2}$.

Машина должна решить задачу оптимизации, которая формулируется так:

Минимизировать $\|\mathbf{w}\|$ *с учетом* $y_i(\mathbf{w}\mathbf{x}_i - b) \geq 1$ для $i = 1, ..., N$. Выражение $y_i(\mathbf{w}\mathbf{x}_i - b) \geq 1$ — это всего лишь компактная запись двух ограничений выше.

Решение этой задачи оптимизации, обозначенной параметрами $\mathbf{w}^*$ и b^*, называется **статистической моделью**, или просто **моделью**. Процесс построения модели называется **обучением**.

Для двумерных векторов задачу и решение можно представить визуально, как показано на рис. 1.1. Синие и оранжевые точки представляют положительные и отрицательные образцы соответственно, а линия, заданная уравнением $\mathbf{w}\mathbf{x} - b = 0$, представляет границу принятия решения.

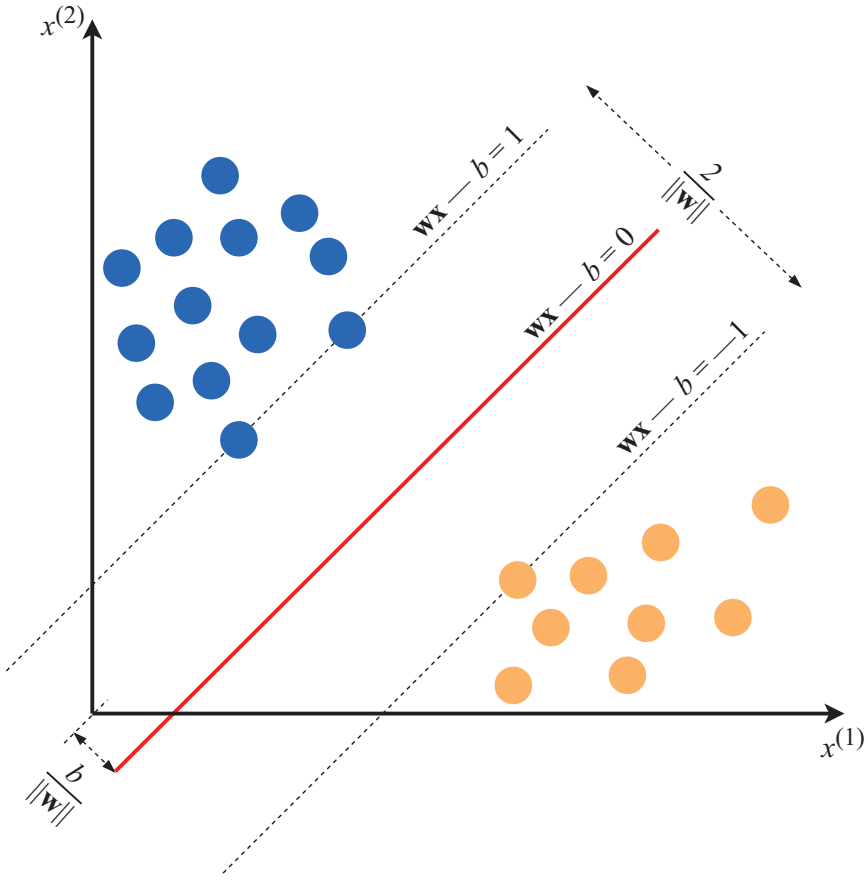


Рис. 1.1. Пример модели SVM для двумерных векторов признаков

Почему минимизация нормы $\mathbf{w}$ дает в результате наибольший зазор между двумя классами? Геометрически уравнения $\mathbf{wx} - b = 1$ и $\mathbf{wx} - b = -1$ определяют две параллельные гиперплоскости, как показано на рис. 1.1. Расстояние между этими гиперплоскостями равно $\frac{2}{\|\mathbf{w}\|}$, поэтому чем меньше норма $\|\mathbf{w}\|$, тем больше расстояние между этими двумя гиперплоскостями.

Так работает метод опорных векторов (SVM). Эта конкретная версия алгоритма строит так называемую *линейную модель*. Она называется линейной, потому что граница принятия решения — это прямая линия (или плоскость, или гиперплоскость). Метод опорных векторов также может включать **ядра**, способные сделать границу решения произвольно нелинейной. В некоторых случаях невозможно полностью

разделить две группы точек из-за шума в данных, ошибок разметки или **аномалий** (данных, сильно отличающихся от «типичного» образца в наборе данных). Для таких случаев есть версия алгоритма SVM, способная включать гиперпараметр штрафа[1] за неправильную классификацию обучающих данных конкретных классов. Более подробно алгоритм SVM мы рассмотрим в главе 3.

Сейчас вы должны усвоить следующее: любой алгоритм решения задачи классификации при построении модели явно или неявно создает границу решения. Граница решения может быть прямой, изогнутой, иметь сложную форму или образовываться наложением некоторых геометрических фигур. Форма границы решения определяет **точность** модели (то есть долю образцов, метки которых предсказываются моделью правильно). Форма границы решения и то, как она алгоритмически или математически вычисляется на основе обучающих данных, отличают один алгоритм обучения от другого.

На практике необходимо учитывать также две другие важные характеристики алгоритма обучения: скорость построения модели и время получения прогноза. Во многих практических ситуациях предпочтение отдается более быстрым алгоритмам, которые строят менее точные модели. Кроме того, иногда предпочтительнее иметь менее точную модель, но быстро вычисляющую прогнозы.

1.4. Почему модель способна работать с новыми данными

Почему модель, полученная в результате машинного обучения, способна правильно предсказывать метки для новых, ранее не встречавшихся ей данных? Чтобы понять это, посмотрите на график на рис. 1.1. Если два класса можно отделить друг от друга границей решения, то, очевидно, данные, принадлежащие каждому классу, расположены в двух разных подпространствах, которые создает граница решения.

Если данные, использованные для обучения, были выбраны случайным образом, независимо друг от друга и с использованием одной и той же процедуры, тогда статистически *весьма вероятно*, что новый отрицательный образец окажется на графике где-то не очень далеко от других отрицательных данных. То же касается нового положительного образца: он, *скорее всего*, окажется где-то среди других положительных данных. В таком случае наша граница решения *с высокой вероят-*

[1] Гиперпараметр — это свойство алгоритма обучения, обычно (но не всегда) имеющее числовое значение. Это значение влияет на работу алгоритма. Значения гиперпараметров не определяются самим алгоритмом из данных. Они должны задаваться аналитиком перед запуском алгоритма.

ностью будет надежно отделять друг от друга новые положительные и отрицательные данные. В других *менее вероятных* ситуациях наша модель будет совершать ошибки, но, поскольку такие ситуации менее вероятны, число ошибок, скорее всего, будет меньше числа правильных прогнозов.

Очевидно, что чем больше набор обучающих данных, тем менее вероятно, что новые данные будут отличаться (и лежать на графике далеко) от данных, использованных для обучения.

Для минимизации вероятности ошибок прогнозирования на новых образцах алгоритм SVM в поисках наибольшего зазора, явным образом пытается провести границу решения так, чтобы она лежала как можно дальше от данных обоих классов.

Читателю, желающему узнать больше об *обучаемости* и понять связь между ошибкой модели, размером обучающего набора, формой математического уравнения, определяющего модель, и временем построения модели, рекомендуется прочитать о *вероятностном приблизительно корректном обучении* (Probably Approximately Correct, PAC). Теория вероятностно-приблизительного корректного обучения поможет проанализировать и понять, сможет ли и при каких условиях алгоритм обучения получить приблизительно корректный классификатор.

2 Обозначения и определения

2.1. Обозначения

Начнем с того, что повторим математические обозначения, которые мы все учили в школе, но некоторые, вероятно, забыли их сразу после выпускного.

2.1.1. Структуры данных

Скаляр — это простое числовое значение, например, 15 или –3.25. Переменные или константы, принимающие скалярные значения, обозначаются наклонным шрифтом, например x или a.

Вектор — это упорядоченный список скалярных значений, называемых атрибутами. Мы будем обозначать **векторы жирным шрифтом**, например, **x** или **w**. Векторы можно изобразить в виде стрелок, указывающих в некоторых направлениях, а также в виде точек в многомерном пространстве. Для примера на рис. 2.1 показаны три двумерных вектора, **a** = [2, 3], **b** = [–2, 5] и **c** = [1, 0]. Атрибуты вектора мы будем обозначать наклонным шрифтом с индексом, например: $w^{(j)}$ или $x^{(j)}$. Индекс j обозначает конкретное **измерение** вектора — позицию атрибута в списке. Например, в векторе **a**, изображенном на рис. 2.1 в виде красной стрелки, $a^{(1)} = 2$ и $a^{(2)} = 3$.

Обозначение $x^{(j)}$ не следует путать с выражением степени, например x^2 (x в квадрате) или x^3 (x в кубе). Если нам понадобится возвести в степень, например в квадрат, атрибут с индексом, мы запишем это так: $(x^{(j)})^2$.

Переменная может иметь два или более индексов: $x_i^{(j)}$ или $x_{i,j}^{(k)}$. Например, обсуждая нейронные сети, мы будем использовать запись $x_{l,u}^{(j)}$ для обозначения входного признака j узла u в слое l.

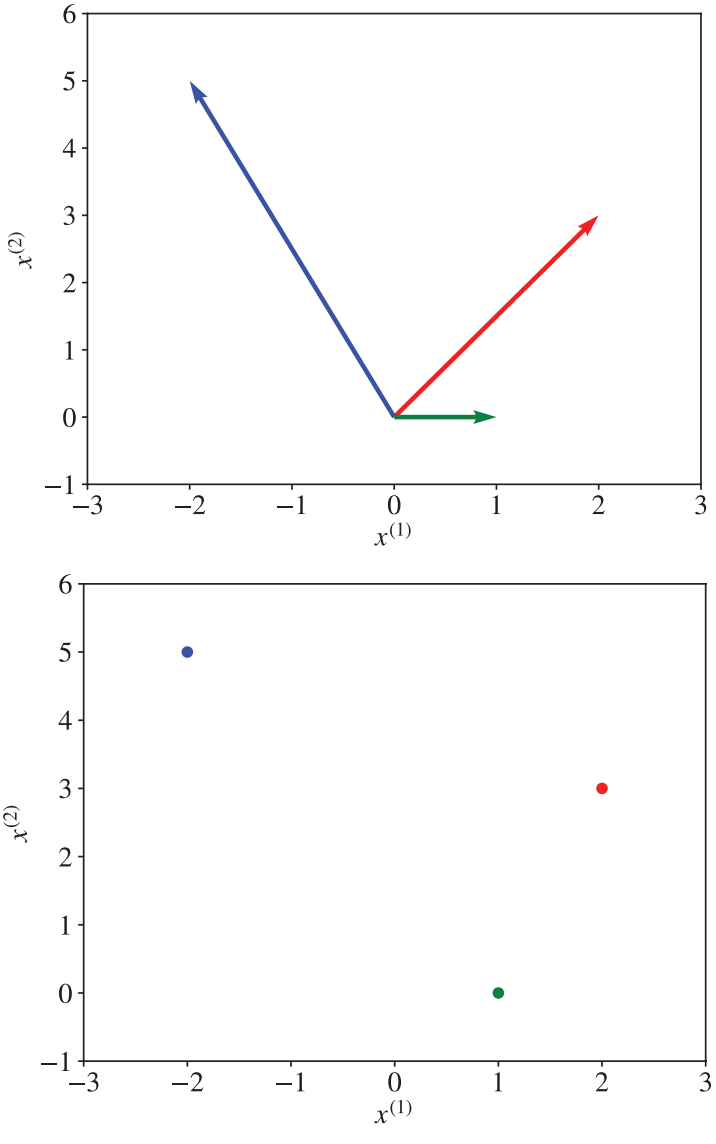


Рис. 2.1. Три вектора, изображенные как стрелки и точки

Матрица — это прямоугольный массив чисел, расположенных в строках и столбцах. Ниже приводится пример матрицы с двумя строками и тремя столбцами:

$$\begin{bmatrix} 2 & 4 & -3 \\ 21 & -6 & -1 \end{bmatrix}.$$

Матрицы будут обозначаться заглавными буквами и жирным шрифтом, например: **A** или **W**.

Множество — это неупорядоченная коллекция уникальных элементов. Мы будем обозначать множества заглавными буквами, например *S*. Множества чисел могут быть конечными (содержать фиксированное количество значений). В этом случае мы будем обозначать их с использованием фигурных скобок, например {1, 3, 18, 23, 235} или $\{x_1, x_2, x_3, x_4, ..., x_n\}$. Множество может быть бесконечным и содержать все значения, принадлежащие некоторому интервалу. Если множество включает все значения между *a* и *b*, в том числе *a* и *b*, мы будем обозначать его с использованием квадратных скобок: [*a*, *b*]. Если множество не включает значения *a* и *b*, такое множество мы будем обозначать с использованием круглых скобок: (*a*, *b*). Например, множество [0, 1] включает в себя такие значения, как 0, 0.0001, 0.25, 0.784, 0.9995 и 1.0. Специальное множество, обозначаемое как $\mathbb{R}$, включает все числа от минус бесконечности до плюс бесконечности.

Если элемент *x* принадлежит множеству $\mathcal{S}$, мы выражаем это как $x \in \mathcal{S}$. Мы можем получить новое множество $\mathcal{S}_3$ как **пересечение** двух множеств $\mathcal{S}_1$ и $\mathcal{S}_2$. Это выражается как $\mathcal{S}_3 \leftarrow \mathcal{S}_1 \cap \mathcal{S}_2$. Например, $\{1, 3, 5, 8\} \cap \{1, 8, 4\}$ дает в результате новое множество {1, 8}.

Новое множество $\mathcal{S}_3$ можно получить как **объединение** двух множеств $\mathcal{S}_1$ и $\mathcal{S}_2$. Мы будем выражать это так: $\mathcal{S}_3 \leftarrow \mathcal{S}_1 \cup \mathcal{S}_2$. Например, $\{1, 3, 5, 8\} \cup \{1, 8, 4\}$ дает в результате новое множество {1, 3, 4, 5, 8}.

2.1.2. Обозначения со знаком суммы

Суммирование элементов коллекции $X = \{x_1, x_2, \ldots x_{n-1}, x_n\}$ или атрибутов вектора $\mathbf{x} = \left[x^{(1)}, x^{(2)}, \ldots x^{(m-1)}, x^{(m)}\right]$ обозначается так:

$$\sum_{i=1}^{n} x_i \overset{\text{def}}{=} x_1 + x_2 + \ldots + x_{n-1} + x_n \quad \text{или так:} \quad \sum_{j=1}^{m} x^{(j)} \overset{\text{def}}{=} x^{(1)} + x^{(2)} + \ldots + x^{(m-1)} + x^{(m)}.$$

Оператор $\overset{\text{def}}{=}$ обозначает «определен как».

2.1.3. Обозначения со знаком произведения

По аналогии с суммированием существует *обозначение, выражающее произведение*. Оно обозначает произведение элементов коллекции или атрибутов вектора:

$$\prod_{i=1}^{n} x_i \overset{\text{def}}{=} x_1 \cdot x_2 \cdot \ldots \cdot x_{n-1} \cdot x_n,$$

где $a \cdot b$ означает умножение a на b. Там, где это возможно, мы будем опускать оператор умножения $(\cdot)$, упрощая запись произведения до ab, также обозначающего умножение a на b.

2.1.4. Операции с множествами

Оператор создания производного множества выглядит следующим образом: $\mathcal{S}' \leftarrow \left\{x^2 \middle| x \in \mathcal{S}, x > 3\right\}$. Это выражение означает, что создается новое множество $\mathcal{S}'$, в которое помещаются квадраты тех элементов x, принадлежащих $\mathcal{S}$, значения которых больше 3.

Оператор мощности $\left|\mathcal{S}\right|$ возвращает число элементов в множестве $\mathcal{S}$.

2.1.5. Операции с векторами

Сумма двух векторов $\mathbf{x} + \mathbf{z}$ определяется как вектор

$$\left[x^{(1)} + z^{(1)},\ x^{(2)} + z^{(2)},\ \ldots,\ x^{(m)} + z^{(m)}\right].$$

Разность двух векторов $\mathbf{x} - \mathbf{z}$ определяется как вектор

$$\left[x^{(1)} - z^{(1)},\ x^{(2)} - z^{(2)},\ \ldots,\ x^{(m)} - z^{(m)}\right].$$

Произведение вектора на скаляр — это вектор. Например, $\mathbf{x}c \overset{\text{def}}{=} \left[cx^{(1)},\ cx^{(2)},\ \ldots,\ cx^{(m)}\right]$.

Скалярное произведение двух векторов — это скаляр. Например, $\mathbf{wx} \overset{\text{def}}{=} \sum_{i=1}^{m} w^{(i)} x^{(i)}$.

В некоторых книгах скалярное произведение записывается как $\mathbf{w} \cdot \mathbf{x}$. Два вектора должны иметь одинаковую размерность. Иначе результат скалярного произведения не определен.

Произведение матрицы $\mathbf{W}$ на вектор $\mathbf{x}$ дает в результате новый вектор. Допустим, у нас есть матрица

$$\mathbf{W} = \begin{bmatrix} w^{(1,1)} & w^{(1,2)} & w^{(1,3)} \\ w^{(2,1)} & w^{(2,2)} & w^{(2,3)} \end{bmatrix}.$$

Когда в операциях с матрицами участвуют векторы, по умолчанию вектор представляется в виде матрицы с одним столбцом. Когда в выражении умножения вектор находится справа от матрицы, он остается вектором-столбцом. Умножить матрицу

на вектор можно, только если количество строк в векторе совпадает с количеством столбцов в матрице. Пусть у нас есть вектор $\mathbf{x} \overset{\text{def}}{=} \begin{bmatrix} x^{(1)}, & x^{(2)}, & x^{(3)} \end{bmatrix}$. Тогда результатом произведения **Wx** будет двумерный вектор:

$$\mathbf{Wx} = \begin{bmatrix} w^{(1,1)} & w^{(1,2)} & w^{(1,3)} \\ w^{(2,1)} & w^{(2,2)} & w^{(2,3)} \end{bmatrix} \overset{def}{\begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \end{bmatrix}} =$$

$$= \begin{bmatrix} w^{(1,1)}x^{(1)} + w^{(1,2)}x^{(2)} + w^{(1,3)}x^{(3)} \\ w^{(2,1)}x^{(1)} + w^{(2,2)}x^{(2)} + w^{(2,3)}x^{(3)} \end{bmatrix} =$$

$$= \begin{bmatrix} \mathbf{w}^{(1)}\mathbf{x} \\ \mathbf{w}^{(2)}\mathbf{x} \end{bmatrix}.$$

Если бы матрица имела, скажем, пять строк, в результате умножения получился бы пятимерный вектор.

Когда в выражении умножения вектор находится слева от матрицы, его нужно **транспонировать** (повернуть) перед умножением. Транспонирование вектора **x** обозначается как $\mathbf{x}^{\mathrm{T}}$ и преобразует вектор-столбец в вектор-строку. Допустим,

$$\mathbf{x} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix}, \text{ тогда } \mathbf{x}^{\mathrm{T}} \overset{\text{def}}{=} \begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}.$$

Умножение вектора **x** на матрицу **W** обозначается как $\mathbf{x}^{\mathrm{T}}\mathbf{W}$,

$$\mathbf{x}^{\mathrm{T}}\mathbf{W} = \begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix} \begin{bmatrix} w^{(1,1)} & w^{(1,2)} & w^{(1,3)} \\ w^{(2,1)} & w^{(2,2)} & w^{(2,3)} \end{bmatrix} =$$

$$\overset{\text{def}}{=} \begin{bmatrix} w^{(1,1)}x^{(1)} + w^{(2,1)}x^{(2)}, & w^{(1,2)}x^{(1)} + w^{(2,2)}x^{(2)}, & w^{(1,3)}x^{(1)} + w^{(2,3)}x^{(2)} \end{bmatrix}.$$

Как видите, умножить вектор на матрицу можно только в том случае, если число измерений в векторе совпадает с числом строк в матрице.

2.1.6. Функции

Функция — это отношение, связывающее каждый элемент x из множества X (**области определения** функции) с единственным элементом y из другого множества Y

(**области значений** функции). Функция обычно имеет имя. Если функция называется f, тогда отношение обозначается как $y = f(x)$ (читается как «эф от икс»), где элемент x является аргументом, или входом, функции, а y — значением, или выходом, функции.

Символ, представляющий аргумент, — это переменная функции (мы часто будем говорить, что f — это функция от переменной x).

Мы говорим, что $f(x)$ имеет **локальный минимум** при $x = c$, если $f(x) \geq f(c)$ для каждого значения x в некотором открытом интервале вокруг $x = c$. **Интервал** — это множество действительных чисел, такое, что любое число, которое находится между двумя числами в множестве, также входит в это множество. Открытый интервал не включает конечные точки и обозначается круглыми скобками. Например, (0, 1) означает «все числа больше 0 и меньше 1». Минимальное значение среди всех локальных минимумов называется **глобальным минимумом** (рис. 2.2).

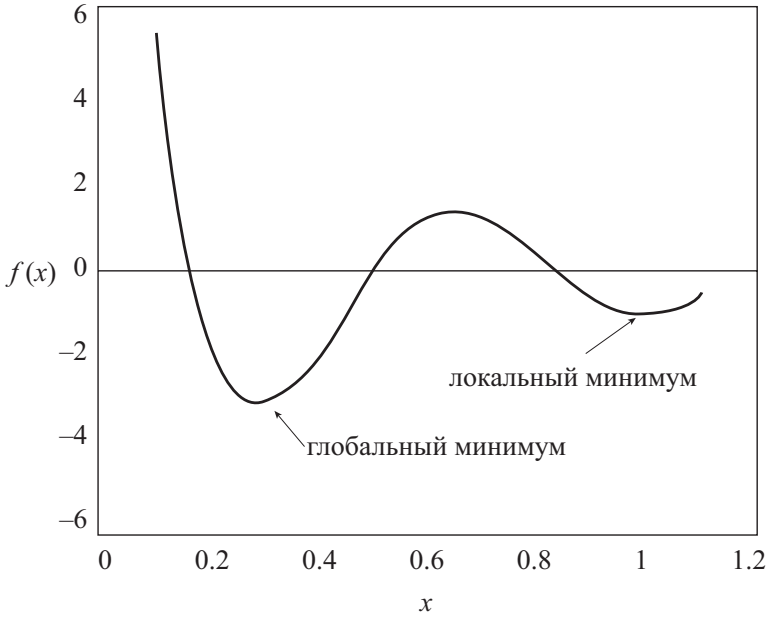


Рис. 2.2. Локальный и глобальный минимумы функции

Векторная функция, обозначаемая как $\mathbf{y} = f(x)$, — это функция, возвращающая вектор **y**. Аргумент такой функции может быть вектором или скаляром.

2.1.7. Операторы max и arg max

Пусть дано множество значений $\mathcal{A} = \{a_1, a_2, ...a_n\}$, тогда оператор $\max_{a \in \mathcal{A}} f(a)$ вернет наибольшее значение $f(a)$ из всех элементов в множестве $\mathcal{A}$. С другой стороны, оператор $\arg\max_{a \in \mathcal{A}} f(a)$ вернет элемент из множества $\mathcal{A}$, который максимизирует $f(a)$.

Иногда, когда множество задано неявно или бесконечно, запись этих операторов можно сократить до $\max_a f(a)$ или $\arg\max_a f(a)$.

Существуют также аналогичные операторы min и arg min.

2.1.8. Оператор присваивания

Выражение $a \leftarrow f(x)$ означает, что переменная a получает новое значение: результат $f(x)$. Мы говорим, что переменной a присваивается новое значение. Аналогично, $\mathbf{a} \leftarrow [a_1, a_2]$ означает, что векторная переменная **a** получает значение двумерного вектора $[a_1, a_2]$.

2.1.9. Производная и градиент

Производная f' функции f — это функция или значение, которое описывает, как быстро f растет (или уменьшается). Если производная имеет постоянное значение, например 5 или –3, значит, функция постоянно растет (или уменьшается) в любой точке x области определения. Если производная f' является функцией, тогда функция f может расти с разной скоростью в разных точках в области определения. Если производная f' положительна в некоторой точке x, значит, функция f растет в этой точке. Если производная от f отрицательна в некоторой точке x, значит, функция уменьшается в этой точке. Производная, равная нулю в точке x, означает, что функция f в этой точке не растет и не уменьшается (то есть это точка минимума или максимума функции f).

Поиск производной называется дифференцированием **дифференцированием**.

Производные для основных функций известны. Например, для $f(x) = x^2$ производная имеет вид $f'(x) = 2x$; для $f(x) = 2x$ производная имеет вид $f'(x) = 2$; для $f(x) = 2$ производная имеет вид $f'(x) = 0$ (производная любой функции $f(x) = c$, где c — константа, равна нулю).

Если функция, которую нужно дифференцировать, не относится к числу основных, ее производную можно найти, используя **правило дифференцирования сложной функции**. Например, если $F(x) = f(g(x))$, где f и g — некоторые функции, тогда $F'(x) = f'(g(x))g'(x)$. Например, если $F(x) = (5x + 1)^2$, тогда

$g(x) = 5x + 1$ и $f(g(x)) = (g(x))^2$. Применяя правило выше, находим $F'(x) = 2(5x + 1)$ $g'(x) = 2(5x + 1)5 = 50x + 10$.

Градиент — это обобщение производной для функций нескольких аргументов (или одного аргумента, представленного вектором или какой-либо другой сложной структурой). Градиент функции — это вектор **частных производных**. Поиск частной производной функции можно рассматривать как процесс поиска производной, когда изменяется только один из входов, а остальные имеют постоянные значения.

Например, если функция определена как $f\left(\left[x^{(1)},\ x^{(2)}\right]\right) = ax^{(1)} + bx^{(2)} + c$, тогда частная производная функции f относительно $x^{(1)}$, обозначаемая как $\frac{\partial f}{\partial x^{(1)}}$, будет определяться формулой

$$\frac{\partial f}{dx^{(1)}} = a + 0 + 0 = a,$$

где a — производная функции $ax^{(1)}$; два нуля соответствуют производным от $bx^{(2)}$ и c, потому что когда вычисляется производная по $x^{(1)}$, $x^{(2)}$ считается константой, а производная любой константы равна нулю.

Аналогично, частная производная функции f относительно $x^{(2)}$, $\frac{\partial f}{\partial x^{(2)}}$, будет определяться формулой

$$\frac{\partial f}{\partial x^{(2)}} = 0 + b + 0 = b.$$

Градиент функции f обозначается как ∇f и задается вектором $\left[\frac{\partial f}{\partial x^{(1)}}, \frac{\partial f}{\partial x^{(2)}}\right]$.

Правило дифференцирования сложной функции применимо и к частным производным, как я покажу в главе 4.

2.2. Случайная величина

Случайная величина, которая обычно обозначается курсивной заглавной буквой, например X, — это величина, возможные значения которой являются числовыми результатами случайного явления. Примерами случайных явлений с числовым результатом могут служить бросок монеты (0 для решки и 1 для орла), бросок игрального кубика или рост первого незнакомца, встретившегося на улице. Существует два типа случайных величин: **дискретные** и **непрерывные**.

Дискретная случайная величина принимает конечное или счетное множество значений, например: *красный*, *желтый*, *синий* или 1, 2, 3, ...

Распределение вероятностей дискретной случайной величины описывается списком вероятностей, связанных с каждым из возможных значений. Этот список вероятностей называется **функцией распределения дискретной случайной величины** (*probability mass function, pmf*). Например: Pr(*X* = *красный*) = 0.3, Pr(*X* = *желтый*) = 0.45, Pr(*X* = *синий*) = 0.25. Каждая вероятность в функции распределения дискретной случайной величины — это значение, большее или равное 0. Сумма вероятностей равна 1 (рис. 2.3а).

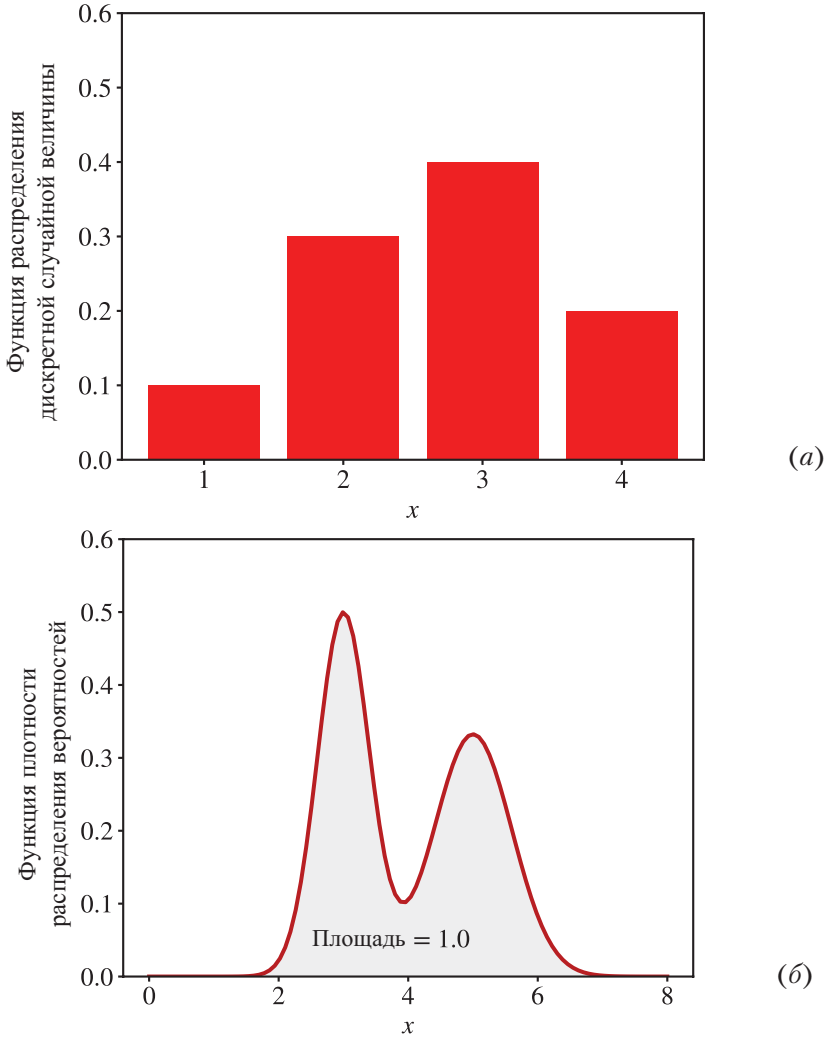


Рис. 2.3. Функция распределения дискретной случайной величины (*а*) и функция плотности распределения вероятностей (*б*)

Непрерывная случайная величина (*continuous random variable*, *CRV*) может принимать бесконечное число возможных значений в некотором интервале. Примерами могут служить рост, вес и время. Поскольку число значений непрерывной случайной величины X бесконечно, вероятность $\Pr(X = c)$ для любого c равна 0. Поэтому распределение вероятностей CRV (непрерывное распределение вероятностей) описывается не списком, а **функцией плотности вероятности** (*probability density function*, *pdf*). Функция плотности вероятности — это функция с неотрицательной областью значений, а площадь под кривой этой функции равна 1 (рис. 2.3б).

Пусть дискретная случайная величина X имеет k возможных значений $\{x_i\}_{i=1}^{k}$. **Ожидание** X, обозначаемое как $\mathbb{E}[X]$, определяется формулой

$$\begin{aligned}\mathbb{E}[X] &\overset{def}{=} \sum_{i=1}^{k}\left[x_i \cdot \Pr(X = x_i)\right] = \\ &= x_1 \cdot \Pr(X = x_1) + x_2 \cdot \Pr(X = x_2) + \cdots + x_k \cdot \Pr(X = x_k),\end{aligned} \tag{2.1}$$

где $\Pr(X = x_i)$ — вероятность, что X имеет значение x_i в соответствии с pmf. Ожидание случайной величины также называется **средним**, или **ожидаемым**, **значением** и часто обозначается буквой μ. Ожидание является одной из наиболее важных **статистических характеристик** случайной величины.

Еще одной важной статистической характеристикой является **стандартное отклонение**, которое определяется как

$$\sigma \overset{\text{def}}{=} \sqrt{\mathbb{E}\left[(X-\mu)^2\right]}.$$

Дисперсия, обозначаемая как σ^2 или $var(X)$, определяется формулой

$$\sigma^2 = \mathbb{E}\left[(X-\mu)^2\right].$$

Для дискретной случайной величины стандартное отклонение определяется как:

$$\sigma = \sqrt{\Pr(X = x_1)(x_1-\mu)^2 + \Pr(X = x_2)(x_2-\mu)^2 + \cdots + \Pr(X = x_k)(x_k-\mu)^2},$$

где $\mu = \mathbb{E}[X]$.

Ожидание непрерывной случайной величины X определяется формулой

$$\mathbb{E}[X] \overset{\text{def}}{=} \int_{\mathbb{R}} x f_X(x)\,dx, \tag{2.2}$$

где f_X — это функция плотности вероятности (pdf) величины X, а $\int_{\mathbb{R}}$ — интеграл от функции xf_X.

Интеграл является эквивалентом суммы всех значений функции, когда функция имеет непрерывную область определения. Он равен площади под кривой функции. Функция плотности вероятности обладает замечательным свойством: площадь под ее кривой всегда равна 1, то есть математически это означает, что $\int_{\mathbb{R}} f_X(x)dx = 1$.

В большинстве случаев f_X неизвестна, но мы можем наблюдать некоторые значения X. В машинном обучении мы называем эти значения **образцами**, а набор этих данных — **выборкой** или **набором данных**.

2.3. Несмещенные оценки

Так как обычно f_X неизвестна, но имеется выборка $S_X = \{x_i\}_{i=1}^N$, мы часто довольствуемся не истинными значениями статистических характеристик распределения вероятностей, такими как ожидание, а их **несмещенными оценками**.

Мы говорим, что $\hat{\theta}(S_X)$ — это несмещенная оценка некоторой статистической характеристики θ, вычисленной по выборке S_X с неизвестным распределением вероятностей, если $\hat{\theta}(S_X)$ обладает следующим свойством:

$$\mathbb{E}\left[\hat{\theta}(S_X)\right] = \theta,$$

где $\hat{\theta}$ — **статистическая характеристика выборки**, полученная по выборке S_X, а не реальная статистическая характеристика, которую можно получить, только зная X; ожидание определяется по всем возможным выборкам из X. Это означает, что при наличии неограниченного количества выборок, таких как S_X, вычислив некоторую несмещенную оценку, такую как $\hat{\mu}$, для каждой выборки и взяв среднее значение для всех этих $\hat{\mu}$, вы получите значение реальной статистической характеристики μ, которую вычислили бы по X.

Можно показать, что несмещенная оценка неизвестного ожидания $\mathbb{E}[X]$ (заданного уравнением 2.1 или 2.2) равна $\frac{1}{N}\sum_{i=1}^N x_i$ (в статистике называется **выборочное среднее**).

2.4. Правило Байеса

Условная вероятность $\Pr(X = x \mid Y = y)$ — это вероятность того, что случайная величина X будет иметь конкретное значение x при условии, что другая случайная

величина Y имеет конкретное значение y. **Правило Байеса** (также известное как **теорема Байеса**) гласит:

$$\Pr\left(X=x\middle|Y=y\right)=\frac{\Pr\left(Y=y\middle|X=x\right)\Pr\left(X=x\right)}{\Pr\left(Y=y\right)}.$$

2.5. Оценка параметров

Правило Байеса удобно использовать, когда есть модель распределения X и эта модель f_θ является функцией, имеющей некоторые параметры в форме вектора θ. Примером такой функции может служить функция Гаусса с двумя параметрами, μ и σ, которая определяется как

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

где $\theta\overset{\text{def}}{=}\left[\mu,\sigma\right]$.

Эта функция имеет все свойства функции плотности распределения вероятностей (pdf)[1]. Следовательно, ее можно использовать как модель неизвестного распределения X. Мы можем получить значения параметров в векторе θ из данных, используя правило Байеса:

$$\Pr\left(\theta=\hat{\theta}\middle|X=x\right)\leftarrow\frac{\Pr\left(X=x\middle|\theta=\hat{\theta}\right)\Pr\left(\theta=\hat{\theta}\right)}{\Pr\left(X=x\right)}=\frac{\Pr\left(X=x\middle|\theta=\hat{\theta}\right)\Pr\left(\theta=\hat{\theta}\right)}{\sum_{\tilde{\theta}}\Pr\left(X=x\middle|\theta=\tilde{\theta}\right)\Pr\left(\theta=\hat{\theta}\right)}. \qquad (2.3)$$

где $\Pr\left(X=x\middle|\theta=\hat{\theta}\right)\overset{\text{def}}{=}f_{\hat{\theta}}$.

Если есть выборка S из X и множество возможных значений для θ конечно, мы легко сможем оценить $\Pr\left(\theta=\hat{\theta}\right)$, итеративно применив правило Байеса к каждому образцу $x\in S$. Начальное значение $\Pr\left(\theta=\hat{\theta}\right)$ можно выбрать таким, что $\sum_{\hat{\theta}}\Pr\left(\theta=\hat{\theta}\right)=1$. Это предположение о вероятностях для различных $\hat{\theta}$ называется **априорной вероятностью**.

Сначала вычислим $\Pr\left(\theta=\hat{\theta}\middle|X=x_1\right)$ для всех возможных значений $\hat{\theta}$. Затем, перед обновлением $\Pr\left(\theta=\hat{\theta}\middle|X=x\right)$, на этот раз для $x=x_2\in S$ с использованием уравне-

[1] На самом деле уравнение 2.3 определяет pdf одного из наиболее часто используемых на практике распределений вероятности, называемого гауссовым распределением, или нормальным распределением, и обозначаемого как $N(\mu,\sigma^2)$.

ния 2.3, заменим априорную вероятность $\Pr\left(\theta = \hat{\theta}\right)$ в уравнении 2.3 новой оценкой $\Pr\left(\theta = \hat{\theta}\right) \leftarrow \frac{1}{N}\sum_{x \in S} \Pr\left(\theta = \hat{\theta} \middle| X = x\right)$.

Лучшие значения параметров θ* в данном примере получаются с использованием принципа **максимума апостериорной вероятности** (maximum a posteriori, MAP):

$$\theta^* = \arg\max_{\theta} \prod_{i=1}^{N} \Pr\left(\theta = \hat{\theta} \middle| X = x_i\right). \tag{2.4}$$

Если множество возможных значений для θ бесконечно, то уравнение 2.4 нужно оптимизировать путем применения процедуры численной оптимизации, такой как градиентный спуск, который мы рассмотрим в главе 4. Обычно оптимизируется натуральный логарифм выражения в правой части уравнения 2.4, потому что логарифм произведения превращается в сумму логарифмов, а компьютерам проще работать с суммой, чем с произведением[1].

2.6. Параметры и гиперпараметры

Гиперпараметр — это свойство алгоритма обучения, обычно (но не всегда) имеющее числовое значение. Это значение влияет на работу алгоритма. Гиперпараметры не вычисляются алгоритмом. Они должны задаваться аналитиком перед запуском алгоритма. Как это сделать, я покажу в главе 5.

Параметры — это переменные, которые определяют модель, обучаемую алгоритмом обучения. Параметры напрямую изменяются алгоритмом обучения на основе обучающих данных. Цель обучения — найти такие значения параметров, которые делают модель оптимальной в определенном смысле.

2.7. Классификация и регрессия

Классификация — это задача автоматического определения **метки** для **неразмеченного образца**. Определение спама — один из ярких примеров классификации.

В машинном обучении задача классификации решается с помощью **алгоритма обучения классификации**, который на входе принимает набор **размеченных данных**

[1] Умножение большого количества чисел может дать очень маленький или очень большой результат. Это часто приводит к проблеме переполнения, когда компьютер оказывается не в состоянии сохранить в памяти такое экстремально маленькое или экстремально большое число.

и создает **модель**, которая принимает неразмеченный образец и возвращает либо его метку непосредственно, либо число, на основании которого аналитик сможет определить метку. Примером такого числа является вероятность.

В задаче классификации метка является членом конечного множества **классов**. Если размер множества классов равен двум («больной»/«здоровый», «спам»/«не_спам»), мы называем такую классификацию **бинарной классификацией** (в некоторых источниках ее также называют **биномиальной**). **Классификация с несколькими классами** (также называемая **мультиномиальной**, или **многоклассовой**) — это задача классификации с тремя или более классами[1].

Некоторые алгоритмы обучения допускают наличие более двух классов, но есть такие, которые по своей природе являются алгоритмами бинарной классификации. Существуют стратегии, позволяющие преобразовать алгоритм обучения бинарной классификации в алгоритм многоклассовой классификации. Об одной из них я расскажу в главе 7.

Регрессия — это задача прогнозирования метки с действительным значением (часто называют также **целевым значением**) для образца без метки. Оценка стоимости дома на основе таких его характеристик, как площадь, количество спален, расположение и т. д., — вот один из примеров регрессии.

Задача регрессии решается с помощью **алгоритма обучения регрессии**, который принимает на входе набор размеченных данных и создает модель, которая может по неразмеченному образцу предсказать его целевое значение.

2.8. Обучение на основе моделей и на основе примеров

Большинство алгоритмов обучения с учителем основаны на моделях. Мы уже видели один такой алгоритм: метод опорных векторов (SVM). Алгоритмы обучения на основе моделей анализируют обучающие данные и создают модель, **параметры** которой определяются в ходе анализа обучающих данных. В SVM мы видели два параметра: $\mathbf{w}^*$ и b^*. После создания модели обучающие данные можно отбросить.

Алгоритмы обучения на основе примеров используют весь набор данных в качестве модели. Одним из наиболее часто используемых на практике алгоритмов обучения на основе примеров является метод **k ближайших соседей** (**k-Nearest Neighbors**, **kNN**). В проблеме классификации, для того чтобы спрогнозировать метку для

[1] При этом каждому образцу соответствует единственная метка.

входного образца, алгоритм kNN просматривает ближайшие окрестности этого образца в пространстве векторов признаков и возвращает метку, которая чаще всего встречается в этой окрестности.

2.9. Поверхностное и глубокое обучение

Алгоритм поверхностного обучения выводит параметры модели непосредственно из признаков обучающих данных. Большинство алгоритмов обучения с учителем являются поверхностными. Известными исключениями являются алгоритмы обучения **нейронных сетей**, особенно те, что строят нейронные сети с несколькими **слоями** между входом и выходом. Такие нейронные сети называются **глубокими нейронными сетями**. В глубоком обучении нейронной сети (или просто глубоком обучении), в отличие от поверхностного обучения, большинство параметров модели выводится не из признаков обучающих данных непосредственно, а из значений, возвращаемых предыдущими слоями.

Не переживайте, если что-то пока остается неясным. Мы подробно рассмотрим нейронные сети в главе 6.

3 Фундаментальные алгоритмы

В этой главе я опишу пять самых известных алгоритмов, которые являются либо очень эффективными сами по себе, либо используются как строительные блоки в других эффективных алгоритмах обучения.

3.1. Линейная регрессия

Линейная регрессия — это популярный алгоритм обучения регрессии, который строит модель, являющуюся линейной комбинацией признаков входного образца.

3.1.1. Задача

Дано: коллекция размеченных данных $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, где N — размер коллекции, $\mathbf{x}_i$ — D-мерный вектор признаков образца $i = 1, ..., N$, y_i — действительное целевое значение[1], и каждый признак $x_i^{(j)}, j = 1, ..., D$ также является действительным числом.

Требуется: сконструировать модель $f_{w,b}(\mathbf{x})$, являющуюся линейной комбинацией признаков образца $\mathbf{x}$:

$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{x}\mathbf{w} + b, \tag{3.1}$$

где $\mathbf{w}$ — D-мерный вектор параметров, а b — действительное число. Запись $f_{\mathbf{w},b}$ означает, что модель f параметризуется двумя значениями: $\mathbf{w}$ и b.

---

[1] Чтобы показать, что y_i является действительным целевым значением, мы пишем $y_i \in \mathbb{R}$, где $\mathbb{R}$ обозначает множество всех действительных чисел, бесконечное множество чисел от минус бесконечности до плюс бесконечности.

Модель будет использоваться для предсказания неизвестного целевого значения y для данного $\mathbf{x}$: $y \leftarrow f_{w,\,b}(\mathbf{x})$. Разные модели, параметризованные разными парами $(\mathbf{w}, b)$, могут давать разные предсказания для одного и того же образца. Нужно найти оптимальные значения $(\mathbf{w}^*, b^*)$. Очевидно, что оптимальные значения параметров определяют модель, которая дает наиболее точные прогнозы.

Обратите внимание, что линейная модель в уравнении. 3.1 очень похожа на модель SVM. Единственное отличие — отсутствие оператора sign. Модели действительно очень похожи. Однако в SVM гиперплоскость играет роль границы принятия решения: она используется для отделения двух групп данных друг от друга. Как следствие, она должна проходить как можно дальше от каждой группы.

В линейной регрессии, напротив, гиперплоскость проводится так, чтобы оказаться как можно ближе ко всем обучающим образцам.

Чтобы понять, почему это последнее требование является обязательным, взгляните на рис. 3.1. Здесь изображена линия регрессии (красным цветом) для одномерных данных (синие точки). Эту линию можно использовать для прогнозирования целевого значения цели $y_{нов.}$ нового образца без метки $x_{нов.}$. Если бы данные были представлены D-мерными векторами признаков ($D > 1$), тогда решение отличалось бы от одномерного случая тем, что регрессионная модель была бы не линией, а плоскостью (в случае двух измерений) или гиперплоскостью (в случае $D > 2$).

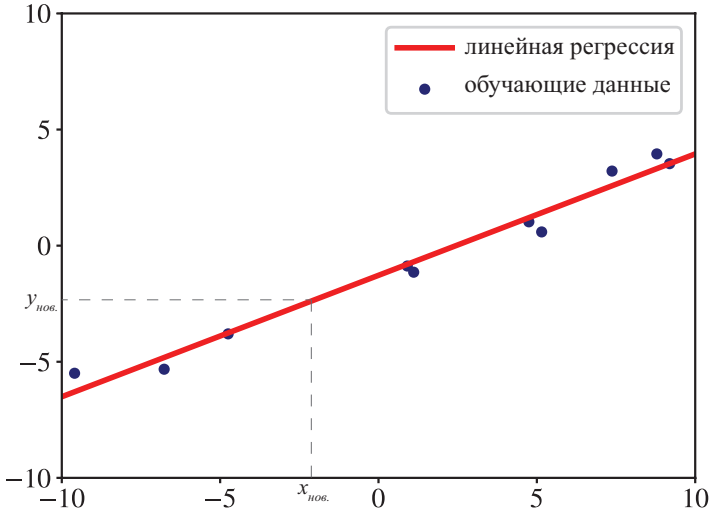


Рис. 3.1. Линейная регрессия для одномерных данных

Теперь должно быть понятно, почему так важно требование к расположению гиперплоскости регрессии как можно ближе к обучающим образцам: если бы красная линия на рис. 3.1 была далека от синих точек, прогноз $y_{нов.}$ имел бы меньше шансов оказаться верным.

3.1.2. Решение

Чтобы удовлетворить это последнее требование, процедура оптимизации, используемая для поиска оптимальных значений $\mathbf{w}^*$ и b^*, должна минимизировать следующее выражение:

$$\frac{1}{N}\sum_{i=1...N}\left(f_{\mathbf{w},b}(\mathbf{x}_i)-y_i\right)^2. \tag{3.2}$$

Выражение, которое требуется минимизировать или максимизировать, в математике называется целевой функцией, или просто целью. Выражение $(f_{w,b}(\mathbf{x}_i)-y_i)^2$ в целевой функции выше называется **функцией потерь**. Она определяет величину штрафа за неправильную классификацию образца i. Эта конкретная функция потерь называется **квадратичной функцией потерь** (squared error loss). Все алгоритмы обучения, основанные на моделях, используют функцию потерь, и в поисках лучшей модели мы пытаемся минимизировать цель, известную как **функция стоимости** (cost function). В линейной регрессии функция стоимости определяется как средняя потеря, также называемая **эмпирическим риском** (empirical risk). Средняя потеря, или эмпирический риск, для модели — это среднее всех штрафов, полученных при применении модели к обучающим данным.

Почему в линейной регрессии используется квадратичная функция потерь? И можно ли взять абсолютное значение разности между истинным целевым значением y_i и прогнозируемым значением $f(x_i)$ и использовать его в качестве штрафа? Можно. Более того, можно использовать любую четную степень вместо квадрата.

Теперь, возможно, вы начинаете понимать, сколько решений, на первый взгляд произвольных, принимается при разработке алгоритма машинного обучения: в данном примере мы решили использовать линейную комбинацию признаков для прогнозирования целевого значения. Однако для объединения значений признаков мы могли бы использовать квадрат или другой многочлен. Мы могли бы также использовать другую функцию потерь, не лишенную смысла: абсолютная разность между $f(x_i)$ и y_i имеет смысл, куб разности — тоже; **бинарная функция потерь** (возвращающая 1, когда $f(x_i)$ и y_i различны, и 0 — когда они одинаковы) тоже имеет смысл, верно?

Приняв другое решение о форме модели, форме функции потерь и о выборе алгоритма минимизации средней потери, чтобы найти лучшие значения параметров, мы в итоге изобрели бы другой алгоритм машинного обучения. Кажется, все просто, не правда ли? Но не спешите изобретать новый алгоритм обучения. Тот факт, что он отличается от имеющихся, не означает, что он будет работать лучше.

Новые алгоритмы обучения изобретают по одной из двух основных причин:

1. Новый алгоритм решает конкретную практическую задачу лучше существующих.
2. Новый алгоритм имеет более надежные теоретические гарантии качества производимой им модели.

Одним из практических обоснований выбора модели линейной формы является ее простота. Зачем использовать сложную модель, если можно использовать простую? Другое обоснование: линейные модели в меньшей степени подвержены эффекту переобучения. **Переобучение** (overfitting) — это свойство модели очень хорошо предсказывать метки данных, использовавшихся для обучения, но часто допускать ошибки при применении к образцам, которые алгоритм обучения не видел прежде.

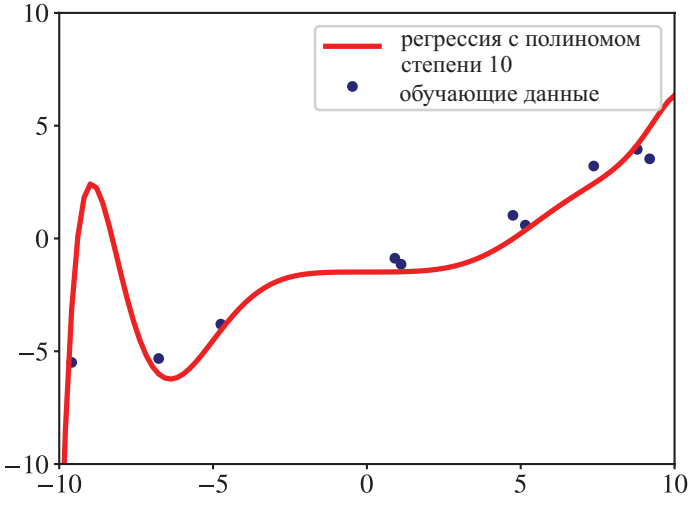


Рис. 3.2. Переобучение

На рис. 3.2 показан пример переобучения в регрессии. Для построения красной линии регрессии использовались те же данные, что и для линии регрессии на рис. 3.1. Разница лишь в том, что на этот раз выполнялась полиномиальная регрессия с по-

линомом степени 10[1]. Линия регрессии почти идеально предсказывает целевые значения для большинства обучающих данных, но, скорее всего, будет допускать значительные ошибки на новых данных, как показано на рис. 3.1 для $x_{нов.}$. Мы еще вернемся к проблеме переобучения в главе 5 и поговорим о том, как его избежать.

Теперь вы знаете одно из преимуществ линейной регрессии: она мало подвержена переобучению. А что можно сказать в отношении выбора квадратичной функции потерь? Почему мы решили, что потери нужно возводить в квадрат? В 1805 году французский математик Адриен-Мари Лежандр (Adrien-Marie Legendre), который первым опубликовал метод подсчета суммы квадратов для оценки качества моделей, заявил, что возведение ошибки в квадрат до суммирования — это *удобно*. Почему он так сказал? Абсолютное значение неудобно, потому что не имеет непрерывной производной, что делает функцию негладкой. Негладкие функции создают ненужные сложности, когда для поиска аналитических решений оптимизационных задач используются методы линейной алгебры. Аналитические решения для нахождения оптимума функции — это простые алгебраические выражения, и они часто предпочтительнее использования сложных численных методов оптимизации, таких как **градиентный спуск** (используется для обучения нейронных сетей, кроме всего прочего).

Очевидно, что квадраты штрафов выгодны еще и потому, что преувеличивают разность между истинным и прогнозируемым целевыми значениями, в соответствии с величиной этой разности. Также можно использовать другие четные степени, такие как 4 или 6,, но работать с их производными сложнее.

Наконец, зачем нам нужна производная средней потери? Вычислив градиент функции в уравнении 3.2, мы сможем затем установить этот градиент в ноль[2] и найти решение системы уравнений, которое даст нам оптимальные значения $\mathbf{w}^*$ и b^*.

3.2. Логистическая регрессия

Сразу должен сказать, что логистическая регрессия — это не регрессия, а алгоритм обучения классификации. Название происходит из статистики и связано с тем, что математическая формулировка логистической регрессии аналогична линейной регрессии.

---

[1] В полиномиальной регрессии степени n, $f(x)$ имеет дополнительный параметр для аргумента x, возведенного в каждую из степеней от 1 до 10, то есть $f(x) =\text{def } w_1x^1 + w_2x^2 + \ldots + w_nx^n + b''$.

[2] Чтобы найти минимум или максимум функции, мы устанавливаем градиент в ноль, потому в точках экстремума функции значение градиента всегда равно нулю. В двумерном случае градиент экстремума — это горизонтальная линия.

Я объясняю суть логистической регрессии на примере бинарной классификации. Однако его можно естественным образом распространить на многоклассовую классификацию.

3.2.1. Задача

Цель логистической регрессии все та же: смоделировать y_i как линейную функцию от x_i, однако для бинарных значений y_i это не так просто. Линейная комбинация признаков, такая как $\mathbf{w}\mathbf{x}_i + b$, — это функция, простирающаяся от минус бесконечности до плюс бесконечности, тогда как y_i имеет только два возможных значения.

Во времена, когда не было компьютеров и все вычисления приходилось выполнять вручную, ученые отдавали предпочтение моделям линейной классификации. В ту пору они подметили, что если определить отрицательную метку как 0, а положительную метку как 1, достаточно найти простую непрерывную функцию с областью значений (0, 1). В этом случае, если значение, возвращаемое моделью для образца **x**, ближе к 0, ему присваивается отрицательная метка; иначе образец маркируется как положительный. Одной из функций, обладающих таким свойством, является **стандартная логистическая функция** (также известная как **логистический сигмоид**):

$$f(x) = \frac{1}{1 + e^{-x}},$$

где e — основание натурального логарифма (также называется *числом Эйлера*; значение e^x в языках программирования также известно как функция $exp(x)$). Ее график изображен на рис. 3.3.

Вот как выглядит модель логистической регрессии:

$$f_{\mathbf{w},b}(\mathbf{x}) \overset{\text{def}}{=} \frac{1}{1 + e^{-(\mathbf{w}\mathbf{x}+b)}}. \tag{3.3}$$

Как видите, здесь присутствует уже знакомый нам член $\mathbf{wx} + b$ из линейной регрессии.

Взглянув на график стандартной логистической функции, можно заметить, насколько хорошо она соответствует нашей цели классификации: если соответствующим образом оптимизировать значения $\mathbf{w}$ и b, результат $f(\mathbf{x})$ можно интерпретировать как вероятность, что y_i будет иметь положительное значение. Например, если она выше или равна пороговому значению 0.5, мы бы сказали, что класс **x** положителен; иначе — отрицателен. На практике могут выбираться другие пороговые значения, в зависимости от решаемой задачи. Мы вернемся к этой теме в главе 5, когда будем говорить об оценке эффективности модели.

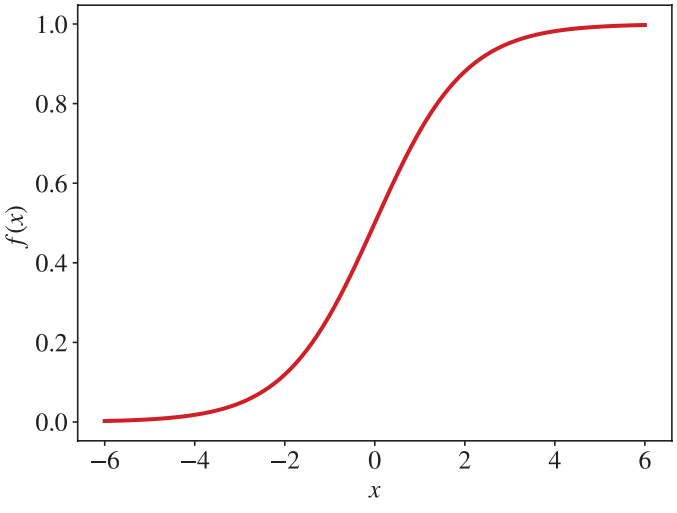


Рис. 3.3. Стандартная логистическая функция

Как теперь найти оптимальные оценки $\mathbf{w}^*$ и b^*? В линейной регрессии мы минимизировали эмпирический риск, который определялся как среднее квадратичной функции потерь, также известное как **среднеквадратичная ошибка** (mean squared error, MSE).

3.2.2. Решение

В логистической регрессии, в отличие от линейной регрессии, максимизируется **правдоподобие** (вероятность) обучающего набора в соответствии с моделью. В статистике функция правдоподобия определяет, насколько правдоподобным выглядит наблюдение (образец) в соответствии с нашей моделью.

Например, допустим, что в нашем обучающем наборе имеется размеченный образец ($\mathbf{x}_i$, y_i). Предположим также, что мы нашли (выбрали) некоторые конкретные значения $\hat{\mathbf{w}}$ и $\hat{b}$ для наших параметров. Если теперь применить модель $f_{\hat{\mathbf{w}},\hat{b}}$ к x_i, используя уравнение 3.3, мы получим некоторое значение $0 < p < 1$. Если y_i является положительным классом, вероятность, что y_i является положительным классом, согласно нашей модели, определяется как p. Аналогично, если y_i является отрицательным классом, вероятность, что он является отрицательным классом, определяется как $1 - p$.

Критерий оптимизации в логистической регрессии называется **максимальным правдоподобием** (maximum likelihood). Вместо того чтобы минимизировать среднюю потерю, как в линейной регрессии, мы теперь максимизируем правдоподобие обучающих данных в соответствии с моделью:

$$L_{\mathbf{w},b} \overset{\text{def}}{=} \prod_{i=1\ldots N} f_{\mathbf{w},b}(\mathbf{x}_i)^{y_i}\left(1-f_{\mathbf{w},b}(\mathbf{x}_i)\right)^{(1-y_i)}. \quad (3.4)$$

Выражение $f_{\mathbf{w},b}(\mathbf{x})^{y_i}\left(1-f_{\mathbf{w},b}(\mathbf{x})\right)^{(1-y_i)}$ может показаться пугающим, но это всего лишь причудливый математический способ сказать: «$f_{\mathbf{w},b}(\mathbf{x})$, когда $y_i = 1$, и $(1 - f_{\mathbf{w},b}(\mathbf{x}))$ иначе». Действительно, если $y_i = 1$, тогда $\left(1-f_{\mathbf{w},b}(\mathbf{x})\right)^{(1-y_i)}$ равно 1, потому что $(1 - y_i) = 0$, а как мы знаем, любое число в степени 0 равно 1. С другой стороны, если $y_i = 0$, тогда $f_{\mathbf{w},b}(\mathbf{x})^{y_i}$ равно 1 по той же причине.

Возможно, вы обратили внимание, что в целевой функции мы использовали оператор произведения Π вместо оператора суммы Σ, который применялся в линейной регрессии. Это связано с тем, что вероятность наблюдения N меток в N образцах является произведением вероятностей каждого наблюдения (при условии, что все наблюдения независимы друг от друга, что в нашем случае действительно так). Можно провести параллель с умножением вероятностей исходов в серии независимых экспериментов в теории вероятностей.

Поскольку в модели используется функция *exp*, на практике удобнее максимизировать **логарифм правдоподобия**, а не правдоподобие. Логарифм правдоподобия определяется следующим образом:

$$Log\,L_{\mathbf{w},b} \overset{\text{def}}{=} \ln\left(L_{\mathbf{w},b}(\mathbf{x})\right) = \sum_{i=1}^{N}\left[y_i \ln f_{\mathbf{w},b}(\mathbf{x}) + (1-y_i)\ln\left(1-f_{\mathbf{w},b}(\mathbf{x})\right)\right].$$

Поскольку ln — **строго возрастающая функция**, ее максимизация равносильна максимизации ее аргумента, а решение этой новой задачи оптимизации равносильно решению исходной задачи.

В отличие от линейной регрессии, задача оптимизации выше не имеет аналитического решения. Поэтому в таких случаях обычно используется процедура численной оптимизации — **градиентный спуск**. Но об этом мы поговорим в следующей главе.

3.3. Обучение дерева решений

Дерево решений — это ациклический **граф**, который можно использовать для принятия решений. В каждом ветвящемся узле графа исследуется *j*-й признак из вектора признаков. Если значение признака ниже определенного порога, выбирается левая ветвь; иначе — правая. По достижении листового узла принимается решение о классе, к которому относится образец.

Как следует из заголовка раздела, дерево решений можно получить из данных.

3.3.1. Задача

Как и прежде, дано: коллекция размеченных данных; метки принадлежат множеству {0, 1}. Нужно построить дерево решений, которое позволило бы предсказать класс по заданному вектору признаков.

3.3.2. Решение

Существуют различные формулировки алгоритма обучения дерева решений. В этой книге мы рассмотрим только один, который называется **ID3**.

Критерием оптимизации в данном случае является среднее логарифмическое правдоподобие:

$$\frac{1}{N}\sum_{i=1}^{N}\Big[y_i \ln f_{ID3}(\mathbf{x}_i)+(1-y_i)\ln\big(1-f_{ID3}(\mathbf{x}_i)\big)\Big] \quad , \tag{3.5}$$

где f_{ID3} — дерево решений.

Пока все выглядит похожим на логистическую регрессию. Однако, в отличие от алгоритма обучения логистической регрессии, который строит **параметрическую модель** $f_{\mathbf{w}^*,b^*}$ путем поиска оптимального решения критерия оптимизации, алгоритм ID3 оптимизирует его *приблизительно*, конструируя **непараметрическую модель** $f_{ID3}(\mathbf{x}) \overset{\text{def}}{=} \Pr(y=1|\mathbf{x})$.

Алгоритм обучения ID3 работает следующим образом. Пусть $\mathcal{S}$ обозначает множество размеченных данных. В начале дерево решений имеет только начальный узел, который содержит все данные: $\mathcal{S} \overset{\text{def}}{=} \{(\mathbf{x}_i,\ y_i)\}_{i=1}^{N}$. Начнем с постоянной модели f_{ID3}^{S}, которая определяется как

$$f_{ID3}^{S} \overset{\text{def}}{=} \frac{1}{|\mathcal{S}|}\sum_{(\mathbf{x},y)\in S} y. \tag{3.6}$$

Вышеупомянутая модель $f_{ID3}^{S}(\mathbf{x})$ будет возвращать один и тот же прогноз для любого входа $\mathbf{x}$. Соответствующее дерево решений, построенное на основе нашего фиктивного набора данных из 12 размеченных данных, показано на рис. 3.4а.

Затем мы ищем все признаки $j = 1, ..., D$ и все пороги t и разбиваем множество $\mathcal{S}$ на два подмножества: $\mathcal{S}_{-} \overset{\text{def}}{=} \big\{(\mathbf{x}, y)\big|(\mathbf{x}, y)\in S, x^{(j)} < t\big\}$ и $\mathcal{S}_{+} \overset{\text{def}}{=} \big\{(\mathbf{x},\ y)\big|(\mathbf{x},\ y)\in S, x^{(j)} \geq t\big\}$. Два новых подмножества образуют два новых листовых узла, и мы оцениваем для

всех возможных пар (j, t), насколько хорошим получилось расщепление на части $\mathcal{S}_-$ и $\mathcal{S}_+$. Наконец, выбираем наилучшие значения (j, t), разбиваем $\mathcal{S}$ на $\mathcal{S}_-$ и $\mathcal{S}_+$, формируем два новых листовых узла и продолжаем рекурсивно разбивать на $\mathcal{S}_-$ и $\mathcal{S}_+$ (или завершаем работу, если ни одно разбиение не дает модели, которая позволяет получить лучший прогноз, чем текущая). Дерево решений после одного расщепления показано на рис. 3.4б.

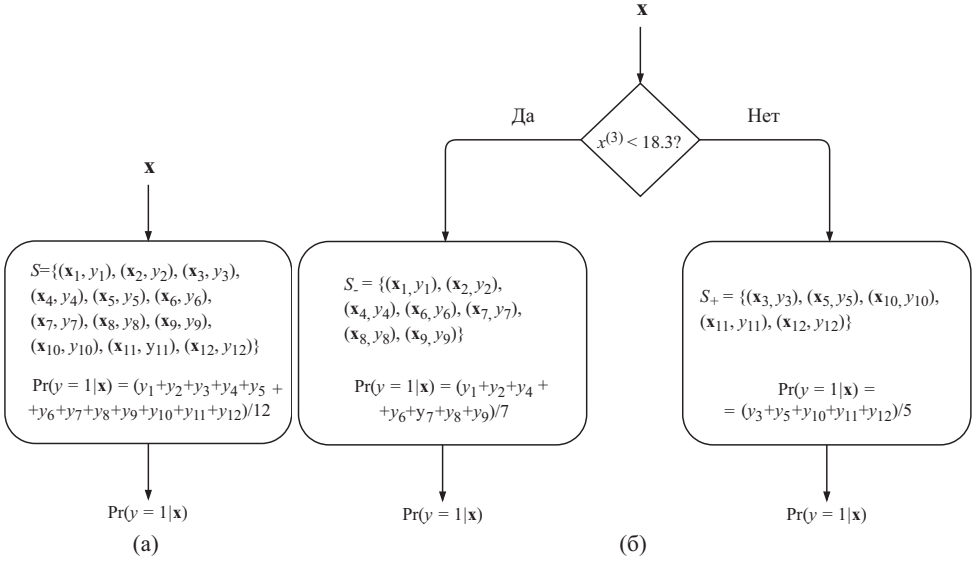


Рис. 3.4. Иллюстрация алгоритма построения дерева решений. Множество S содержит 12 размеченных данных. (а) Вначале дерево решений содержит только начальный узел; он дает один и тот же прогноз для любого входа. (б) Дерево решений после первого расщепления; оно проверяет, является ли признак 3 меньше 18.3, и, в зависимости от результата, прогноз выполняется одним из двух листовых узлов

Теперь у вас наверняка возник вопрос, что означают слова «оценить, насколько хорошим получилось расщепление». В ID3 качество расщепления оценивается с использованием критерия, называемого **энтропией**. Энтропия — это мера неопределенности случайной величины. Она достигает своего максимума, когда все значения случайной величины равновероятны. Энтропия достигает своего минимума, когда случайная величина может иметь только одно значение. Энтропия множества данных $\mathcal{S}$ определяется как

$$H\left(\mathcal{S}\right) \stackrel{\text{def}}{=} -f_{ID3}^{S} \ln f_{ID3}^{S} - \left(1 - f_{ID3}^{S}\right) \ln\left(1 - f_{ID3}^{S}\right).$$

Когда мы разбиваем множество данных по некоторому признаку j и порогу t, энтропия разбиения $H(\mathcal{S}_-, \mathcal{S}_+)$ определяется как простая взвешенная сумма двух энтропий:

$$H(\mathcal{S}_-, \mathcal{S}_+) \stackrel{\text{def}}{=} \frac{|\mathcal{S}_-|}{|\mathcal{S}|} H(\mathcal{S}_-) + \frac{|\mathcal{S}_+|}{|\mathcal{S}|} H(\mathcal{S}_+). \tag{3.7}$$

Итак, в ID3, на каждом шаге, в каждом листовом узле мы находим расщепление, минимизирующее энтропию, заданную уравнением 3.7, или останавливаемся на этом листовом узле.

Алгоритм останавливается на листовом узле в любой из следующих ситуаций:

- Все примеры в листовом узле правильно классифицируются моделью (уравнение 3.6).
- Невозможно найти атрибут для расщепления.
- Расщепление уменьшает энтропию ниже некоторого значения ε (которое нужно определить экспериментально[1]).
- Дерево достигает некоторой максимальной глубины d (также должна определяться экспериментально).

Поскольку в ID3 решение о расщеплении набора данных в каждой итерации является локальным (не зависит от будущих расщеплений), алгоритм не гарантирует оптимального решения. Модель можно улучшить, использовав в процессе поиска оптимального дерева решений такие методы, как *возврат* (backtracking), хотя и за счет увеличения времени построения модели.

Наиболее широко используемая версия алгоритма обучения дерева решений называется **C4.5**. Она имеет несколько дополнительных особенностей по сравнению с ID3:

- принимает непрерывные и дискретные признаки;
- поддерживает возможность обработки неполных данных;
- решает проблему переобучения с использованием восходящего метода, известного как «подрезка» (отсечение ветвей).

Подрезка заключается в том, чтобы выполнить обратный обход только что созданного дерева и удалить ветви, которые не вносят существенного вклада в уменьшение ошибки, заменив их листовыми узлами.

---

[1] Я покажу, как это сделать, в главе 5, в разделе, посвященном настройке гиперпараметров.

Смысл критерия расщепления на основе энтропии очевиден: энтропия достигает минимальной величины 0, когда все данные в S имеют одинаковую метку; с другой стороны, энтропия достигает максимальной величины 1, когда ровно половина примеров в S имеет метку 1, что делает такой лист бесполезным для классификации. Единственный оставшийся вопрос — как этот алгоритм приблизительно максимизирует средний логарифм правдоподобия. Я предлагаю читателям выяснить это самостоятельно.

3.4. Метод опорных векторов

Я уже представлял метод опорных векторов (SVM) во введении, поэтому здесь восполню лишь несколько пробелов. Вот два важных вопроса, которые мы должны задать:

1. Как быть, если исходные данные настолько искажены помехами, что невозможно найти гиперплоскость, четко разделяющую положительные и отрицательные данные?
2. Что если данные нельзя разделить с помощью плоскости, но можно с помощью полинома более высокого порядка?

Обе эти ситуации изображены на рис. 3.5. Слева показан случай, когда данные нельзя разделить прямой линией из-за шума (аномальных выбросов или неправильно размеченных данных). Справа границей решения является окружность, а не прямая линия.

Напомню, что, используя метод опорных векторов, мы должны удовлетворить следующие ограничения:

$$\begin{aligned} \mathbf{w}\mathbf{x}_i - b \geq +1, \text{ если } y_i = +1, \\ \mathbf{w}\mathbf{x}_i - b \leq -1, \text{ если } y_i = -1. \end{aligned} \tag{3.8}$$

Также мы должны минимизировать $\|\mathbf{w}\|$, чтобы гиперплоскость была одинаково удалена от ближайших данных каждого класса. Минимизация $\|\mathbf{w}\|$ эквивалентна минимизации $\frac{1}{2}\|\mathbf{w}\|^2$, и использование этого члена позволит нам в дальнейшем выполнять оптимизацию квадратичного программирования. Соответственно, задача оптимизации в SVM выглядит следующим образом:

$$\min \frac{1}{2}\|\mathbf{w}\|^2, \text{ такое, что } y_i(\mathbf{x}_i\mathbf{w} - b) - 1 \geq 0, \; i = 1, \ldots, N. \tag{3.9}$$

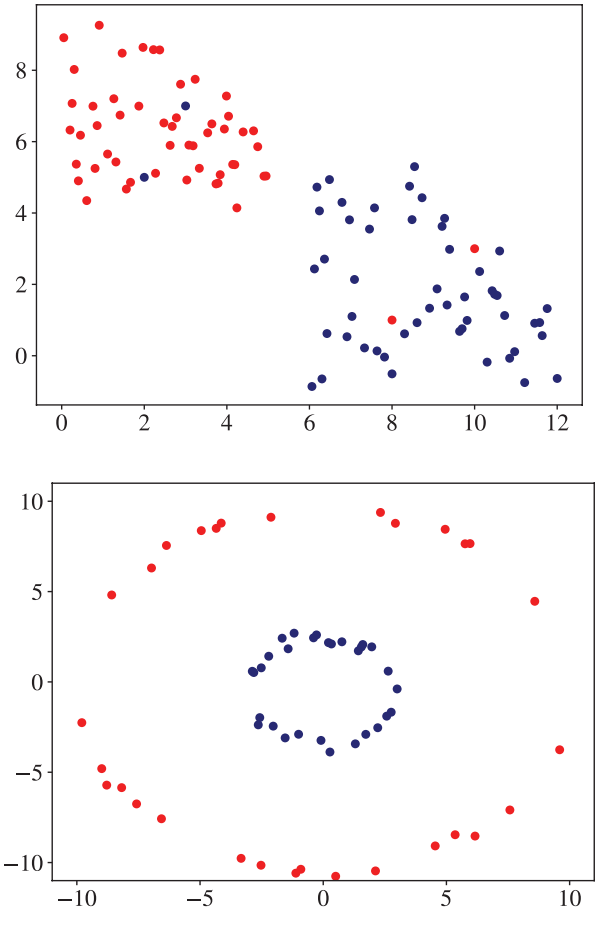


Рис. 3.5. Случаи, неразделимые линейно. Сверху: присутствует шум. Снизу: естественная нелинейность

3.4.1. Работа с шумом

Чтобы распространить SVM на случаи, когда данные невозможно разделить линейно, введем **кусочно-линейную функцию потерь** (hinge loss function): $\max(0,1 - y_i(\mathbf{w}\mathbf{x}_i - b))$.

Кусочно-линейная функция потерь равна нулю, если выполнены условия 3.8, то есть если $\mathbf{w}\mathbf{x}_i$ лежит с правильной стороны от границы решения. Для данных, лежащих с неправильной стороны, значение функции пропорционально расстоянию от границы решения.

Затем минимизируем следующую функцию стоимости:

$$C\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N}\max\left(0, 1 - y_i\left(\mathbf{w}\mathbf{x}_i - b\right)\right),$$

где гиперпараметр C определяет компромисс между увеличением размера границы решения и гарантией местонахождения каждого $\mathbf{x}_i$ с правильной стороны от границы решения. Значение C обычно выбирается экспериментально, как и гиперпараметры ϵ и d в ID3.

Алгоритм SVM, оптимизирующий кусочно-линейную функцию потерь, называют алгоритмом **SVM с мягким зазором** (soft-margin SVM), тогда как алгоритм в оригинальной формулировке называют алгоритмом **SVM с жестким зазором** (hard-margin SVM).

Как нетрудно заметить, при достаточно высоких значениях C второй член в функции стоимости становится пренебрежимо малым, поэтому алгоритм SVM будет пытаться найти наибольший зазор, полностью игнорируя ошибочную классификацию. По мере уменьшения значения C ошибки классификации становятся более дорогостоящими, поэтому алгоритм SVM будет пытаться делать меньше ошибок, жертвуя размером зазора. Как уже говорилось, больший зазор дает лучшее обобщение. Следовательно, C регулирует компромисс между хорошей классификацией обучающих данных (минимальный эмпирический риск) и хорошей классификацией данных в будущем (обобщение).

3.4.2. Работа с естественной нелинейностью

SVM можно адаптировать для работы с наборами данных, которые нельзя разделить гиперплоскостью в исходном пространстве. Действительно, если удастся преобразовать исходное пространство в пространство более высокой размерности, можно надеяться, что данные станут линейно разделимыми в этом преобразованном пространстве. Использование функции для *неявного* преобразования исходного пространства в пространство более высокой размерности в ходе оптимизации функции стоимости в SVM называется **ядерным трюком** (kernel trick).

На рис. 3.6 показан эффект применения ядерного трюка. Как видите, двумерные данные, не разделимые линейно, можно преобразовать в линейно разделимые трехмерные данные, используя специальное отображение $\phi: \mathbf{x} \mapsto \phi(\mathbf{x})$, где $\phi(\mathbf{x})$ — вектор с более высокой размерностью, чем $\mathbf{x}$. Например, к двумерным данным, изображенным на рис. 3.5 (справа), можно применить отображение, проецирующее двумерные данные $\mathbf{x} = [q, p]$ в трехмерное пространство (рис. 3.6), которое выглядит так: $\phi([q, p]) \overset{\text{def}}{=} (q^2, \sqrt{2}qp, p^2)$, где $\cdot^2$ — это возведение в ква-

драт. После преобразования данные становятся линейно разделимыми в новом пространстве.

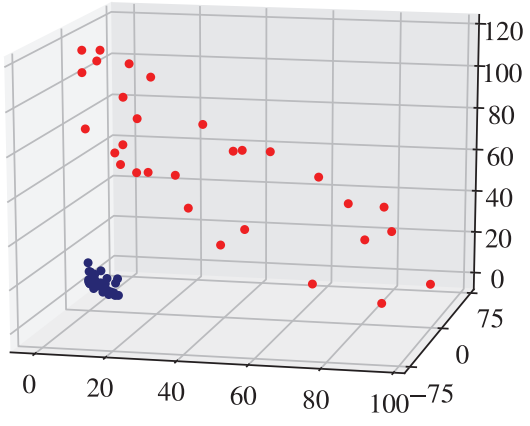


Рис. 3.6. Данные, представленные на рис. 3.5 (снизу), становятся линейно разделимыми после преобразования в трехмерное пространство

Однако заранее неизвестно, какое отображение подойдет для наших данных. Если преобразовывать все входные данные в векторы с более высокой размерностью и применять к ним SVM, опробуя все возможные функции отображения, вычисления могут стать очень неэффективными и мы никогда не решим задачу классификации.

К счастью, ученые выяснили, как использовать **функции ядра́** (или просто **я́дра**) для эффективной работы в многомерных пространствах *без явного преобразования*. Чтобы понять, как работают ядра, прежде нужно посмотреть, как алгоритм оптимизации для SVM находит оптимальные значения для **w** и b.

Для решения задачи оптимизации в уравнении 3.9 традиционно используется **метод множителей Лагранжа**. Вместо оригинальной задачи из уравнения 3.9 проще решить эквивалентную задачу, сформулированную так:

$$\max_{\alpha_1 \ldots \alpha_N} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} y_i \alpha_i \left(\mathbf{x}_i \mathbf{x}_k \right) y_k \alpha_k \text{ при условии, что } \sum_{i=1}^{N} \alpha_i y_i = 0 \text{ и } \alpha_i \geq 0, i = 1, \ldots, N,$$

где α_i называются множителями Лагранжа. В такой формулировке задача оптимизации превращается в выпуклую задачу квадратичной оптимизации, которая эффективно решается применением алгоритмов квадратичного программирования.

Обратите внимание, что в формулировке выше присутствует член $\mathbf{x}_i\mathbf{x}_k$, это единственное место, где используются векторы признаков. Чтобы преобразовать исходное векторное пространство в пространство с большим числом измерений, нужно преобразовать $\mathbf{x}_i$ в $\phi(\mathbf{x}_i)$ и $\mathbf{x}_k$ в $\phi(\mathbf{x}_k)$, а затем перемножить $\phi(\mathbf{x}_i)$ и $\phi(\mathbf{x}_k)$. Эти вычисления могут оказаться очень дорогостоящими.

С другой стороны, нас интересует только результат скалярного произведения $\mathbf{x}_i\mathbf{x}_k$, который, как мы знаем, является действительным числом. Нам все равно, как будет получено это число, лишь бы оно было верным. Используя функцию ядра, можно избавиться от дорогостоящего преобразования исходных векторов признаков в векторы с более высокой размерностью и избежать необходимости вычислять их скалярное произведение. Мы заменим эти вычисления простой операцией с исходными векторами признаков, которая даст тот же результат. Например, вместо преобразования (q_1, p_1) в $(q_1^2,\ \sqrt{2}q_1p_1,\ p_1^2)$ и (q_2, p_2) в $(q_2^2,\ \sqrt{2}q_2p_2,\ p_2^2)$ и последующего вычисления скалярного произведения $(q_1^2,\ \sqrt{2}q_1p_1,\ p_1^2)$ и $(q_2^2,\ \sqrt{2}q_2p_2,\ p_2^2)$, чтобы получить $(q_1^2q_2^2 + 2q_1q_2p_1p_2 + p_1^2p_2^2)$, можно найти скалярное произведение (q_1, p_1) и (q_2, p_2), чтобы получить $(q_1q_2 + p_1p_2)$, а затем возвести в квадрат, чтобы получить тот же результат $(q_1^2q_2^2 + 2q_1q_2p_1p_2 + p_1^2p_2^2)$.

Это был пример функции ядра, и мы использовали квадратичное ядро $k(\mathbf{x}_i,\ \mathbf{x}_k) \overset{\text{def}}{=} (\mathbf{x}_i\mathbf{x}_k)^2$. Существует несколько функций ядра, из которых наиболее широко используется **ядро RBF**:

$$k(\mathbf{x},\ \mathbf{x}') = \exp\left(-\frac{\left\|\mathbf{x}-\mathbf{x}'\right\|^2}{2\sigma^2}\right),$$

где $\left\|\mathbf{x}-\mathbf{x}'\right\|^2$ — квадрат **евклидова расстояния** между двумя векторами признаков. Евклидово расстояние определяется следующим уравнением:

$$d\left(\mathbf{x}_i,\mathbf{x}_k\right) \overset{\text{def}}{=} \sqrt{\left(x_i^{(1)}-x_k^{(1)}\right)^2+\left(x_i^{(2)}+x_k^{(2)}\right)^2+\cdots+\left(x_i^{(N)}-x_k^{(N)}\right)^2} = \sqrt{\sum_{j=1}^{D}\left(x_i^{(j)}-x_k^{(j)}\right)^2}.$$

Можно показать, что пространство признаков ядра RBF (Radial Basis Function — радиальная базисная функция) имеет бесконечное число измерений. Изменяя гиперпараметр σ, аналитик может выбирать между гладкой или изогнутой границей решения в исходном пространстве.

3.5. Метод k ближайших соседей

Метод k ближайших соседей (k-Nearest Neighbors, kNN) — это непараметрический алгоритм обучения. В отличие от других алгоритмов обучения, позволяющих от-

брасывать обучающие данные после построения модели, метод kNN сохраняет все обучающие примеры в памяти. Когда появляется новый, ранее не встречавшийся образец **x**, алгоритм kNN находит k обучающих данных, наиболее близких к **x**, и возвращает наиболее часто встречающуюся метку в случае классификации или среднее значение метки в случае регрессии.

Близость двух данных определяется функцией расстояния. Например, на практике часто используется евклидово расстояние, показанное выше. Также нередко используется еще одна функция расстояния — отрицательное **косинусное сходство**. Косинусное сходство, которое определяется как

$$s\left(\mathbf{x}_i, \mathbf{x}_k\right) \overset{\text{def}}{=} \cos\left(\angle\left(\mathbf{x}_i, \mathbf{x}_k\right)\right) = \frac{\sum_{j=1}^{D} x_i^{(j)} x_k^{(j)}}{\sqrt{\sum_{j=1}^{D}\left(x_i^{(j)}\right)^2}\sqrt{\sum_{j=1}^{D}\left(x_k^{(j)}\right)^2}},$$

является мерой сходства направлений двух векторов. Если угол между двумя векторами равен 0 градусов, значит, они указывают в одном направлении и косинусное сходство равно 1. Если векторы ортогональны, косинусное сходство равно 0. Для векторов, указывающих в противоположных направлениях, косинусное сходство равно –1. Чтобы использовать косинусное сходство в качестве меры расстояния, его значение нужно умножить на –1. В числе других популярных мер расстояния можно назвать расстояние Чебышева, расстояние Махаланобиса и расстояние Хемминга. Выбор меры, а также значения для k должен сделать аналитик перед запуском алгоритма. То есть все это — гиперпараметры. Меру расстояния также можно вывести из данных. Мы поговорим об этом в главе 10.

4 Анатомия алгоритмов обучения

4.1. Строительные блоки алгоритмов обучения

Читая предыдущую главу, вы могли заметить, что все рассмотренные там методы состоят из трех частей:

1. Функция потерь.
2. Критерий оптимизации, основанный на функции потерь (например, функция стоимости).
3. Процедура оптимизации, использующая обучающие данные для поиска решения критерия оптимизации.

Эти строительные блоки присутствуют во всех методах обучения. В предыдущей главе вы видели, что одни методы предполагают явную оптимизацию определенного критерия (линейная и логистическая регрессия, SVM). Другие, включая обучение дерева решений и kNN, оптимизируют критерий неявно. Обучение дерева решений и kNN являются одними из самых старых методов машинного обучения и были изобретены экспериментально на основе интуитивных представлений, без наличия конкретных глобальных критериев оптимизации, которые (как это часто случалось в истории науки) были разработаны позже, чтобы объяснить, как работают эти методы.

В современной литературе по машинному обучению часто можно встретить понятия **градиентный спуск** или **стохастический градиентный спуск**. Это два наиболее часто используемых метода оптимизации, которые применяются в случаях использования дифференцируемого критерия оптимизации.

Градиентный спуск — это метод итеративной оптимизации для поиска минимума функции. Чтобы найти *локальный* минимум функции с использованием градиентного спуска, нужно выбрать некоторую случайную точку и шагать в направлении отрицательных значений градиента (или аппроксимации градиента) функции в текущей точке.

Градиентный спуск можно использовать для поиска оптимальных параметров линейной и логистической регрессии, SVM, а также нейронных сетей, которые мы рассмотрим позже. Для многих моделей, таких как логистическая регрессия и SVM, критерий оптимизации является *выпуклым*. Выпуклые функции имеют только один минимум, который является глобальным. Критерии оптимизации для нейронных сетей не являются выпуклыми, но на практике часто достаточно найти хотя бы локальный минимум.

Давайте посмотрим, как работает градиентный спуск.

4.2. Градиентный спуск

В этом разделе я покажу, как градиентный спуск находит решение задачи линейной регрессии[1]. Свое описание я проиллюстрирую с помощью кода на Python, а также графиков, показывающих, как решение улучшается после нескольких итераций градиентного спуска. В примере я использую набор данных с единственным признаком. Однако критерий оптимизации будет иметь два параметра: w и b. Распространение примера на многомерные обучающие данные делается просто: нужно использовать переменные $w^{(1)}$, $w^{(2)}$ и b в случае с двумерными данными, $w^{(1)}$, $w^{(2)}$, $w^{(3)}$ и b в случае с трехмерными данными и т. д.

Чтобы придать примеру практический характер, я использую реальный набор данных (его можно найти в вики для книги) со следующими столбцами: ежегодные расходы различных компаний на рекламу по радио и их ежегодные объемы продаж в пересчете на проданные штуки. Нам нужно построить регрессионную модель, которую можно использовать для прогнозирования продаж в зависимости от расходов на рекламу. Каждая строка в наборе данных представляет одну конкретную компанию.

В наборе содержатся данные по 200 компаниям, соответственно, у нас имеется 200 обучающих данных в форме $(x_i, y_i) = (Затраты_i, Продажи_i)$. На рис. 4.1 показана двумерная диаграмма со всеми образцами.

---

[1] Как известно, линейная регрессия имеет аналитическое решение. То есть для решения задач этого конкретного типа градиентный спуск не требуется. Однако линейная регрессия является идеальным примером для объяснения градиентного спуска.

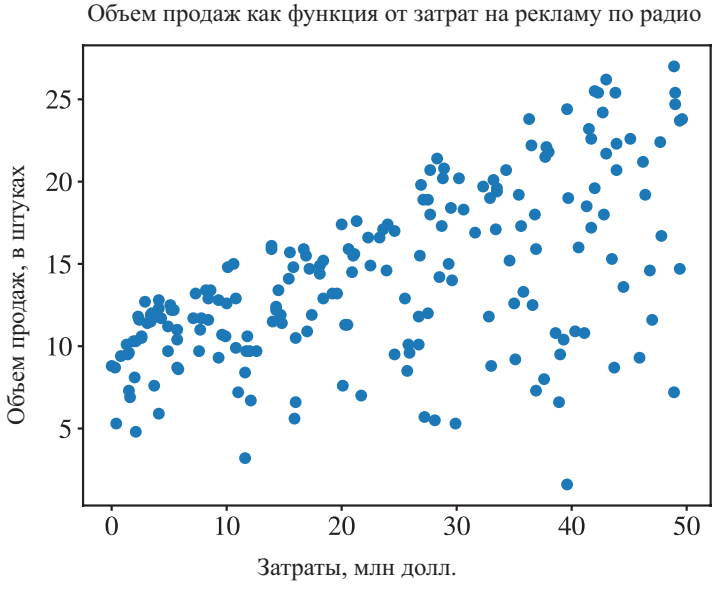


Рис. 4.1. Исходные данные. Ось Y соответствует объему продаж в штуках (который мы должны предсказать), ось X соответствует признаку: затратам на рекламу по радио в млн долл.

Компания	Затраты, M$	Продажи, в штуках
1	37.8	22.1
2	39.3	10.4
3	45.9	9.3
4	41.3	18.5
...	...	...

Напомню, что модель линейной регрессии выглядит следующим образом: $f(x) = wx + b$. Мы не знаем оптимальных значений w и b и должны определить их из данных. Для этого мы найдем такие значения w и b, которые минимизируют среднеквадратичную ошибку:

$$l \overset{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \left(wx_i + b\right)\right)^2.$$

Градиентный спуск начинается с вычисления частной производной для каждого параметра:

$$\frac{\partial l}{\partial w} = \frac{1}{N}\sum_{i=1}^{N} -2x_i\left(y_i - \left(wx_i + b\right)\right);$$
$$\frac{\partial l}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} -2\left(y_i - \left(wx_i + b\right)\right). \tag{4.1}$$

Чтобы найти частную производную члена $(y_i - (wx + b))^2$ относительно w, я применил правило дифференцирования сложной функции. Здесь мы имеем составную функцию $f = f_2(f_1)$, где $f_1 = y_i - (wx + b)$ и $f_2 = f_1^2$. Чтобы найти частную производную f относительно w, нужно сначала найти частную производную f относительно f_2, которая равна $2(y_i - (wx + b))$ (из теории вычислений мы знаем, что производная $\frac{\partial}{\partial x}x^2 = 2x$), а затем умножить ее на частную производную $y_i - (wx + b)$ относительно w, равную $-x$. То есть в целом получаем $\frac{\partial l}{\partial w} = \frac{1}{N}\sum_{i=1}^{N} -2x_i\left(y_i - \left(wx_i + b\right)\right)$. Аналогично вычисляется частная производная l относительно b, $\frac{\partial l}{\partial b}$.

Градиентный спуск выполняется этапами, или **эпохами**. В каждой эпохе производится уточнение каждого параметра с использованием всего обучающего набора.

В начале, в первую эпоху, мы инициализируем[1] $w \leftarrow 0$ и $b \leftarrow 0$. Частные производные, $\frac{\partial l}{\partial w}$ и $\frac{\partial l}{\partial b}$, заданные уравнениями 4.1, равны соответственно $\frac{-2}{N}\sum_{i=1}^{N} x_i y_i$ и $\frac{-2}{N}\sum_{i=1}^{N} y_i$. В каждую эпоху с использованием частных производных производится уточнение w и b. Скорость обучения α контролирует размер уточнения:

$$w \leftarrow w - \alpha\frac{\partial l}{\partial w};$$
$$b \leftarrow b - \alpha\frac{\partial l}{\partial b}. \tag{4.2}$$

Мы вычитаем частные производные из значений параметров (а не прибавляем), потому что производные являются индикаторами роста функции. Если производная положительна в некоторой точке[2], значит, функция возрастает в этой точке.

---

[1] В сложных моделях, таких как нейронные сети, имеющих тысячи параметров, инициализация параметров может существенно повлиять на решение, найденное градиентным спуском. Существуют разные методы инициализации (случайными значениями, нулями, небольшими значениями, близкими к нулю, и т. д.), и этот важный выбор должен сделать аналитик.

[2] Точка задается текущими значениями параметров.

Поскольку мы минимизируем целевую функцию, для положительного значения производной мы должны скорректировать параметр в противоположном направлении (сместить влево на оси координат). Для отрицательного значения производной (функция убывает) мы должны сместить параметр вправо, чтобы еще больше уменьшить значение функции. Вычитание отрицательного значения из параметра смещает его вправо.

В следующую эпоху мы повторно вычисляем производные, используя уравнение 4.1 с измененными значениями w и b; процесс продолжается до схождения. Обычно требуется пройти много эпох, пока не обнаружится, что значения w и b мало меняются после каждой эпохи; после этого процесс останавливается.

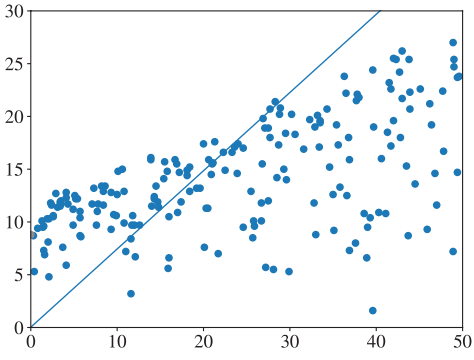
Трудно представить инженера по машинному обучению, который не использует язык программирования Python. Поэтому, если вы ждали удобного момента для изучения Python, он настал. Ниже я покажу, как запрограммировать градиентный спуск на Python.

Ниже показана функция, корректирующая параметры w и b в ходе одной эпохи:

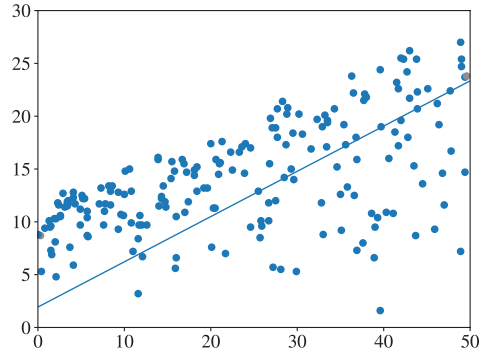
```
def update_w_and_b(spendings, sales, w, b, alpha):
    dl_dw = 0.0
    dl_db = 0.0
    N = len(spendings)

    for i in range(N):
        dl_dw += -2*spendings[i]*(sales[i] - (w*spendings[i] + b))
        dl_db += -2*(sales[i] - (w*spendings[i] + b))

    # скорректировать w и b
    w = w - (1/float(N))*dl_dw*alpha
    b = b - (1/float(N))*dl_db*alpha

    return w, b
```

Далее показана функция, которая в цикле выполняет множество эпох:

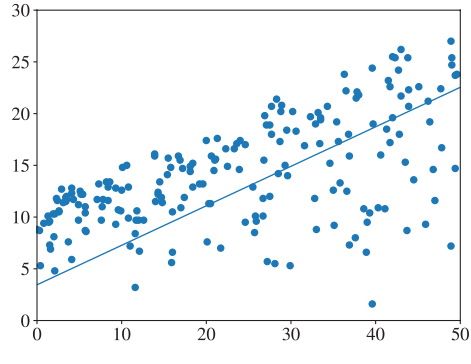
```
def train(spendings, sales, w, b, alpha, epochs):
    for e in range(epochs):
        w, b = update_w_and_b(spendings, sales, w, b, alpha)

        # вывести информацию о продвижении вперед
        if e % 400 == 0:
            print("epoch:", e, "loss: ", avg_loss(spendings, sales, w, b))

    return w, b
```



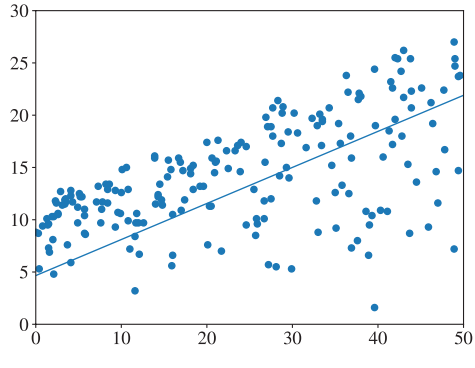
Эпоха 0



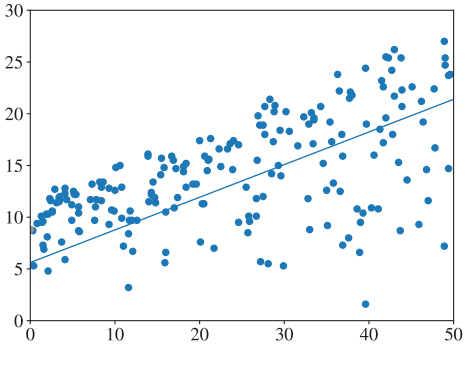
Эпоха 400



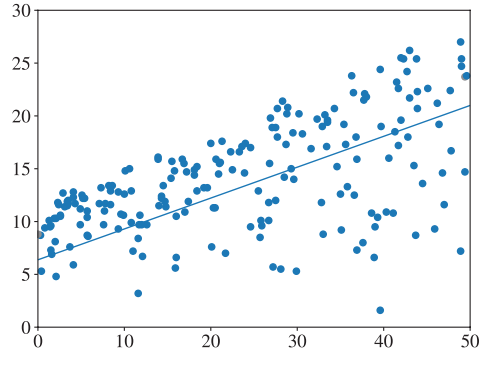
Эпоха 800



Эпоха 1200



Эпоха 1600



Эпоха 3000

Рис. 4.2. Эволюция линии регрессии с течением эпох градиентного спуска

Функция `avg_loss`, используемая в листинге выше, вычисляет среднеквадратичную ошибку. Вот как она определяется:

```
25 def avg_loss(spendings, sales, w, b):
26     N = len(spendings)
27     total_error = 0.0
28     for i in range(N):
29         total_error += (sales[i] - (w*spendings[i] + b))**2
30     return total_error / float(N)
```

Если вызвать функцию `train` с параметрами `alpha` = 0.001, `w` = 0.0, `b` = 0.0 и `epochs` = 15000, мы увидим следующие результаты (показана лишь часть из них):

```
epoch: 0 loss: 92.32078294903626
epoch: 400 loss: 33.79131790081576
epoch: 800 loss: 27.9918542960729
epoch: 1200 loss: 24.33481690722147
epoch: 1600 loss: 22.028754937538633
...
epoch: 2800 loss: 19.07940244306619
```

Как видите, средняя потеря уменьшается по мере прохождения функции `train` через эпохи. На рис. 4.2 показано, как эволюционировала линия регрессии с течением эпох.

Наконец, после получения оптимальных значений параметров w и b нам недостает только функции, выполняющей предсказание:

```
31 def predict(x, w, b):
32     return w*x + b
```

Попробуйте выполнить следующий код:

```
33 w, b = train(x, y, 0.0, 0.0, 0.001, 15000)
34 x_new = 23.0
35 y_new = predict(x_new, w, b)
36 print(y_new)
```

Он должен вывести 13.97.

Градиентный спуск чувствителен к выбору скорости обучения α. Кроме того, он медленно сходится на больших наборах данных. К счастью, было предложено несколько существенных улучшений этого алгоритма.

Мини-пакетный стохастический градиентный спуск (Minibatch Stochastic Gradient Descent, minibatch SGD) — это версия алгоритма, ускоряющая вычисления за счет аппроксимации градиента с использованием небольших пакетов (подмножеств) обучающих данных. Сам алгоритм SGD тоже включает различные «усовершенство-

вания». **Метод адаптивного градиента** (adagrad) — это версия SGD, которая масштабирует α для каждого параметра в соответствии с историей градиентов. В результате α уменьшается при очень больших градиентах и наоборот. **Метод моментов** — это метод, позволяющий ускорить SGD за счет ориентации градиентного спуска в соответствующем направлении и уменьшения колебаний. В обучении нейронных сетей также часто используются такие варианты SGD, как **RMSprop** и **Adam**.

Обратите внимание, что градиентный спуск и его варианты не являются алгоритмами машинного обучения. Они решают задачи минимизации, в которых минимизируемая функция имеет градиент (в большинстве точек в своей области определения).

4.3. Как работают инженеры, занимающиеся машинным обучением

Если вы не научный сотрудник и не работаете в крупной компании с большим бюджетом, выделяемым на научные исследования и разработки, вам едва ли придется самим заниматься реализацией алгоритмов машинного обучения. Вам не придется заниматься реализацией градиентного спуска или любого другого метода оптимизации. Вы, скорее всего, будете использовать библиотеки, большинство из которых распространяется с открытым исходным кодом. Библиотека — это коллекция алгоритмов и вспомогательных инструментов, отличающаяся надежностью и эффективностью. На практике чаще всего используется библиотека машинного обучения с открытым исходным кодом — *scikit-learn*. Она написана на Python и C. Вот как можно реализовать линейную регрессию с использованием scikit-learn:

```
 1 def train(x, y):
 2     from sklearn.linear_model import LinearRegression
 3     model = LinearRegression().fit(x,y)
 4     return model
 5
 6 model = train(x,y)
 7
 8 x_new = 23.0
 9 y_new = model.predict(x_new)
10 print(y_new)
```

В результате вы получите тот же результат 13.97. Легко, да? При желании вы можете заменить `LinearRegression` другим алгоритмом обучения регрессии, ничего не меняя в остальном коде. Все просто. То же можно сказать о классификации. Вы легко сможете заменить алгоритм `LogisticRegression` алгоритмом `SVC` (именно так называется реализация метода опорных векторов в библиотеке scikit-learn),

`DecisionTreeClassifier`, `NearestNeighbors` и многими другими алгоритмами классификации, реализованными в scikit-learn.

4.4. Особенности алгоритмов обучения

Здесь я отмечу некоторые практические особенности, отличающие один алгоритм обучения от другого. Вы уже знаете, что разные алгоритмы обучения могут иметь разные гиперпараметры (C в SVM, ϵ и d в ID3). Алгоритмы оптимизации, такие как градиентный спуск, тоже могут иметь гиперпараметры, например α.

Некоторые алгоритмы, такие как обучение дерева решений, могут принимать качественные признаки. Например, если в наборе данных имеется признак «цвет», принимающий такие значения, как «красный», «желтый» или «зеленый», вы можете оставить его как есть. Но SVM, логистическая и линейная регрессия, а также kNN (с метриками косинусного сходства или евклидова расстояния) ожидают, что все признаки будут иметь числовые значения. Все алгоритмы, реализованные в scikit-learn, работают только с числовыми признаками. В следующей главе я покажу, как преобразовать качественные признаки в числовые.

Некоторые алгоритмы, такие как SVM, позволяют аналитикам добавлять весовые коэффициенты для каждого класса. Эти весовые коэффициенты влияют на определение границы решения. Если какой-либо класс имеет большой вес, алгоритм обучения постарается не допускать ошибок в классификации обучающих данных, принадлежащих этому классу (обычно за счет ошибок в других данных). Это может быть важно, если представители какого-то класса находятся в меньшинстве в обучающих данных и вам нужно избежать, насколько это возможно, неправильной их классификации.

Некоторые модели классификации, такие как SVM и kNN, выводят для заданного вектора признаков только класс. Другие, такие как логистическая регрессия или деревья решений, также могут возвращать оценку от 0 до 1, которую можно интерпретировать как степень уверенности модели в прогнозе или вероятность, что входной образец принадлежит определенному классу[1].

Некоторые алгоритмы классификации (например, обучения дерева решений, логистической регрессии или SVM) строят модель, используя сразу весь набор данных. При появлении дополнительных размеченных данных вам придется пересобрать модель заново. Другие алгоритмы (такие как наивный байесов-

[1] При необходимости оценку для прогнозов SVM и kNN можно сгенерировать искусственно, использовав несложные методы.

ский классификатор, многослойный персептрон, `SGDClassifier`/`SGDRegressor`, `PassiveAggressiveClassifier`/`PassiveAggressiveRegressor` в scikit-learn) могут обучаться итеративно, принимая обучающие данные пакетами. При появлении новых обучающих данных вы можете обновить модель, использовав только новые данные.

Наконец, некоторые алгоритмы, такие как обучение дерева решений, SVM и kNN, могут использоваться как для классификации, так и для регрессии, в то время как другие способны решать только задачи одного вида: либо классификацию, либо регрессию, но не обе.

Обычно каждая библиотека сопровождается документацией, где объясняется, какую задачу решает каждый алгоритм, какие входные значения допустимы и что возвращает модель. Документация также содержит информацию о гиперпараметрах.

5 Практические основы

До сих пор я лишь мимоходом упомянул некоторые проблемы, которые аналитик должен учитывать, работая над задачей машинного обучения: проектирование признаков, переобучение и настройка гиперпараметров. В этой главе мы поговорим об этих и других проблемах, которые необходимо разрешить, прежде чем вы сможете ввести инструкцию `model = LogisticRegression().fit(x, y)`.

5.1. Проектирование признаков

Когда менеджер по продукту говорит вам: «Нужно предсказать, останется ли у нас конкретный клиент. Вот вам журналы, фиксирующие взаимодействия клиента с нашим продуктом за пять лет», — вы не сможете просто взять эти данные, передать их библиотеке и получить прогноз. Вы должны сначала сконструировать **набор данных**.

Как рассказывалось в первой главе, набор данных — это коллекция **размеченных образцов** $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Каждый элемент $\mathbf{x}_i$ из N называется **вектором признаков**. Вектор признаков — это вектор, в котором каждое измерение $j = 1, ..., D$ содержит значение, которое как-то описывает образец. Это значение называется **признаком** и обозначается как $x^{(j)}$.

Проблема преобразования исходной информации в набор данных называется **проектированием признаков**. В большинстве случаев проектирование признаков — трудоемкий процесс, требующий от аналитика творческого подхода и, предпочтительно, знания предметной области.

Например, для преобразования журналов, фиксирующих взаимодействия пользователя с компьютерной системой, можно создать признаки, содержащие различную статистическую информацию о пользователе, извлеченную из журналов. К при-

меру, один признак мог бы содержать цену подписки; другие признаки — частоту соединений в день, неделю и год. Еще один признак мог бы содержать среднюю продолжительность сеанса в секундах или среднее время ответа на один запрос и т. д. Все, что доступно для измерения, можно использовать как признак. Задача аналитика — создать **информативные** признаки, которые позволят алгоритму обучения построить модель, хорошо предсказывающую метки в обучающих данных. Высокоинформативные признаки также называют признаками с большой **прогнозирующей способностью**. Например, средняя продолжительность сеанса пользователя имеет высокую прогнозирующую способность для предсказания — будет ли пользователь продолжать использовать приложение в будущем.

Мы говорим, что модель имеет **малое смещение**, если она дает хорошие результаты на обучающих данных. То есть модель делает немногочисленные ошибки, когда применяется для прогнозирования меток данных, использованных для построения модели.

5.1.1. Унитарное кодирование

Некоторые алгоритмы обучения работают только с векторами числовых признаков. Если какой-то признак в наборе данных является качественным, например «цвет» или «день недели», его можно преобразовать в несколько бинарных признаков.

Если образец имеет качественный признак «цвет» с тремя возможными значениями: «красный», «желтый», «зеленый», его можно преобразовать в вектор с тремя числовыми значениями с помощью унитарного кодирования (известного также как кодирование с одним активным состоянием, one-hot encoding):

$$\begin{aligned} \text{красный} &= [1, 0, 0] \\ \text{желтый} &= [0, 1, 0] \\ \text{зеленый} &= [0, 0, 1]. \end{aligned} \tag{5.1}$$

Такой подход увеличивает размерность векторов признаков. Не следует пытаться уменьшить размерность, преобразовав значение «красный» в 1, «желтый» в 2 и «зеленый» в 3, потому что это будет означать наличие упорядоченности в этой категории и важность этого конкретного порядка для принятия решений. Если порядок значений признака неважен, использование упорядоченных чисел в качестве значений может запутать алгоритм обучения[1], потому что алгоритм будет

[1] Когда порядок значений некоторого качественного признака действительно имеет значение, тогда можно заменить эти значения порядковыми числами, оставив единственный признак. Например, если признак представляет качество статьи и имеет значения {*плохо*, *удовлетворительно*, *хорошо*, *отлично*}, тогда можно заменить эти качественные значения числами, например {1, 2, 3, 4}.

пытаться найти упорядоченность там, где ее нет, что потенциально может привести к переобучению.

5.1.2. Биннинг

Противоположная ситуация, реже встречающаяся на практике, — когда имеется числовой признак, но его хотелось бы преобразовать в качественный. **Биннинг** (binning, в литературе также встречается bucketing или просто дискретизация) — это процесс преобразования признака с непрерывным диапазоном значений в несколько бинарных признаков, которые иногда называют корзинами (или категориями), обычно с разбивкой по диапазонам. Например, вместо представления возраста, как единственного действительного признака, аналитик мог бы разбить возраст на дискретные диапазоны: все возрасты от 0 до 5 лет могут быть объединены в одну категорию, от 6 до 10 лет — во вторую категорию, от 11 до 15 лет — в третью, и т. д.

Например, пусть признак $j = 4$ представляет возраст. Применяя биннинг, признак можно заменить соответствующими категориями. Допустим, мы добавили три новые категории, «age_bin1», «age_bin2» и «age_bin3», с индексами $j = 123$, $j = 124$ и $j = 125$ соответственно. Теперь, если для некоторого образца x_i признак $x_i^{(4)} = 7$, тогда мы можем установить признак $x_i^{(124)} = 1$; если $x_i^{(4)} = 13$ — установить признак $x_i^{(125)} = 1$ и т. д.

В некоторых случаях хорошо продуманная дискретизация может помочь алгоритму обучиться на меньшем количестве данных, потому что, используя такой подход, мы «подсказываем» алгоритму обучения, что если значение признака попадает в определенный диапазон, точное значение этого признака не имеет значения.

5.1.3. Нормализация

Нормализация — это процесс преобразования фактического диапазона значений числового признака в стандартный диапазон значений, обычно в интервале $[-1, 1]$ или $[0, 1]$.

Например, предположим, что естественные значения некоторого признака изменяются в диапазоне от 350 до 1450. Вычитая 350 из каждого значения признака и деля результат на 1100, можно нормализовать эти значения, преобразовав их в диапазон $[0, 1]$.

В общем случае формула нормализации выглядит так:

$$\bar{x}^{(j)} = \frac{x^{(j)} - \min^{(j)}}{\max^{(j)} - \min^{(j)}},$$

где $\min^{(j)}$ и $\max^{(j)}$ — минимальное и максимальное значения признака j в наборе данных соответственно.

Зачем нужна нормализация? Нормализация данных не является строгим требованием. Однако на практике она может способствовать увеличению скорости обучения. Вспомните пример градиентного спуска из предыдущей главы. Представьте, что у вас есть двумерный вектор признаков. Корректируя параметры $w^{(1)}$ и $w^{(2)}$, вы используете частные производные среднеквадратичной ошибки относительно $w^{(1)}$ и $w^{(2)}$. Если $x^{(1)}$ находится в диапазоне [0, 1000], а $x^{(2)}$ — в диапазоне [0, 0.0001], то при корректировке будет преобладать производная по признаку с большим значением.

Кроме того, часто полезно гарантировать относительно небольшой диапазон и примерно одинаковые изменения входных данных, чтобы избежать проблем, которые возникают у компьютеров при работе с очень маленькими или очень большими числами (так называемое **числовое переполнение**).

5.1.4. Стандартизация

Стандартизация (или **нормализация z-оценки**) — это процедура, в ходе которой значения признаков масштабируются таким образом, что приобретают свойства *стандартного нормального распределения* с $\mu = 0$ и $\sigma = 1$, где μ — среднее значение признака (усредняется по всем образцам в наборе данных), а σ — стандартное отклонение от среднего.

Стандартные оценки (или z-оценки) признаков вычисляются следующим образом:

$$\hat{x}^{(j)} = \frac{x^{(j)} - \mu^{(j)}}{\sigma^{(j)}}.$$

Возникает вопрос, когда следует использовать нормализацию, а когда стандартизацию. На этот вопрос нет однозначного ответа. Обычно, если набор данных не слишком большой и есть время, можно попробовать оба варианта и посмотреть, какой лучше подходит для решаемой задачи.

Если нет времени на эксперименты, используйте следующие эмпирические правила:

- в алгоритмах обучения без учителя на практике чаще выгоднее использовать стандартизацию, а не нормализацию;
- стандартизация также предпочтительнее для признаков, распределение значений которых близко к нормальному распределению (то есть если график распределения имеет колоколообразную форму);

- и снова стандартизация предпочтительнее для признаков, которые иногда могут иметь экстремально большие или экстремально малые (аномальные) значения; предпочтительность стандартизации объясняется тем, что нормализация «втискивает» нормальные значения в очень узкий диапазон;
- во всех остальных случаях предпочтительнее нормализация.

Масштабирование признаков обычно хорошо сказывается на большинстве алгоритмов обучения. Однако современные реализации алгоритмов, которые можно найти в популярных библиотеках, вполне устойчивы к признакам с разными диапазонами значений.

5.1.5. Работа с отсутствующими значениями признаков

Иногда аналитик получает данные в виде набора с уже определенными признаками. В некоторых образцах значения отдельных признаков могут отсутствовать. Это часто случается, когда набор данных создавался вручную, и человек, работавший с ним, забывал заполнять некоторые значения или вообще не измерял их.

Вот несколько типичных способов работы с отсутствующими значениями признаков:

- удалить данные с отсутствующими значениями признаков из набора данных (этот способ можно использовать, если набор данных достаточно велик и можно пожертвовать несколькими обучающими образцами);
- использовать алгоритм обучения, умеющий работать с отсутствующими значениями (зависит от библиотеки и конкретной реализации алгоритма);
- использовать вычислительные методы восстановления данных.

5.1.6. Методы восстановления данных

Один из методов восстановления данных заключается в замене отсутствующего значения признака средним значением, вычисленным по всему набору данных:

$$\hat{x}^{(j)} \leftarrow \frac{1}{N}\sum_{i=1}^{N} x_i^{(j)}.$$

Другой метод заключается в замене отсутствующего значения значением, выходящим за пределы диапазона нормальных значений. Например, если нормальные значения находятся в диапазоне $[0, 1]$, отсутствующее значение можно установить равным 2 или -1. Идея состоит в том, чтобы позволить алгоритму обучения самому решить, как лучше поступить, если значение признака значительно отличается

от типичных значений. Как вариант, отсутствующее значение можно заменить значением середины диапазона. Например, если значения признака находятся в диапазоне [–1, 1], отсутствующее значение можно установить равным 0. В данном случае идея заключается в том, что значение середины диапазона не окажет значительного влияния на прогноз.

Более продвинутый метод — решить задачу регрессии, использовав отсутствующее значение в качестве целевой переменной. Используя все остальные признаки $\left[x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(j-1)}, x_i^{(j+1)}, \ldots, x_i^{(D)}\right]$, можно сформировать вектор признаков $\hat{\mathbf{x}}_i$, установить $\hat{y}_i \leftarrow x^{(j)}$, где j — признак с отсутствующим значением. Затем построить регрессионную модель для прогнозирования $\hat{y}$ по $\hat{\mathbf{x}}$. Разумеется, для построения обучающих данных $\left(\hat{\mathbf{x}}, \hat{y}\right)$ должны использоваться только те данные из исходного набора данных, в которых присутствует значение признака j.

Наконец, если имеется достаточно большой набор данных и значения отсутствуют лишь в нескольких признаках, можно увеличить размерность векторов признаков, добавив бинарный признак для каждого признака с отсутствующими значениями. Например, пусть в D-мерном наборе данных признак $j = 12$ имеет отсутствующие значения. Для каждого вектора признаков $\mathbf{x}$ тогда можно добавить признак $j = D + 1$ со значением 1, если значение 12-го признака присутствует в $\mathbf{x}$, и 0 в противном случае. Тогда недостающее значение можно заменить нулем или любым другим числом по вашему выбору.

На этапе прогнозирования, если образец имеет отсутствующее значение, следует использовать тот же метод восстановления данных, который применялся к обучающим данным, чтобы заполнить недостающие значения признаков.

Заранее нельзя сказать, какой метод восстановления данных окажется лучшим в конкретной ситуации. Попробуйте применить несколько методов, постройте несколько моделей и выберите ту, которая показывает лучшие результаты.

5.2. Выбор алгоритма обучения

Выбор алгоритма машинного обучения иногда может оказаться сложной задачей. При наличии времени можно попробовать их все. Но обычно на решение задачи отводится ограниченное время. Попробуйте задать себе несколько вопросов, прежде чем приступить к работе над задачей. В зависимости от ответов вы сможете сузить круг алгоритмов и опробовать их на своих данных.

- Объяснимость.

  Нужно ли объяснять вашу модель нетехническим специалистам? Большинство точных алгоритмов обучения — это так называемые «черные ящики». Они

обучают модели, допускающие очень мало ошибок, но иногда очень трудно понять и еще сложнее объяснить, почему модель сделала тот или иной прогноз. Примерами таких моделей являются нейронные сети и ансамблевые модели.

С другой стороны, алгоритмы обучения kNN, линейной регрессии или дерева решений создают модели, которые дают пусть не всегда самые точные, зато легко объяснимые результаты.

- Возможность сохранения набора данных в оперативной памяти.

  Можно ли набор данных целиком загрузить в оперативную память сервера или персонального компьютера? Если да, тогда появляется возможность выбора из широкого спектра алгоритмов. В противном случае предпочтение следует отдавать **инкрементальным алгоритмам обучения**, способным совершенствовать модель постепенно, по мере добавления новых данных.

- Число признаков и данных.

  Сколько обучающих данных присутствует в наборе данных? Сколько признаков имеет каждый образец? Некоторые алгоритмы, включая **нейронные сети** и **градиентный бустинг** (мы рассмотрим их позже), могут обрабатывать огромное количество данных с миллионами признаков. Другие, такие как SVM, могут быть очень ограниченными в этом отношении.

- Качественные и количественные признаки.

  Состоят ли данные только из качественных или только из числовых признаков либо имеют признаки обоих видов? В зависимости от ответа некоторые алгоритмы могут быть неспособны обработать набор данных непосредственно, и вам придется преобразовать качественные признаки в числовые.

- Нелинейность данных.

  Являются ли данные линейно разделимыми или их можно моделировать с помощью линейной модели? Если да, тогда хорошим выбором могут оказаться SVM с линейным ядром, логистическая или линейная регрессия. Иначе более предпочтительными могут оказаться глубокие нейронные сети или ансамблевые алгоритмы, рассматриваемые в главах 6 и 7.

- Скорость обучения.

  Сколько времени можно отпустить алгоритму обучения для построения модели? Как известно, нейронные сети обучаются очень долго. Простые алгоритмы, такие как логистическая и линейная регрессия или деревья решений, действуют намного быстрее. Специализированные библиотеки содержат очень эффективные реализации некоторых алгоритмов; возможно, вы предпочтете поискать

в интернете, чтобы найти такие библиотеки. Некоторые алгоритмы, такие как случайные леса, при выполнении на многоядерных процессорах могут производить вычисления параллельно, поэтому при наличии десятков ядер способны строить модели намного быстрее.

- Скорость прогнозирования.

  Как быстро обученная модель должна генерировать прогнозы? Будет ли модель использоваться в окружении, где требуется очень высокая пропускная способность? Одни алгоритмы, такие как SVM, линейная и логистическая регрессия и нейронные сети (некоторых типов), чрезвычайно быстро генерируют прогноз. Другие, такие как kNN, ансамблевые алгоритмы и очень глубокие или рекуррентные нейронные сети, работают медленнее[1].

Чтобы не надеяться на удачу в выборе лучшего алгоритма для ваших данных, можно воспользоваться популярным способом: протестировать несколько алгоритмов на **контрольном наборе**. Мы поговорим об этом далее. В качестве альтернативы при использовании библиотеки scikit-learn можно попробовать схему выбора алгоритма, показанную на рис. 5.1.

5.3. Три набора

До сих пор я использовал выражения «набор данных» и «обучающий набор» взаимозаменяемо. Однако на практике аналитики работают с тремя наборами размеченных данных:

1. Обучающий набор.
2. Контрольный набор.
3. Тестовый набор.

После получения набора размеченных данных первое, что следует сделать, — перемешать данные и разбить их на три выборки: обучающую, контрольную и тестовую. Обучающая выборка обычно самая большая; она используется для построения модели. Выборки для проверки и тестирования имеют примерно одинаковые размеры и намного меньше обучающей выборки. Алгоритм обучения не должен использовать данные из этих двух выборок для построения модели. Вот почему эти две выборки часто называют **отложенными выборками** (holdout sets).

---

[1] Скорость прогнозирования kNN и ансамблевых методов, реализованных в современных библиотеках, достаточно высока, поэтому не бойтесь использовать эти алгоритмы в своей практике.

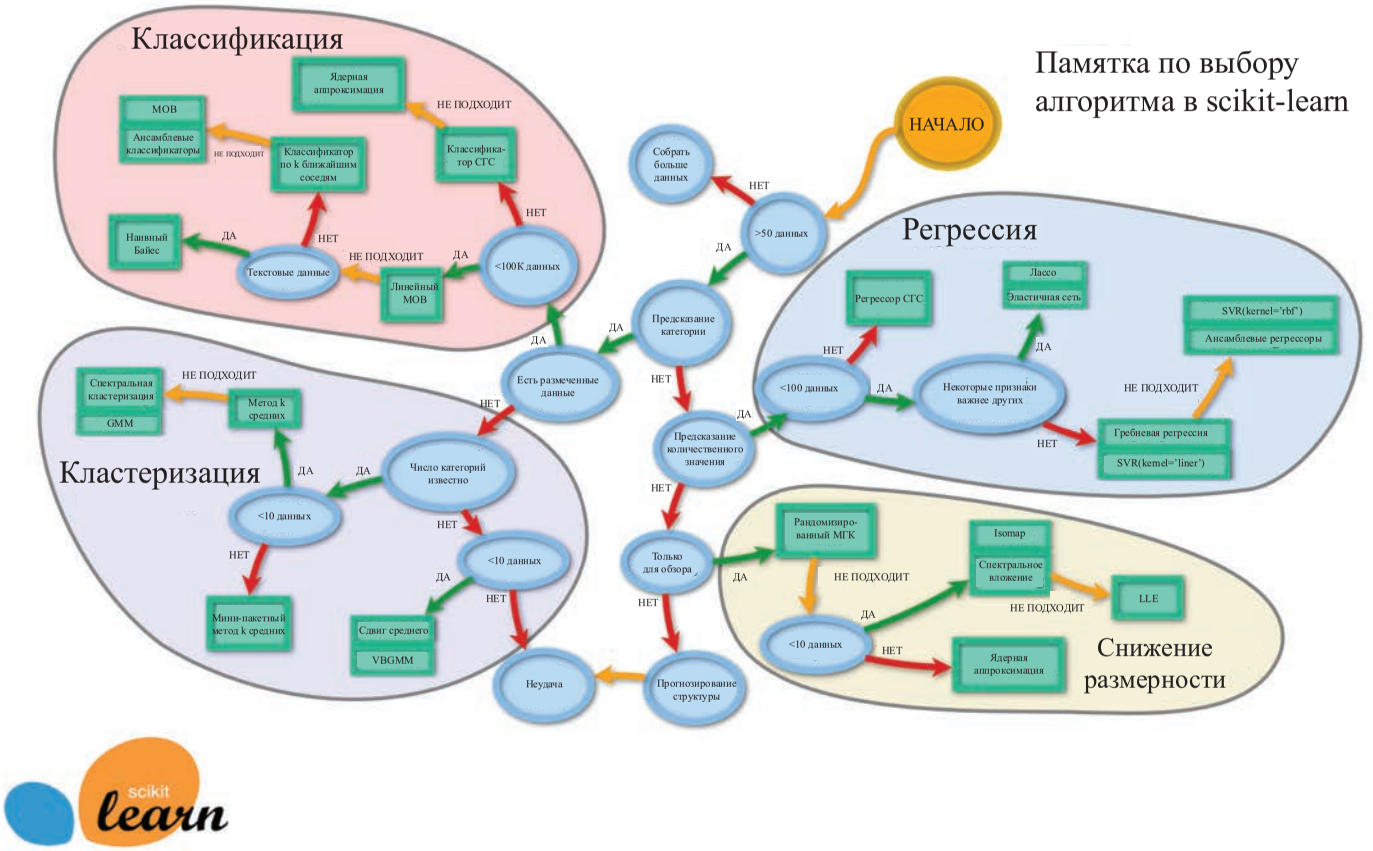


Рис. 5.1. Диаграмма выбора машинного алгоритма из имеющихся в библиотеке scikit-learn

Не существует оптимальной пропорции деления исходного набора данных на три выборки. В прошлом широко применялось эмпирическое правило, согласно которому 70 % исходных данных использовались для обучения, 15 % для проверки и 15 % для тестирования. Однако в эпоху больших данных наборы часто включают миллионы образцов. В таких случаях разумнее оставить 95 % для обучения и по 2.5 % выделить для контроля/тестирования.

Возможно, вам интересно, почему нужно использовать три выборки, а не одну. Ответ прост: нежелательно, чтобы модель давала хорошие прогнозы только для данных, которые алгоритм видел в процессе обучения. Тривиальный алгоритм, просто запоминающий все обучающие данные, а затем использующий память, чтобы «предсказать» их метки, не будет ошибаться, предсказывая метки обучающих данных, но такой алгоритм бесполезен на практике. В действительности нам нужно, чтобы модель давала достаточно точные прогнозы для данных, которые алгоритм обучения не видел: нам нужна высокая эффективность на отложенной выборке.

Зачем же нужны две контрольные выборки, а не одна? Контрольная выборка используется для: 1) выбора алгоритма обучения и 2) поиска оптимальных значений гиперпараметров. Тестовая выборка используется для оценки модели перед передачей ее клиенту или запуском в продакшн.

5.4. Недообучение и переобучение

Выше я упомянул понятие **смещения**. Я говорил, что модель имеет малое смещение, если дает хорошие результаты на обучающих данных. Если модель допускает много ошибок на обучающих данных, мы говорим, что модель имеет **большое смещение** или что модель **недообучена**. Недообученность — это неспособность модели более или менее точно предсказывать метки данных, на которых она обучалась. Причин недообучения может быть несколько, наиболее важными из них являются:

- модель слишком проста для данных (например, линейные модели часто страдают недообученностью);
- спроектированные вами признаки недостаточно информативны.

Первую причину легко показать на примере одномерной регрессии: набор данных может напоминать изогнутую линию, а модель описывает прямую линию. Вторую причину можно проиллюстрировать так: допустим, требуется предсказать наличие у пациента рака; в вашем распоряжении есть такие признаки, как рост, кровяное давление и частота сердечных сокращений. Очевидно, что эти три признака не являются хорошими прогностическими признаками рака,

поэтому модель, использующая их, не сможет выявить значимые связи между этими признаками и метками.

Чтобы решить проблему недообучения, можно попробовать использовать более сложную модель или спроектировать признаки с более высокой прогнозирующей способностью.

Переобучение — другая проблема, которой может страдать модель. Переобученная модель слишком хорошо предсказывает обучающие данные, но плохо — данные из обеих контрольных выборок или хотя бы из одной из них. Я уже приводил пример переобучения в главе 3. Причин переобучения может быть несколько, наиболее важными из них являются:

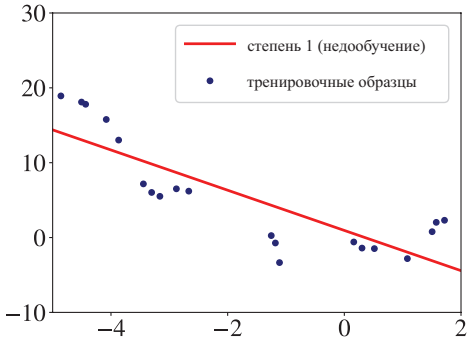
- модель слишком сложна для данных (например, очень высокое дерево решений или очень глубокая или широкая нейронная сеть часто бывают переобучены);
- слишком много признаков, но мало обучающих данных.

В литературе можно встретить другое название проблемы переобучения: проблема **высокой дисперсии**. Этот термин происходит из статистики. Дисперсия — это ошибка модели, вызванная ее чувствительностью к небольшим колебаниям в обучающем наборе. Если обучающие данные отобрать чуть иначе, в результате может получиться совершенно другая модель. Вот почему переобученная модель показывает плохие результаты на тестовых данных: тестовые и обучающие данные выбираются из исходного набора данных независимо друг от друга.

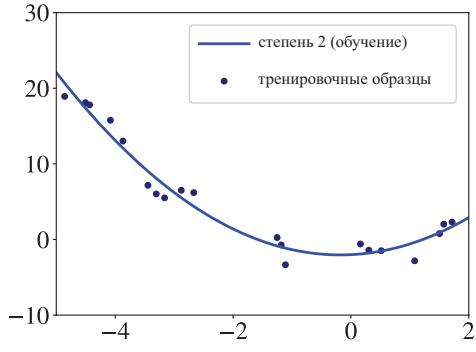
Даже самую простую линейную модель можно переобучить. Обычно такое происходит, когда данные многомерны, а количество обучающих данных относительно невелико. Фактически, когда векторы признаков слишком многомерны, линейный алгоритм обучения может построить модель, которая присваивает ненулевые значения большинству элементов $w^{(j)}$ в векторе параметров $\mathbf{w}$, пытаясь учесть очень сложные взаимосвязи между всеми доступными признаками, чтобы предсказать метки обучающих данных максимально точно.

Такая сложная модель, скорее всего, будет плохо предсказывать метки контрольных данных. Это связано с тем, что, пытаясь точно предсказать метки всех обучающих данных, модель также изучит специфические особенности обучающего набора: шум в значениях признаков обучающих данных, несовершенство выборки из-за малого размера набора данных и другие артефакты, не свойственные решаемой задаче, но присутствующие в обучающем наборе.

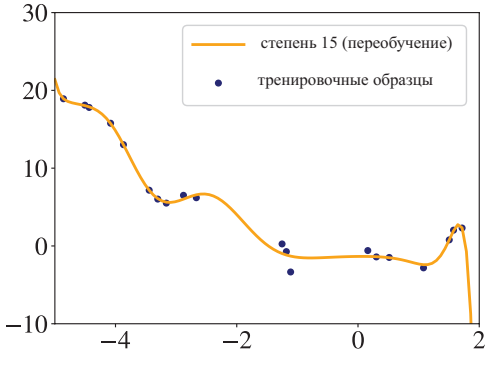
На рис. 5.2 изображен набор одномерных данных, для которого созданы недообученная, хорошо обученная и переобученная регрессионные модели.



Недообучение



Хорошее обучение



Переобучение

Рис. 5.2. Примеры недообучения (линейная модель), хорошего обучения (квадратичная модель) и переобучения (полином степени 15)

Вот некоторые методы борьбы с переобучением:

1. Попробовать более простую модель (линейную регрессию вместо полиномиальной, метод опорных векторов (SVM) с линейным ядром вместо радиальных базисных функций (RBF), нейронную сеть с меньшим числом слоев/узлов).
2. Уменьшить размерность данных в наборе данных (например, с помощью одного из методов уменьшения размерности, рассматриваемых в главе 9).
3. Добавить больше обучающих данных, если возможно.
4. Применить регуляризацию к модели.

Регуляризация — один из самых широко используемых приемов в борьбе с переобучением.

5.5. Регуляризация

Регуляризация — это собирательный термин, охватывающий методы, позволяющие алгоритмам обучения строить менее сложные модели. На практике это часто приводит к небольшому увеличению смещения, но значительно уменьшает дисперсию. Эта проблема известна в литературе как **дилемма смещения-дисперсии**.

На практике наиболее широко используются **L1-** и **L2-регуляризация**. Идея довольно проста. Чтобы создать регуляризованную модель, нужно модифицировать целевую функцию, добавив штрафной член, значение которого увеличивается с ростом сложности модели.

Я покажу регуляризацию на примере линейной регрессии. Но тот же принцип с успехом можно применить к широкому спектру моделей.

Напомню, как выглядит целевая функция регрессии:

$$\min_{\mathbf{w},b} \frac{1}{N}\sum_{i=1}^{N}\left(f_{\mathbf{w},b}(\mathbf{x}_i)-y_i\right)^2. \tag{5.2}$$

L1-регуляризованная целевая функция имеет вид

$$\min_{\mathbf{w},b}\left[C|\mathbf{w}|+\frac{1}{N}\sum_{i=1}^{N}\left(f_{\mathbf{w},b}(\mathbf{x}_i)-y_i\right)^2\right], \tag{5.3}$$

где $|\mathbf{w}| \overset{\text{def}}{=} \sum_{j=1}^{D}\left|w^{(j)}\right|$, а C является гиперпараметром, определяющим важность регуляризации. Если установить C равным нулю, модель превратится в стандартную нерегуляризованную модель линейной регрессии. С другой стороны, если присвоить гиперпараметру C большое положительное значение, алгоритм обучения постарается присвоить большинству элементов $w^{(j)}$ очень маленькие значения или ноль, чтобы минимизировать целевую функцию. В результате модель сильно упростится, что может привести к недообучению. Ваша роль как аналитика состоит в том, чтобы найти такое значение гиперпараметра C, которое не слишком увеличивает смещение и уменьшает дисперсию до приемлемого уровня. В следующем разделе я покажу, как это сделать.

L2-регуляризованная целевая функция имеет вид

$$\min_{\mathbf{w},b}\left[C\|\mathbf{w}\|^2+\frac{1}{N}\sum_{i=1}^{N}\left(f_{\mathbf{w},b}(x_i)-y_i\right)^2\right], \text{ где } \|\mathbf{w}\|^2 \overset{\text{def}}{=} \sum_{j=1}^{D}\left(w^{(j)}\right)^2. \tag{5.4}$$

С практической точки зрения L1-регуляризация создает **разреженную модель** — модель, большинство параметров которой (в случае линейной модели — боль-

шинство значений $w^{(j)}$) равно нулю, при условии что гиперпараметр C достаточно велик. Таким образом, L1 производит **отбор признаков**, решая, какие признаки важны для прогнозирования, а какие нет. Это может пригодиться, когда требуется упростить объяснение модели. Однако если единственная цель — увеличение качества прогнозирования на контрольных данных, то обычно лучшие результаты дает L2.

Методы регуляризации L1 и L2 также объединяются в так называемую **модель регуляризации эластичных сетей**, в которой L1- и L2-регуляризация являются частными случаями. В литературе также можно встретить название **гребневая регуляризация** (ridge regularization), подразумевающее L2-регуляризацию, и **лассо** (lasso), соответствующее L1-регуляризации.

Методы регуляризации L1 и L2 широко используются с линейными моделями, а также часто применяются к нейронным сетям и моделям многих других типов, которые напрямую минимизируют целевую функцию.

При создании нейронных сетей также нередко используются два других метода регуляризации: **прореживание** (dropout) и **пакетная нормализация**. Существуют также нематематические методы, имеющие эффект регуляризации: **расширение данных** (data augmentation) и **ранняя остановка**. Мы поговорим об этих методах в главе 8.

5.6. Оценка эффективности модели

После того как алгоритм построит модель на основе обучающих данных, как мы можем узнать, насколько хорошей получилась эта модель? Для оценки модели используется тестовый набор данных.

Тестовый набор содержит данные, которые алгоритм обучения не видел прежде, поэтому, если модель покажет хорошие результаты при прогнозировании меток данных из тестового набора, можно сказать, что модель имеет *хорошую обобщающую способность*, или просто хороша.

Строго говоря, для оценки эффективности моделей специалисты по машинному обучению используют разные формальные метрики и инструменты. В случае регрессии модель оценивается довольно просто. Хорошо обученная модель регрессии дает прогнозные значения, близкие к наблюдаемым. В отсутствие информативных признаков обычно используется **модель средних** (mean model), которая всегда прогнозирует средние значения меток в обучающих данных. Соответственно, оцениваемая регрессионная модель должна быть лучше модели средних. Если это

так, тогда на следующем шаге сравнивается эффективность модели на обучающих и тестовых данных.

Для этого вычисляется среднеквадратичная ошибка[1] (Mean Squared Error, MSE) отдельно для обучающих и тестовых данных. Если среднеквадратичная ошибка модели на тестовых данных *существенно выше* среднеквадратичной ошибки на обучающих данных, это явный признак переобучения. Проблему могут решить регуляризация или более тщательный подбор гиперпараметров. Смысл слов «существенно выше» зависит от решаемой задачи и определяется аналитиком совместно с лицом, принимающим решение, заказавшим модель.

В случае с классификацией все немного сложнее. Вот метрики и инструменты, наиболее широко используемые для оценки моделей классификации:

- матрица ошибок;
- правильность;
- правильность с учетом цены (cost-sensitive accuracy);
- точность/полнота;
- площадь под кривой рабочей характеристики приемника (Receiver Operating Characteristic, ROC).

Для простоты иллюстрации я использую задачу бинарной классификации. А там, где это необходимо, я покажу, как распространить решение на случай многоклассовой классификации.

5.6.1. Матрица ошибок

Матрица ошибок — это таблица, описывающая успешность классификации данных, принадлежащих разным классам. Одна ось матрицы ошибок — метка, предсказанная моделью, а другая ось — фактическая метка. В задаче бинарной классификации есть два класса. Допустим, модель делает выбор из двух классов: «спам» и «не_спам»:

	спам (предсказание)	**не_спам (предсказание)**
спам (факт)	23 (TP)	1 (FN)
не_спам (факт)	12 (FP)	556 (TN)

[1] Или функция средней потери любого другого типа, имеющего смысл.

Приведенная выше матрица ошибок показывает, что из 24 данных «спам» модель правильно классифицировала 23. В этом случае мы говорим, что у нас есть 23 **истинно положительных** (true positives, TP) результата, или TP = 23. Модель неправильно классифицировала 1 образец как не_спам. То есть мы имеем 1 **ложноотрицательный** (false negative, FN) результат, или FN = 1. Аналогично, из 568 данных, фактически не являющихся спамом, 556 были классифицированы правильно. То есть мы имеем 556 **истинно отрицательных** (true negatives, TN), или TN = 556. Наконец, 12 данных, фактически не являющихся спамом, были классифицированы неверно, а значит, мы имеем 12 **ложноположительных** (false positives, FP), или FP = 12.

Матрица ошибок для случая многоклассовой классификации имеет число строк и столбцов по числу разных классов. Она может помочь вам определить тенденции в распределении ошибок. Например, матрица ошибок может показать, что модель, обученная распознаванию видов животных, имеет тенденцию ошибочно распознавать «пантеру» как «кошку» или «крысу» как «мышь». В таком случае можно попробовать добавить больше размеченных данных этих видов животных, чтобы помочь алгоритму обучения «увидеть» разницу между ними. Также можно попробовать добавить дополнительные признаки, которые алгоритм обучения сможет использовать для построения модели, способной лучше различать эти виды животных.

Матрица ошибок используется для вычисления двух других метрик: **точность** и **полнота**.

5.6.2. Точность/полнота

Для оценки эффективности модели часто используются две метрики — **точность** и **полнота**. Точность — это отношение **истинно положительных** прогнозов к общему количеству положительных прогнозов:

$$\text{точность} \overset{\text{def}}{=} \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

Полнота — это отношение **истинно положительных** прогнозов к общему количеству положительных данных:

$$\text{полнота} \overset{\text{def}}{=} \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

Чтобы понять значимость и важность точности и полноты для оценки модели, часто полезно рассматривать задачу прогнозирования как задачу поиска документов в базе данных с помощью запросов. Точность — это доля релевантных документов

в списке, возвращаемом запросом. Полнота — это доля релевантных документов, возвращаемых поисковым механизмом, в общем количестве релевантных документов, которые могли бы быть возвращены.

В случае с задачей классификации спама нам нужна высокая точность (желательно исключить ошибочную классификацию допустимого сообщения как спам), и мы готовы поступиться полнотой (то есть мы допускаем, что некоторые сообщения, являющиеся спамом, окажутся в папке для входящих сообщений).

На практике почти всегда приходится выбирать между высокой точностью или высокой полнотой. Добиться высоких значений обеих метрик одновременно обычно невозможно. Мы можем повысить тот или другой показатель разными способами:

- назначая более высокие весовые коэффициенты образцам (к примеру, алгоритм SVM принимает на вход весовые коэффициенты для классов);
- настраивая гиперпараметры для получения более высокого значения точности или полноты на контрольном наборе;
- изменяя порог принятия решения для алгоритмов, возвращающих вероятности классов; например, при использовании логистической регрессии или дерева решений, для того чтобы повысить точность (за счет снижения полноты), можно считать прогноз положительным, только если вероятность, возвращаемая моделью, выше 0.9.

Даже притом, что точность и полнота определены для случая бинарной классификации, эти оценки всегда можно распространить на модель многоклассовой классификации. Для этого сначала нужно выбрать класс для оценивания. И затем считать все данные, принадлежащие выбранному классу, положительными, а все остальные данные — отрицательными.

5.6.3. Правильность

Правильность определяется количеством правильно классифицированных данных, деленным на общее количество классифицированных примеров. В терминах матрицы ошибок она определяется как

$$\text{правильность} \stackrel{\text{def}}{=} \frac{TP + TN}{TP + TN + FP + FN}. \tag{5.5}$$

Оценка правильности может пригодиться, когда одинаково важны ошибки в прогнозировании всех классов. Это не относится к задаче классификации спам/

не_спам. Например, вы терпимее будете относиться к ложноотрицательным результатам, чем к ложноположительным. Ложноположительный результат в классификации спама — это когда ваш друг отправил вам электронное письмо, но модель распознала его как спам и спрятала от вас. С другой стороны, ложноотрицательный результат — намного меньшая проблема: если ваша модель не распознает небольшой процент спама, это не будет иметь большого значения.

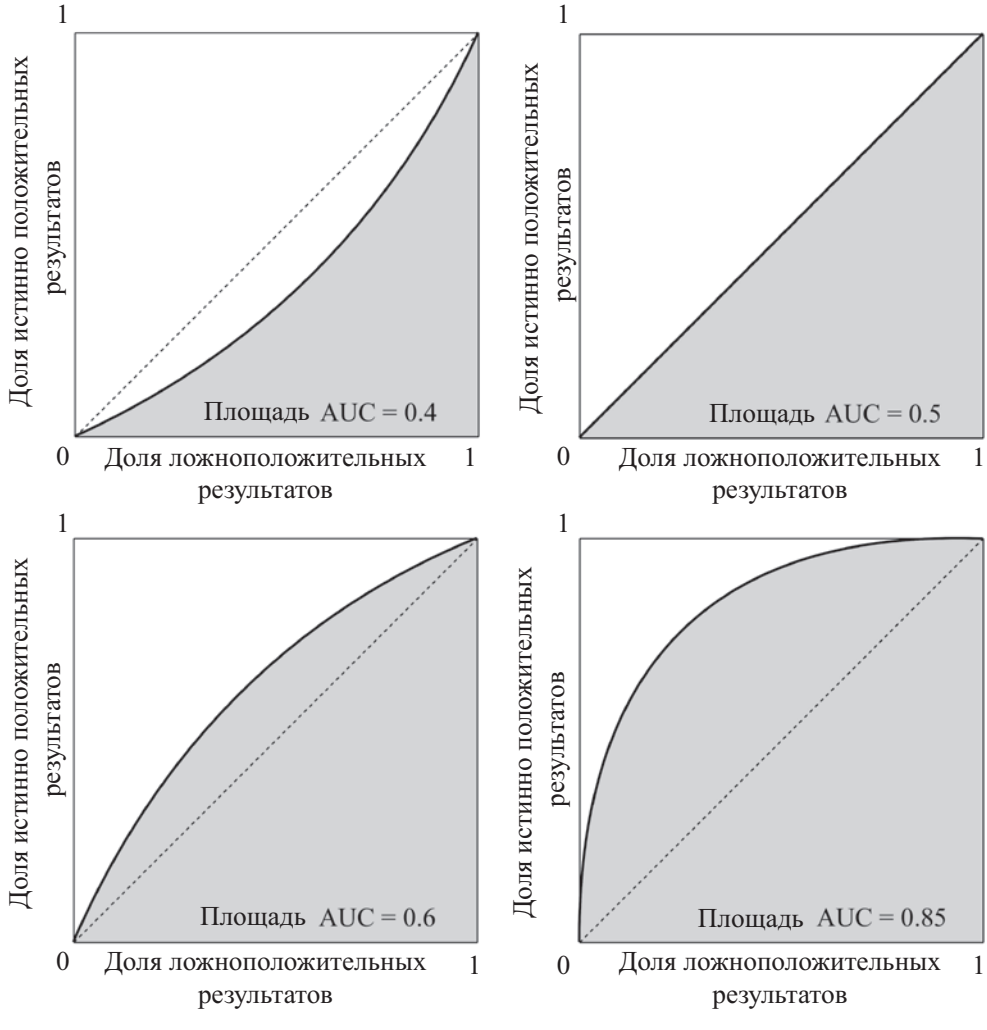


Рис. 5.3. Площадь под кривой рабочей характеристики (окрашена в серый цвет)

5.6.4. Правильность с учетом цены

В случаях, когда разные классы имеют разную важность, может пригодиться метрика **правильность с учетом цены** (cost-sensitive accuracy). Чтобы получить оценку правильности с учетом цены, нужно присвоить цену (положительное число) обоим типам ошибок: FP и FN. Затем вычислить значения TP, TN, FP, FN, как обычно, умножить значения FP и FN на соответствующие цены и определить правильность с помощью уравнения 5.5.

5.6.5. Площадь под ROC-кривой (AUC)

ROC-кривая (термин receiver operating characteristic — рабочая характеристика приемника — происходит из радиолокации) — это широко используемый метод оценки эффективности классификационных моделей. Чтобы создать общую картину эффективности классификации, ROC-кривые используют комбинацию **доли истинно положительных результатов** (определяется в точности как **полнота**) и **доли ложноположительных результатов** (доля отрицательных данных, классифицированных неправильно).

Доли истинно положительных (True Positive Rate, TPR) и ложноположительных (False Positive Rate, FPR) результатов определяются как

$$\mathrm{TPR} \overset{\text{def}}{=} \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \text{ и } \mathrm{FPR} \overset{\text{def}}{=} \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}.$$

ROC-кривые можно использовать только для оценки классификаторов, которые возвращают некоторую оценку достоверности (или вероятность) прогноза. Например, с использованием ROC-кривых можно оценить логистическую регрессию, нейронные сети и деревья решений (включая ансамблевые модели, основанные на деревьях решений).

Чтобы начертить ROC-кривую, сначала нужно дискретизировать диапазон оценок достоверности. Если для данной модели диапазон равен [0, 1], его можно дискретизировать следующим образом: [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]. Затем каждое дискретное значение используется в качестве порога при прогнозировании, с использованием модели, меток данных в наборе данных. Например, чтобы вычислить TPR и FPR для порога 0.7, нужно применить модель к каждому образцу, получить оценку достоверности, и, если оценка выше или равна 0.7, для этого образца выбирается положительный класс; иначе — отрицательный.

Взгляните на рис. 5.3. Легко заметить, что для порога 0 все прогнозы будут положительными, поэтому обе метрики, TPR и FPR, будут равны 1 (верхний правый угол). С другой стороны, для порога 1 положительные прогнозы невозможны, поэтому оценки TPR и FPR будут равны 0, что соответствует левому нижнему углу.

Чем больше площадь под ROC-кривой (area under ROC curve, AUC), тем эффективнее классификатор. Классификатор с AUC выше 0.5 лучше классификатора, действующего методом случайного выбора. Если AUC ниже 0.5, значит, с моделью что-то не так. Идеальный классификатор будет иметь AUC, равную 1. Обычно, имея хорошую модель, можно получить эффективный классификатор, выбирая значение порога, который дает значение TPR, близкое к 1, и удерживает значение FPR около 0.

ROC-кривые пользуются большой популярностью, потому что относительно просты для понимания, способны охватывать несколько аспектов классификации (с учетом ложных положительных и отрицательных результатов) и позволяют визуально и с минимальными усилиями сравнивать эффективность разных моделей.

5.7. Настройка гиперпараметров

Когда я знакомил вас с обучающими алгоритмами, то говорил, что вы, как аналитик, должны выбирать правильные значения для гиперпараметров алгоритма, таких как ϵ и d для ID3, C для SVM или α для градиентного спуска. Но что понимается под «правильными значениями»? Какое значение является правильным и как его найти? В данном разделе я отвечаю на эти важные вопросы.

Как вы уже знаете, гиперпараметры не оптимизируются самим алгоритмом обучения. Аналитик должен «настроить» гиперпараметры экспериментально, подобрав лучшую комбинацию значений, по одному для каждого гиперпараметра.

Один из типичных способов сделать это, если имеется достаточный объем данных, чтобы выделить приличный контрольный набор (в котором каждый класс представлен как минимум парой десятков данных), а количество гиперпараметров и их диапазон не слишком велики, — использовать **поиск по сетке**.

Поиск по сетке — самый простой метод настройки гиперпараметров. Допустим, вы обучаете SVM и у вас есть два гиперпараметра: параметр штрафа C (положительное действительное число) и ядро («linear» или «rbf»).

Если вы впервые работаете с конкретным набором данных, вы не знаете возможный диапазон значений для C. На практике часто применяется простой трюк — ис-

пользуется логарифмическая шкала. Например, вы можете попробовать следующие значения для *C*: [0.001, 0.01, 0.1, 1, 10, 100, 1000]. В этом случае у вас есть 14 комбинаций гиперпараметров: [(0.001, «linear»), (0.01, «linear»), (0,1, «linear»), (1, «linear»), (10, «linear» »), (100, «linear»), (1000, «linear»), (0.001, «rbf»), (0.01, «rbf»), (0.1, «rbf»), (1, «rbf»), (10, «rbf»), (100, «rbf»), (1000, «rbf»)].

Далее вы обучаете 14 моделей на обучающем наборе, по одной для каждой комбинации гиперпараметров. Затем оцениваете эффективность каждой модели на контрольных данных, используя одну из метрик, о которых мы говорили в предыдущем разделе (или какую-то другую метрику, важную для вас). И наконец, сохраняете модель, обладающую лучшей эффективностью в соответствии с выбранной метрикой.

Определив лучшую пару гиперпараметров, вы можете попробовать другие значения, близкие к лучшим. Иногда это позволяет получить еще более эффективную модель.

В заключение вы оцениваете выбранную модель с использованием тестового набора.

Очевидно, что проверка всех комбинаций гиперпараметров, особенно если их несколько, может занять много времени, особенно для больших наборов данных. Существуют более эффективные методы, такие как **случайный поиск** и **байесовская оптимизация гиперпараметров**.

Случайный поиск отличается от поиска по сетке тем, что от вас не требуется создавать дискретный набор значений для каждого гиперпараметра; вместо этого вы произвольно определяете статистическое распределение каждого гиперпараметра, из которого случайным образом выбираются значения для опробования, и задаете общее количество комбинаций, которые нужно опробовать.

Байесовские методы отличаются от случайного поиска или поиска по сетке тем, что выбор следующих значений для оценки они производят на основе результатов прошлых испытаний. Идея состоит в том, чтобы ограничить количество дорогостоящих оптимизаций целевой функции, выбирая следующие значения гиперпараметра на основе тех, которые давали хороший результат в прошлом.

Существуют также **методы на основе градиента**, **методы эволюционной оптимизации** и другие методы алгоритмической настройки гиперпараметров. Большинство современных библиотек машинного обучения реализуют один или несколько таких методов. Существуют также специализированные библиотеки настроек гиперпа-

раметров, которые могут помочь в настройке гиперпараметров для практически любого алгоритма обучения, включая созданные вами.

5.7.1. Перекрестная проверка

Когда нет возможности выделить приличного размера контрольный набор для настройки гиперпараметров, часто используется метод **перекрестной проверки**. Если обучающих данных немного, затруднительно выделить два дополнительных набора данных, контрольный и тестовый. Вы наверняка предпочтете использовать больше данных для обучения модели. В таком случае вы можете разбить данные только на два набора: обучающий и тестовый, а затем использовать прием перекрестной проверки на обучающем наборе, чтобы сымитировать контрольный набор.

Перекрестная проверка выполняется следующим образом. Сначала фиксируются значения гиперпараметров для оценки. Затем обучающий набор разбивается на несколько выборок одинакового размера. Каждая выборка называется *блоком*. На практике обычно используется перекрестная проверка с пятью блоками. Для перекрестной проверки с пятью блоками нужно случайным образом разбить обучающие данные на пять блоков: $\{F_1, F_2, ..., F_5\}$. Каждый F_k, $k = 1, ..., 5$ содержит 20 % обучающих данных. Затем обучается пять отдельных моделей, как описывается далее. Для обучения первой модели, f_1, используются все данные из блоков F_2, F_3, F_4 и F_5, а данные из F_1 играют роль контрольного набора. Для обучения второй модели, f_2, используются данные из блоков F_1, F_3, F_4 и F_5, а данные из F_2 используются для проверки. Аналогично строятся остальные модели и для каждого контрольного набора, от F_1 до F_5, вычисляются значения метрики. Затем пять значений метрики усредняются, чтобы получить окончательное значение.

Для поиска оптимальных значений гиперпараметров перекрестную проверку можно совместить с поиском по сетке. Определив искомые значения, можно использовать весь обучающий набор для построения модели с лучшими значениями гиперпараметров, найденными путем перекрестной проверки. В завершение производится оценка модели с использованием тестового набора.

6 Нейронные сети и глубокое обучение

Начнем с того, что вы уже знаете, что такое нейронная сеть, и знаете, как ее построить. Да, это логистическая регрессия! Фактически, модель логистической регрессии, или, скорее, ее обобщение для многоклассовой классификации, называемое моделью регрессии softmax, является стандартным узлом в нейронной сети.

6.1. Нейронные сети

Понимая, как работает линейная регрессия, логистическая регрессия и градиентный спуск, вы легко освоите нейронные сети.

Нейронная сеть (neural network, NN), так же как модель регрессии или SVM, — это всего лишь математическая функция:

$$y = f_{NN}(\mathbf{x}).$$

Функция f_{NN} имеет особую форму: это **вложенная функция** (вроде матрешки). Вы, наверное, уже слышали о **слоях** в нейронных сетях. Итак, для трехслойной нейронной сети, возвращающей скаляр, f_{NN} выглядит так:

$$y = f_{NN}(\mathbf{x}) = f_3(\boldsymbol{f}_2(\boldsymbol{f}_1(\mathbf{x}))).$$

$\boldsymbol{f}_1$ и $\boldsymbol{f}_2$ в уравнении выше — это векторные функции, которые определяются как

$$\boldsymbol{f}_l(\mathbf{z}) \overset{\text{def}}{=} \boldsymbol{g}_l(\mathbf{W}_l\mathbf{z} + \mathbf{b}_l), \tag{6.1}$$

где l — это индекс слоя и может охватывать от 1 до любого количества слоев. Функция $\mathbf{g}_l$ называется **функцией активации**. Это фиксированная, обычно нелинейная

функция, выбранная аналитиком до начала обучения. Параметры $\mathbf{W}_l$ (матрица) и $\mathbf{b}_l$ (вектор) определяются для каждого слоя в ходе обучения, с использованием уже знакомого вам градиентного спуска для оптимизации, в зависимости от решаемой задачи, конкретной функции стоимости (такой как среднеквадратичная ошибка — MSE). Сравните уравнение 6.1 с уравнением логистической регрессии, заменив $\mathbf{g}_l$ сигмоидной функцией, и вы не увидите никакой разницы. Функция f_3 является скалярной функцией для задачи регрессии, но также может быть векторной функцией, в зависимости от конкретной задачи.

Возможно, вам интересно, почему вместо вектора $\mathbf{w}_l$ используется матрица $\mathbf{W}_l$. Причина в том, что $\mathbf{g}_l$ — векторная функция. Каждая строка $\mathbf{w}_{l,u}$ (u означает «unit» — узел) в матрице $\mathbf{W}_l$ является вектором той же размерности, что и $\mathbf{z}$. Пусть $a_{l,u} = \mathbf{w}_{l,u}\mathbf{z} + b_{l,u}$. На выходе $\boldsymbol{f}_l(\mathbf{z})$ возвращает вектор $\left[g_l\left(a_{l,1}\right), g_l\left(a_{l,2}\right), \ldots, g_l\left(a_{l,size_l}\right)\right]$, где $\mathbf{g}_l$ — некоторая скалярная функция[1], а $size_l$ — количество узлов в слое l. Чтобы сделать обсуждение более конкретным, рассмотрим одну из архитектур нейронных сетей, называемую **многослойным перцептроном** и часто упоминаемую как **классическая нейронная сеть**.

6.1.1. Пример многослойного перцептрона

Далее мы подробно рассмотрим одну конкретную конфигурацию нейронных сетей, называемую **нейронными сетями прямого распространения** (Feedforward Neural Network, FFNN), и еще более конкретно — архитектуру, называемую **многослойным перцептроном** (Multilayer Perceptron, MLP). В качестве иллюстрации рассмотрим MLP с тремя слоями. Наша сеть принимает на вход двумерный вектор признаков и выводит число. Эта FFNN может быть моделью регрессии или классификации, в зависимости от функции активации, используемой в третьем выходном слое.

Наш многослойный перцептрон изображен на рис. 6.1. Графически нейронную сеть можно представить в виде комбинации связанных **узлов**, логически организованных в один или несколько **слоев**. Каждый узел представлен кружком или прямоугольником. Входящая стрелка представляет вход узла и указывает, откуда поступают данные на этот вход. Исходящая стрелка представляет выход узла.

Выход каждого узла является результатом математической операции, выполняемой внутри прямоугольника. Узлы, обозначенные кружками, ничего не делают с входными данными; они просто пересылают их на свой выход.

Вот что происходит в каждом узле, обозначенном прямоугольником. Сначала все входы узла объединяются, и из них формируется входной вектор. Затем узел при-

[1] Скалярная функция возвращает скаляр — простое число, а не вектор.

меняет линейное преобразование к входному вектору, точно так же, как это делает модель линейной регрессии со своим вектором входных признаков. И в заключение применяет функцию активации g к результату линейного преобразования, получая выходное значение — действительное число. В классической нейронной сети прямого распространения (FFNN) выходное значение узла в некотором слое становится входным значением для всех узлов в следующем слое.

На рис. 6.1 функция активации g_l имеет один индекс: l — индекс слоя, которому принадлежит узел. Обычно все узлы в одном слое уровня используют одну и ту же функцию активации, но это не обязательно. Каждый слой может иметь различное количество узлов. Каждый узел имеет свои параметры $\mathbf{w}_{l,u}$ и $b_{l,u}$, где u — индекс узла, а l — индекс слоя. Вектор $\mathbf{y}_{l-1}$ в каждом узле определяется как

$$\left[y_{l-1}^{(1)},\ y_{l-1}^{(2)},\ y_{l-1}^{(3)},\ y_{l-1}^{(4)}\right].$$

Вектор $\mathbf{x}$ в первом слое определяется как

$$\left[x^{(1)},\ \ldots,\ x^{(D)}\right].$$

Как можно заметить на рис. 6.1, в многослойном перцептроне все выходы одного слоя подключены к каждому входу последующего слоя. Такая архитектура называется **полносвязанной**. Нейронная сеть также может содержать **полносвязанные слои**, то есть слои, чьи узлы получают на входе выходные данные от каждого из узлов в предыдущем слое.

6.1.2. Архитектура нейронных сетей прямого распространения

Для решения задач регрессии или классификации, которые рассматривались в предыдущих главах, достаточно, чтобы последний (самый правый) слой нейронной сети содержал единственный узел. Если функция активации g_{last} последнего узла является линейной, тогда нейронная сеть является регрессионной моделью. Если g_{last} является логистической функцией, тогда нейронная сеть является моделью бинарной классификации.

Аналитик данных может выбрать для $g_{l,u}$ любую дифференцируемую математическую функцию[1]. Последнее свойство существенно для градиентного спуска, используемого для поиска значений параметров $\mathbf{w}_{l,u}$ и $b_{l,u}$ для всех l и u. Основная

[1] Функция должна быть дифференцируемой по всей области значений или в большинстве точек в этой области. Например, ReLU не дифференцируется в точке 0.

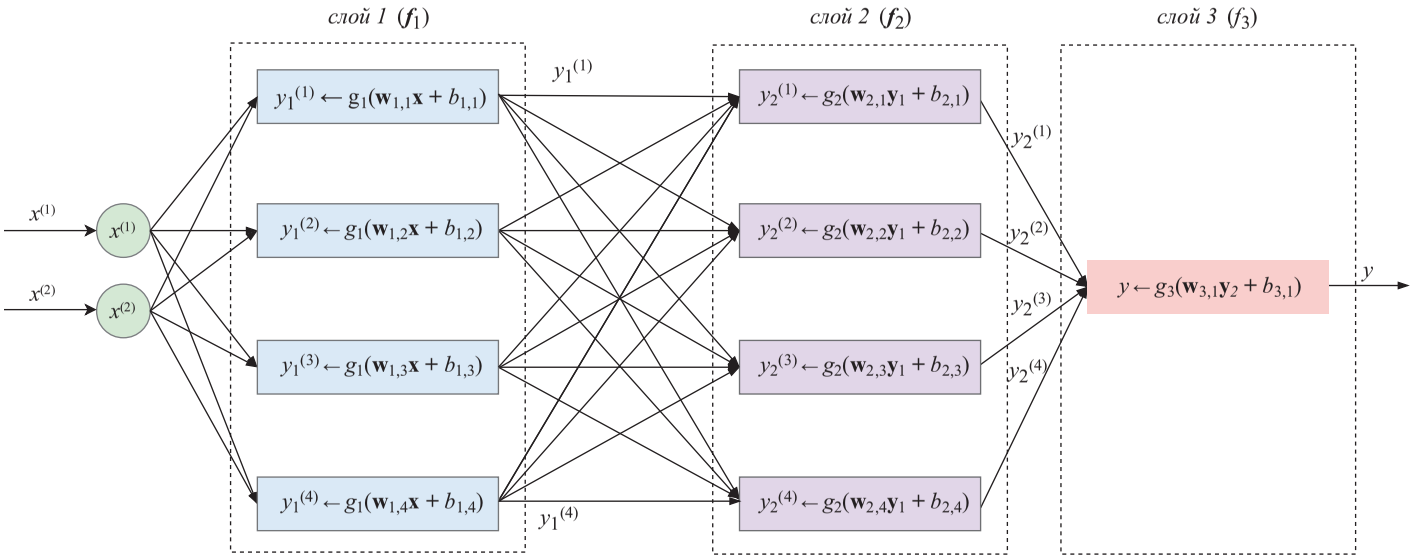


Рис. 6.1. Многослойный перцептрон с двумерным входом, двумя слоями по четыре узла в каждом и выходным слоем с единственным узлом

цель нелинейных компонентов в функции f_{NN} состоит в том, чтобы позволить нейронной сети аппроксимировать нелинейные функции. В отсутствие нелинейных компонентов f_{NN} была бы линейной, независимо от количества слоев. Причина в том, что $\mathbf{W}_l\mathbf{z} + \mathbf{b}_l$ является линейной функцией, а линейная функция от линейной функции также является линейной.

В роли функций активации часто используется уже известная вам логистическая функция, а также **TanH** и **ReLU**. Первая — это функция гиперболического тангенса, напоминающая логистическую функцию, но имеющая область определения в диапазоне от –1 до 1 (не включая их). Последняя представляет собой спрямленную линейную единичную функцию, которая равна нулю, когда на входе получает отрицательное значение z, и само значение z в противном случае:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

$$relu(z) = \begin{cases} 0 & \text{если } z < 0 \\ z & \text{иначе} \end{cases}.$$

Как я уже говорил выше, $\mathbf{W}_l$ в выражении $\mathbf{W}_l\mathbf{z} + \mathbf{b}_l$ — это матрица, а $\mathbf{b}_l$ — вектор. Этим оно отличается от линейной регрессии $\mathbf{wz} + b$. Каждой строке u в матрице $\mathbf{W}_l$ соответствует вектор параметров $\mathbf{w}_{l,u}$. Размерность вектора $\mathbf{w}_{l,u}$ равна количеству узлов в слое $l - 1$. Операция $\mathbf{W}_l\mathbf{z}$ дает вектор $\boldsymbol{a}_l \overset{\text{def}}{=} \left[\mathbf{w}_{l,1}\mathbf{z}, \mathbf{w}_{l,2}\mathbf{z}, \ldots, \mathbf{w}_{l,size_l}\mathbf{z}\right]$. Тогда сумма $\mathbf{a}_l + \mathbf{b}_l$ дает $size_l$-размерный вектор $\mathbf{c}_l$. Наконец, функция $\mathbf{g}_l(\mathbf{c}_l)$ создает и возвращает вектор $\mathbf{y}_l \overset{\text{def}}{=} \left[y_l^{(1)}, y_l^{(2)}, \ldots, y_l^{(size_l)}\right]$.

6.2. Глубокое обучение

Под глубоким обучением подразумевается обучение нейронных сетей, имеющих больше двух невыходных слоев. В прошлом обучение таких сетей усложнялось все больше с ростом количества слоев. В числе самых больших проблем назывались **взрывной рост градиента** и **затухание градиента**, поскольку для определения параметров сети использовался градиентный спуск.

Если проблема взрывного роста градиента решалась относительно просто, с применением таких простых методов, как **ограничение градиента** и L1- или L2-регуляризация, то проблема затухания градиента оставалась неразрешимой в течение десятилетий.

Что такое затухание градиента и чем обусловлена эта проблема? Для обновления значений параметров в нейронных сетях обычно используется алгоритм, называе-

мый **обратным распространением**. Обратное распространение — это эффективный алгоритм вычисления градиентов в нейронных сетях с использованием правила дифференцирования сложных функций. В главе 4 мы уже видели, как это правило используется для вычисления частных производных составной функции. В каждой итерации обучения, в процессе градиентного спуска, параметры нейронной сети обновляются пропорционально частной производной функции стоимости для текущего параметра. Проблема в том, что иногда градиент оказывается исчезающе малым, что фактически мешает изменению значений некоторых параметров. В худшем случае это может полностью остановить обучение нейронной сети.

Традиционные функции активации, такие как функция гиперболического тангенса, о которой я упоминал выше, имеют градиенты в диапазоне (0, 1), при этом градиенты вычисляются на этапе обратного распространения по правилу дифференцирования сложных функций. В результате для вычисления градиентов предыдущих слоев (расположенных левее) в n-слойной сети производится перемножение n этих небольших чисел, из-за чего градиент экспоненциально уменьшается с увеличением n. Это приводит к тому, что более ранние слои обучаются намного медленнее, если вообще обучаются.

Однако современные реализации алгоритмов обучения нейронных сетей позволяют эффективно обучать очень глубокие нейронные сети (до нескольких сотен слоев). Это объясняется внедрением целого комплекса усовершенствований, включая ReLU, LSTM (и другие вентильные узлы; мы рассмотрим их ниже), а также таких методов, как **соединения с пропуском слоя** (skip connections), используемые в **остаточных нейронных сетях** (residual neural networks), а также усовершенствованные версии алгоритма градиентного спуска.

Итак, поскольку проблемы затухания и взрывного роста градиента в настоящее время практически решены (или их влияние сведено к минимуму), под термином «глубокое обучение» подразумевается обучение нейронных сетей с использованием современного алгоритмического и математического инструментария, независимо от глубины нейронной сети. На практике многие бизнес-задачи можно решить с помощью нейронных сетей, имеющих 2–3 слоя между входным и выходным слоями. Слои, не являющиеся ни входными, ни выходными, часто называют **скрытыми слоями**.

6.2.1. Сверточная нейронная сеть

Возможно, вы заметили, что количество параметров, которые может иметь MLP (многослойный перцептрон), быстро растет по мере увеличения сети. В частности, при добавлении одного слоя в сеть добавляется $(size_{l-1}+1) \cdot size_l$ параметров (матрица $\mathbf{W}_l$ плюс вектор $\mathbf{b}_l$). То есть если в существующую нейронную сеть добавить

еще один слой с 1000 узлов, в модель добавится более миллиона дополнительных параметров. Оптимизация таких больших моделей является очень сложной вычислительной задачей.

Когда в роли обучающих данных используются изображения, входные данные получаются слишком многомерными[1]. Если вы решите построить модель классификации изображений с использованием MLP, проблема оптимизации почти наверняка станет неразрешимой.

Сверточная нейронная сеть (Convolutional Neural Network, CNN) — это особый вид FFNN, который значительно сокращает количество параметров в глубокой нейронной сети с большим количеством узлов практически без потери качества модели. Сверточные нейронные сети нашли применение в обработке изображений и текста, где они побили многие ранее установленные рекорды.

Так как сверточные нейронные сети были изобретены для обработки изображений, я объясню, как они работают, на примере классификации изображений.

Возможно, вы замечали, что пикселы, расположенные в изображениях близко друг от друга, обычно представляют информацию одного и того же типа: небо, вода, листья, мех, кирпичи и т. д. Исключением из правила являются края: части изображения, где два разных объекта «касаются» друг друга.

Если обучить нейронную сеть распознаванию областей с одной и той же информацией, а также краев, эти знания позволят нейронной сети распознавать объекты, представленные на изображении. Например, если нейронная сеть обнаружит несколько областей кожи с краями, имеющими овальную форму, с цветом кожи внутри и голубым цветом снаружи, то, скорее всего, это лицо на фоне неба. Если целью является обнаружение людей на изображениях, нейронная сеть, скорее всего, преуспеет в этом.

Учитывая, что наиболее важная информация занимает на изображении ограниченную площадь, мы можем разделить изображение на квадратные фрагменты, используя метод скользящего окна[2]. Затем обучить несколько небольших регрессионных моделей одновременно, передавая каждой квадратный фрагмент. Цель каждой небольшой регрессионной модели — научиться обнаруживать определенный шаблон во фрагменте на входе. Например, одна небольшая модель может научиться определять небо; другая — траву, третья — края зданий и т. д.

---

[1] Каждый пиксел в изображении — это отдельный признак. Изображение размером 100 на 100 пикселов имеет 10 000 признаков.

[2] Представьте, что рассматриваете банкноту в микроскоп. Чтобы рассмотреть всю банкноту, нужно постепенно перемещать ее слева направо и сверху вниз. В каждый момент времени вы видите только часть банкноты. Этот подход называется методом **скользящего окна**.

Небольшие регрессионные модели в CNN напоминают модель на рис. 6.1, но имеют только по одному слою. Чтобы обнаружить какой-либо шаблон, модель регрессии должна определить параметры матрицы **F** (от слова «filter» — фильтр) размером $p \times p$, где p — размер фрагмента. Для простоты предположим, что входное изображение черно-белое, число 1 представляет черные, а число 0 — белые пикселы. Предположим также, что фрагменты имеют размер 3 на 3 пиксела ($p = 3$). Некоторый фрагмент может выглядеть как следующая матрица **P** (от слова «patch» — фрагмент):

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Фрагмент выше представляет шаблон с изображением креста. Небольшая регрессионная модель, которая будет обнаруживать такие шаблоны (и только их), должна обучить матрицу **F** размером 3 на 3, в которой параметры в позициях, соответствующих единицам во входном фрагменте, будут положительными числами, а параметры в позициях, соответствующих нулям, будут иметь значения, близкие к нулю. Если вычислить **свертку** матриц **P** и **F**, полученное значение будет тем больше, чем больше **F** похожа на **P**. Для иллюстрации свертки двух матриц предположим, что **F** выглядит следующим образом:

$$\mathbf{F} = \begin{bmatrix} 0 & 2 & 3 \\ 2 & 4 & 1 \\ 0 & 3 & 0 \end{bmatrix}.$$

Оператор свертки `conv` определен только для матриц, имеющих одинаковое количество строк и столбцов. Для наших матриц **P** и **F** свертка вычисляется, как показано на рис. 6.2.

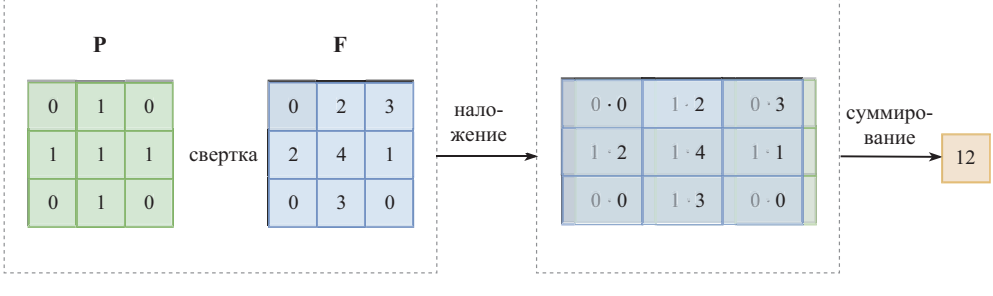


Рис. 6.2. Свертка двух матриц

Если на вход подать фрагмент **P**, имеющий другой шаблон, например изображение буквы L,

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix},$$

тогда свертка с **F** даст в результате меньшее значение: 5. То есть чем больше фрагмент «похож» на фильтр, тем выше значение операции свертки. Для удобства имеется также параметр смещения b, связанный с каждым фильтром **F**, который добавляется к результату свертки перед применением нелинейности (функция активации).

Один слой в CNN состоит из нескольких сверточных фильтров (каждый со своим собственным параметром смещения), в точности как один слой в классической FFNN состоит из нескольких узлов. Каждый фильтр в первом (самом левом) слое скользит — *свертывает* — по входному изображению слева направо, сверху вниз, и в каждой итерации вычисляет значение свертки.

Этот процесс изображен на рис. 6.3, где показаны шесть шагов перемещения одного фильтра, выполняющего свертку изображения.

Матрица фильтра (по одной для каждого фильтра в каждом слое) и значения смещения являются обучаемыми параметрами, которые оптимизируются с использованием градиентного спуска с обратным распространением.

Нелинейность применяется к сумме свертки и смещения. Как правило, во всех скрытых слоях используется функция активации ReLU. Функция активации в выходном слое зависит от решаемой задачи.

Поскольку в каждом слое l может иметься $size_l$ фильтров, выходной слой l будет состоять из $size_l$ матриц, по одной для каждого фильтра.

Если CNN имеет один сверточный слой, следующий за другим сверточным слоем, то последующий слой $l + 1$ будет обрабатывать выходные данные предыдущего слоя l, как коллекцию $size_l$ матриц изображения. Такая коллекция называется **томом**. Размер коллекции называется глубиной тома. Каждый фильтр в слое $l + 1$ выполняет свертку всего тома. Свертка фрагмента тома — это просто сумма сверток соответствующих фрагментов отдельных матриц, из которых состоит том.

На рис. 6.4 показан пример свертки фрагмента тома с глубиной 3. Значение свертки, –3, было получено в результате вычисления выражения:

$(-2 \cdot 3 + 3 \cdot 1 + 5 \cdot 4 + (-1) \cdot 1) + (-2 \cdot 2 + 3 \cdot (-1) + 5 \cdot (-3) + (-1) \cdot 1) + (-2 \cdot 1 + + 3 \cdot (-1) + 5 \cdot 2 + (-1) \cdot (-1)) + (-2).$

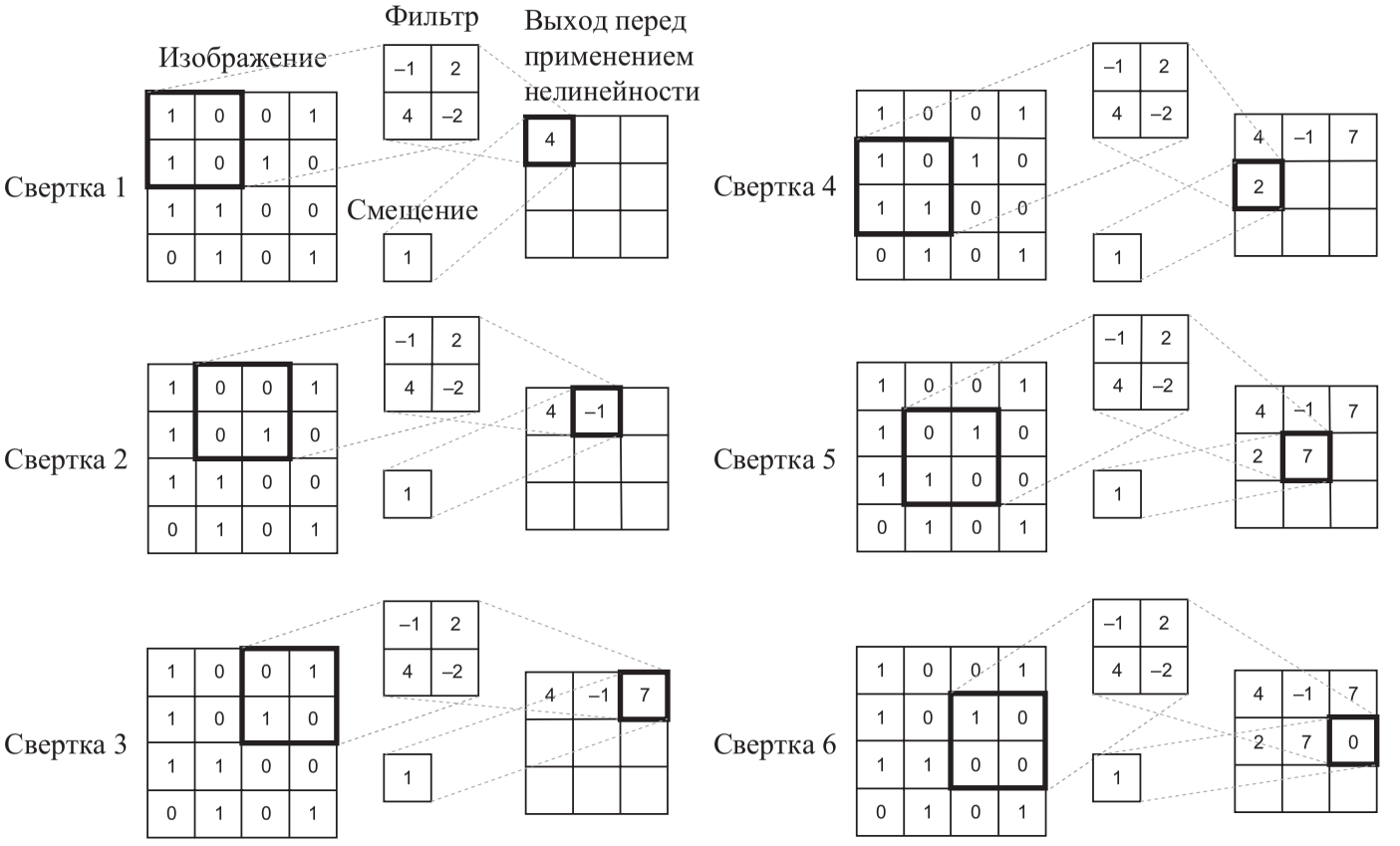


Рис. 6.3. Фильтр, свертывающий изображение

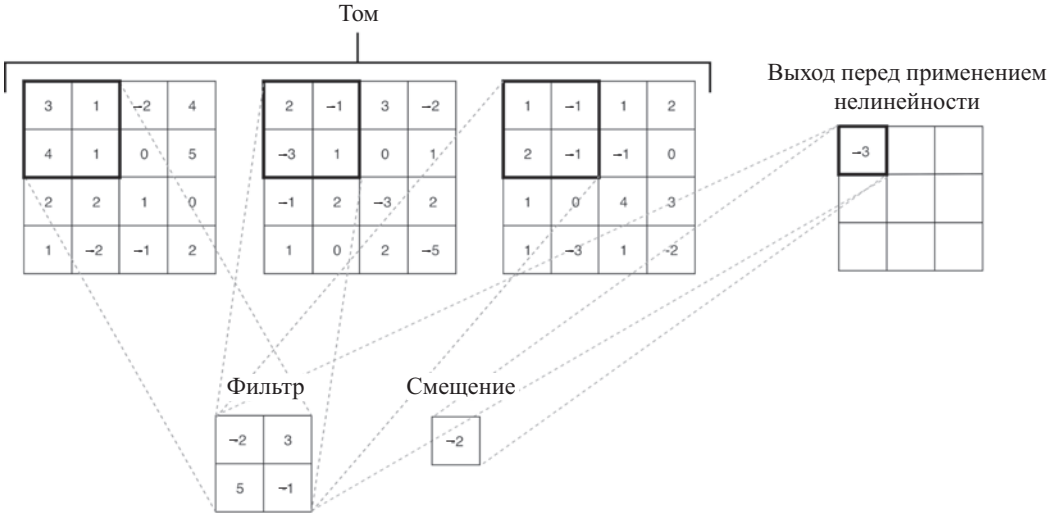


Рис. 6.4. Свертка тома состоит из трех матриц

В системах распознавания изображений сверточные нейронные сети часто получают на входе тома, потому что изображения обычно представлены тремя каналами: R, G и B, каждый из которых представляет монохромное изображение.

Свертки имеют два важных свойства — **шаг** и **дополнение**. Шаг — это величина одного шага смещения окна. На рис. 6.3 шаг равен 1, то есть фильтр смещается вправо и вниз на одну ячейку за раз. На рис. 6.5 показан пример свертки с шагом 2. Как видите, чем больше шаг, тем меньше выходная матрица.

Дополнение позволяет получить увеличенную выходную матрицу; это ширина рамки с дополнительными ячейками, которые добавляются вокруг изображения (или тома) перед сверткой с помощью фильтра. Обычно дополнительные ячейки, формирующие дополнение, содержат нули. На рис. 6.3 дополнение равно 0, поэтому дополнительные ячейки не добавляются в изображение. На рис. 6.6 задан шаг, равный 2, и дополнение, равное 1, поэтому вокруг изображения добавляется рамка шириной в 1 ячейку. Как видите, выходная матрица увеличилась с увеличением дополнения[1].

[1] Для экономии места на рис. 6.6 показаны только первые две из девяти сверток.

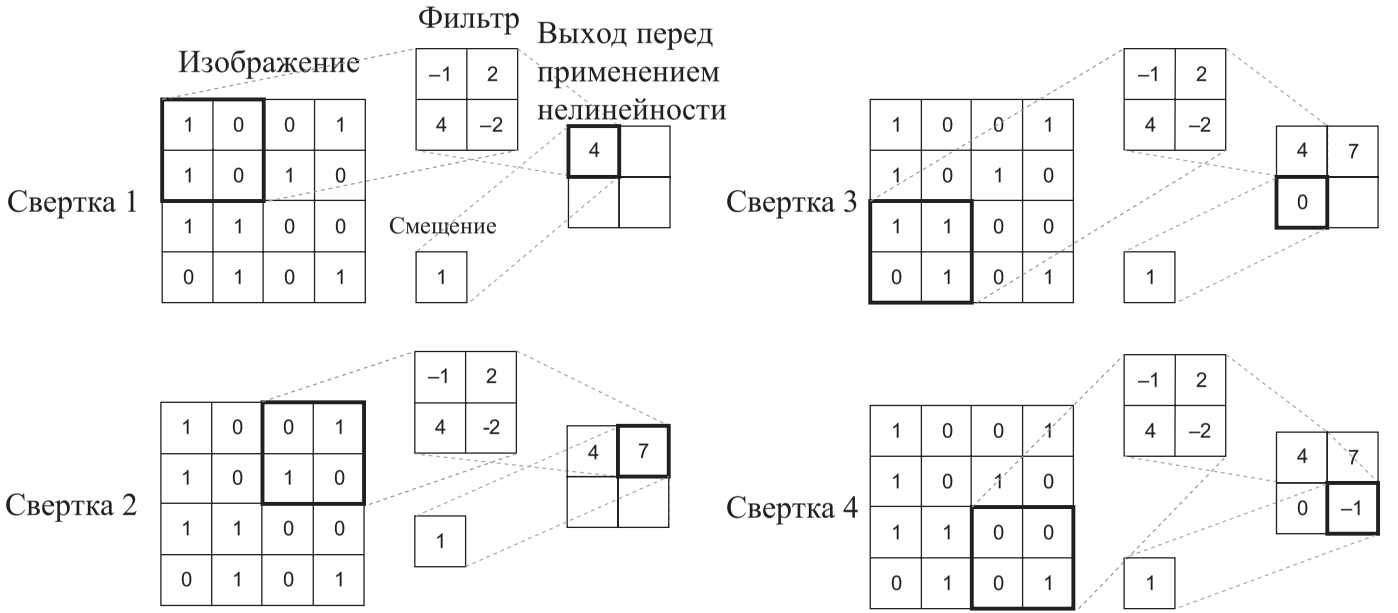


Рис. 6.5. Свертка с шагом 2

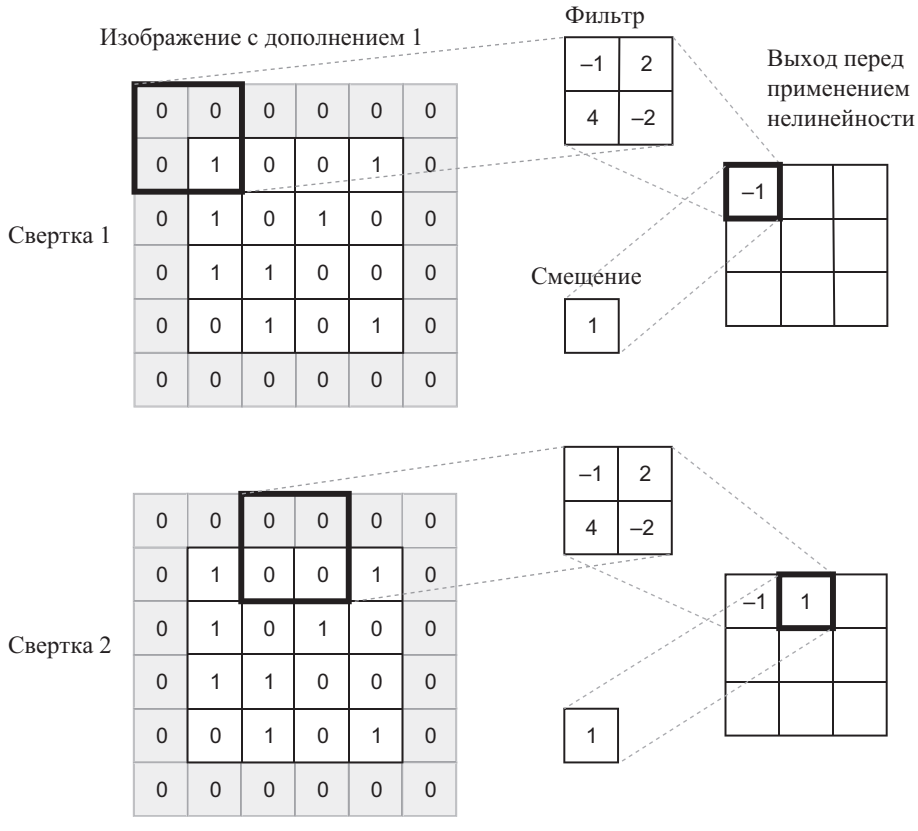


Рис. 6.6. Свертка с шагом 2 и дополнением 1

На рис. 6.7 показан пример изображения с дополнением 2. Дополнение может пригодиться при использовании более крупных фильтров, позволяя им лучше «сканировать» границы изображения.

Этот раздел был бы неполным без описания приема **подвыборки** (pooling, в русскоязычной литературе также встречаются термины «субдискретизация» и «пулинг»), очень часто используемого в CNN. Подвыборка действует во многом подобно свертке, потому что фильтр применяется методом скользящего окна. Однако вместо применения обучаемого фильтра к входной матрице или тóму слой подвыборки применяет фиксированный оператор, обычно max (возвращающий максимальное значение) или average (возвращающий среднее значение). Подобно свертке, операция подвыборки имеет гиперпараметры: размер фильтра и шаг. Пример подвыборки с оператором max с размером фильтра 2 и шагом 2 показан на рис. 6.8.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	0	1	0	1	0	0	0
0	0	1	1	0	0	0	0
0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Рис. 6.7. Изображение с дополнением 2

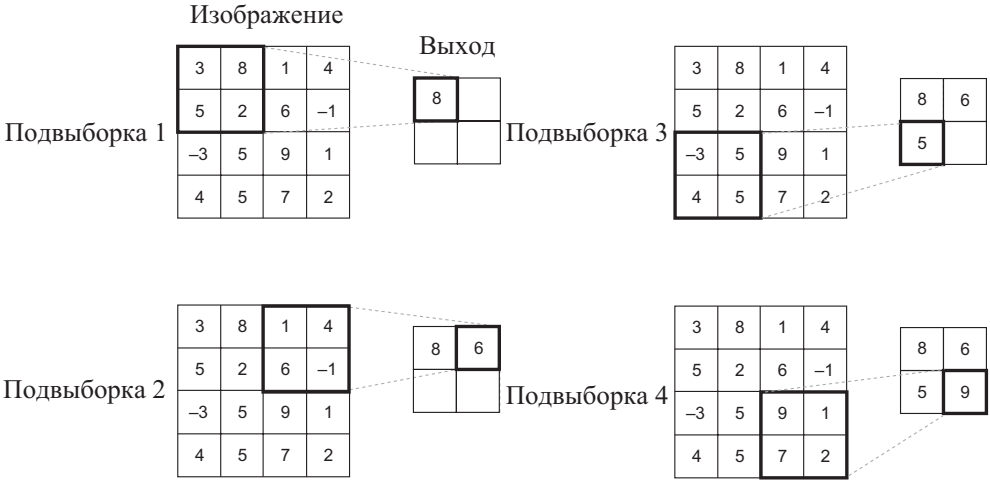


Рис. 6.8. Подвыборка с размером фильтра 2 и шагом 2

Как правило, слой подвыборки следует за сверточным слоем и получает на входе выходные данные свертки. Когда подвыборка применяется к то́му, каждая матрица в этом томе обрабатывается независимо от других. То есть в результате применения подвыборки к то́му получается том с той же глубиной.

Как видите, операция подвыборки имеет только гиперпараметры и не имеет обучаемых параметров. На практике обычно используются фильтры с размером 2

или 3 и с шагом 2. Подвыборка с определением максимального значения более популярна, чем с определением среднего, и часто дает лучшие результаты.

Обычно подвыборка способствует повышению точности модели, а также повышает скорость обучения за счет уменьшения количества вычислений требуемых для применения свертки. (Как показано на рис. 6.8, при размере фильтра 2 и шаге 2 размерность тома на входе следующего слоя уменьшается вдвое.)

6.2.2. Рекуррентная нейронная сеть

Рекуррентные нейронные сети (Recurrent Neural Network, RNN) используются для маркировки, классификации или генерации последовательностей. Последовательность — это матрица, каждая строка которой является вектором признаков и в которой порядок строк имеет значение. Под маркировкой последовательности подразумевается предсказание класса для каждого вектора признаков в последовательности. Под классификацией последовательности — предсказание класса для всей последовательности. Сгенерировать последовательность означает вывести другую последовательность (возможно, другой длины), некоторым образом связанную с входной последовательностью.

Рекуррентные нейронные сети часто используются для обработки текста, потому что предложения и тексты являются естественными последовательностями слов и знаков препинания или последовательностями символов. По той же причине рекуррентные нейронные сети также используются в обработке речи.

Рекуррентная нейронная сеть не является сетью прямого распространения: она содержит циклы. Идея состоит в том, что каждый узел u рекуррентного слоя l имеет вещественное **состояние** $h_{l,u}$. Состояние можно рассматривать как память узла. В RNN каждый узел u в каждом слое l имеет два входа: вектор состояний из предыдущего слоя $l-1$ и вектор состояний из этого же слоя l, но из *предыдущего временного шага*.

Для иллюстрации рассмотрим первый и второй рекуррентные слои в сети RNN. Первый (самый левый) слой получает на входе вектор признаков. Второй слой получает на входе выходные данные из первого слоя.

Эта ситуация схематически изображена на рис. 6.9. Как я уже говорил, каждый обучающий образец представлен матрицей, в которой каждая строка является вектором признаков. Для простоты будем рассматривать эту матрицу как последовательность векторов $\mathbf{X} = \left[\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^{t-1}, \mathbf{x}^t, \mathbf{x}^{t+1}, \ldots, \mathbf{x}^{length_x}\right]$, где $length_x$ — длина входной последовательности. Если входной образец $\mathbf{X}$ является текстовым предложением, тогда вектор признаков $\mathbf{x}^t$ для каждого $t = 1, \ldots, length_x$ представляет слово в позиции t в предложении.

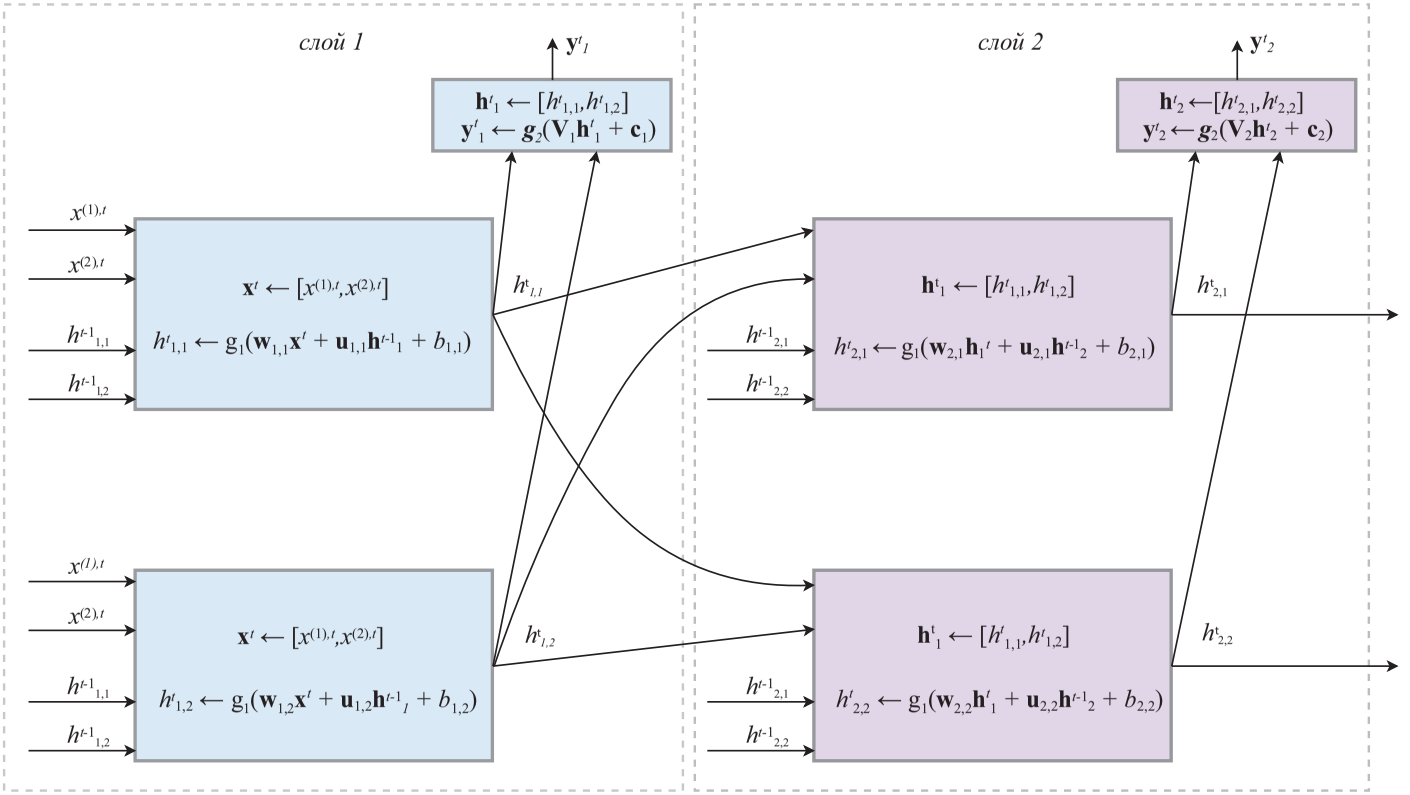


Рис. 6.9. Первые два слоя в рекуррентной нейронной сети.
На вход подается двумерный вектор признаков; каждый слой имеет два узла

Как показано на рис. 6.9, в RNN векторы признаков из входного образца последовательно «читаются» нейронной сетью в порядке временных шагов. Индекс t обозначает временной шаг. Чтобы обновить состояние $h_{l,u}^t$ на каждом временном шаге t в каждом узле u каждого слоя l, сначала вычисляется линейная комбинация входного вектора признаков с вектором состояния $\mathbf{h}_{l,u}^{t-1}$ этого же слоя из предыдущего временного шага $t-1$. Линейная комбинация двух векторов вычисляется с использованием двух векторов параметров $\mathbf{w}_{l,u}$, $\mathbf{u}_{l,u}$ и параметра $b_{l,u}$. Значение $h_{l,u}^t$ получается применением функции активации g_1 к результату линейной комбинации. Обычно в роли функции g_1 выступает *tanh*. Выход $\mathbf{y}_l^t$ обычно является вектором, который вычисляется сразу для всего слоя l. Чтобы получить $\mathbf{y}_l^t$, используется функция активации $\mathbf{g}_2$, которая на входе принимает вектор и возвращает другой вектор той же размерности. Функция $\mathbf{g}_2$ применяется к линейной комбинации вектора значений состояния h_l^t, вычисленных с использованием матрицы параметров $\mathbf{V}_l$ и вектора параметров $\mathbf{c}_l$. В классификации на роль $\mathbf{g}_2$ обычно выбирается **функция softmax**:

$$\sigma(\mathbf{z}) \overset{\text{def}}{=} \left[\sigma^{(1)}, \ldots, \sigma^{(D)}\right], \text{ где } \sigma^{(j)} \overset{\text{def}}{=} \frac{\exp\left(z^{(j)}\right)}{\sum_{k=1}^{D}\exp\left(z^{(k)}\right)}.$$

Функция softmax — это обобщение сигмоидной функции на многомерные выходы. Она имеет свойство $\sum_{j=1}^{D}\sigma^{(j)} = 1$ и $\sigma^{(j)} > 0$ для всех j.

Размерность $\mathbf{V}_l$ выбирается аналитиком таким образом, чтобы произведение матрицы $\mathbf{V}_l$ на вектор $\mathbf{h}_l^t$ давало вектор той же размерности, что и вектор $\mathbf{c}_l$. Этот выбор зависит от размерности выходной метки $\mathbf{y}$ в данных обучения. (До сих пор мы рассматривали только одномерные метки, но в следующих главах мы увидим, что метки могут быть многомерными.)

Значения $\mathbf{w}_{l,u}$, $\mathbf{u}_{l,u}$, $b_{l,u}$, $\mathbf{V}_{l,u}$ и $\mathbf{c}_{l,u}$ определяются по обучающим данным с использованием градиентного спуска с обратным распространением. Для обучения моделей RNN используется специальная версия обратного распространения, называемая **обратным распространением во времени**.

Обе функции, *tanh* и *softmax*, страдают проблемой затухания градиента. Даже если наша сеть RNN имеет только один или два рекуррентных слоя, из-за последовательного характера входных данных обратное распространение «развертывает» сеть с течением времени. С точки зрения вычисления градиента это означает, что чем длиннее входная последовательность, тем глубже получается развернутая сеть.

Другая проблема, характерная для RNN, заключается в обработке долгосрочных зависимостей. По мере увеличения длины входной последовательности векторы признаков, находящиеся в начале последовательности, постепенно «забываются», потому что состояние всех узлов, которые играют роль памяти сети, в значительной

мере зависит от векторов признаков, прочитанных последними. Следовательно, при обработке текста или речи причинно-следственная связь между удаленными словами в длинном предложении может быть потеряна. Наиболее эффективными рекуррентными моделями нейронных сетей, используемыми на практике, являются **вентильные RNN**. К ним относятся сети с **долгой краткосрочной** памятью (Long Short-Term Memory, LSTM) и сети с **вентильными рекуррентными узлами** (Gated Recurrent Unit, GRU).

Благодаря использованию вентильных узлов сети RNN получают способность хранить информацию в своих узлах для будущего использования почти так же, как хранятся биты в памяти компьютера. Разница лишь в том, что операции чтения, записи и стирания информации, хранящейся в каждом узле, контролируются функциями активации, которые принимают значения в диапазоне (0, 1). Обученная нейронная сеть может «прочитать» входную последовательность векторов признаков и на некотором раннем временном шаге t решить сохранить конкретную информацию о векторах признаков. Эта информация о более ранних векторах признаков может позже использоваться моделью для обработки векторов признаков в конце входной последовательности. Например, если текст на входе начинается со слова *она*, модель RNN для обработки текстов может решить запомнить род местоимения, чтобы правильно интерпретировать слово *ее*, следующее далее в предложении.

Решение о том, какую информацию хранить и когда разрешать чтение, запись и удаление, принимают узлы. Эти решения принимаются на основе данных и реализуются через идею *вентилей* (gates). Есть несколько архитектур управляемых узлов. Простая, но эффективная называется **минимальным вентильным узлом** и состоит из ячейки памяти и вентиля забывания.

Давайте посмотрим, как действует узел GRU с математической точки зрения, взяв в качестве примера первый слой RNN (тот, который принимает входную последовательность векторов признаков). Минимальный вентильный узел u в слое l имеет два входа: вектор значений ячеек памяти из всех узлов в том же слое из предыдущего временного шага $\mathbf{h}_l^{t-1}$ и вектор признаков $\mathbf{x}^t$. Он использует эти два вектора следующим образом (все операции, представленные ниже, выполняются узлом последовательно, друг за другом):

$$\tilde{h}_{l,u}^t \leftarrow g_1\left(\mathbf{w}_{l,u}\mathbf{x}^t + \mathbf{u}_{l,u}\mathbf{h}_l^{t-1} + b_{l,u}\right),$$
$$\Gamma_{l,u}^t \leftarrow g_2\left(\mathbf{m}_{l,u}\mathbf{x}^t + \mathbf{o}_{l,u}\mathbf{h}^{t-1} + a_{l,u}\right),$$
$$h_{l,u}^t \leftarrow \Gamma_{l,u}^t \tilde{h}_l^t + \left(1 - \Gamma_{l,u}^t\right) h_l^{t-1},$$
$$\mathbf{h}_l^t \leftarrow \left[h_{l,1}^t, \ldots, h_{l,size_l}^t\right],$$
$$\mathbf{y}_l^t \leftarrow \boldsymbol{g}_3\left(\mathbf{V}_l\mathbf{h}_l^t + \mathbf{c}_{l,u}\right),$$

где g_1 — функция активации *tanh*, g_2 называется управляющей функцией и реализуется как сигмоидная функция, принимающая значения в диапазоне (0, 1). Если вентиль $\Gamma_{l,u}$ близок к 0, тогда ячейка памяти сохраняет значение, полученное на предыдущем временном шаге h_l^{t-1}. Если вентиль $\Gamma_{l,u}$ близок к 1, значение ячейки памяти затирается новым значением $\tilde{h}_{l,u}^t$ (см. третью сверху операцию). Как и в стандартных сетях RNN, в роли $\mathbf{g}_3$ обычно используется функция softmax.

Управляемый узел принимает входные данные и хранит их в течение некоторого времени. Это эквивалентно применению к входу функции тождества ($f(x) = x$). Поскольку производная функции тождества является константой, когда сеть с управляемыми узлами обучается с обратным распространением во времени, градиент не затухает.

К другим важным расширениям RNN относятся: **двунаправленные RNN**, RNN с механизмом **внимания** и модели **RNN преобразования последовательностей в последовательности** (sequence-to-sequence). Последние, например, часто используются для реализации нейронных моделей машинного перевода и других моделей преобразования текста в текст. Обобщением RNN является **рекурсивная нейронная сеть**.

7 Проблемы и решения

7.1. Ядерная регрессия

Ранее мы говорили о линейной регрессии, но что если исходные данные имеют форму, отличную от прямой линии? В таких случаях может помочь полиномиальная регрессия. Допустим, у нас есть одномерные данные $\{(x_i, y_i)\}_{i=1}^{N}$. Мы могли бы попытаться найти квадратичную линию $y = w_1 x_i + w_2 x_i^2 + b$, описывающую наши данные. Определив функцию стоимости как среднеквадратичную ошибку (MSE), можно выполнить градиентный спуск и найти значения параметров w_1, w_2 и b, минимизирующие эту функцию. В одно- или двумерном пространстве легко увидеть, соответствует ли функция данным. Но если входные данные представлены D-мерным вектором признаков с $D > 3$, тогда найти правильный полином будет сложно.

Ядерная регрессия (kernel regression) является непараметрическим методом. Это означает отсутствие параметров, которые должны определяться в процессе обучения. Модель основана на самих данных (как kNN). В простейшем случае ядерная регрессия подбирает такую модель:

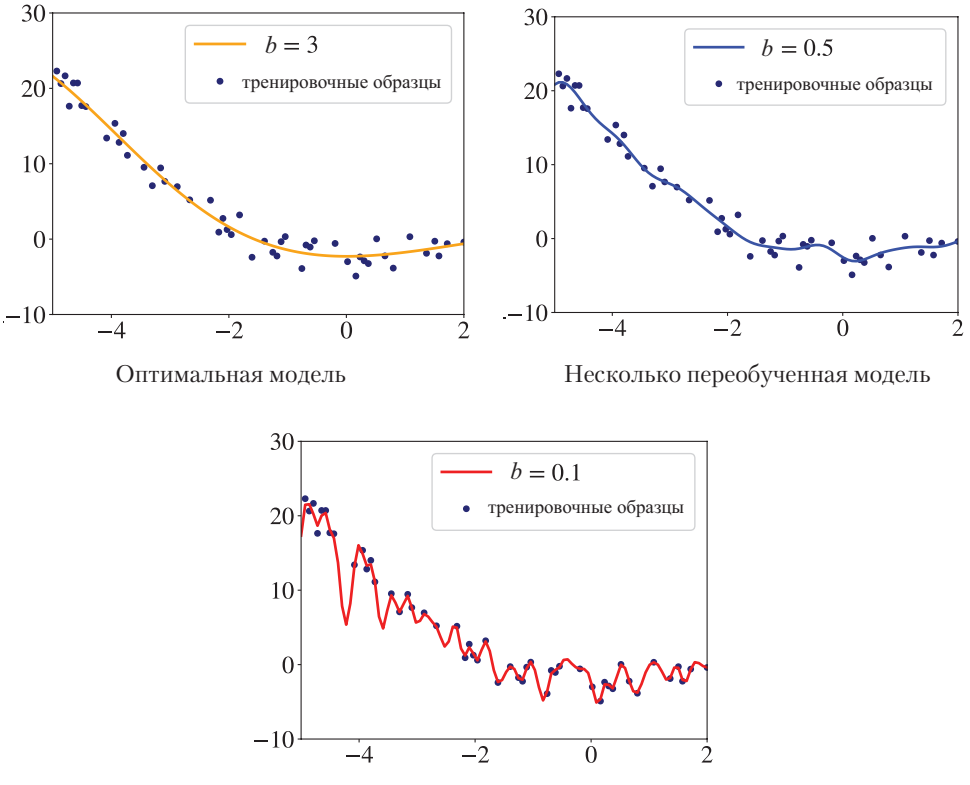
$$f(x) = \frac{1}{N}\sum_{i=1}^{N} w_i y_i, \text{ где } w_i = \frac{Nk\left(\frac{x_i - x}{b}\right)}{\sum_{l=1}^{N} k\left(\frac{x_l - x}{b}\right)}. \tag{7.1}$$

Функция $k(\cdot)$ называется **ядром** (kernel). Ядро играет роль функции подобия: значения коэффициентов w_i тем выше, чем ближе значение x к x_i, и наоборот. Ядро

может принимать разные формы. На практике наиболее часто используется ядро Гаусса:

$$k(z) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-z^2}{2}\right).$$

Значение b является гиперпараметром, который настраивается с использованием контрольного набора (путем применения модели, построенной с определенным значением b, к образцам в контрольном наборе и вычисления среднеквадратичной ошибки). На рис. 7.1 показано, как b влияет на форму линии регрессии.



Оптимальная модель

Несколько переобученная модель

Сильно переобученная модель

Рис. 7.1. Пример линий ядерной регрессии с ядром Гаусса для трех значений b

В случае, когда входные данные являются многомерными векторами признаков, члены $x_i - x$ и $x_l - x$ в уравнении 7.1 следует заменить евклидовым расстоянием $\|\mathbf{x}_i - \mathbf{x}\|$ и $\|\mathbf{x}_l - \mathbf{x}\|$ соответственно.

7.2. Многоклассовая классификация

Многие задачи классификации можно определить с использованием двух классов, но некоторые определены с использованием большего количества классов, что требует адаптации алгоритмов машинного обучения.

В многоклассовой классификации метка может быть одним из C классов: $y \in \{1, \ldots, C\}$. Многие алгоритмы машинного обучения являются бинарными, например SVM. Некоторые алгоритмы можно модифицировать для решения задач с множеством классов. ID3 и другие алгоритмы обучения деревьев решений можно изменить, как показано ниже:

$$f_{ID3}^{S} \stackrel{\text{def}}{=} \Pr(y_i = c|\mathbf{x}) = \frac{1}{|\mathcal{S}|} \sum_{\{y|(\mathbf{x}, y)\in S,\ y=c\}} y,$$

для всех $c \in \{1, \ldots, C\}$, где $\mathcal{S}$ — листовой узел, в котором происходит прогнозирование.

Логистическую регрессию можно естественным образом распространить на задачи многоклассового обучения, заменив сигмоидную функцию **функцией softmax**, которую мы уже видели в главе 6.

Алгоритм kNN тоже легко распространить на случай многоклассовой классификации: отыскав k ближайших данных для входа $\mathbf{x}$, нужно вернуть класс, которому принадлежит больше всего данных среди k.

Алгоритм SVM не получится естественным образом распространить на задачи многоклассовой классификации. Другие алгоритмы работают намного эффективнее, когда определены для двух классов. Что делать, если требуется решить задачу многоклассовой классификации, но алгоритм обучения поддерживает только бинарную классификацию? В таких случаях часто используется стратегия, которая называется «**один против всех**». Идея состоит в том, чтобы преобразовать задачу многоклассовой классификации в C задач бинарной классификации и построить C бинарных классификаторов. Например, если есть три класса $y \in \{1, 2, 3\}$, нужно создать копии исходных наборов данных и модифицировать их. В первой копии заменить на 0 все метки, не равные 1. Во второй копии заменить на 0 все метки, не равные 2. В третьей копии заменить на 0 все метки, не равные 3. После этого останется только решить три задачи бинарной классификации и обучить модели различать метки 1 и 0, 2 и 0 и 3 и 0.

После получения трех моделей для классификации нового входного вектора $\mathbf{x}$, они применяются к входным данным и дают три прогноза. После этого остается только выбрать прогноз принадлежности к ненулевому классу, который является *наиболее достоверным*. Как вы помните, модель логистической регрессии возвращает не метку, а оценку (от 0 до 1), которую можно интерпретировать как вероятность, что

метка является положительной. Этот показатель можно также интерпретировать как достоверность прогноза. В SVM аналогом достоверности является расстояние d от входа $\mathbf{x}$ до границы решения:

$$d \overset{\text{def}}{=} \frac{\mathbf{w}^{*}\mathbf{x} + b^{*}}{\|w\|}.$$

Чем больше расстояние, тем достовернее прогноз. Большинство алгоритмов обучения можно либо естественным путем преобразовать для многоклассового случая, либо с их помощью получить результат, который затем использовать в стратегии «один против всех».

7.3. Одноклассовая классификация

Иногда в наличии имеются только данные одного класса и нужно обучить модель, которая будет отличать данные этого класса от всех остальных данных.

Одноклассовая классификация, также известная как **унарная классификация**, или **моделирование класса**, решает задачу идентификации объектов определенного класса среди всех объектов через обучение на наборе, содержащем только объекты этого класса. Эта задача сложнее и отличается от традиционной задачи классификации, целью которой является выявление различий между двумя или более классами с помощью обучающего набора, содержащего объекты всех классов. Типичным примером задачи одноклассовой классификации может служить классификация допустимого трафика в защищенной компьютерной сети. В этом сценарии обычно имеется очень немного примеров трафика, порождаемого атакующим злоумышленником, если такие примеры вообще есть. Зато примеров допустимого трафика часто сколько угодно. Алгоритмы обучения одноклассовой классификации используются для обнаружения выбросов, аномалий и новых данных.

Есть несколько алгоритмов обучения одноклассовой классификации. На практике наиболее широко используются **одноклассовые** версии алгоритма **Гаусса**, ***k* средних**, **kNN** и **SVM**.

Идея одноклассового алгоритма Гаусса состоит в моделировании данных, как если бы они были получены из распределения Гаусса, точнее, из *многомерного нормального распределения* (Multivariate Normal Distribution, MND). Функция плотности вероятности (probability density unction, pdf) для MND задается следующим уравнением:

$$f_{\mu,\Sigma}(\mathbf{x}) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}{\sqrt{(2\pi)^{D}|\Sigma|}},$$

где $f_{\mu,\Sigma}(\mathbf{x})$ возвращает плотность вероятности, соответствующую входному вектору признаков $\mathbf{x}$. Плотность вероятности можно интерпретировать как достоверность, что образец $\mathbf{x}$ был взят из распределения, которое мы смоделировали как MND. Значения $\boldsymbol{\mu}$ (вектор) и $\boldsymbol{\Sigma}$ (матрица) являются параметрами, которые требуется определить по обучающим данным. Критерий **максимального правдоподобия** (аналогичный тому, что используется в задаче логистической регрессии) оптимизирован для поиска оптимальных значений этих двух параметров. $|\boldsymbol{\Sigma}| \stackrel{\text{def}}{=} \det \boldsymbol{\Sigma}$ — *определитель* матрицы $\boldsymbol{\Sigma}$; обозначение $\boldsymbol{\Sigma}^{-1}$ означает матрицу, *обратную* матрице $\boldsymbol{\Sigma}$.

Если вы не знакомы с терминами *определитель* и *обратная матрица*, не беспокойтесь. Это стандартные операции над векторами и матрицами из области математики — *теории матриц*. Если у вас появится желание узнать, что это такое, почитайте «Википедию», где хорошо объясняются эти понятия.

На практике числа в векторе $\boldsymbol{\mu}$ определяют место, где находится центр кривой гауссова распределения, а числа в $\boldsymbol{\Sigma}$ определяют форму кривой. На рис. 7.2 показан пример гауссовой модели для случая с обучающим набором, состоящим из двумерных векторов признаков.

После построения модели и определения параметров $\boldsymbol{\mu}$ и $\boldsymbol{\Sigma}$ по обучающим данным спрогнозировать вероятность каждого входа $\mathbf{x}$ можно с помощью $f_{\mu,\Sigma}(\mathbf{x})$. Если полученная вероятность выше определенного порога, мы предсказываем, что образец принадлежит нашему классу; иначе он классифицируется как аномальный. Значение порога определяется экспериментально или с использованием «обоснованного предположения».

Когда данные имеют более сложную форму, можно использовать более сложный алгоритм, состоящий из нескольких гауссовых моделей (называется смесью гауссовых распределений). В этом случае по данным определяется большее число параметров: по одному $\boldsymbol{\mu}$ и $\boldsymbol{\Sigma}$ для каждого гауссова распределения, а также параметры, управляющие объединением нескольких гауссовых моделей в одно значение плотности распределения вероятности. В главе 9 мы рассмотрим применение смеси гауссовых распределений к задаче кластеризации.

Одноклассовые версии k средних и kNN основаны на принципах, схожих с одноклассовой версией алгоритма Гаусса: создается некоторая модель данных и затем определяется порог принятия решения о сходстве нового вектора признаков с другими образцами, согласно модели. В первом случае все обучающие примеры группируются с использованием алгоритма кластеризации k **средних**, и, когда появляется новый образец $\mathbf{x}$, расстояние $d(\mathbf{x})$ вычисляется как минимальное расстояние между $\mathbf{x}$ и центром каждого кластера. Если $d(\mathbf{x})$ меньше определенного порога, значит, $\mathbf{x}$ принадлежит этому классу.

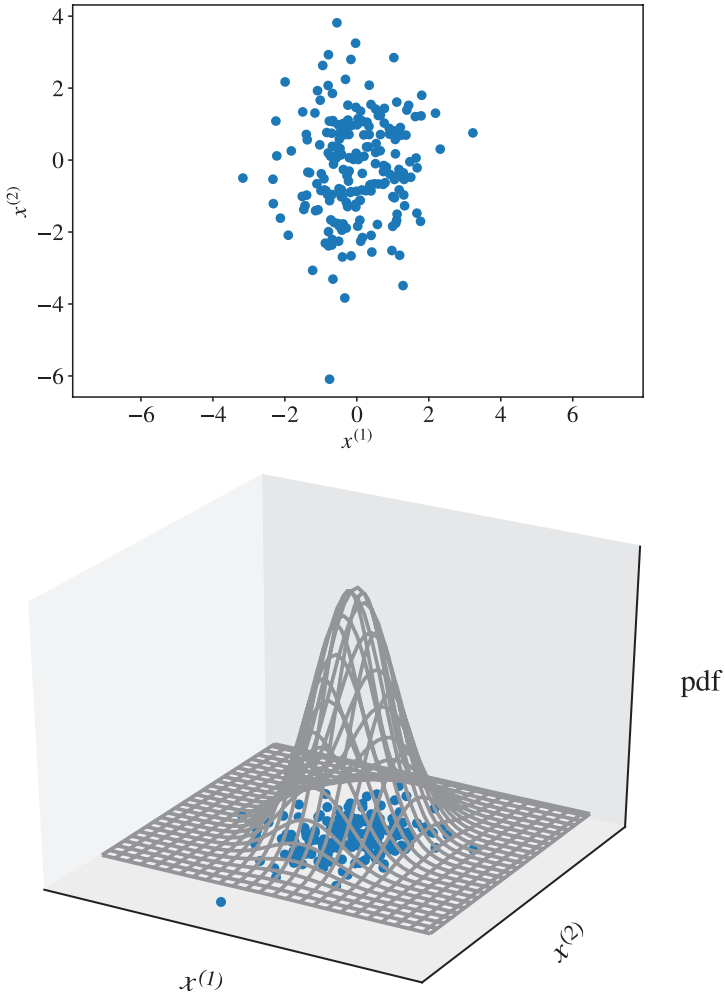


Рис. 7.2. Решение задачи одноклассовой классификации с использованием одноклассовой версии метода Гаусса. Сверху: двумерные векторы признаков. Снизу: кривая многомерного нормального распределения, которая максимизирует правдоподобие данных сверху

Одноклассовая версия SVM, в зависимости от формулировки задачи, пытается либо 1) отделить гиперплоскостью все обучающие данные от начала координат (в пространстве признаков) и максимизировать расстояние от этой гиперплоскости до начала координат, либо 2) определить сферическую границу вокруг данных, минимизируя объем этой гиперсферы. Я оставляю изучение одноклассовых версий алгоритмов kNN, *k* средних и SVM как самостоятельное упражнение.

7.4. Классификация с многими метками

Иногда для описания образца из набора данных подходит более одной метки. В данном случае речь идет о **классификации с многими метками**.

Например, для описания изображения на рис. 7.3 можем использовать одновременно несколько меток: «хвойный лес», «горы», «дорога».

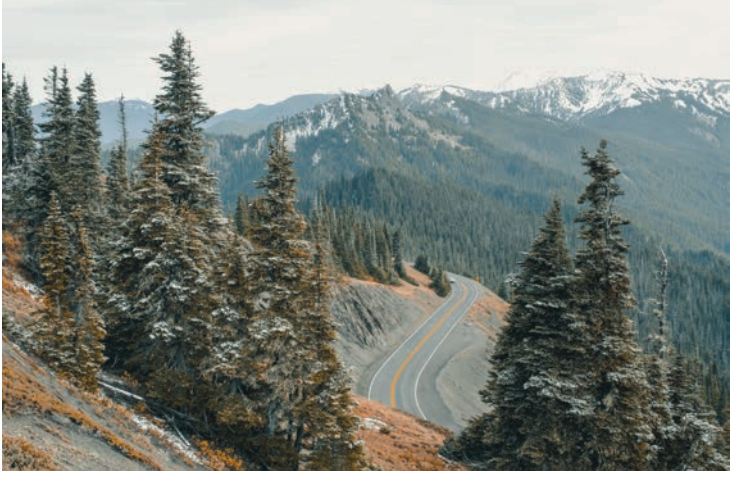


Рис. 7.3. Изображение с метками «хвойный лес», «горы» и «дорога». Фотограф Кейт Лагадия (Cate Lagadia)

Если число возможных значений для меток велико, но все они имеют одинаковую природу, как теги, каждый размеченный образец можно преобразовать в несколько размеченных данных, по одному для каждой метки. Все эти новые данные будут иметь одинаковые векторы признаков и только одну метку. В результате задача превращается в задачу многоклассовой классификации. Решить ее можно, используя стратегию «один против всех». Единственное отличие от обычной задачи многоклассовой классификации заключается в появлении нового гиперпараметра: порога. Если оценка подобия для какой-то метки выше порогового значения, эта метка присваивается входному вектору признаков. В этом сценарии одному вектору признаков может быть присвоено несколько меток. Значение порога выбирается с использованием контрольного набора.

Для решения задачи классификации с многими метками аналогично можно применять алгоритмы, которые естественным образом преобразуются в многоклассовые (деревья решений, логистическая регрессия, нейронные сети и др.). Они возвращают оценку для каждого класса, поэтому мы можем определить порог

и затем присвоить одному вектору признаков несколько меток, для которых оценка близости превышает этот порог.

Нейронные сети можно естественным образом обучить классификации с многими метками, используя в качестве функции стоимости **бинарную перекрестную энтропию** (binary cross-entropy). Выходной слой нейронной сети в этом случае имеет по одному узлу на метку. Каждый узел в выходном слое имеет сигмоидную функцию активации. Соответственно, каждая метка l является бинарной $\left(y_{i,l} \in \{0,1\}\right)$, где $l = 1, ..., L$ и $i = 1, ..., N$. Бинарная перекрестная энтропия определяет вероятность $\hat{y}_{i,l}$, что образец $\mathbf{x}_i$ имеет метку l, определяется как $-\left(y_{i,l}\ln\left(\hat{y}_{i,l}\right)+\left(1-y_{i,l}\right)\ln\left(1-\hat{y}_{i,l}\right)\right)$.

Критерий минимизации — простое среднее значение всех членов бинарной перекрестной энтропии во всех обучающих образцах и всех их метках.

В случаях, когда число возможных значений меток невелико, можно попробовать преобразовать задачу классификации с многими метками в задачу многоклассовой классификации. Представьте следующую задачу. Требуется присвоить изображениям метки двух типов. Метки первого типа могут иметь два возможных значения: {*фото*, *живопись*}; метки второго типа могут иметь три возможных значения: {*портрет*, *пейзаж*, *другое*}. Для каждой комбинации двух исходных классов можно создать новый фиктивный класс, например:

Фиктивный класс	Истинный класс 1	Истинный класс 2
1	фото	портрет
2	фото	пейзаж
3	фото	другое
4	живопись	портрет
5	живопись	пейзаж
6	живопись	другое

Теперь мы имеем те же самые размеченные данные, но заменили набор истинных меток одной фиктивной меткой со значениями от 1 до 6. На практике такой подход дает хорошие результаты, когда возможных комбинаций классов не слишком много. В противном случае необходимо использовать гораздо больше данных для обучения, чтобы компенсировать увеличение набора классов.

Основное преимущество этого последнего подхода в том, что метки остаются коррелированными, в отличие от методов, описанных выше, которые предсказывают каждую метку независимо друг от друга. Во многих задачах корреляция между метками может быть существенным фактором. Например, представьте, что нужно классифицировать

электронную почту как *спам* и *не_спам*, и одновременно как *обычная* и *важная*. Вы наверняка пожелали бы исключить такие прогнозы, как [*спам*, *важная*].

7.5. Обучение ансамбля

Фундаментальные алгоритмы, которые мы рассмотрели в главе 3, имеют свои ограничения. Из-за простоты иногда они не могут создать модель, достаточно эффективную для вашей задачи. В таких случаях можно попробовать использовать глубокие нейронные сети. Однако на практике глубокие нейронные сети требуют значительного объема размеченных данных, которых у вас может не быть. Другой способ повысить эффективность простых алгоритмов обучения — использовать **обучение ансамбля**.

Обучение ансамбля — это парадигма обучения, которая основана на обучении не одной сверхправильной модели, а большого числа моделей с низкой правильностью и объединении прогнозов, данных этими *слабыми* моделями, для получения более правильной **метамодели**.

Модели с низкой правильностью обычно обучаются **слабыми алгоритмами обучения**, которые не способны обучать сложные модели и поэтому показывают высокую скорость работы на этапах обучения и прогнозирования. Наиболее часто в роли слабого алгоритма используется алгоритм обучения дерева решений, который обычно прекращает разбивать обучающий набор после нескольких итераций. В результате получаются мелкие и не очень правильные деревья, но, как гласит идея обучения ансамбля, если деревья не идентичны и каждое дерево хотя бы немного лучше случайного угадывания, мы можем получить высокую правильность, объединив большое количество таких деревьев.

Чтобы получить окончательный прогноз для входа **x**, прогнозы всех слабых моделей объединяются с использованием некоторого метода взвешенного голосования. Конкретная форма взвешивания голосов зависит от алгоритма, но сама суть не зависит от него: если по совокупности слабые модели предсказывают, что электронное письмо является спамом, мы присваиваем образцу **x** метку *спам*.

Двумя основными методами обучения ансамблей являются **бустинг** (boosting — форсирование) и **бэггинг** (bagging — агрегирование)[1].

7.5.1. Бустинг и бэггинг

Метод бустинга заключается в использовании исходных обучающих данных и итеративного создания нескольких моделей с применением слабого алгоритма.

[1] Переводы терминов boosting и bagging неточные и не прижившиеся.

Каждая новая модель отличается от предыдущих тем, что, конструируя ее, слабый алгоритм пытается «исправить» ошибки, допускаемые предыдущими моделями. Окончательная **ансамблевая модель** представляет собой комбинацию этих многочисленных слабых моделей, построенных итеративно.

Суть бэггинга заключается в создании множества «копий» обучающих данных (каждая копия немного отличается от других) и последующем применении слабого алгоритма к каждой копии с целью получить несколько слабых моделей, а затем объединить их. Широко используемым и эффективным алгоритмом машинного обучения, основанным на идее бэггинга, является **случайный лес**.

7.5.2. Случайный лес

«Классический» алгоритм бэггинга работает следующим образом. Из имеющегося обучающего набора создается B случайных выборок $\mathcal{S}_b$ (для каждого $b = 1, ..., B$) и на основе каждой выборки $\mathcal{S}_b$ строится модель f_b дерева решений. Чтобы получить выборку $\mathcal{S}_b$ для некоторого b, производится **выборка с заменой**. То есть сначала создается пустая выборка, а затем из обучающего набора выбирается случайный образец, и его точная копия помещается в $\mathcal{S}_b$, при этом сам образец остается в исходном обучающем наборе. Выбор данных продолжается, пока не выполнится условие $|\mathcal{S}_b| = \mathrm{N}$.

В результате обучения получается B деревьев решений. Прогноз для нового образца $\mathbf{x}$, в случае регрессии, определяется как среднее из B прогнозов

$$y \leftarrow \hat{f}(\mathbf{x}) \overset{\text{def}}{=} \frac{1}{B}\sum_{b=1}^{B} f_b(\mathbf{x}),$$

или большинством голосов в случае классификации.

Случайный лес имеет только одно отличие от классического бэггинга. Он использует модифицированный алгоритм обучения дерева, который при каждом расщеплении в процессе обучения проверяет случайное подмножество признаков. Это делается с целью устранить корреляцию между деревьями: если один или несколько признаков имеют большую прогнозирующую способность, многие деревья будут выбирать их для расщепления данных. Это приведет к появлению в «лесу» большого числа коррелированных деревьев. Корреляция по признакам с большой прогнозирующей способностью препятствует повышению точности предсказания. Высокая эффективность ансамбля моделей объясняется тем, что хорошие модели, вероятнее всего, согласятся с одним и тем же прогнозом, а плохие — не согласятся и дадут разные прогнозы. Корреляция сделает плохие модели более склонными к согласию, что исказит картину голосования или повлияет на среднее значение.

Наиболее важными гиперпараметрами для настройки являются количество деревьев B и размер случайного подмножества признаков, которые необходимо учитывать при каждом расщеплении.

Случайный лес — один из наиболее широко используемых алгоритмов обучения ансамблей. Чем обусловлена его эффективность? Причина в том, что, используя несколько выборок из исходного набора данных, мы уменьшаем **дисперсию** конечной модели. Помните, что низкая дисперсия означает слабую предрасположенность к **переобучению**. Переобучение происходит, когда модель пытается объяснить небольшие вариации в наборе данных, потому что набор данных является лишь небольшой выборкой из всех возможных примеров явления, которое мы пытаемся смоделировать. В случае неудачного подхода к формированию обучающего набора в него могут попасть некоторые нежелательные (но неизбежные) артефакты: шум, аномальные и чрезмерно или недостаточно представительные данные. Создавая несколько случайных выборок с заменой обучающего набора, мы уменьшаем влияние этих артефактов.

7.5.3. Градиентный бустинг

Другой эффективный алгоритм обучения ансамблей, основанный на идее бустинга, — **градиентный бустинг**. Сначала рассмотрим применение градиентного бустинга в регрессии. Построение эффективной регрессионной модели мы начнем с константной модели $f = f_0$ (как мы это делали в ID3):

$$f = f_0(\mathbf{x}) \overset{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N} y_i.$$

Затем изменим метки во всех образцах $i = 1, ..., N$ в обучающем наборе:

$$\hat{y}_i \leftarrow y_i - f(\mathbf{x}_i), \tag{7.2}$$

где $\hat{y}_i$ называется **остатком** и является новой меткой образца $\mathbf{x}_i$.

Теперь используем модифицированный обучающий набор с остатками вместо оригинальных меток, чтобы построить новую модель дерева решений, f_1. Модель бустинга теперь определяется как $f \overset{\text{def}}{=} f_0 + \alpha f_1$, где α — скорость обучения (гиперпараметр).

Затем пересчитаем остатки с использованием уравнения 7.2, заменим метки в обучающих данных еще раз, обучим новую модель дерева решений f_2, переопределим модель бустинга как $f \overset{\text{def}}{=} f_0 + \alpha f_1 + \alpha f_2$ и будем повторять процесс, пока не объединим предопределенное максимальное число M деревьев.

Давайте интуитивно разберемся в том, что тут происходит. Вычисляя остатки, мы определяем, насколько хорошо (или плохо) предсказывается цель каждого обучающего образца текущей моделью f. Затем мы обучим другое дерево для исправления ошибок текущей модели (именно поэтому мы используем остатки вместо фактических меток) и добавим новое дерево в существующую модель с некоторым весом α. В результате каждое новое дерево, добавленное в модель, частично исправляет ошибки, допущенные предыдущими деревьями. Процесс продолжается, пока не будет объединено максимальное количество M (еще один гиперпараметр) деревьев.

Теперь попробуем ответить на вопрос, почему этот алгоритм называется *градиентным* бустингом. В градиентном бустинге мы не вычисляем градиент, в отличие от того, что мы делали в главе 4, решая задачу линейной регрессии. Чтобы увидеть сходство между градиентным бустингом и градиентным спуском, вспомните, для чего мы вычисляли градиент в линейной регрессии: чтобы узнать направление изменения значений параметров для минимизации функции стоимости MSE. Градиент показывает направление, но не показывает, как далеко идти в этом направлении, поэтому в каждой итерации мы делали небольшой шаг, а затем вновь определяли направление. То же происходит в градиентном бустинге, только вместо непосредственного вычисления градиента мы используем его оценку в форме остатков: они показывают, как следует скорректировать модель, чтобы уменьшить ошибку (остаток).

В градиентном бустинге доступны для настройки три основных гиперпараметра: количество деревьев, скорость обучения и глубина деревьев. Все три влияют на точность модели. Глубина деревьев также влияет на скорость обучения и прогнозирования: чем меньше глубина, тем быстрее.

Можно показать, что обучение по остаткам оптимизирует общую модель f для критерия среднеквадратичной ошибки. Здесь можно заметить отличие от бэггинга: бустинг уменьшает смещение (или недообученность) вместо дисперсии. Как результат, бустинг подвержен переобучению. Однако, настраивая глубину и количество деревьев, можно в значительной степени избежать переобучения.

В задачах классификации градиентный бустинг применяется аналогично, но шаги немного отличаются. Рассмотрим случай бинарной классификации. Предположим, есть M деревьев решений регрессии. По аналогии с логистической регрессией прогноз ансамбля деревьев решений моделируется с использованием сигмоидной функции:

$$\Pr\left(y = 1 \middle| \mathbf{x},\ f\right) \stackrel{\text{def}}{=} \frac{1}{1 + e^{-f(\mathbf{x})}},$$

где $f\left(\mathbf{x}\right) \stackrel{\text{def}}{=} \sum_{m=1}^{M} f_m\left(\mathbf{x}\right)$ и f_m — дерево регрессии.

И снова, как в логистической регрессии, при попытке найти модель f, максимизирующую $L_f = \sum_{i=1}^{N} \ln\left[\Pr\left(y_i = 1 | \mathbf{x}_i, f\right)\right]$, применяется принцип максимального правдоподобия. Точно так же, чтобы избежать числового переполнения, мы максимизируем сумму логарифмов правдоподобий, а не произведение правдоподобий.

Алгоритм начинает работу с начальной константной модели $f = f_0 = \frac{p}{1-p}$, где $p = \frac{1}{N}\sum_{i=1}^{N} y_i$. (Можно показать, что такая инициализация оптимальна для сигмоидной функции.) Затем в каждой итерации m в модель добавляется новое дерево f_m. Чтобы найти наилучшее дерево f_m, сначала вычисляется частная производная g_i текущей модели для каждого $i = 1, ..., N$:

$$g_i = \frac{dL_f}{df},$$

где f — модель ансамблевого классификатора, построенная на предыдущей итерации $m - 1$. Чтобы вычислить g_i, нужно найти производные от $\ln\left[\Pr\left(y_i = 1 | \mathbf{x}_i, f\right)\right]$ по f для всех i. Обратите внимание, что $\ln\left[\Pr\left(y_i = 1 | \mathbf{x}_i, f\right)\right] \stackrel{\text{def}}{=} \ln\left[\frac{1}{1 + e^{-f(\mathbf{x}_i)}}\right]$. Производная по f правого члена в предыдущем уравнении равна

$$\frac{1}{e^{f(\mathbf{x}_i)} + 1}.$$

Затем выполняется преобразование обучающего набора заменой исходной метки y_i соответствующей частной производной g_i, и на основе преобразованного обучающего набора строится новое дерево f_m. Далее определяется оптимальный шаг обновления ρ_m как:

$$\rho_m \leftarrow \arg\max_{\rho} L_{f + \rho f m}.$$

В конце итерации m мы обновляем модель ансамбля f, добавляя новое дерево f_m:

$$f \leftarrow f + \alpha \rho_m f_m.$$

Итерации продолжаются, пока не выполнится условие $m = M$, после чего обучение прекращается и в результате получается модель ансамбля f.

Градиентный бустинг является одним из самых мощных алгоритмов машинного обучения. Не только потому, что создает очень точные модели, но и потому, что способен обрабатывать огромные наборы данных с миллионами данных и признаков. Как правило, он превосходит в точности случайный лес, но из-за последовательной природы может обучаться намного медленнее.

7.6. Обучение маркировке последовательностей

Последовательность — один из наиболее распространенных типов структурированных данных. Мы общаемся, используя последовательности слов и предложений, мы выполняем действия в определенной последовательности, наши гены, музыка, которую мы слушаем, видеофильмы, которые смотрим, наши наблюдения за непрерывным процессом, таким как движение автомобиля или изменение цен акций на бирже, — все это последовательности.

Маркировка последовательностей — это задача автоматического назначения меток элементам последовательности. Обучающим набором для задачи маркировки последовательностей является пара списков (**X**, **Y**), где **X** — список векторов признаков, по одному для каждого шага во времени, **Y** — список меток той же длины. Например, **X** может содержать слова в предложении, такие как [«большой», «красивый», «автомобиль»], а **Y** может содержать соответствующие части речи, такие как [«прилагательное», «прилагательное», «существительное»]). Более формально: для образца *i*, $\mathbf{X}_i = \left[\mathbf{x}_i^1, \mathbf{x}_i^2, \ldots, \mathbf{x}_i^{size_i}\right]$, где $size_i$ — длина последовательности в образце *i*, $\mathbf{Y}_i = \left[y_i^1, y_i^2, \ldots, y_i^{size_i}\right]$ и $y_i \in \{1, 2, \ldots, C\}$.

Вы уже знаете, что для маркировки последовательностей можно использовать RNN. На каждом временном шаге *t* она читает входной вектор признаков $\mathbf{x}_i^{(t)}$, а последний рекуррентный слой выводит метку $y_{last}^{(t)}$ (в случае бинарной маркировки) или $\mathbf{y}_{last}^{(t)}$ (в случае многоклассовой маркировки или маркировки с многими метками).

Однако RNN — не единственная возможная модель для маркировки последовательностей. Модель, которая называется **условные случайные поля** (Conditional Random Fields, CRF), является очень эффективной альтернативой, которая часто дает хорошие результаты для векторов признаков, имеющих много информативных признаков. Например, представьте, что нам поставили задачу реализовать **извлечение именованных сущностей**, и мы решили построить модель, которая маркировала бы каждое слово в предложении, например «Я еду в Санкт-Петербург», одним из следующих классов: {*местоположение*, *имя*, *название_компании*, *другое*}. Если наши векторы признаков (представляющие слова) содержат такие бинарные признаки, как «слово начинается с заглавной буквы» и «слово присутствует в списке местоположений», такие признаки будут очень информативными и помогут классифицировать слова *Санкт* и *Петербург* как местоположение (так же, как и дефис между ними).

Известно, что определение признаков вручную — трудоемкий процесс, требующий значительных знаний в предметной области.

CRF — очень интересная модель, и ее можно рассматривать как обобщение логистической регрессии на последовательности. Однако на практике для маркировки последовательностей лучше использовать глубокие двунаправленные вентильные RNN. CRF значительно медленнее в обучении, что затрудняет ее применение к большим обучающим наборам (с сотнями тысяч данных). Кроме того, там, где имеются большие обучающие наборы, глубокие нейронные сети выглядят особенно привлекательно.

7.7. Обучение преобразованию последовательностей в последовательности

Обучение преобразованию последовательностей в последовательности (sequence-to-sequence, сокращенно seq2seq) является обобщением задачи маркировки последовательностей. В seq2seq $\mathbf{X}_i$ и $\mathbf{X}_i$ могут иметь разную длину. Модели seq2seq нашли применение в машинном переводе (где, например, вход — это предложение на английском языке, а выход — соответствующее предложение на русском), диалоговых интерфейсах (где вход — это вопрос, введенный пользователем, а выход — ответ машины), обобщении текста, исправлении орфографических ошибок и многих других сферах.

Многие, но не все задачи обучения seq2seq в настоящее время лучше всего решаются с помощью нейронных сетей. Все сетевые архитектуры, используемые в seq2seq, состоят из двух частей: **кодировщика** и **декодировщика**.

Кодировщик — это нейронная сеть, принимающая последовательность. Это может быть рекуррентная сеть (RNN), сверточная (CNN) или сеть с какой-то другой архитектурой. Роль кодировщика состоит в том, чтобы прочитать входные данные и сгенерировать некое состояние (аналогичное состоянию в RNN), которое можно рассматривать как числовое представление *смысла* входных данных, с которым может работать машина. Смысл изображения, текста, видеоролика или чего-то еще обычно представляется как вектор или матрица с действительными числами. Этот вектор (или матрица) на жаргоне машинного обучения называется **вложением** (embedding) входных данных.

Декодировщик — это другая нейронная сеть, которая принимает на входе вложение и генерирует выходную последовательность. Как вы уже могли догадаться, это вложение создается кодировщиком. Чтобы создать выходную последовательность, декодировщик берет вектор признаков $\mathbf{x}^{(0)}$ *начала последовательности* (обычно все нули), генерирует первый выход $\mathbf{y}^{(1)}$, обновляет свое состояние, комбинируя вложение и вход $\mathbf{x}^{(0)}$, а затем использует выход $\mathbf{y}^{(1)}$ в качестве своего следующего входа $\mathbf{x}^{(1)}$. Для простоты будем считать, что $\mathbf{y}^{(t)}$ имеет ту же размерность, что и $\mathbf{x}^{(t)}$,

хотя это необязательно. Как мы видели в главе 6, каждый слой RNN может генерировать много выходов сразу: один может использоваться, чтобы сгенерировать метку $\mathbf{y}^{(t)}$, а другой, с другой размерностью, может использоваться как $\mathbf{x}^{(t)}$.

Кодировщик и декодировщик обучаются одновременно. Ошибки на выходе декодировщика распространяются в кодировщик посредством механизма обратного распространения.

На рис. 7.4 изображена традиционная архитектура seq2seq. Для получения более точных прогнозов можно использовать архитектуры с механизмом **внимания**. Механизм внимания реализуется с помощью дополнительного набора параметров, которые объединяют некоторую информацию, полученную от кодировщика (в RNN эта информация представлена списком векторов состояния последнего рекуррентного уровня из всех временных шагов кодировщика), с текущим состоянием декодировщика для получения метки. Это обеспечивает даже лучшую сохранность долгосрочных зависимостей, в сравнении с применением вентильных узлов и двунаправленных RNN.

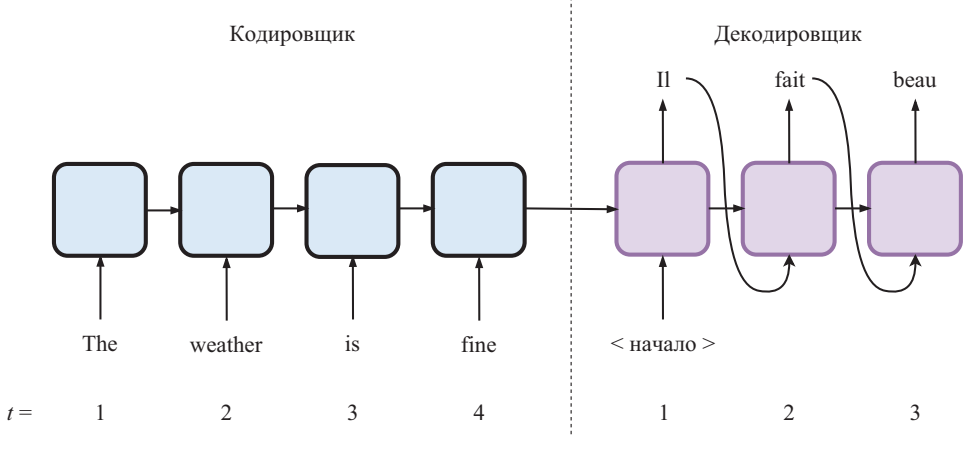


Рис. 7.4. Традиционная архитектура для обучения seq2seq. Векторное представление, которое обычно определяется состоянием последнего слоя кодировщика, передается из левой подсети в правую

Архитектура seq2seq с механизмом внимания показана на рис. 7.5.

Обучение seq2seq — относительно новая область исследований. Постоянно обнаруживаются и публикуются новые архитектуры сетей. Обучение таких архитектур может быть сложной задачей, потому что количество настраиваемых гиперпара-

метров и других архитектурных решений может быть огромным. Дополнительные сведения, ссылки на руководства и примеры кода ищите в вики для книги.

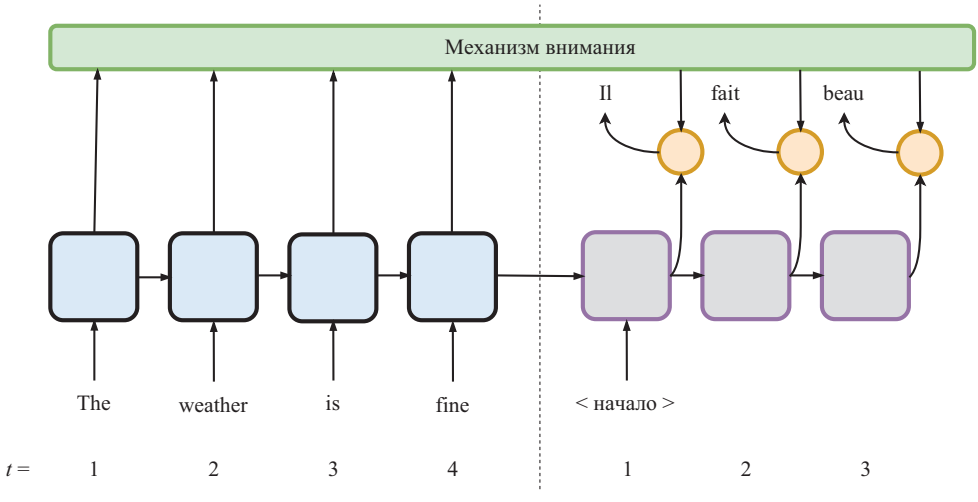


Рис. 7.5. Архитектура seq2seq с механизмом внимания

7.8. Активное обучение

Активное обучение — интересная парадигма обучения с учителем. Она обычно применяется, когда получение размеченных данных обходится слишком дорого. Это часто имеет место в медицинской или финансовой областях, где для маркировки данных о пациентах или клиентах может потребоваться привлекать экспертов. Идея состоит в том, чтобы начать обучение с относительно небольшого количества размеченных и большого количества неразмеченных образцов, а затем маркировать только те образцы, которые в наибольшей степени способствуют повышению качества модели.

Есть несколько стратегий активного обучения. Здесь мы рассмотрим следующие две:

1. На основе плотности и неопределенности данных.
2. На основе метода опорных векторов.

Первая стратегия применяет текущую модель f, обученную с использованием имеющихся размеченных данных, к каждому из остальных неразмеченных данных (или, чтобы сэкономить время, к некоторой случайной выборке из них). Для каждого неразмеченного образца $\mathbf{x}$ вычисляется оценка его важности:

плотность $(\mathbf{x}) \cdot$ неопределенность$_f(\mathbf{x})$. Плотность отражает количество ближайших данных, окружающих **x**, а *неопределенность*$_f$**(x)** — насколько неопределенным является прогноз модели *f* для **x**. В бинарной классификации с сигмоидной функцией чем ближе прогнозная оценка к 0.5, тем неопределеннее прогноз. В SVM: чем ближе образец к границе решения, тем неопределеннее прогноз.

В многоклассовой классификации в качестве меры неопределенности можно использовать *энтропию*:

$$\mathrm{H}_f(\mathbf{x}) = -\sum_{c=1}^{C} \Pr\left(y^{(c)}; f(\mathbf{x})\right) \ln\left[\Pr\left(y^{(c)}; f(\mathbf{x})\right)\right],$$

где $\Pr\left(y^{(c)}; f(\mathbf{x})\right)$ — оценка вероятности, с какой модель *f* отнесет образец **x** к классу $y^{(c)}$. Как видите, если для каждого $y^{(c)}$ выполняется условие $f\left(y^{(c)}\right) = \frac{1}{C}$, тогда модель является полностью неопределенной, а энтропия имеет максимальное значение 1; с другой стороны, если для некоторого $y^{(c)}$ выполняется условие $f\left(y^{(c)}\right) = 1$, тогда модель полностью определена относительно класса $y^{(c)}$, а энтропия имеет минимальное значение 0.

Плотность для образца **x** можно получить, взяв среднее значение расстояний от **x** до каждого из *k* ближайших к нему соседей (где *k* является гиперпараметром).

После вычисления оценок важности для всех неразмеченных данных выбираем наибольшую и просим эксперта присвоить метку соответствующему образцу. Затем добавляем новый размеченный образец в обучающий набор, строим модель заново и продолжаем процесс, пока не будет удовлетворен некоторый критерий остановки. Критерий остановки можно выбрать заранее (например, максимальное количество обращений к эксперту, исходя из выделенного бюджета) или положиться на некоторую метрику, определяющую качество работы модели.

Стратегия активного обучения с использованием метода опорных векторов заключается в построении модели SVM с использованием размеченных данных, после чего эксперту предлагается присвоить метку неразмеченному образцу, находящемуся ближе всех к гиперплоскости, разделяющей два класса. Идея в том, что, если образец ближе всех лежит к гиперплоскости, значит, он наименее определен и его маркировка внесет наибольший вклад в определение точек, через которые проходит истинная (та, которую мы ищем) гиперплоскость.

Некоторые стратегии активного обучения могут включать стоимость обращения к эксперту для маркировки. Другие *учатся* спрашивать мнение эксперта. Стратегия «запрос комитетом» (query by committee) предполагает обучение нескольких мо-

делей с применением разных методов, после чего эксперту предлагается присвоить метку образцу, на котором модели разошлись во мнении больше всего. Некоторые стратегии пытаются выбирать для маркировки такие данные, чтобы максимально уменьшить смещение или дисперсию модели.

7.9. Обучение с частичным привлечением учителя

В обучении с частичным привлечением учителя (Semi-Supervised Learning, SSL) точно так же маркируется небольшая часть набора данных; большинство данных остаются неразмеченными. Цель — использовать большое количество неразмеченных данных для повышения эффективности модели без запроса дополнительных размеченных данных.

В истории науки было несколько попыток решить эту задачу. Но ни одно из решений не получило общего признания и широкого применения на практике. Чаще других упоминается метод SSL, который называется **самообучением**. В этом методе с помощью алгоритма обучения и размеченных данных строится начальная модель. Затем эта модель применяется ко всем неразмеченным примерам и производится их маркировка. Если показатель достоверности прогноза для некоторого неразмеченного примера **x** выше некоторого порогового значения (выбранного экспериментально), этот размеченный пример добавляется в обучающий набор, производится повторное обучение модели, и так повторяется, пока не будет удовлетворен критерий остановки. Критерием остановки, например, может служить отсутствие роста правильности модели в течение последних m итераций.

Метод, описанный выше, может обеспечить некоторое улучшение модели по сравнению с использованием только исходного размеченного набора данных, но обычно это улучшение не особо впечатляет. Кроме того, качество модели может даже снизиться. Это зависит от свойств статистического распределения, из которого взяты данные, которые обычно неизвестны.

С другой стороны, недавние достижения в обучении нейронных сетей привели к довольно впечатляющим результатам. Например, было показано, что для некоторых наборов данных, таких как MNIST (тестовый набор, состоящий из размеченных изображений рукописных цифр от 0 до 9, известный в сфере распознавания образов), модель, обученная с частичным привлечением учителя, показывает почти идеальную эффективность при наличии 10 размеченных образцов на класс (всего 100 размеченных данных). Для сравнения, MNIST содержит 70 000 размеченных образцов (60 000 для обучения и 10 000 для тестирования). Архитектура нейрон-

ной сети, которая достигла такой замечательной производительности, называется **лестничной сетью**. Чтобы понять идею, лежащую в основе лестничных сетей, нужно сначала разобраться с понятием **автокодировщик**.

Автокодировщик — это нейронная сеть прямого распространения с архитектурой кодировщик-декодировщик. Она обучается восстанавливать свой вход. Соответственно обучающий образец — это пара $(\mathbf{x}, \mathbf{x})$. Нам нужно, чтобы, выход $\hat{\mathbf{x}}$ модели $f(\mathbf{x})$ был максимально похож на вход $\mathbf{x}$.

Важно отметить, что сеть автокодировщика напоминает песочные часы, имея узкий слой «горлышка» в середине, который содержит вложение D-мерного входного вектора. Обычно слой вложения имеет гораздо меньше узлов, чем D. Цель декодировщика — восстановить входной вектор признаков из этого вложения. Теоретически достаточно 10 узлов в узком слое, чтобы успешно закодировать изображения MNIST. В типичном автокодировщике, схематически изображенном на рис. 7.6, в роли функции стоимости обычно используется либо среднеквадратическая ошибка (когда признаками могут быть любые числа), либо бинарная перекрестная энтропия (когда признаки имеют бинарную природу и узлы последнего слоя декодировщика имеют сигмоидную функцию активации). Когда используется среднеквадратическая ошибка, она определяется так:

$$\frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{x}_i - f\left(\mathbf{x}_i\right)\right\|^2,$$

где $\left\|\mathbf{x}_i - f\left(\mathbf{x}_i\right)\right\|$ — евклидово расстояние между двумя векторами.

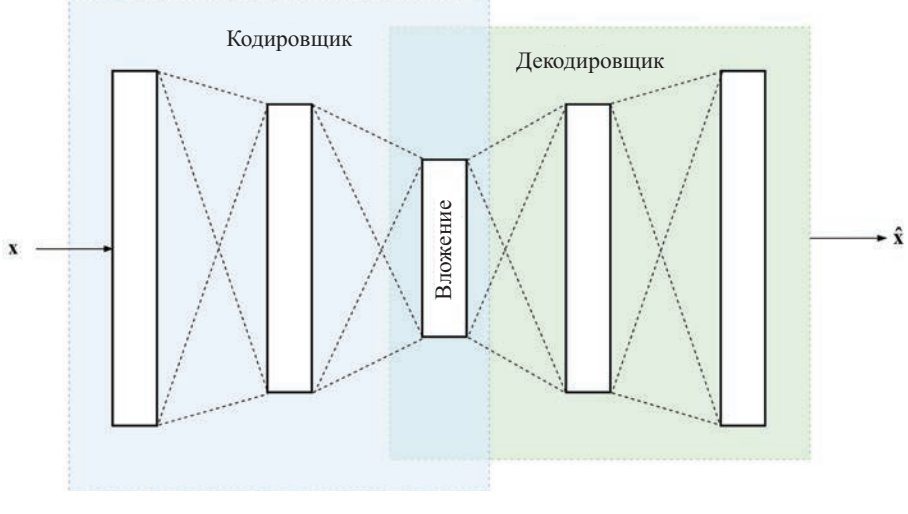


Рис. 7.6. Автокодировщик

Автокодировщик с шумоподавлением искажает левый элемент **x** в обучающем образце (**x**, **x**), добавляя в признаки некоторые случайные возмущения. В случае, когда данные являются черно-белыми изображениями — с пикселами, представленными значениями от 0 до 1, — в каждый признак обычно добавляется **нормальный гауссов шум**. Для каждого признака j во входном векторе **x** значение шума $n^{(j)}$ выбирается из следующего распределения:

$$n^{(j)} \sim \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(-\mu)^2}{2\sigma^2}\right),$$

где символ ~ означает «выбирается из», π — это константа и $\theta \stackrel{\text{def}}{=} [\mu, \sigma]$ — гиперпараметр. Новое искаженное значение признака $x^{(j)}$ получается путем сложения $x^{(j)} + n^{(j)}$.

Лестничная сеть — это усовершенствованный шумоподавляющий автокодировщик. Кодировщик и декодировщик имеют одинаковое количество слоев. Слой узкого горлышка используется непосредственно для прогнозирования метки (с помощью функции активации softmax). Сеть имеет несколько функций стоимости. Для каждого слоя l в кодировщике и соответствующего слоя l в декодировщике применяется функция C_d^l, штрафующая за разность между выходами двух уровней (квадрат евклидова расстояния). Когда во время обучения используется размеченный образец, применяется другая функция стоимости, C_c, которая штрафует за ошибку в прогнозировании метки (отрицательный логарифм правдоподобия). Комбинированная функция стоимости, $C_c + \sum_{l=1}^{L} \lambda_l C_d^l$ (усредненная по всем образцам в пакете), оптимизируется с помощью мини-пакетного стохастического градиентного спуска с обратным распространением. Гиперпараметры λ_l для каждого слоя l определяют компромисс между стоимостью классификации и кодирования-декодирования.

В лестничной сети шумом искажается не только вход, но и выход каждого слоя кодировщика (в процессе обучения). При применении обученной модели к новому входу **x** для предсказания его метки этот вход не искажается.

Существуют и другие методы обучения с частичным привлечением учителя, не связанные с нейронными сетями. Один из них предполагает построение модели с использованием размеченных данных и кластеризацию размеченных и неразмеченных данных вместе с использованием любого метода кластеризации (некоторые из них мы рассмотрим в главе 9). Для каждого нового образца выводится прогноз — метка, присущая большинству данных в кластере, которому он принадлежит.

Еще один метод, называемый S3VM, основан на использовании SVM. При его использовании строится одна модель SVM для каждого возможного способа маркировки неразмеченных данных, а затем выбирается модель с наибольшим зазором. В статье о S3VM (ссылку на которую вы найдете в вики) описывается подход, который позволяет решить эту проблему без фактического перебора всех возможных способов маркировки.

7.10. Обучение с первого раза

Эта глава была бы неполной без упоминания двух других важных парадигм обучения с учителем. Одна из них — **обучение с первого раза** (one-shot learning). В обучении с первого раза, которое обычно применяется для распознавания лиц, требуется построить модель, которая распознает, что две фотографии одного человека действительно представляют одного и того же человека. Если показать модели две фотографии двух разных людей, она должна распознать, что это два разных человека.

Можно попробовать решить эту задачу традиционным путем и построить бинарный классификатор, который принимает два изображения на входе и предсказывает либо истинное значение (если два изображения представляют одного и того же человека), либо ложное (когда два изображения принадлежат разным людям). Однако на практике это приведет к тому, что нейронная сеть окажется в два раза больше типичной нейронной сети, потому что для представления каждого из двух изображений нужна своя подсеть. Обучить такую сеть будет сложно не только из-за ее размера, но и потому, что получить положительные данные гораздо сложнее, чем отрицательные. То есть задача крайне несбалансированная.

Одним из эффективных решений этой задачи является обучение **сиамской нейронной сети** (Siamese Neural Network, SNN). SNN можно реализовать как нейронную сеть любого типа: CNN, RNN или MLP. Сеть принимает изображения по одному; поэтому размер сети не удваивается. Чтобы получить из сети бинарный классификатор «тот_же_человек»/«другой», который принимает на вход только одно изображение за раз, сеть обучается особым образом.

Для обучения SNN мы используем функцию **триплетной потери** (triplet loss). Например, пусть есть три изображения лица: изображение A (для привязки), изображение P (положительное изображение) и изображение N (отрицательное изображение). A и P — два разных изображения одного и того же человека; N — изображение другого человека. Каждый обучающий образец i теперь является триплетом (A_i, P_i, N_i).

Допустим, у нас есть модель нейронной сети *f*, которая принимает изображение лица на входе и выводит его векторное представление. Потеря триплета для образца *i* определяется как

$$\max(\|f(A_i) - f(P_i)\|^2 - \|f(A_i) - f(N_i)\|^2 + \alpha, 0). \quad (7.3)$$

Функция стоимости определяется как среднее потери триплета:

$$\frac{1}{N}\sum_{i=1}^{N}\max\left(\left\|f\left(A_i\right) - f\left(P_i\right)\right\|^2 - \left\|f\left(A_i\right) - f\left(N_i\right)\right\|^2 + \alpha,\ 0\right),$$

где α — положительный гиперпараметр. Очевидно, что значение $\|f(A) - f(P)\|^2$ низко, когда нейронная сеть выводит одинаковые векторные представления для *A* и *P*; значение $\|f(A_i) - f(N_i)\|^2$ высоко, когда векторные представления изображений двух разных людей различны. Если модель работает правильно, тогда член $m = (\|f(A_i) - f(P_i)\|^2 - \|f(A_i) - f(N_i)\|^2$ всегда будет отрицательным, потому что большое значение будет вычитаться из маленького. Увеличивая α, можно еще уменьшить член *m*, чтобы убедиться, что модель научилась надежно распознавать одинаковые и разные лица. Если значение *m* недостаточно мало, то из-за α стоимость получится положительной и параметры модели скорректируются на этапе обратного распространения.

Вместо выбора случайного изображения *N* лучший способ создания триплетов для обучения — использовать текущую модель после нескольких эпох обучения и отыскивать кандидатов на *N*, которые похожи на *A* и *P*, согласно этой модели. Использование случайных данных в качестве *N* значительно замедлит процесс обучения, поскольку нейронная сеть легко будет находить различия между изображениями двух случайных людей, из-за чего средняя потеря триплета в большинстве случаев будет низкой и параметры будут обновляться недостаточно быстро.

Чтобы построить SNN, сначала нужно выбрать архитектуру нейронной сети. Например, для обработки изображений часто выбирается CNN. Чтобы вычислить средние потери триплета в нашем примере, мы последовательно применяем модель к *A*, затем к *P*, затем к *N*, а потом вычисляем потери, используя уравнение 7.3. Процедура повторяется для всех триплетов в партии, а затем вычисляется стоимость; градиентный спуск с обратным распространением распространяет стоимость через сеть и тем самым корректирует ее параметры.

Многие заблуждаются, думая, что для обучения с первого раза нужен только один обучающий образец каждой сущности. В действительности, чтобы модель идентификации получилась точной, необходимо подобрать несколько образцов для каждого человека. Это обучение называется «с первого раза» из-за сферы применения таких моделей: для идентификации по лицу. Например, такую модель

можно использовать для разблокировки телефона. Если модель хорошо обучена, вам достаточно будет хранить в телефоне *только одну свою фотографию*, и она будет уверенно узнавать вас, а также отличать других людей. Имея модель, которая способна определить, принадлежат ли два изображения A и $\hat{A}$ одному и тому же лицу, мы сравниваем $\left\| f(A) - f\left(\hat{A}\right) \right\|^2$ с τ — гиперпараметром.

7.11. Обучение без подготовки

В завершение главы я упомяну метод **обучения без подготовки**. Это относительно новая область исследований, поэтому пока отсутствуют алгоритмы, доказавшие свою практическую ценность. Я только обрисую основную идею и оставлю знакомство с различными алгоритмами вам, как самостоятельное упражнение. В обучении без подготовки (Zero-Shot Learning, ZSL) модель обучается назначать метки объектам. Наиболее часто этот метод обучения применяется для назначения меток изображениям. Однако, в отличие от стандартной классификации, модель должна уметь прогнозировать метки, отсутствующие в обучающих данных. Как такое возможно?

Хитрость заключается в использовании вложений, которые представляют не только входные данные **x**, но и выходные данные y. Представьте, что у нас есть модель, которая для любого слова может сгенерировать вектор вложения, обладающий следующим свойством: если по значению слово y_i похоже на слово y_k, векторы вложения этих двух слов должны быть одинаковыми. Например, если y_i — это *Париж*, а y_k — *Рим*, они будут иметь похожие векторные представления; с другой стороны, если y_k — это *картошка*, векторы вложения слов y_i и y_k должны быть разными. Такие векторы вложения называют **вложениями слов** (word embeddings) и обычно сравниваются с использованием мер косинусного сходства[1].

Вложения слов обладают важным свойством: каждое измерение во вложении представляет определенную особенность значения слова. Например, если вложение слов имеет четыре измерения (обычно их намного больше, от 50 до 300), эти измерения могут представлять такие особенности значения, как *животность*, *абстрактность*, *кислость* и *желтизна* (да, выглядит забавно, но это всего лишь пример). Согласно такому определению, слово *пчела* будет иметь такое вложение: [1, 0, 0, 1]. Слово *желтый* — такое: [0, 1, 0, 1]. Слово *единорог* — такое: [1, 1, 0, 0]. Значения для каждого вложения получаются с помощью специальной процедуры обучения, применяемой к обширному текстовому корпусу.

[1] В главе 10 я покажу, как получать вложения слов из данных.

Теперь, продолжая нашу задачу классификации, заменим метку y_i каждого образца i в обучающем наборе ее вложением слова и обучим модель классификации с многими метками, которая предсказывает вложения слов. Чтобы получить метку для нового образца $\mathbf{x}$, нужно применить модель f к $\mathbf{x}$, получить векторное представление $\hat{y}$, а затем найти среди всех слов те, вложения которых наиболее похожи на $\hat{y}$, используя косинусное сходство.

Почему это работает? Возьмем, к примеру, зебру. Это млекопитающее белого цвета с полосами. Теперь возьмем рыбу-клоуна: это не млекопитающее, имеет оранжевую окраску с полосами. Теперь возьмем тигра: это млекопитающее оранжевого цвета с полосами. Если эти три признака присутствуют во вложениях слов, CNN научится обнаруживать эти признаки на изображениях. Даже если метка *тигр* отсутствовала в обучающих данных, но другие объекты, включая зебру и рыбу-клоуна, присутствовали, то CNN, скорее всего, научится определять понятия *млекопитающее*, *оранжевый* и *полосы* для предсказания меток этих объектов. Когда мы передадим в модель изображение тигра, эти признаки будут правильно идентифицированы на изображении, и, скорее всего, самым близким вложением слова из нашего словаря окажется вложение слова *тигр*.

8 Продвинутые методики

В этой главе описываются методики, которые могут пригодиться вам в вашей практике для решения некоторых задач. Она называется «Продвинутые методики» не потому, что представленные здесь методы более сложные, а потому что они применяются для решения узкоспециализированных задач. Во многих ситуациях эти методики вам едва ли потребуются, но иногда они оказываются очень полезными.

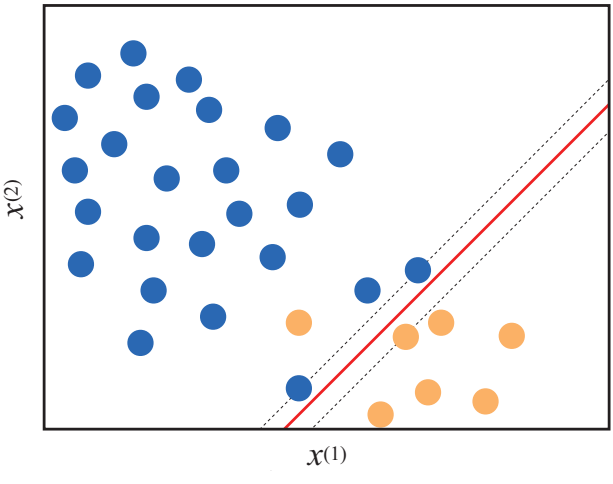
8.1. Работа с несбалансированными наборами данных

На практике некоторые классы часто оказываются недостаточно представленными в обучающих данных. Примером могут служить наборы данных, используемые для обучения классификатора, различающего законные и мошеннические транзакции в электронной коммерции: данные законных транзакций встречаются гораздо чаще. При использовании SVM с мягким зазором можно определить стоимость для неправильно классифицированных данных. Поскольку в обучающих данных всегда присутствует шум, высока вероятность, что многие данные законных транзакций окажутся не на той стороне границы и это будет способствовать увеличению цены.

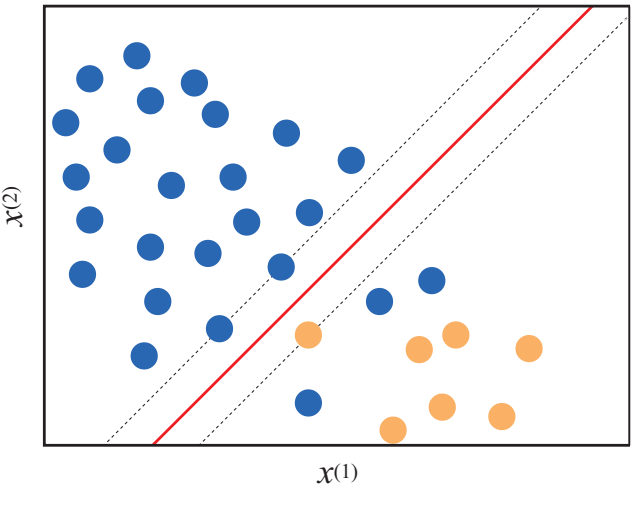
Алгоритм SVM попытается сместить гиперплоскость, чтобы избежать как можно большего числа ошибок в классификации данных. При этом стремление правильно классифицировать данные, принадлежащие к классу подавляющего большинства, увеличивает риск неправильной классификации «мошеннических» данных, которые находятся в меньшинстве. Эта ситуация показана на рис. 8.1а. Эта проблема наблюдается в большинстве алгоритмов обучения, применяемых к несбалансированным наборам данных.

Если увеличить стоимость ошибочной классификации данных из класса меньшинства, то модель постарается избежать неправильной классификации этих данных, но ценой неправильной классификации некоторых данных из класса большинства, как показано на рис. 8.1б.

Некоторые реализации SVM позволяют задавать вес каждого класса. Алгоритм обучения учтет эту информацию при поиске лучшей гиперплоскости.



(а)



(б)

Рис. 8.1. Иллюстрация проблемы несбалансированности набора данных. (а) Оба класса имеют одинаковый вес; (б) данные в классе меньшинства имеют более высокий вес

Если алгоритм обучения не позволяет задавать вес классов, можно попробовать метод **увеличения выборки** (oversampling). Он позволяет увеличить важность некоторого класса путем создания нескольких копий данных из этого класса.

Противоположный подход, **сокращение выборки** (undersampling), состоит в случайном исключении из обучающего набора некоторых данных из класса большинства.

Также можно попробовать создать синтетические данные, случайно выбирая значения признаков из нескольких данных класса меньшинства и объединяя их в новые данные этого класса. Есть два популярных алгоритма увеличения выборки с образцами из класса меньшинства путем создания синтетических примеров: **метод расширения выборки меньшинства синтетическими образцами** (Synthetic Minority Oversampling Technique, SMOTE) и **метод адаптивной синтетической выборки** (Adaptive Synthetic Sampling, ADASYN).

Методы SMOTE и ADASYN во многих отношениях действуют одинаково. Для данного образца $\mathbf{x}_i$ класса меньшинства они выбирают k ближайших его соседей (обозначим этот набор из k данных как $\mathcal{S}_k$), а затем создают синтетический образец $\mathbf{x}_{нов.}$ как $\mathbf{x}_i + \lambda(\mathbf{x}_{zi} - \mathbf{x}_i)$, где $\mathbf{x}_{zi}$ — представитель класса меньшинства, выбранный случайным образом из $\mathcal{S}_k$. Гиперпараметр интерполяции λ — это любое число в диапазоне [0, 1].

Оба метода, SMOTE и ADASYN, случайно выбирают все возможные $\mathbf{x}_i$ в наборе данных. В ADASYN число синтетических данных, генерируемых для каждого $\mathbf{x}_i$, пропорционально количеству данных в $\mathcal{S}_k$, не принадлежащих классу меньшинства. Соответственно, в областях, где данные класса меньшинства встречаются редко, генерируется больше синтетических данных.

Некоторые алгоритмы менее чувствительны к проблеме несбалансированности набора данных. Деревья решений, а также случайный лес и градиентный бустинг часто показывают неплохую эффективность на несбалансированных наборах данных.

8.2. Объединение моделей

Ансамблевые алгоритмы, такие как случайный лес, обычно объединяют модели одинаковой природы. Они увеличивают эффективность, объединяя сотни слабых моделей. На практике иногда можно получить дополнительный выигрыш в эффективности, комбинируя сильные модели, созданные с использованием *различных* алгоритмов обучения. В этом случае обычно используются только две или три такие модели.

Три типичных способа объединения моделей: 1) усреднение, 2) большинство голосов и 3) штабелирование.

Усреднение может применяться к моделям регрессии, а также к моделям классификации, которые дают классификационные баллы. В этом случае все модели, назовем их *базовыми моделями*, применяются к входу **x**, а их прогнозы усредняются. Чтобы увидеть, обеспечивает ли модель «усреднения» более высокое качество прогнозирования, чем каждый отдельный алгоритм, ее можно проверить на тестовом наборе с использованием метрики по вашему выбору.

Метод **большинства голосов** используется в моделях классификации. Согласно этому методу, все базовые модели применяются к входу **x**, а в качестве общего результата возвращается класс, выбранный большинством моделей. Если несколько классов получили одинаковое число голосов, результат выбирается случайным образом или возвращается сообщение об ошибке (если неправильная классификация имеет высокую цену).

Штабелирование заключается в создании метамодели, которая принимает на вход выходы базовых моделей. Допустим, нужно объединить классификаторы f_1 и f_2, которые выбирают предсказание из одного набора классов. Чтобы создать обучающий образец $\left(\hat{\mathbf{x}}_i,\ \hat{y}_i\right)$ для суммирующей модели, нужно задать $\hat{\mathbf{x}}_i = \left[f_1(\mathbf{x}),\ f_2(\mathbf{x})\right]$ и $\hat{y}_i = y_i$.

Если какие-то из базовых моделей возвращают не только класс, но и оценку правдоподобия для каждого класса, эти оценки тоже можно использовать как признаки.

Для обучения штабелирующей модели рекомендуется использовать данные из обучающего набора, а настройку гиперпараметров производить с помощью перекрестной проверки.

Разумеется, эффективность штабелирующей модели следует проверить на контрольном наборе и сравнить с эффективностью базовых моделей.

Увеличение эффективности объединения нескольких моделей обусловлено тем, что при согласии нескольких сильных некоррелированных моделей высока вероятность, что они сойдутся во мнении в отношении правильного результата. Ключевое слово здесь «некоррелированный». В идеале базовые модели должны быть получены с использованием разных признаков или алгоритмов с разной природой — например, SVM и случайный лес. Объединение разных версий алгоритма дерева решений или нескольких SVM с разными гиперпараметрами может не дать значительного увеличения качества прогнозирования.

8.3. Обучение нейронных сетей

Одним из сложных аспектов в обучении нейронной сети является преобразование данных во входы, которые сеть сможет обрабатывать. Если сеть должна обрабатывать изображения, прежде всего необходимо привести все изображения к одному размеру. После этого обычно выполняется сначала стандартизация пикселов, а затем нормализация в диапазон [0, 1].

Текст необходимо разбить на лексемы (то есть на такие части, как слова, знаки препинания и другие символы). Для CNN и RNN каждая лексема преобразуется в вектор с использованием унитарного кодирования, в результате чего текст превращается в список векторов. Другой, часто лучший способ представления лексем заключается в создании **вложений слов**. Для многослойного перцептрона неплохие результаты, особенно для длинных текстов (длиннее, чем SMS-сообщения и твиты), можно получить, преобразуя тексты в векторы с использованием подхода «мешок слов».

Выбор конкретной архитектуры нейронной сети является сложной задачей. Например, для задачи обучения seq2seq уже создано множество архитектур и почти каждый год появляются новые. Я советую отыскать современные решения, подходящие для вашей конкретной задачи, с помощью поисковых систем Google Scholar или Microsoft Academic, которые позволяют находить научные публикации по ключевым словам и диапазонам времени. Если вы не имеете ничего против менее современных архитектур, поищите уже реализованные архитектуры на GitHub и выберите такую, которую можно применить к вашим данным с минимальными изменениями.

На практике преимущество современной архитектуры перед более старой не так важно, когда выполняются предварительная обработка, очистка и нормализация данных и создается больший обучающий набор. Современные архитектуры нейронных сетей являются результатом сотрудничества ученых из нескольких лабораторий и компаний; такие модели могут быть очень сложными для самостоятельной реализации и обычно требуют большой вычислительной мощности для обучения. Время, потраченное на попытки воспроизвести результаты недавней научной работы, может не стоить того. Это время лучше потратить на создание решения на основе пусть и менее современной, но стабильной модели, и получение большего количества обучающих данных.

Определившись с выбором архитектуры сети, нужно также определиться с количеством слоев, их типами и размерами. Рекомендую начать с одного или двух слоев, обучить модель и посмотреть, хорошо ли она предсказывает обучающие данные (имеет малое смещение). Если результат не удовлетворит вас, постепенно

увеличивайте размер каждого слоя и их количество, пока модель не будет идеально прогнозировать обучающие данные. Если модель дает высокое качество на обучающих данных, но низкое на контрольных (имеет высокую дисперсию), добавьте в модель регуляризацию. Если после добавления регуляризации модель начала плохо предсказывать обучающие данные, немного увеличьте размер сети. Продолжайте итеративно наращивать сеть и регуляризовать, пока модель не достигнет достаточно высокого качества предсказания обучающих и контрольных данных в соответствии с вашей метрикой.

8.4. Продвинутая регуляризация

В нейронных сетях кроме L1- и L2-регуляризации можно также использовать типы регуляризации, характерные для нейронных сетей: **прореживание**, **ранняя остановка** и **пакетная нормализация**. Последний технически не является методом регуляризации, но он часто оказывает регулирующее влияние на модель.

Идея прореживания очень проста. Каждый раз, передавая в сеть обучающий образец, нужно на время отключать некоторые узлы. Чем выше процент отключенных узлов, тем выше эффект регуляризации. Библиотеки нейронных сетей позволяют добавлять прореживающий слой между двумя соседними слоями или определить в самом слое параметр, управляющий прореживанием. Параметр прореживания может иметь значение в диапазоне [0, 1], и его необходимо подобрать экспериментально, по результатам прогнозирования на контрольных данных.

Ранняя остановка — это метод обучения нейронной сети, предусматривающий сохранение предварительной модели после каждой эпохи и оценку ее эффективности на контрольном наборе. Как рассказывалось в разделе о градиентном спуске в главе 4, с увеличением количества эпох ошибка уменьшается. Уменьшение ошибки означает высокое качество модели на обучающих данных. Однако в какой-то момент, после некоторой эпохи, может начать развиваться эффект переобучения: ошибка продолжает снижаться, но качество модели на контрольных данных ухудшается. Если сохранять в файле версию модели после каждой эпохи, вы можете прекратить обучение, как только начнет наблюдаться снижение качества прогнозирования на контрольном наборе. Как вариант, можно продолжить процесс обучения в течение определенного количества эпох, а затем выбрать лучшую модель. Модели, сохраненные после каждой эпохи, называются *контрольными точками*. Некоторые специалисты, использующие машинное обучение на практике, очень часто используют этот прием; другие пытаются найти подходящий метод регуляризации, чтобы избежать такого нежелательного поведения.

Пакетная нормализация (которую правильнее называть пакетной стандартизацией) — это метод, заключающийся в стандартизации выходов каждого слоя перед передачей их узлам последующего слоя. На практике пакетная нормализация увеличивает скорость и стабильность обучения, а также дает некоторый эффект регуляризации. Поэтому всегда полезно попробовать использовать пакетную нормализацию. В библиотеках нейронных сетей часто есть возможность вставить слой пакетной нормализации между двумя соседними слоями.

Еще один метод регуляризации, который можно применять не только к нейронным сетям, но и практически к любому алгоритму обучения, называется **расширением данных** (data augmentation). Этот метод часто используется для регуляризации моделей, работающих с изображениями. Получив исходный размеченный обучающий набор, вы можете создать синтетический образец из исходного образца, применяя различные преобразования: слегка увеличивая размеры исходного образца, поворачивая, переворачивая, затемняя и т. д. В таких синтетических образцах сохраняется оригинальная метка. На практике это часто дает увеличение качества модели.

8.5. Обработка нескольких входов

На практике часто приходится работать с мультимодальными данными. Например, на вход могут подаваться изображение и текст, а бинарный выход может определять, описывает ли текст это изображение.

Алгоритмы поверхностного обучения трудно адаптировать для работы с мультимодальными данными. Тем не менее это возможно. Можно попробовать обучить две поверхностные модели — одну для изображений, а другую для текста. После этого можно применить метод объединения моделей, описанный выше.

Если не получается разделить задачу на две независимые подзадачи, можно попытаться преобразовать в вектор каждый вход (применяя соответствующий метод проектирования признаков), а затем просто объединить два вектора признаков, чтобы сформировать один более широкий вектор признаков. Например, если изображение имеет признаки $[i^{(1)}, i^{(2)}, i^{(3)}]$, а текст имеет признаки $[t^{(1)}, t^{(2)}, t^{(3)}, t^{(4)}]$, тогда объединенный вектор признаков будет иметь вид $[i^{(1)}, i^{(2)}, i^{(3)}, t^{(1)}, t^{(2)}, t^{(3)}, t^{(4)}]$.

Нейронные сети дают больше гибкости. Можно создать две подсети, по одной для входа каждого типа. Например, подсеть CNN может читать изображение, а подсеть RNN — текст. Последние слои обеих подсетей будут возвращать вложение: CNN — вложение изображения, а RNN — вложение текста. После этого можно объединить (конкатенировать) два вложения и добавить слой классификации, такой

как softmax или sigmoid, принимающий объединенные вложения. Библиотеки нейронных сетей предлагают простые в использовании инструменты, которые позволяют объединять или усреднять выходы слоев из нескольких подсетей.

8.6. Обработка нескольких выходов

В некоторых задачах требуется предсказать несколько выходов для одного входа. В предыдущей главе мы рассмотрели классификацию с многими метками. Некоторые задачи с несколькими выходами можно эффективно преобразовать в задачу классификации с многими метками. Особенно если они имеют метки одинаковой природы (например, теги). Или можно создать фиктивные метки, как комбинации оригинальных меток.

Однако в некоторых случаях с мультимодальными выходами нет возможности перечислить их комбинации. Рассмотрим следующий пример: нужно построить модель, которая обнаруживает объект на изображении и возвращает его координаты. Кроме того, модель должна возвращать тег, описывающий объект, например: «человек», «кошка» или «хомяк». В обучающих образцах имеется вектор признаков, представляющий изображение. Метка представлена вектором координат объекта и еще одним вектором с закодированным тегом.

В ситуациях, подобных этой, можно создать одну подсеть, которая будет работать как кодировщик и читать входное изображение, используя, например, один или несколько сверточных слоев. Последним слоем кодировщика будет вложение изображения. Поверх слоя с вложением можно добавить еще две подсети, одна из которых будет принимать вложение и предсказывать координаты объекта. Эта первая подсеть может иметь выходной слой ReLU, что является хорошим выбором для прогнозирования положительных действительных чисел, таких как координаты, и использовать в качестве функции стоимости среднеквадратичную ошибку C_1. Вторая подсеть будет принимать то же вложение и прогнозировать вероятность каждой метки. Эта вторая подсеть может иметь выходной слой softmax, который хорошо подходит для прогнозирования вероятностей, и использовать в качестве функции стоимости усредненный отрицательный логарифм правдоподобия C_2 (также называется стоимостью перекрестной энтропии).

Очевидно, что в этом случае важно, насколько точно предсказываются и координаты, и метка. Однако невозможно оптимизировать сразу две функции стоимости. Есть риск, что оптимизация одного критерия пойдет в ущерб другому. В такой ситуации можно добавить еще один гиперпараметр γ в диапазоне (0, 1) и опреде-

лить комбинированную функцию стоимости как $\gamma C_1 + (1 - \gamma)C_2$, а затем настроить значение γ с использованием контрольных данных, как еще один гиперпараметр.

8.7. Перенос обучения

Перенос обучения (transfer learning) — это тот случай, когда нейронные сети имеют уникальное преимущество перед поверхностными моделями. При переносе обучения выбирается существующая модель, обученная на некотором наборе данных, и адаптируется для прогнозирования данных из другого набора данных, отличного от того, на котором была построена модель. Этот второй набор данных не похож на контрольные наборы, которые используются для проверки и тестирования. Он может представлять какое-то другое явление или, как говорят специалисты по машинному обучению, он может происходить из другого статистического распределения.

Например, представьте, что вы обучили модель распознавать (и маркировать) диких животных, использовав большой набор размеченных данных. Через некоторое время вам предложили решить еще одну задачу: построить модель, которая распознавала бы домашних животных. С поверхностными алгоритмами обучения у вас не так много вариантов: вам придется создать еще один большой набор размеченных данных, но уже для домашних животных.

При использовании нейронных сетей ситуация обстоит намного лучше. Можно воспользоваться переносом обучения, который в нейронных сетях работает следующим образом.

1. На основе оригинального набора данных (с дикими животными) строится глубокая модель.
2. Собирается намного меньший набор размеченных данных для второй модели (с домашними животными).
3. С конца первой модели удаляется один или несколько слоев. Обычно это слои, отвечающие за классификацию или регрессию, и следуют за слоем, создающим векторное представление (вложение).
4. Удаленные слои заменяются новыми, адаптированными для решения новой задачи.
5. Фиксируются («замораживаются») параметры слоев, доставшихся от первой модели.
6. С использованием небольшого набора размеченных данных и градиентного спуска производится обучение параметров новых слоев.

В интернете можно найти множество глубоких моделей для задач распознавания образов, и среди них вам вполне вероятно удастся найти ту, которая окажется полезной для вашей задачи. Скачайте эту модель, удалите несколько последних слоев (количество слоев для удаления — это гиперпараметр), добавьте свои слои прогнозирования и обучите полученную модель на ваших данных.

Даже если вы не найдете готовые модели, прием переноса обучения все равно сможет помочь вам в ситуациях, когда задача требует очень дорогостоящего набора размеченных данных и есть возможность получить другой, более доступный набор размеченных данных. Допустим, вы строите модель классификации документов и получили от заказчика список, содержащий тысячу категорий. В этом случае вам нужно будет заплатить кому-то, чтобы тот: а) прочитал, понял и запомнил различия между категориями и б) прочитал до миллиона документов и разметил их.

Чтобы сэкономить на маркировке такого большого количества данных, вы можете использовать страницы «Википедии» в качестве обучающего набора данных и построить свою первую модель. Метки для страницы в «Википедии» можно получить автоматически, взяв, например, категории, к которым относится страница. Обучив первую модель предсказывать категории из «Википедии», вы сможете «настроить» эту модель, чтобы предсказать категории из списка вашего заказчика. Для решения поставленной задачи вам, вероятно, понадобится гораздо меньше размеченных данных, чем если бы вы начали решать ее с нуля.

8.8. Эффективность алгоритмов

Не все алгоритмы, способные решить задачу, являются практичными. Некоторые могут работать слишком медленно. Некоторые задачи можно решить с помощью быстрого алгоритма, для других задач быстрых алгоритмов может и вовсе не существовать.

Раздел информатики, называемый *анализом алгоритмов*, занимается проблемами определения и сравнения сложности алгоритмов. Для классификации алгоритмов в соответствии с ростом их требований к времени выполнения или объему памяти при увеличении размера входных данных используется критерий **О большое**.

Например, допустим, что у нас есть задача поиска двух самых удаленных друг от друга одномерных данных в множестве S размера N. Один алгоритм для решения этой задачи мог бы выглядеть так (здесь и далее приводится реализация на Python):

```
1 def find_max_distance(S):
2     result = None
3     max_distance = 0
4     for x1 in S:
5         for x2 in S:
6             if abs(x1 - x2) >= max_distance:
7                 max_distance = abs(x1 - x2)
8                 result = (x1, x2)
9     return result
```

Алгоритм выше перебирает все значения в $\mathcal{S}$ и в каждой итерации первого цикла повторно перебирает все значения в $\mathcal{S}$. То есть этот алгоритм выполняет N^2 сравнений чисел. Если время выполнения сравнения, вычисления абсолютного значения и присваивания принять за единицу времени, тогда временная сложность (или просто сложность) этого алгоритма в худшем случае составит $5N^2$. (В каждой итерации мы имеем одно сравнение, две операции вычисления абсолютного значения и две операции присваивания.) Когда измеряется сложность алгоритма в худшем случае, используется запись **О большое**. Сложность алгоритма выше можно записать как $O(N^2)$; константы, такие как число 5, игнорируются.

Ту же задачу можно решить с помощью другого алгоритма:

```
1 def find_max_distance(S):
2     result = None
3     min_x = float("inf")
4     max_x = float("-inf")
5     for x in S:
6         if x < min_x:
7             min_x = x
8         if x > max_x:
9             max_x = x
10     result = (max_x, min_x)
11     return result
```

Этот алгоритм предусматривает обход элементов множества $\mathcal{S}$ только один раз, поэтому он имеет сложность $O(N)$. В таких случаях мы говорим, что этот алгоритм *эффективнее* предыдущего.

Алгоритм считается эффективным, когда его сложность находится в полиномиальной зависимости от объема входных данных. Следовательно, эффективны оба алгоритма, со сложностью $O(N)$ и $O(N^2)$, потому что N — это полином степени 1, а N^2 — полином степени 2. Однако для очень больших объемов входных данных алгоритм $O(N^2)$ может оказаться слишком медленным. В эпоху больших данных ученые часто стремятся отыскать алгоритмы $O(\log N)$.

С практической точки зрения, реализуя свой алгоритм, *старайтесь по возможности избегать циклов*. Например, вместо циклов используйте операции с матрицами и векторами. В Python для вычисления **wx** вы должны написать:

```
1 import numpy
2 wx = numpy.dot(w,x),
```

а не

```
1 wx = 0
2 for i in range(N):
3     wx += w[i]*x[i]
```

Используйте подходящие структуры данных. Если порядок элементов в коллекции не имеет значения, используйте `set` вместо `list`. В Python операция проверки принадлежности конкретного экземпляра **x** множеству $\mathcal{S}$ выполняется более эффективно, когда $\mathcal{S}$ объявлено как множество `set`, и менее эффективно, когда $\mathcal{S}$ объявлено как список `list`.

Другая важная структура данных, которую можно использовать для увеличения эффективности кода на Python, — это словарь `dict`. В других языках он называется также ассоциативным массивом. Словарь позволяет определить набор пар ключ/значение и обеспечивает очень быстрый поиск ключей.

Если у вас нет абсолютной уверенности в своих действиях, всегда старайтесь использовать популярные библиотеки для научных вычислений. Пакеты для Python, такие как numpy, scipy и scikit-learn, были созданы опытными учеными и инженерами и реализуют очень эффективные алгоритмы. Они предлагают множество методов, написанных на языке программирования C, для максимальной эффективности.

Если вам понадобится выполнить обход элементов очень большой коллекции, используйте **генераторы**, которые создают функцию, возвращающую один элемент за раз, а не все сразу.

Используйте пакет *cProfile* для поиска неэффективных фрагментов в своем коде на Python.

Наконец, когда в вашем коде не осталось ничего, что можно было бы улучшить с алгоритмической точки зрения, у вас все равно остается возможность увеличить скорость выполнения кода, если задействовать:

- пакет *multiprocessing* для параллельного выполнения сразу нескольких вычислений;
- *PyPy*, *Numba* или другие похожие инструменты для компиляции кода на Python в быстрый и оптимизированный машинный код.

9 Обучение без учителя

Обучение без учителя используется для решения задач, когда отсутствуют размеченные данные. Это свойство делает данный подход труднопримени́мым для решения многих практических задач. Отсутствие меток, представляющих желаемый результат, означает отсутствие надежной контрольной точки для оценки качества модели. В этой книге я представляю только методы обучения без учителя, позволяющие строить модели, которые можно оценить исключительно на основе данных, а не человеческих суждений.

9.1. Оценка плотности

Оценка плотности — это задача моделирования функции плотности вероятности (probability density function, pdf) неизвестного распределения, из которого был получен набор данных. Этот метод может пригодиться во многих случаях, в частности для обнаружения новизны или факта вторжения. В главе 7 мы уже использовали оценку функции плотности вероятности, когда решали задачу одноклассовой классификации. Тогда мы решили, что наша модель будет **параметрической**, а точнее, многомерным нормальным распределением (Multivariate Normal Distribution, MND). Это решение было несколько произвольным, потому что, если фактическое распределение, из которого получен набор данных, отличается от MVN, модель, скорее всего, будет далека от идеальной. Мы также знаем, что модели могут быть непараметрическими. Мы использовали **непараметрическую модель** в ядерной регрессии. Как оказывается, тот же подход можно применять для оценки плотности.

Пусть $\{x_i\}_{i=1}^N$ — набор одномерных данных (решение задачи для многомерного случая выглядит аналогично), полученный из распределения с неизвестной функцией плотности вероятности f с $x_i \in \mathbb{R}$ для всех $i = 1, ..., N$. Нам требуется смоделировать форму f. Наша ядерная модель f, обозначенная как $\hat{f}_b$, определяется как

$$\hat{f}_b(x) = \frac{1}{Nb}\sum_{i=1}^{N} k\left(\frac{x - x_i}{b}\right), \tag{9.1}$$

где b — гиперпараметр, управляющий компромиссом между смещением и дисперсией модели, а k — ядро. Так же как в главе 7, используем ядро Гаусса:

$$k(z)=\frac{1}{\sqrt{2\pi}}\exp\left(\frac{-z^2}{2}\right).$$

Требуется найти значение b, минимизирующее разницу между фактической формой f и формой нашей модели $\hat{f}_b$. Для оценки разницы разумно выбрать **средний накопленный квадрат ошибки** (Mean Integrated Squared Error, MISE):

$$\text{MISE}(b)=\mathbb{E}\left[\int_{\mathbb{R}}\left(\hat{f}_b(x)-f(x)\right)^2 dx\right]. \tag{9.2}$$

В уравнении 9.2 мы возводим в квадрат разницу между фактической pdf f и нашей моделью $\hat{f}_b$. Интеграл $\int_{\mathbb{R}}$ заменяет суммирование $\sum_{i=1}^{N}$ в среднеквадратичной ошибке, а оператор ожидания $\mathbb{E}$ заменяет среднее значение $\frac{1}{N}$.

Действительно, когда потеря является функцией с непрерывной областью значений, такой как $\left(\hat{f}_b(x)-f(x)\right)^2$, мы должны заменить суммирование интегралом. Операция ожидания $\mathbb{E}$ означает, что b должно быть оптимальным для всех возможных реализаций обучающего набора $\{x_i\}_{i=1}^{N}$. Это важно, потому что $\hat{f}_b$ определена на конечной выборке некоторого распределения вероятностей, а фактическая pdf f определена в бесконечной области (множество $\mathbb{R}$).

Теперь перепишем правую сторону уравнения 9.2, как показано ниже:

$$\mathbb{E}\left[\int_{\mathbb{R}}\hat{f}_b^2(x)dx\right]-2\mathbb{E}\left[\int_{\mathbb{R}}\hat{f}_b(x)f(x)dx\right]+\mathbb{E}\left[\int_{\mathbb{R}}f(x)^2dx\right].$$

Третий член в формуле выше не зависит от b, следовательно, его можно игнорировать. Несмещенная оценка первого слагаемого задается как $\int_{\mathbb{R}}\hat{f}_b^2(x)dx$, тогда как несмещенную оценку второго слагаемого можно аппроксимировать **перекрестной проверкой** $-\frac{2}{N}\sum_{i=1}^{N}\hat{f}_b^{(i)}(x_i)$, где $\hat{f}_b^{(i)}$ — ядерная модель f, вычисленная на обучающем наборе после исключения образца x_i.

Член $\sum_{i=1}^{N}\hat{f}_b^{(i)}(x_i)$ известен в статистике как *оценка с одним отделяемым объектом*, форма перекрестной проверки, в которой каждый блок состоит из одного образца. Возможно, вы заметили, что член $\int_{\mathbb{R}}\hat{f}_b(x)f(x)dx$ (назовем его a) — это ожидаемое значение функции $\hat{f}_b$, потому что f — это функция плотности вероятности (pdf). Можно показать, что оценка с одним отделяемым объектом является несмещенной оценкой $\mathbb{E}[a]$.

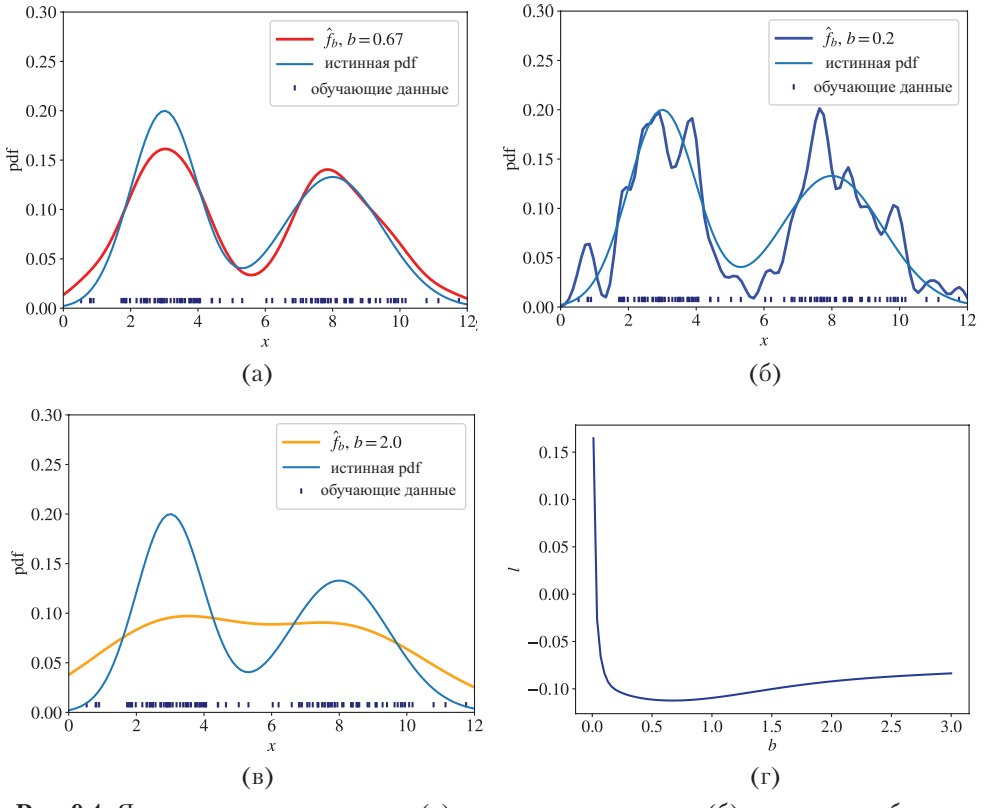


Рис. 9.1. Ядерная оценка плотности: (а) хорошее соответствие; (б) модель переобучена; (в) модель недообучена; (г) кривая поиска по сетке для выбора оптимального значения b

Теперь, чтобы найти оптимальное значение b^* для b, минимизируем стоимость, которая определяется как

$$\int_{\mathbb{R}} \hat{f}_b^2(x)dx - \frac{2}{N}\sum_{i=1}^{N} \hat{f}_b^{(i)}(x_i).$$

Найти b^* можно с помощью поиска по сетке. Для D-мерных векторов признаков член ошибки $x - x_i$ в уравнении 9.1 можно заменить евклидовым расстоянием $\|\mathbf{x} - \mathbf{x}_i\|$. На рис. 9.1 показаны оценки для одной и той же pdf, полученные с тремя разными значениями b из набора данных со 100 образцами, а также кривая поиска по сетке. Мы выбираем значение b^*, соответствующее нижней точке на кривой поиска по сетке.

9.2. Кластеризация

Кластеризация — это задача обучения выбору меток для данных с использованием набора неразмеченных данных. Поскольку набор данных не содержит меток,

оценить оптимальность полученной модели намного сложнее, чем в обучении с учителем.

Существует множество алгоритмов кластеризации, и, к сожалению, сложно сказать, какой из них окажется лучшим для того или иного набора данных. Обычно качество работы каждого алгоритма зависит от неизвестных свойств распределения, из которого был получен набор данных. В этой главе я опишу наиболее полезные и широко используемые алгоритмы кластеризации.

9.2.1. Кластеризация k средних

Алгоритм **кластеризации *k* средних** работает следующим образом. Сначала выбирается k — количество кластеров. Затем в пространстве признаков случайным образом выбираются k векторов признаков, называемых **центроидами**.

Затем, с использованием некоторой метрики, например евклидова расстояния, вычисляется расстояние от каждого образца $\mathbf{x}$ до каждого центроида. Затем каждому образцу ставится в соответствие ближайший центроид (как если бы мы назначали каждому образцу метку с идентификатором центроида). Для каждого центроида вычисляется средний вектор признаков данных, связанных с ним. Эти средние векторы признаков становятся новыми местоположениями центроидов.

Далее снова вычисляется расстояние от каждого образца до каждого центроида и изменяется его метка. Процедура повторяется до тех пор, пока после очередного пересчета местоположений центроидов связи не останутся неизменными. Модель представляет собой список связей между идентификаторами центроидов и образцами.

Начальное положение центроидов влияет на конечные положения, поэтому два прогона алгоритма k средних могут дать две разные модели. Некоторые варианты k средних вычисляют начальные положения центроидов, исходя из некоторых свойств набора данных.

На рис. 9.2 показан один прогон алгоритма k средних. Кружки на рис. 9.2 представляют двумерные векторы признаков; квадраты — движущиеся центроиды. Различные цвета фона представляют области, в которых все точки принадлежат одному кластеру.

Значение k — количество кластеров — является гиперпараметром, который выбирается аналитиком. Существует несколько методов выбора k, но ни один не является оптимальным. Большинство из этих методов требуют, чтобы аналитик сделал «обоснованное предположение», просматривая некоторые метрики или визуально изучая разбиение на кластеры. В этой главе я представлю один из подходов, помогающий выбрать достаточно хорошее значение для k, не требующий просматривать данные и делать какие-либо предположения.

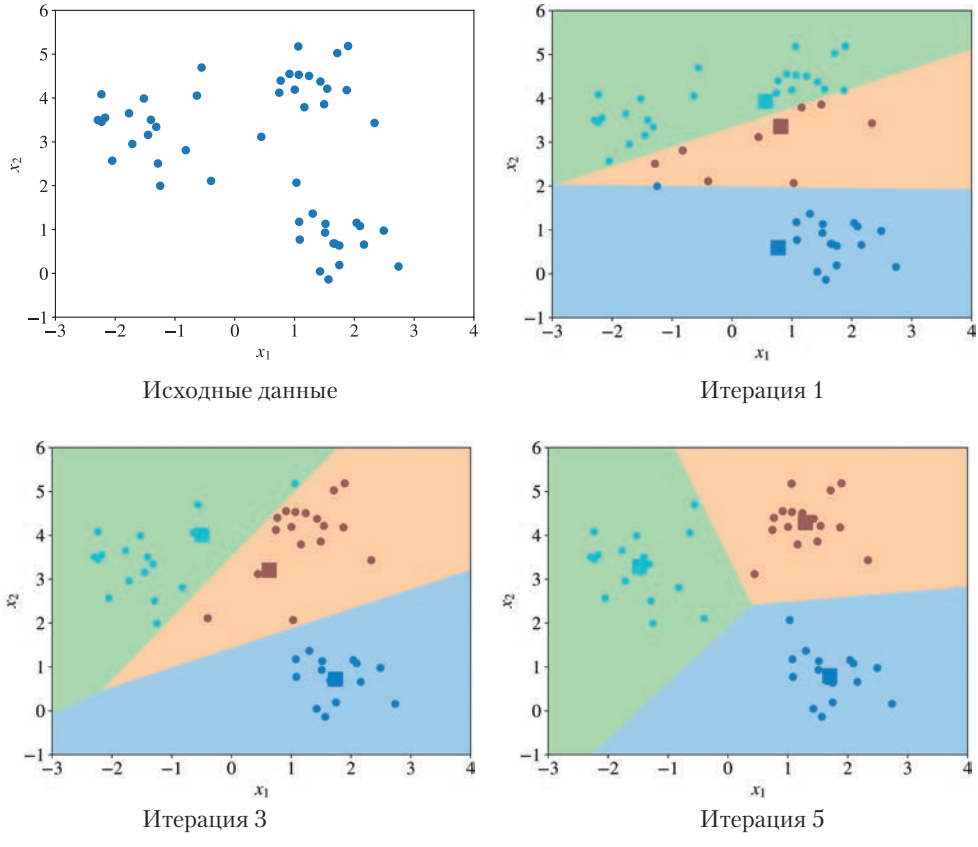


Рис. 9.2. Ход выполнения алгоритма k средних для $k = 3$

9.2.2. DBSCAN и HDBSCAN

В отличие от k средних и других подобных алгоритмов, основанных на центроидах, алгоритм DBSCAN осуществляет кластеризацию на основе плотности. Алгоритм DBSCAN не требует угадывать количество кластеров, вместо этого нужно определить два гиперпараметра: ϵ и n. В самом начале выбирается случайный образец $\mathbf{x}$ и маркируется как принадлежащий кластеру 1. Затем подсчитывается число данных, расстояние от которых до $\mathbf{x}$ меньше или равно ϵ. Если это число больше или равно n, тогда все эти ϵ-соседи помещаются в один и тот же кластер 1. Затем проверяется каждый член кластера 1 и выявляются соответствующие им ϵ-соседи. Если какой-либо член кластера 1 имеет n или больше ϵ-соседей, все они тоже добавляются в кластер 1. Процесс расширения кластера 1 продолжается, пока не останется данных, которые можно было бы добавить в него. После этого из набора данных выбирается другой образец, не принадлежащий ни одному кластеру,

и помещается в кластер 2. Так продолжается до тех пор, пока все данные не будут принадлежать какому-либо кластеру или не будут отмечены как аномальные. Аномальный образец — это образец, имеющий меньше n ϵ-соседей.

Преимущество алгоритма DBSCAN состоит в том, что он может создавать кластеры произвольной формы, в то время как k средних и другие алгоритмы на основе центроидов создают кластеры, имеющие форму гиперсферы. Очевидный недостаток DBSCAN в том, что он имеет два гиперпараметра, выбор хороших значений для которых (особенно для ϵ) может вызывать затруднения. Кроме того, с фиксированным гиперпараметром ϵ алгоритм кластеризации не может эффективно работать с кластерами, имеющими различную плотность.

HDBSCAN — это алгоритм кластеризации, сохраняющий преимущества DBSCAN и устраняющий необходимость выбирать значение ϵ. Алгоритм способен строить кластеры, имеющие разную плотность. HDBSCAN — это искусная комбинация множества идей, так что полное описание алгоритма выходит за рамки этой книги.

HDBSCAN имеет только один важный гиперпараметр: n, минимальное количество данных в одном кластере. Этот гиперпараметр относительно легко выбрать, основываясь на интуиции. HDBSCAN имеет очень быстрые реализации, способные эффективно работать с миллионами данных. Современные реализации k средних намного быстрее, чем HDBSCAN, однако качества последнего могут перевесить его недостатки при решении многих практических задач. Я рекомендую всегда пробовать использовать HDBSCAN для ваших данных в первую очередь.

9.2.3. Определение числа кластеров

Самый важный вопрос — сколько кластеров в наборе данных? Когда векторы признаков одно-, дву- или трехмерные, можно нарисовать распределение данных на графике и увидеть «облака» точек в пространстве признаков. Каждое облако — это потенциальный кластер. Однако для D-мерных данных, с $D > 3$, нарисовать такой график проблематично[1].

Один из способов определения разумного количества кластеров основан на идее **прогнозирующей силы**. Суть состоит в том, чтобы разделить данные на обучающий и тестовый наборы, как это делается в обучении с учителем. Выделив обучающий и тестовый наборы, $\mathcal{S}_{tr}$ с размером N_{tr} и $\mathcal{S}_{te}$ с размером N_{te} соответственно, вы фикси-

[1] Некоторые аналитики строят несколько двумерных графиков, на которых одновременно присутствует только пара признаков. Это тоже может дать представление о количестве кластеров. Однако такой подход страдает субъективностью, подвержен ошибкам и считается обоснованным предположением, а не научным методом.

руете количество кластеров k, запускаете алгоритм кластеризации C на наборах $\mathcal{S}_{tr}$ и $\mathcal{S}_{te}$ и получаете результаты кластеризации $C(\mathcal{S}_{tr}, k)$ и $C(\mathcal{S}_{te}, k)$.

Пусть A — результат кластеризации $C(\mathcal{S}_{tr}, k)$, полученный для обучающего набора. Кластеры в A можно рассматривать как области. Если образец попадает в одну из этих областей, значит, он принадлежит некоторому конкретному кластеру. Например, если применить алгоритм k средних к некоторому набору данных, в результате получится разбиение пространства признаков на k многоугольных областей, как показано на рис. 9.2.

Определим *матрицу* $N_{te} \times N_{te}$ *совместной принадлежности* $\mathbf{D}[A, \mathcal{S}_{te}]$, элементы которой $\mathbf{D}[A, \mathcal{S}_{te}]^{(i,i')} = 1$ тогда и только тогда, когда данные $\mathbf{x}_i$ и $\mathbf{x}_{i'}$ из тестового набора принадлежат тому же кластеру, согласно разбиению A. В противном случае $\mathbf{D}[A, \mathcal{S}_{te}]^{(i,i')} = 0$.

А теперь прервемся и посмотрим, что у нас получилось. Мы создали разбиение A, *использовав обучающий набор* данных, на k кластеров. Затем построили матрицу совместной принадлежности, которая указывает, принадлежат ли два образца *из тестового набора* одному кластеру в A.

Очевидно, что если величина k является разумной, тогда два образца, принадлежащие одному кластеру в решении $C(\mathcal{S}_{te}, k)$, скорее всего, будут принадлежать одному кластеру в решении и $C(\mathcal{S}_{tr}, k)$. С другой стороны, если значение k не является разумным (слишком высокое или слишком низкое), тогда разбиения на основе обучающих и тестовых данных, вероятно, будут менее согласованными.

На рис. 9.3 показаны использованные данные, а рис. 9.4 иллюстрирует идею. Графики на рис. 9.4а и 9.4б показывают результаты $C(\mathcal{S}_{tr}, 4)$ и $C(\mathcal{S}_{te}, 4)$ с соответствующими областями кластеров. На рис. 9.4в показаны тестовые данные, нанесенные на области кластеров, полученных в ходе кластеризации обучающих данных. На рис. 9.4в можно видеть, что оранжевые тестовые данные больше не принадлежат одному кластеру в соответствии с областями, полученными на обучающих данных. В результате в матрице $\mathbf{D}[A, \mathcal{S}_{te}]$ появляется множество нулей, что в свою очередь показывает, что $k = 4$, вероятно, не лучшее число кластеров.

Более формально прогнозирующая сила числа кластеров k определяется как

$$ps(k) \stackrel{\text{def}}{=} \min_{j=1,\ldots,k} \frac{1}{|A_j|(|A_j|-1)} \sum_{i,i' \in A_j} \mathbf{D}[A, \mathcal{S}_{te}]^{(i,i')},$$

где $A \stackrel{\text{def}}{=} C(\mathcal{S}_{tr}, k)$, A_j — j-й кластер из разбиения $C(\mathcal{S}_{te}, k)$ и $|A_j|$ — число данных в кластере A_j.

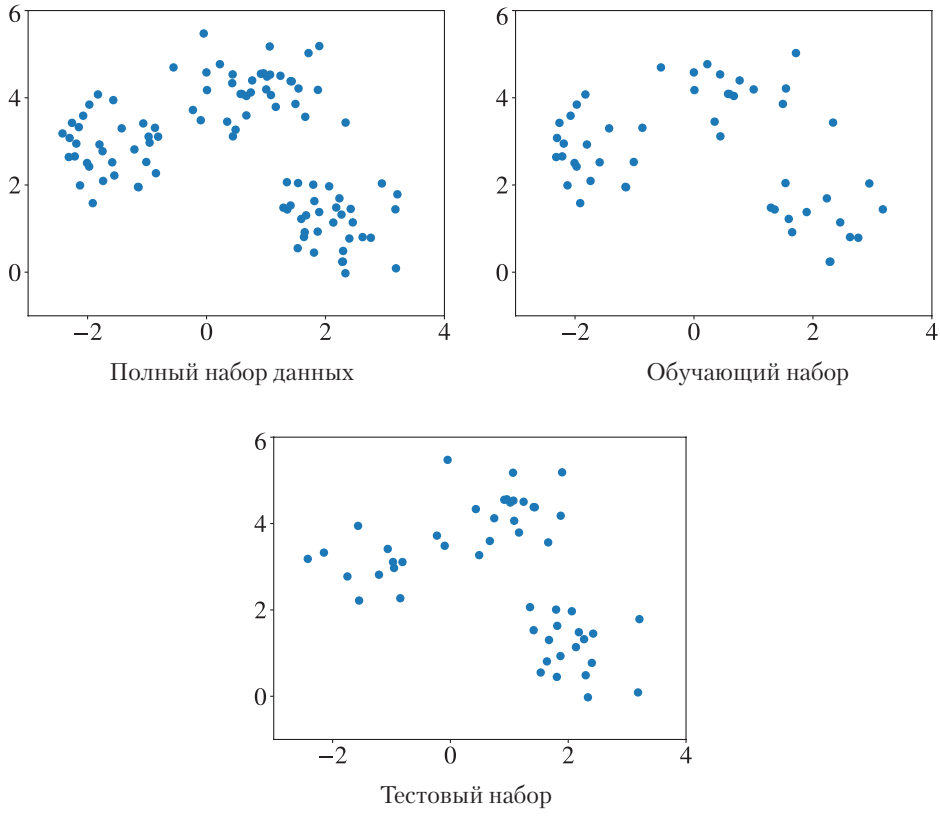


Рис. 9.3. Данные, использованные в задаче кластеризации, решение которой показано на рис. 9.4

С учетом разбиения $C(\mathcal{S}_{tr}, k)$ для каждого тестового кластера вычисляется доля пар в нем, которые также попали в один и тот же кластер, определяемый центроидом для обучающего набора. Прогнозирующая сила определяется как минимум этой величины для k тестовых кластеров.

Как показывают эксперименты, разумное количество кластеров является наибольшим k при $ps(k)$ выше 0.8. На рис 9.5 показаны примеры определения прогнозирующей силы разных значений k для данных, делящихся на два, три и четыре кластера.

Для недетерминированных алгоритмов кластеризации, таких как k средних, которые могут генерировать разные варианты разбиения, в зависимости от начальных положений центроидов, рекомендуется выполнить несколько прогонов алгоритма кластеризации для одного и того же k и вычислить среднюю прогнозирующую силу $\overline{ps}(k)$.

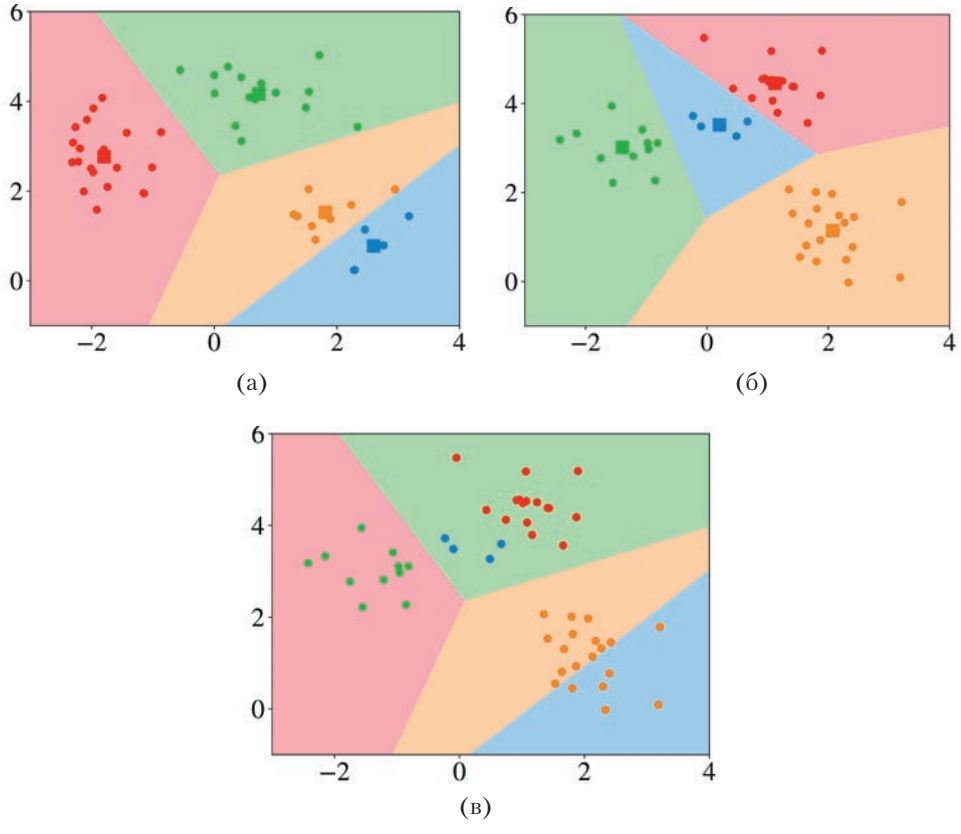


Рис. 9.4. Результат кластеризации для $k = 4$: (а) результат кластеризации обучающего набора; (б) результат кластеризации тестового набора; (в) тестовые данные, нанесенные поверх результатов кластеризации обучающего набора

Другой эффективный метод оценки количества кластеров называется **статистикой разрывов** (gap statistic). К другим, менее автоматизированным методам, которые все еще используются некоторыми аналитиками, относятся **метод «локтя»** (elbow method) и **метод среднего силуэта** (average silhouette).

9.2.4. Другие алгоритмы кластеризации

Методы DBSCAN и k средних реализуют так называемую **жесткую кластеризацию**, в которой каждый образец может принадлежать только одному кластеру. **Модель смеси гауссовых распределений** (Gaussian Mixture Model, GMM) допускает включение каждого образца в несколько кластеров с разным **баллом членства** (HDBSCAN тоже поддерживает такую возможность). Кластеризация

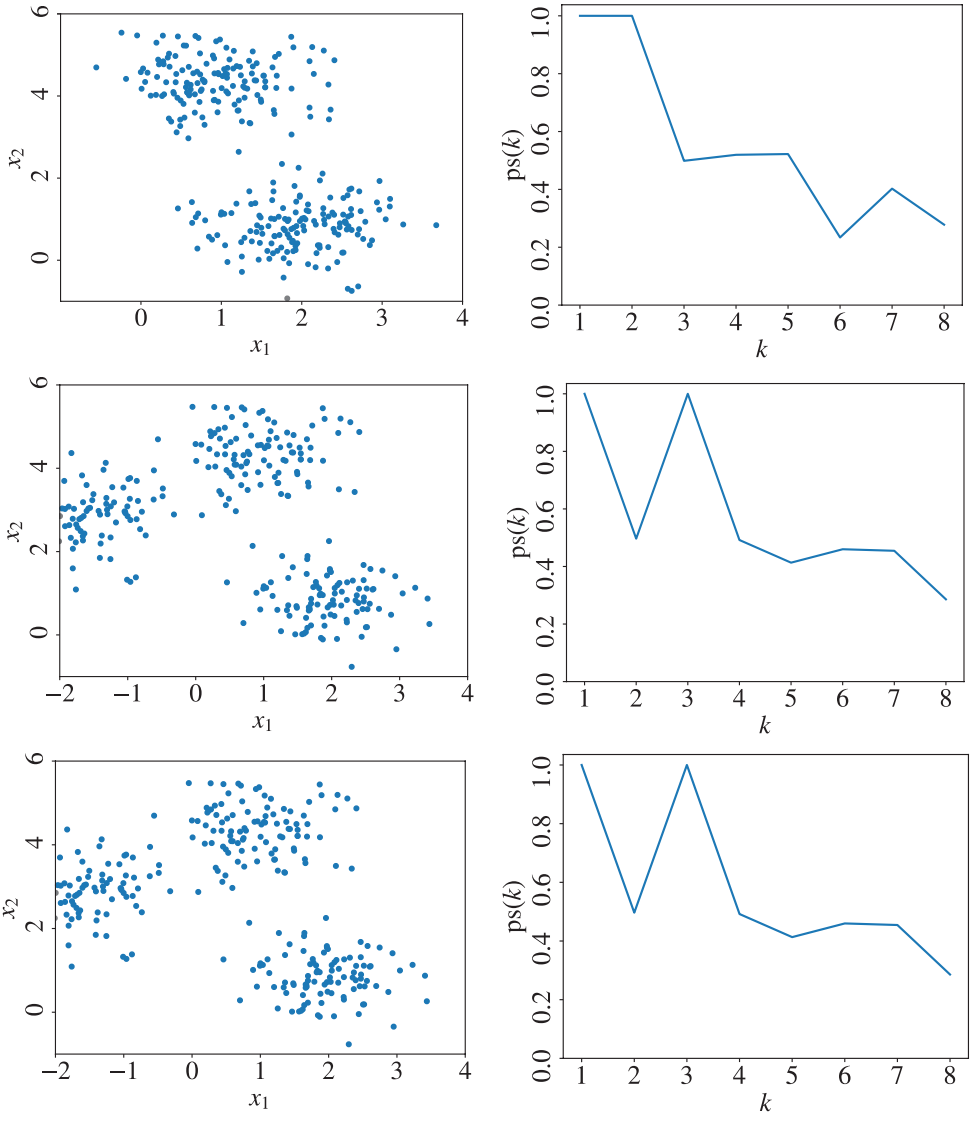


Рис. 9.5. Прогнозирующая сила разных значений k для данных, делящихся на два, три и четыре кластера

методом GMM очень похожа на оценку плотности на основе модели. В GMM вместо одного многомерного нормального распределения (MND) используется взвешенная сумма нескольких MND:

$$f_X = \sum_{j=1}^{k} \phi_j f_{\mu_j, \mathbf{\Sigma}_j},$$

где f_{μ_j, Σ_j} — многомерное нормальное распределение j, а ϕ_j — его вес в сумме. Значения параметров $\boldsymbol{\mu}_j$, $\boldsymbol{\Sigma}_j$ и ϕ_j для всех $j = 1, ..., k$ получаются с использованием **алгоритма максимизации ожидания** (Expectation Maximization, EM), оптимизирующего критерий **максимального правдоподобия**.

И снова ради простоты рассмотрим пример с одномерными данными. Также предположим, что есть два кластера: $k = 2$. В этом случае мы имеем два распределения Гаусса

$$f\left(x \middle| \mu_1, \sigma_1^2\right) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp - \frac{\left(x - \mu_1\right)^2}{2\sigma_1^2} \text{ и } f\left(x \middle| \mu_2, \sigma_2^2\right) = \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp - \frac{\left(x - \mu_2\right)^2}{2\sigma_2^2}, \tag{9.3}$$

где $f\left(x \middle| \mu_1, \sigma_1^2\right)$ и $f\left(x \middle| \mu_1, \sigma_2^2\right)$ — две функции плотности вероятности (pdf), определяющие вероятность $X = x$.

Используем алгоритм EM (максимизации ожидания) для оценки μ_1, σ_1^2, μ_2, σ_2^2, ϕ_1 и ϕ_2. Параметры ϕ_1 и ϕ_2 больше пригодятся для оценки плотности, чем для кластеризации, как мы увидим ниже.

Алгоритм EM действует, как описывается далее. Сначала выбираем наугад начальные значения μ_1, σ_1^2, μ_2 и σ_2^2 и устанавливаем $\phi_1 = \phi_2 = 1/2$ (в общем случае $1/k$ для каждого ϕ_j, $j \in 1, ..., k$).

В каждой итерации алгоритма EM выполняются четыре шага:

1. Для всех $i = 1, ..., N$ вычисляется вероятность для каждого x_i с использованием уравнения 9.3:

$$f\left(x_i \middle| \mu_1, \sigma_1^2\right) \leftarrow \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp - \frac{\left(x_i - \mu_1\right)^2}{2\sigma_1^2} \text{ и } f\left(x_i \middle| \mu_2, \sigma_2^2\right) \leftarrow \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp - \frac{\left(x_i - \mu_2\right)^2}{2\sigma_2^2}.$$

2. С использованием **правила Байеса** для каждого образца x_i вычисляется вероятность $b_i^{(j)}$ его принадлежности кластеру $j \in \{1, 2\}$ (то есть вероятность, что образец получен из распределения Гаусса j):

$$b_i^{(j)} \leftarrow \frac{f\left(x_i \middle| \mu_j, \sigma_j^2\right)\phi_j}{f\left(x_i \middle| \mu_1, \sigma_1^2\right)\phi_1 + f\left(x_i \middle| \mu_2, \sigma_2^2\right)\phi_2}.$$

Параметр ϕ_j отражает вероятность, что распределение Гаусса j с параметрами μ_j и σ_j^2 могло дать наш набор данных. Вот почему вначале мы устанавливаем $\phi_1 = \phi_2 = 1/2$: мы не знаем, насколько вероятно каждое из двух распределений, и отражаем нашу неосведомленность, устанавливая вероятность того и другого равной одной второй.

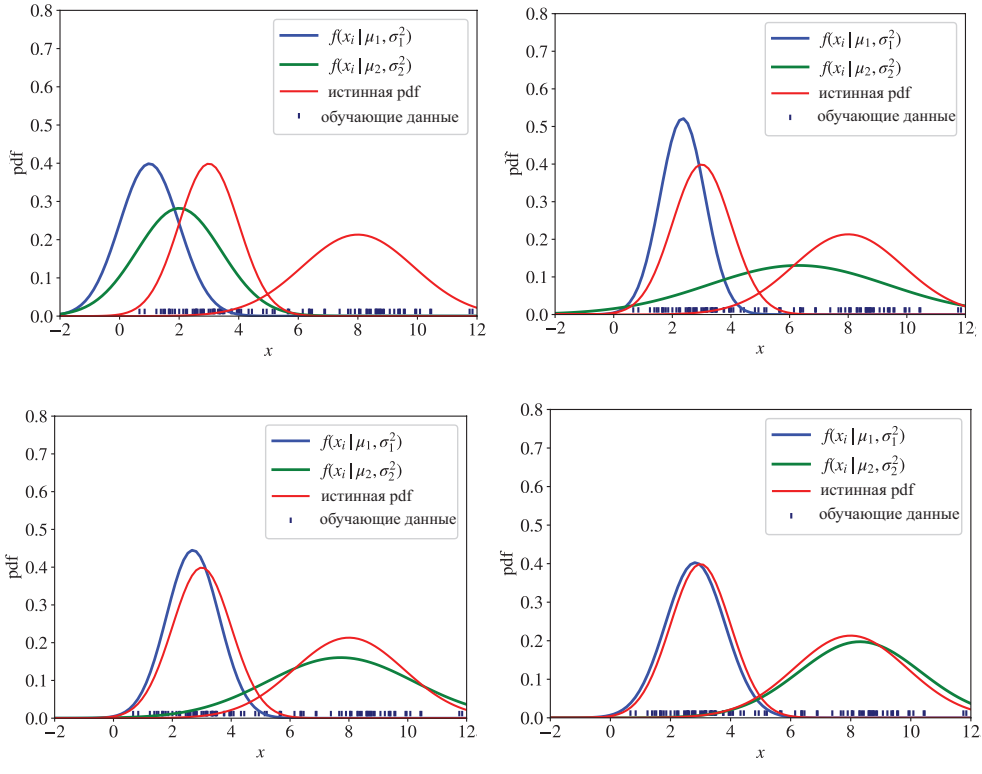


Рис. 9.6. Ход создания модели смеси гауссовых распределений с использованием алгоритма EM для двух кластеров (k = 2)

3. Вычисляются новые значения μ_j и σ_j^2, $j \in \{1, 2\}$ как

$$\mu_j \leftarrow \frac{\sum_{i=1}^{N} b_i^{(j)} x_i}{\sum_{i=1}^{N} b_i^{(j)}} \text{ и } \sigma_j^2 \leftarrow \frac{\sum_{i=1}^{N} b_i^{(j)} \left(x_i - \mu_j\right)^2}{\sum_{i=1}^{N} b_i^{(j)}}. \tag{9.4}$$

4. Корректируется ϕ_j, $j \in \{1, 2\}$ как

$$\phi_j \leftarrow \frac{1}{N} \sum_{i=1}^{N} b_i^{(j)}.$$

Шаги 1–4 повторяются, пока значения μ_j и σ_j^2 не перестанут существенно изменяться: например, изменение становится меньше, чем некоторое пороговое значение ϵ. Этот процесс иллюстрирует рис. 9.6.

Возможно, вы обратили внимание, что алгоритм EM очень похож на алгоритм *k* средних: сначала случайно выбираются кластеры, затем в цикле корректируются

параметры каждого кластера путем усреднения данных, связанных с этим кластером. Единственное отличие GMM от EM состоит в том, что образец x_i связывается с кластером j **не жестко**: x_i принадлежит кластеру j с вероятностью $b_i^{(j)}$. Вот почему новые значения для μ_j и σ_j^2 в уравнении 9.4 вычисляются не как средние (как в методе k средних), а как *средневзвешенные*, с весами $b_i^{(j)}$.

После определения значений параметров μ_j и σ_j^2 для каждого кластера j, оценка вероятности принадлежности образца x к кластеру j определяется как $f\left(x\middle|\mu_j, \sigma_j^2\right)$.

Описанный алгоритм легко распространить на случай D-мерных данных ($D > 1$). Единственное отличие — вместо дисперсии σ^2 потребуется использовать ковариационную матрицу $\mathbf{\Sigma}$, которая параметризует полиномиальное нормальное распределение (MND).

В отличие от метода k средних, где кластеры могут быть только сферическими, кластеры в GMM имеют эллиптическую форму с произвольным удлинением и углом поворота. Эти свойства определяются значениями в ковариационной матрице.

Не существует общепризнанного метода выбора правильного значения k в GMM. Я советую сначала разбить набор данных на обучающий и тестовый наборы. Затем выбрать разные значения k, для каждого построить модель f_{tr}^k на основе обучающих данных и по окончании выбрать значение k, максимизирующее вероятности принадлежности данных в тестовом наборе:

$$\arg\max_k \prod_{i=1}^{|N_{te}|} f_{tr}^k\left(\mathbf{x}_i\right),$$

где $|N_{te}|$ — размер тестового набора.

В литературе можно найти описание многих других алгоритмов кластеризации. Отдельно стоит упомянуть алгоритмы **спектральной** и **иерархической кластеризации**. Они могут оказаться более оптимальными для некоторых наборов данных. Однако в большинстве случаев вполне достаточно алгоритмов k средних, HDBSCAN и модели гауссовой смеси.

9.3. Сокращение размерности

Современные алгоритмы машинного обучения, такие как ансамблевые алгоритмы и нейронные сети, хорошо справляются с многомерными образцами, имеющими очень большую размерность, вплоть до миллионов признаков. В настоящее время, на современном уровне развития вычислительной техники с мощными графическими процессорами, методы сокращения размерности используются реже, чем

в прошлом. Чаще всего этот прием применяется для визуализации данных: люди могут интерпретировать графики, имеющие не больше трех измерений.

Другая ситуация, где можно извлечь выгоду из сокращения размерности, — когда нужно построить интерпретируемую модель и вы ограничены в выборе алгоритмов обучения. Например, вам разрешено использовать только обучение деревьев решений или линейную регрессию. Сокращая размерность данных и выясняя, какое качество исходного образца отражает каждый новый признак в сокращенном пространстве признаков, вы получаете возможность использовать более простые алгоритмы. Сокращение размерности устраняет избыточные или сильно коррелированные признаки, а также уменьшает долю шума в данных — все это способствует увеличению интерпретируемости модели.

На практике для сокращения размерности широко используются три основных метода: **метод главных компонент** (Principal Component Analysis, PCA), **равномерная аппроксимация и проекция многообразия** (Uniform Manifold Approximation and Projection, UMAP) и **автокодировщики**.

Я уже рассказывал об автокодировщиках в главе 7. Используя этот подход, в качестве вектора с уменьшенной размерностью можно использовать низкоразмерный выход **слоя узкого горлышка** автокодировщика, который представляет вектор входных объектов с высокой размерностью. Известно, что этот низкоразмерный вектор представляет значимую информацию, содержащуюся во входном векторе, благодаря чему автокодировщик способен восстановить входной вектор признаков, опираясь исключительно на слой узкого горлышка.

9.3.1. Метод главных компонент

Метод главных компонент (Principal Component Analysis, PCA) является одним из старейших методов сокращения размерности. Математический аппарат, лежащий в его основе, опирается на выполнение операций с матрицами, о которых я ничего не говорил в главе 2, поэтому оставлю знакомство с PCA как самостоятельное упражнение. Здесь я приведу только краткий пример применения этого метода.

Пусть имеется массив двумерных данных, как показано на рис. 9.7а. Основными компонентами являются векторы, определяющие новую систему координат, в которой первая ось направлена в сторону наибольшей дисперсии в данных. Вторая ось ортогональна первой и направлена в сторону второй наибольшей дисперсии в данных. Если бы данные были трехмерными, можно было бы провести третью ось, ортогональную к первой и ко второй осям, направленную в сторону третьей наибольшей дисперсии, и т. д. На рис. 9.7б показаны две основные компоненты в виде стрелок. Длина стрелки отражает дисперсию в данном направлении.

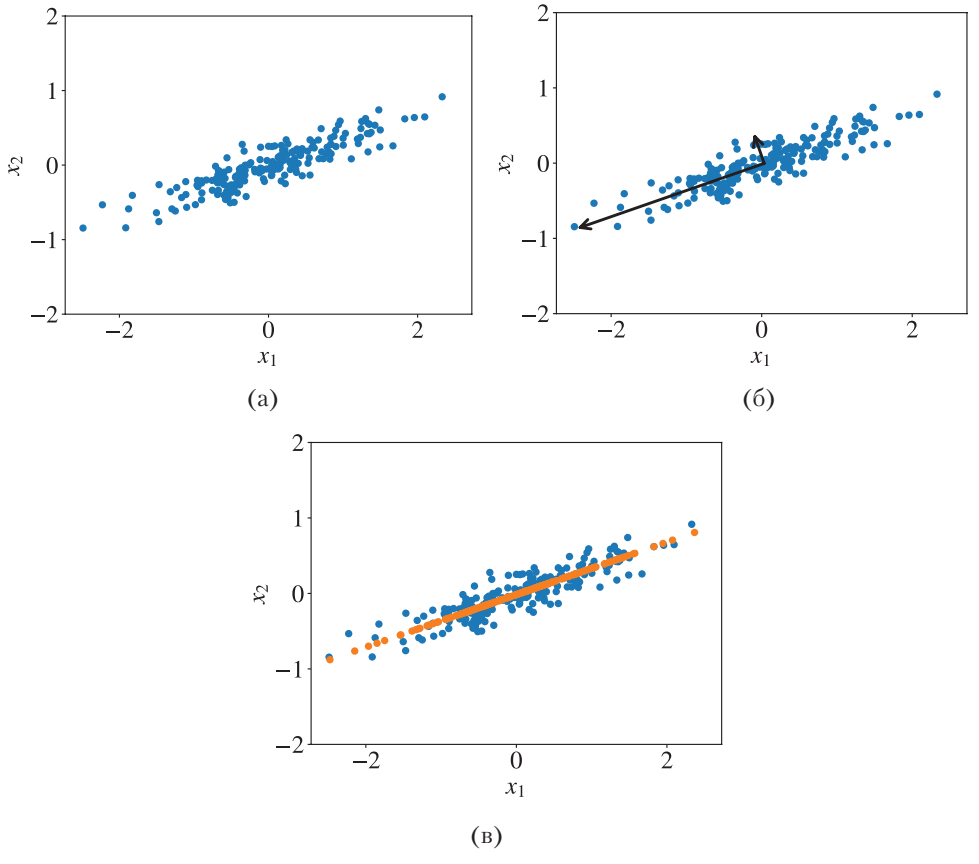


Рис. 9.7. Метод главных компонент: (а) исходные данные; (б) две главные компоненты в виде векторов; (в) проекция данных на первую главную компоненту

Теперь, чтобы сократить размерность данных до $D_{нов.} < \mathrm{D}$, нужно выбрать $D_{нов.}$ самых больших главных компонент и спроецировать на них точки данных. Для нашей двумерной иллюстрации можно принять $D_{нов.} = 1$, спроецировать данные на первую главную компоненту и получить оранжевые точки, как на рис. 9.7в.

Чтобы описать каждую оранжевую точку, нужна только одна координата вместо двух: координата относительно первой главной компоненты. Когда данные имеют очень большую размерность, на практике часто случается так, что первые две или три главные компоненты «объясняют» большую часть дисперсии в данных, поэтому, отображая данные на двух- или трехмерном графике, мы действительно можем получить наглядную картину, отражающую основные свойства многомерных данных.

9.3.2. UMAP

Многие современные алгоритмы сокращения размерности, особенно разработанные специально для целей визуализации, такие как **t-SNE** и **UMAP**, основываются на схожих идеях. Сначала разрабатывается метрика сходства двух данных. Для целей визуализации, помимо евклидова расстояния между двумя образцами, эта метрика сходства часто отражает некоторые локальные свойства двух данных, такие как плотность других данных вокруг них.

В UMAP такая метрика сходства w определяется как

$$w\left(\mathbf{x}_i, \mathbf{x}_j\right) \stackrel{\text{def}}{=} w_i\left(\mathbf{x}_i, \mathbf{x}_j\right) + w_j\left(\mathbf{x}_j, \mathbf{x}_i\right) - w_i\left(\mathbf{x}_i, \mathbf{x}_j\right) w_j\left(\mathbf{x}_j, \mathbf{x}_i\right). \tag{9.5}$$

Функция $w_i(\mathbf{x}_i, \mathbf{x}_j)$ определяется как

$$w_i\left(\mathbf{x}_i, \mathbf{x}_j\right) \stackrel{\text{def}}{=} \exp\left(-\frac{d\left(\mathbf{x}_i, \mathbf{x}_j\right) - \rho_i}{\sigma_i}\right),$$

где $d(\mathbf{x}_i, \mathbf{x}_j)$ — евклидово расстояние между двумя образцами, ρ_i — расстояние от $\mathbf{x}_i$ до ближайшего соседа, σ_i — расстояние от $\mathbf{x}_i$ до k-го ближайшего соседа (k — гиперпараметр алгоритма).

Можно показать, что метрика, определяемая уравнением 9.5, изменяется в диапазоне от 0 до 1 и симметрична, то есть $w(\mathbf{x}_i, \mathbf{x}_j) = w(\mathbf{x}_j, \mathbf{x}_i)$.

Обозначим через w сходство двух данных в исходном многомерном пространстве, и пусть w' представляет сходство, заданное тем же уравнением 9.5, в новом пространстве с меньшей размерностью.

Чтобы продолжить, я должен ввести понятие **нечеткого множества**. Нечеткое множество является обобщением множества. Для каждого элемента x нечеткого множества $\mathcal{S}$ существует функция принадлежности $\mu_S(x) \in [0, 1]$, которая определяет *степень принадлежности* x к множеству $\mathcal{S}$. Мы говорим, что x в малой степени принадлежит нечеткому множеству $\mathcal{S}$, если значение $\mu_S(x)$ близко к нулю. С другой стороны, если значение $\mu_S(x)$ близко к 1, тогда x имеет сильную принадлежность к $\mathcal{S}$. Если $\mu_S(x) = 1$ для всех $x \in \mathcal{S}$, то нечеткое множество $\mathcal{S}$ становится эквивалентным нормальному, четкому множеству.

Теперь посмотрим, зачем нам здесь нужно это понятие нечеткого множества.

Поскольку значения w и w' лежат в диапазоне от 0 до 1, мы можем рассматривать $w(\mathbf{x}_i, \mathbf{x}_j)$ как метрику принадлежности пары данных $(\mathbf{x}_i, \mathbf{x}_j)$ некоторому нечеткому

множеству. То же самое можно сказать и о w'. Сходство двух нечетких множеств называется перекрестной энтропией нечетких множеств и определяется как

$$C_{w,w'} = \sum_{i=1}^{N}\sum_{j=1}^{N}\left[w\left(\mathbf{x}_i,\mathbf{x}_j\right)\ln\left(\frac{w\left(\mathbf{x}_i,\mathbf{x}_j\right)}{w'\left(\mathbf{x}'_i,\mathbf{x}'_j\right)}\right)+\left(1-w\left(\mathbf{x}_i,\mathbf{x}_j\right)\right)\ln\left(\frac{1-w\left(\mathbf{x}_i,\mathbf{x}_j\right)}{1-w'\left(\mathbf{x}'_i,\mathbf{x}'_j\right)}\right)\right], \quad (9.6)$$

где $\mathbf{x}'$ — низкоразмерная «версия» оригинального образца $\mathbf{x}$ с бóльшим числом измерений.

Параметры $\mathbf{x}'_i$ (для всех $i = 1, ..., N$) в уравнении 9.6 представляют искомые низкоразмерные данные. Их можно вычислить с помощью градиентного спуска, путем минимизации $C_{w,w'}$.

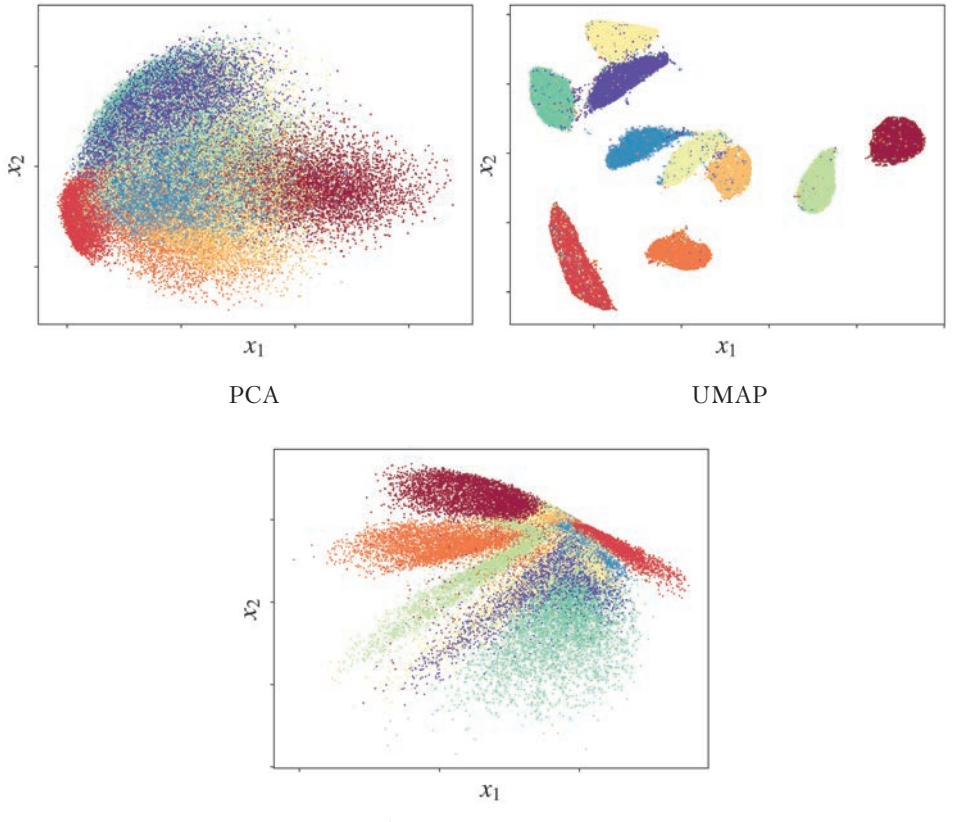


Рис. 9.8. Сокращение размерности набора данных MNIST с использованием трех разных методов

На рис. 9.8 показан результат сокращения размерности набора данных MNIST, содержащего изображения рукописных цифр. Набор MNIST часто используется для сравнительного анализа разных систем обработки изображений; он содержит 70 000 размеченных данных. Десять разных цветов на диаграмме соответствуют десяти классам. Каждая точка соответствует конкретному образцу в наборе данных. Как видите, UMAP визуально лучше разделяет данные (не забывайте, что этот метод не использует метки). На практике UMAP немного медленнее метода главных компонент (PCA), но быстрее автокодировщика.

9.4. Обнаружение аномалий

Задача **обнаружения аномалий** заключается в выявлении в некоем наборе данных таких данных, которые сильно отличаются от типичного представителя из этого набора. Мы уже познакомились с некоторыми методами, способными помочь решить эту задачу: автокодировщики и обучение классификатора с единственным классом. В случае с автокодировщиком мы обучаем его на наборе данных, а затем, когда требуется предсказать, является ли образец аномальным, используем модель автокодировщика и реконструируем образец из слоя узкого горлышка. Модель вряд ли сможет реконструировать аномальный образец.

В случае с классификатором с одним классом модель предсказывает, что входной образец принадлежит классу или является аномальным.

10 Другие формы обучения

10.1. Определение метрик

Я уже говорил, что чаще всего в качестве метрик сходства (или различий) между двумя векторами признаков используются **евклидово расстояние** и **косинусное сходство**. Такой выбор метрики кажется логичным, но необоснованным, как и выбор квадрата ошибки в линейной регрессии (или сама линейная форма регрессии). Тот факт, что качество метрики зависит от набора данных, явно показывает, что ни одна метрика не является идеальной.

Вы можете *создать* метрику, которая лучше подходит для вашего набора данных, и интегрировать ее в любой алгоритм обучения, которому требуется метрика, например, *k* средних или kNN. Как без опробования всех возможных вариантов узнать, какое уравнение будет хорошей метрикой? Как вы уже наверняка догадались, метрику можно вывести из самих данных.

Напомню, как определяется евклидово расстояние между двумя векторами признаков, $\mathbf{x}$ и $\mathbf{x}'$:

$$d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\| \stackrel{\text{def}}{=} \sqrt{(\mathbf{x} - \mathbf{x}')^2} = \sqrt{(\mathbf{x} - \mathbf{x}')(\mathbf{x} - \mathbf{x}')}.$$

Эту метрику можно немного изменить и сделать ее параметризуемой, а затем определить эти параметры из данных. Рассмотрим следующую модификацию:

$$d_{\mathbf{A}}(\mathbf{x}, \mathbf{x}') = \|(\mathbf{x} - \mathbf{x}')\|_{\mathbf{A}} \stackrel{\text{def}}{=} \sqrt{(\mathbf{x} - \mathbf{x}')^{\mathsf{T}} \mathbf{A} (\mathbf{x} - \mathbf{x}')},$$

где $\mathbf{A}$ — матрица $D \times D$. Пусть $D = 3$. Если принять, что $\mathbf{A}$ является единичной матрицей,

$$\mathbf{A} \stackrel{\text{def}}{=} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

тогда $d_{\mathbf{A}}$ становится евклидовым расстоянием. Если $\mathbf{A}$ является неединичной диагональной матрицей, такой как

$$\mathbf{A} \stackrel{\text{def}}{=} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

тогда разные измерения будут иметь в метрике разную важность. (В примере выше второе измерение является более важным в вычислении метрики.) В общем случае, чтобы называться метрикой, функция двух переменных должна удовлетворять трем условиям:

1. $d(\mathbf{x}, \mathbf{x}') \geq 0$, неотрицательность;
2. $d(\mathbf{x}, \mathbf{x}') \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{x})$, неравенство треугольника;
3. $d(\mathbf{x}, \mathbf{x}') = d(\mathbf{x}', \mathbf{x})$, симметрия.

Для удовлетворения первых двух условий матрица $\mathbf{A}$ должна быть *положительно полуопределенной*. Положительно полуопределенную матрицу можно рассматривать как обобщение понятия неотрицательного действительного числа. Любая положительно полуопределенная матрица $\mathbf{M}$ удовлетворяет условию

$$\mathbf{z}^T\mathbf{M}\mathbf{z} \geq 0,$$

для любого вектора $\mathbf{z}$, размерность которого совпадает с числом строк и столбцов в $\mathbf{M}$.

Указанное выше свойство следует из определения положительно полуопределенной матрицы. Доказательство выполнения второго условия, когда матрица $\mathbf{A}$ является положительно полуопределенной, можно найти на веб-сайте книги.

Чтобы удовлетворить третье условие, можно просто взять $(d(\mathbf{x}, \mathbf{x}') + d(\mathbf{x}', \mathbf{x}))/2$.

Допустим, что у нас есть неразмеченное множество $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N$. Чтобы построить обучающие данные для задачи определения метрики, вручную создаются два набора. В первый набор $\mathcal{S}$ включаются пары данных $(\mathbf{x}_i, \mathbf{x}_k)$, если $\mathbf{x}_i$ и $\mathbf{x}_k$ похожи (с субъективной точки зрения). Во второй набор $\mathcal{D}$ включаются пары данных $(\mathbf{x}_i, \mathbf{x}_k)$, если $\mathbf{x}_i$ и $\mathbf{x}_k$ непохожи.

Чтобы получить матрицу параметров $\mathbf{A}$ из данных, нужно найти положительную полуопределенную матрицу $\mathbf{A}$, которая решает следующую задачу оптимизации:

$$\min_{\mathbf{A}} \sum_{(\mathbf{x}_i,\mathbf{x}_k)\in\mathcal{S}} \left\|(\mathbf{x}-\mathbf{x}')\right\|_{\mathbf{A}}^2 \text{ так, что } \sum_{(\mathbf{x}_i,\mathbf{x}_k)\in\mathcal{D}} \left\|(\mathbf{x}-\mathbf{x}')\right\|_{\mathbf{A}} \geq c,$$

где c — положительная константа (может быть любым числом).

Решение этой задачи оптимизации находится с помощью модифицированного градиентного спуска, гарантирующего, что найденная матрица **A** является положительно полуопределенной. Я оставляю описание алгоритма за рамками этой книги для самостоятельного изучения.

Следует отметить, что **обучение с первого раза** с использованием **сиамских сетей** и **триплетных потерь** можно рассматривать как задачу определения метрики: пары изображений одного человека принадлежат множеству $\mathcal{S}$, а пары случайных изображений принадлежат $\mathcal{D}$.

Существует много других способов определения метрики, в том числе нелинейных и ядерных. Однако метода, представленного в книге, а также адаптации метода обучения с первого раза должно быть достаточно для большинства практических применений.

10.2. Определение ранга

Определение ранга является задачей обучения с учителем. Среди прочего определение ранга часто используется для оптимизации результатов поиска, возвращаемых поисковой системой в ответ на запрос. В оптимизации рангов результатов поиска размеченный образец $\mathcal{X}_i$ в обучающем наборе с размером N является ранжированной коллекцией документов размера r_i (метки определяют ранг документов). Вектор признаков представляет каждый документ в коллекции. Цель обучения — найти функцию ранжирования f, возвращающую значения, которые можно использовать для ранжирования документов. Для каждого обучающего образца идеальная функция f будет возвращать значения, которые определяют тот же ранг, что и метки.

Каждый образец $\mathcal{X}_i$, $i = 1, ..., N$ — это коллекция векторов признаков с метками: $\mathcal{X}_i = \left\{\left(\mathbf{x}_{i,j},\ y_{i,j}\right)\right\}_{j=1}^{r_i}$. Элементы в векторе признаков $\mathbf{x}_{ij}$ представляют документ $j = 1, ..., r_i$. Например, $x_{i,j}^{(1)}$ может представлять давность создания документа, $x_{i,j}^{(2)}$ — наличие искомых слов в названии документа, $x_{i,j}^{(3)}$ — размер документа и т. д. Метка y_{ij} может быть рангом $(1, 2, ..., r_i)$ или оценкой. Например, чем ниже оценка, тем выше рейтинг документа.

Существует три подхода к решению этой задачи: **поточечный**, **попарный** и **списочный**.

При использовании поточечного подхода каждый обучающий образец преобразуется в несколько данных: по одному на документ. Задача обучения в этом случае превращается в стандартную задачу обучения с учителем, регрессии или логистической регрессии. В каждом образце $(\mathbf{x}, y)$ в задаче поточечного обучения $\mathbf{x}$ — это вектор признаков некоторого документа, а y — исходная оценка (если $y_{i,j}$ — оценка) или синтетическая оценка, полученная из ранжирования (чем выше ранг, тем ниже синтетическая оценка). В этом случае можно использовать любой алгоритм обучения с учителем. Однако решение обычно получается далеким от идеального. В основном это связано с тем, что каждый документ рассматривается изолированно, в то время как исходное ранжирование (заданное метками $y_{i,j}$ в исходном обучающем наборе) может оптимизировать позиции всего набора документов. Например, если мы уже присвоили высокий ранг странице из «Википедии» в некоторой коллекции документов, мы бы предпочли не давать высокий ранг другой странице с этого же сайта в ответ на тот же запрос.

При использовании попарного подхода документы тоже рассматриваются изолированно, однако в этом случае рассматривается сразу пара документов. Для пары документов $(\mathbf{x}_i, \mathbf{x}_k)$ строится модель f, которая получает пару $(\mathbf{x}_i, \mathbf{x}_k)$ и возвращает значение, близкое к 1, если ранг документа $\mathbf{x}_i$ должен быть выше ранга документа $\mathbf{x}_k$; в противном случае f возвращает значение, близкое к 0. Во время тестирования окончательный ранг для неразмеченного образца X получается путем агрегирования прогнозов для всех пар документов в X. Попарный подход работает лучше поточечного, но все еще далек от совершенства.

Современные алгоритмы обучения, такие как **LambdaMART**, реализуют подход, основанный на списках. При использовании списочного подхода модель оптимизируется непосредственно по некоторому показателю, который отражает качество ранжирования. Существуют разные метрики для оценки ранжирования результатов в поисковых системах, в том числе точность и полнота. Одна из популярных метрик, которая сочетает в себе точность и полноту, называется **усредненной средней точностью** (Mean Average Precision, MAP).

Чтобы определить MAP, попросим экспертов (в Google их называют *ранжировщиками*) изучить коллекцию результатов поиска для запроса и присвоить оценки релевантности каждому результату. Метки могут быть бинарными (1 для «релевантных» и 0 для «нерелевантных») или иметь некоторый масштаб, скажем, от 1 до 5: чем выше значение, тем более релевантным для запроса является документ. Пусть наши эксперты выполнят такую маркировку для коллекции из 100 запросов. Теперь проверим нашу модель ранжирования для этой коллекции. **Точность** модели для некоторого запроса определяется как

$$\text{точность} = \frac{\left|\{\text{релевантные документы}\} \cap \{\text{найденные документы}\}\right|}{\left|\{\text{найденные документы}\}\right|},$$

где запись $|\cdot|$ означает «число документов». Метрика **средней точности**, AveP, определяется для ранжированной коллекции документов, возвращаемых поисковой системой в ответ на запрос q как

$$\text{Ave}P(q) = \frac{\sum_{k=1}^{n}\left(P(k)\cdot rel(k)\right)}{\left|\{\text{релевантные документы}\}\right|},$$

где n — количество найденных документов, $P(k)$ обозначает точность, вычисленную для k первых результатов поиска, возвращаемых моделью ранжирования в ответ на запрос, $rel(k)$ — это индикаторная функция, равная 1, если элемент с рангом k является релевантным документом (по мнению экспертов), и ноль в противном случае. Наконец, MAP для коллекции поисковых запросов размера Q определяется как

$$\text{MAP} = \frac{\sum_{q=1}^{Q}\text{Ave}P(q)}{Q}.$$

Теперь вернемся к алгоритму LambdaMART. Он реализует попарный подход и использует градиентный бустинг для обучения функции ранжирования $h(\mathbf{x})$. Затем бинарная модель $f(\mathbf{x}_i, \mathbf{x}_k)$, предсказывающая, что документ $\mathbf{x}_i$ должен иметь более высокий ранг, чем документ $\mathbf{x}_k$ (для того же поискового запроса), задается сигмоидой с гиперпараметром α:

$$f(\mathbf{x}_i, \mathbf{x}_k) \stackrel{\text{def}}{=} \frac{1}{1+\exp\left(h(\mathbf{x_i}) - h(\mathbf{x_k})\right)\alpha}.$$

И снова, как во многих моделях, предсказывающих вероятность, функция стоимости является перекрестной энтропией, вычисляемой с использованием модели f. В нашем градиентном бустинге мы объединяем несколько деревьев регрессии, чтобы построить функцию h, пытаясь минимизировать стоимость. Напомню, что в градиентном бустинге мы добавляем дерево в модель, чтобы уменьшить ошибку текущей модели на обучающих данных. Для задачи классификации мы вычисляем производную функции стоимости, чтобы заменить фактические метки обучающих данных этими производными. Алгоритм LambdaMART работает аналогично, за одним исключением. Он заменяет фактический градиент комбинацией градиента и еще одного фактора, который зависит от метрики, такой как MAP. Этот фактор изменяет исходный градиент, увеличивая или уменьшая его, что способствует улучшению метрики.

Это блестящая идея, и не многие алгоритмы обучения с учителем могут похвастаться тем, что оптимизируют метрику напрямую. Оптимизация метрики — это то, в чем мы действительно заинтересованы, однако типичный алгоритм обучения с учителем оптимизирует стоимость вместо метрики (это делается потому, что

обычно метрики не дифференцируемы). В обучении с учителем после создания модели, оптимизирующей функцию стоимости, мы часто пытаемся настроить гиперпараметры, чтобы улучшить метрику. Алгоритм LambdaMART оптимизирует метрику непосредственно.

Остается вопрос: как построить ранжированный список результатов, опираясь на прогнозы модели f, которая предсказывает, должен ли ее первый вход иметь более высокий ранг, чем второй. В общем случае это сложная вычислительная задача, и существует множество реализаций механизмов ранжирования, способных преобразовывать попарные сравнения в список ранжирования.

Самый простой подход — использовать существующий алгоритм сортировки. Алгоритмы сортировки сортируют коллекцию чисел по возрастанию или убыванию. (Самый простой алгоритм сортировки называется *пузырьковой сортировкой*. Его часто преподают в вузах.) Обычно алгоритмы сортировки итеративно сравнивают пары чисел в коллекции и меняют их местами, исходя из результатов сравнения. Если внедрить функцию f в алгоритм сортировки, чтобы выполнить это сравнение, алгоритм сортировки будет сортировать документы, а не числа.

10.3. Обучение делать рекомендации

Обучение делать рекомендации — это подход к созданию рекомендательных систем. Обычно есть пользователь, который потребляет контент, есть история потребления и требуется предложить этому пользователю новый контент, который мог бы ему понравиться. Это может быть фильм на «Нетфликсе» или книга на «Амазоне».

Для реализации рекомендаций традиционно используются два подхода: **фильтрация контента** и **совместная фильтрация**.

Фильтрация контента заключается в изучении того, что нравится пользователям, на основе описания потребляемого ими контента. Например, если пользователь новостного сайта часто читает статьи по науке и технике, мы могли бы предложить ему больше документов по науке и технике. В более общем случае мы могли бы создать обучающий набор *для каждого пользователя* и добавить в него статьи, каждая из которых представлена в виде вектора признаков $\mathbf{x}$, а также метки y, отражающей то, что пользователь недавно читал данную статью. После этого остается только построить модель каждого пользователя и регулярно проверять каждый новый контент, чтобы определить, заинтересует ли он конкретного пользователя.

Подход на основе контента имеет много ограничений. Например, пользователь может оказаться в так называемом пузыре фильтров: система всегда будет пред-

лагать пользователю информацию, которая выглядит очень похожей на то, что он уже потреблял. В результате пользователь может оказаться в полной изоляции от информации, которая не соответствует его точке зрения или дополняет ее. В итоге пользователи могут просто перестать следовать таким рекомендациям, что нежелательно.

Совместная фильтрация имеет важное преимущество перед фильтрацией контента: рекомендации для конкретного пользователя вычисляются на основе того, что потребляют или оценивают другие пользователи. Например, если два пользователя дали высокие оценки одним и тем же десяти фильмам, тогда весьма вероятно, что пользователь 1 высоко оценит новые фильмы, рекомендованные на основе вкусов пользователя 2, и наоборот. Недостаток этого подхода заключается в игнорировании содержания рекомендуемых элементов.

При совместной фильтрации информация о пользовательских предпочтениях организована в виде матрицы. Каждая строка соответствует пользователю, а каждый столбец — контенту, который пользователь оценил или потребил. Обычно эта матрица огромна и чрезвычайно разрежена, то есть большинство ее ячеек не заполнены (или заполнены нулями). Причина разреженности заключается в том, что большинство пользователей потребляют или оценивают лишь небольшую часть доступного контента. На основе таких разреженных данных очень сложно дать содержательные рекомендации.

В большинстве действующих рекомендательных систем используется гибридный подход: они сочетают рекомендации, полученные с помощью моделей фильтрации контента и совместной фильтрации.

Я уже отмечал, что модель рекомендаций на основе контента можно построить с использованием модели классификации или регрессии, которая предсказывает отношение пользователя к контенту на основе признаков этого контента. Примерами признаков могут служить слова в книгах или статьях, которые понравились пользователю, цена, актуальность контента, личность автора контента и т. д.

Двумя наиболее эффективными алгоритмами обучения системы рекомендаций считаются **метод факторизации** (Factorization Machines, FM) и **автокодировщики с шумоподавлением** (Denoising Autoencoders, DAE).

10.3.1. Метод факторизации

Метод факторизации — это относительно новый вид алгоритмов. Он специально разрабатывался для разреженных наборов данных. Проиллюстрируем его применение на примере.

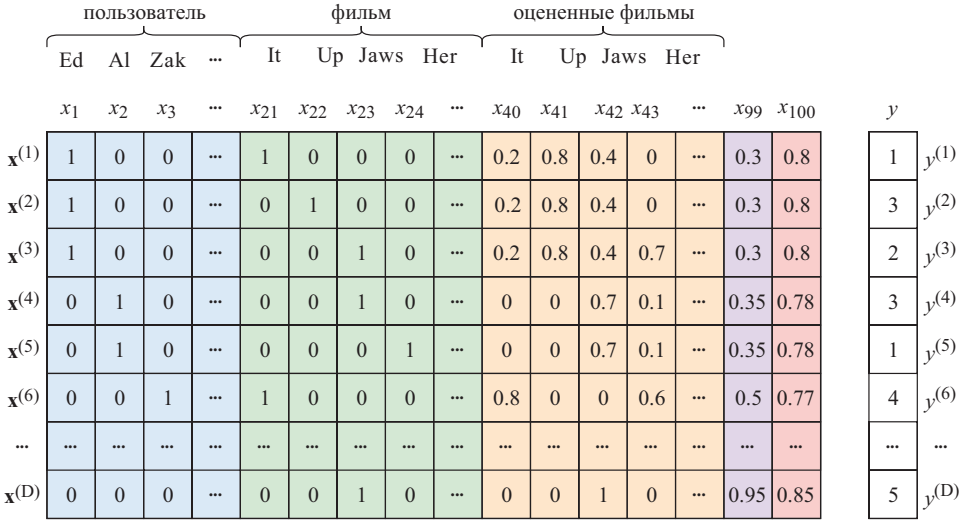


Рис. 10.1. Пример разреженных векторов признаков **x** и соответствующих им меток y

На рис. 10.1 показан пример разреженных векторов признаков с метками. Каждый вектор признаков представляет информацию об одном конкретном пользователе и одном конкретном фильме. Признаки в синей области представляют пользователя. Векторы, представляющие пользователей, получены методом унитарного кодирования. Элементы в зеленой области представляют фильмы. Эти векторы также получены методом унитарного кодирования. Признаки в желтой области представляют оценки, которые пользователь из синей области присвоил каждому оцененному им фильму. Признак x_{99} представляет долю фильмов, получивших Оскара, среди просмотренных пользователем. Признак x_{100} представляет процент общей протяженности фильма, просмотренного пользователем из синей области, до того, как он оценил фильм в зеленой области. Цель y представляет оценку, данную пользователем из синей области фильму из зеленой области.

В настоящих рекомендательных системах количество пользователей может исчисляться миллионами, поэтому матрица на рис. 10.1 будет насчитывать сотни миллионов строк. Количество признаков может составлять сотни тысяч, в зависимости от богатства выбора контента и вашей изобретательности, как аналитика, в разработке признаков. Элементы x_{99} и x_{100} были определены вручную, в процессе проектирования признаков, и в этом примере я остановлюсь только на двух признаках.

Попытка обучить регрессионную модель или модель классификации на таком чрезвычайно разреженном наборе данных приведет к плохому обобщению. Метод факторизации решает эту задачу по-другому.

Модель метода факторизации определяется следующим образом:

$$f(\mathbf{x}) \overset{\text{def}}{=} b + \sum_{i=1}^{D} w_i x_i + \sum_{i=1}^{D} \sum_{j=i+1}^{D} (\mathbf{v}_i \mathbf{v}_j) x_i x_j,$$

где b и w_i, $i = 1, ..., D$ — скалярные параметры, подобные тем, что используются в линейной регрессии. Векторы $\mathbf{v}_i$ — это k-мерные векторы **факторов**. k является гиперпараметром и обычно имеет значение намного меньше, чем D. Выражение $\mathbf{v}_i\mathbf{v}_j$ — это скалярное произведение i-го и j-го векторов факторов.

Как видите, вместо попытки отыскать один большой вектор параметров, который может плохо отражать взаимосвязи между признаками из-за разреженности, мы дополняем его дополнительными параметрами, отражающими парные взаимосвязи $x_i x_j$ между признаками. Однако вместо параметра $w_{i,j}$ для каждой взаимосвязи, что добавило бы огромное[1] количество новых параметров в модель, мы факторизуем $w_{i,j}$ на $\mathbf{v}_i\mathbf{v}_j$, добавляя $D_k << D(D-1)$ параметров в модель[2].

В зависимости от задачи функция потерь может быть квадратом ошибок потерь (для регрессии) или кусочно-линейной функцией потерь. Для классификации с $y \in \{-1, +1\}$, прогнозирование с кусочно-линейной функцией потерь или логистическими потерями осуществляется по формуле $y = \text{sign}(f(x))$. Логистические потери определяются как

$$loss(f(\mathbf{x}), y) = \frac{1}{\ln 2} \ln\left(1 + e^{-yf(\mathbf{x})}\right).$$

Для оптимизации средней потери можно использовать градиентный спуск. В примере на рис. 10.1 метки принадлежат множеству {1, 2, 3, 4, 5}, соответственно, это задача классификации с многими классами. Мы можем использовать стратегию «один против всех», чтобы преобразовать эту задачу в пять задач бинарной классификации.

10.3.2. Автокодировщики с шумоподавлением

В главе 7 вы узнали, что такое автокодировщик с шумоподавлением: это нейронная сеть, которая восстанавливает входные данные из слоя узкого горлышка. Тот факт, что входные данные искажены шумом, а выход должен быть свободен от него, делает автокодировщики с шумоподавлением идеальным инструментом для построения модели рекомендации.

---

[1] Точнее, добавилось бы $D(D-1)$ параметров $w_{i,j}$.

[2] Запись << означает «намного меньше, чем».

Идея очень проста: новые фильмы, которые могут понравиться пользователю, выглядят так, будто они были удалены из полного набора предпочтительных фильмов в результате какого-то процесса. Цель автокодировщика с шумоподавлением — восстановить эти удаленные элементы.

Чтобы подготовить обучающий набор для автокодировщика с шумоподавлением, нужно удалить синие и зеленые элементы из обучающего набора на рис. 10.1. Поскольку после этого некоторые примеры будут повторять друг друга, их нужно удалить и оставить только уникальные элементы.

Затем производится обучение автокодировщика восстановлению поврежденного входа, с заменой нулями некоторых ненулевых признаков в желтой области случайным образом во время обучения.

Во время прогнозирования нужно создать вектор признаков, представляющий пользователя, содержащий неповрежденные признаки из желтой области, а также элементы, созданные вручную, такие как x_{99} и x_{100}. С помощью обученной модели DAE восстановить неповрежденный вход и вернуть пользователю рекомендованные фильмы, имеющие самые высокие оценки в выходе модели.

Другая эффективная модель совместной фильтрации — FFNN с двумя входами и одним выходом. Как говорилось в главе 8, нейронные сети хорошо справляются сразу с несколькими входами. Обучающим образцом здесь является триплет (**u**, **m**, r). Входной вектор **u** представляет пользователя и получен **методом унитарного кодирования**. Второй входной вектор **m** представляет фильм и тоже получен методом унитарного кодирования. Выходной слой может быть сигмоидой (в этом случае метка r принадлежит диапазону [0, 1]) или ReLU, когда r может принадлежать некоторому типичному диапазону, например [1, 5].

10.4. Самообучение с учителем: вложения слов

Мы уже обсуждали вложения слов в главе 7. Напомню, что **вложения** — это векторы признаков, представляющие слова. Главное их свойство заключается в том, что похожие слова имеют похожие векторы признаков. Вопрос, который наверняка возник у вас — откуда берутся эти вложения. Ответ: они извлекаются из данных.

Есть много алгоритмов для создания вложений слов. Здесь мы рассмотрим только один из них: **word2vec**, и только одну версию, которая называется **skip-gram**, хорошо зарекомендовавшую себя на практике. В интернете можно найти готовые вложения word2vec для многих языков, доступные для скачивания.

Главная цель при создании вложений слов — получить модель, которую можно использовать для преобразования векторов, полученных методом унитарного кодирования, во вложения слов. Пусть словарь содержит 10 000 слов. При использовании унитарного кодирования для каждого слова получается 10 000-мерный вектор с нулями во всех элементах, кроме одного, содержащего 1. Разные слова имеют значение 1 в разных измерениях.

Рассмотрим предложение: «Я почти закончил читать книгу по машинному обучению». Теперь рассмотрим то же предложение, из которого мы удалили одно слово, например «книгу». Теперь предложение выглядит так: «Я почти закончил читать · по машинному обучению». Теперь оставим только три слова перед · и три после: «почти закончил читать · по машинному обучению». Посмотрите на этот фрагмент из семи слов с · в центре. А теперь представьте, что я предложил вам угадать, что означает ·. Вы почти наверняка дадите ответ: «книга», «статья» или «документ». Именно так слова из контекста позволяют предсказать слово, которое они окружают. Машина тоже может определить, что слова «книга», «документ» и «статья» имеют схожее значение: потому что они имеют схожий контекст во множестве текстов.

Как оказывается, обратное тоже верно: слово может предсказать контекст, который его окружает. Часть «почти закончил читать · по машинному обучению» называется скипграммой (skip-gram) с размером окна 7 (3 + 1 + 3). Используя документы, доступные в интернете, можно создать сотни миллионов скипграмм.

Обозначим скипграмму следующим образом: $[\mathbf{x}_{-3}, \mathbf{x}_{-2}, \mathbf{x}_{-1}, \mathbf{x}, \mathbf{x}_{+1}, \mathbf{x}_{+2}, \mathbf{x}_{+3}]$. В нашем предложении $\mathbf{x}_{-3}$ — это вектор унитарного кодирования для слова «почти», $\mathbf{x}_{-2}$ соответствует «закончил», $\mathbf{x}$ — пропущенное слово (·), $\mathbf{x}_{+1}$ — «по» и т. д. Скипграмма с размером окна 5 будет выглядеть следующим образом: $[\mathbf{x}_{-2}, \mathbf{x}_{-1}, \mathbf{x}, \mathbf{x}_{+1}, \mathbf{x}_{+2}]$.

Модель скипграммы с размером окна 5 схематически изображена на рис. 10.2. Это полносвязанная сеть, подобная многослойному перцептрону. Входное слово обозначено в скипграмме как ·. Нейронная сеть должна научиться предсказывать контекстные слова скипграммы с учетом центрального слова.

Теперь должно быть понятно, почему обучение такого рода называется **самообучением с учителем**: размеченные данные извлекаются из неразмеченных данных, таких как текст.

В выходном слое используется функция активации softmax. Функция стоимости — отрицательное логарифмическое правдоподобие. Вложение слова возвращается слоем вложения после передачи на вход модели вектора унитарного кодирования этого слова.

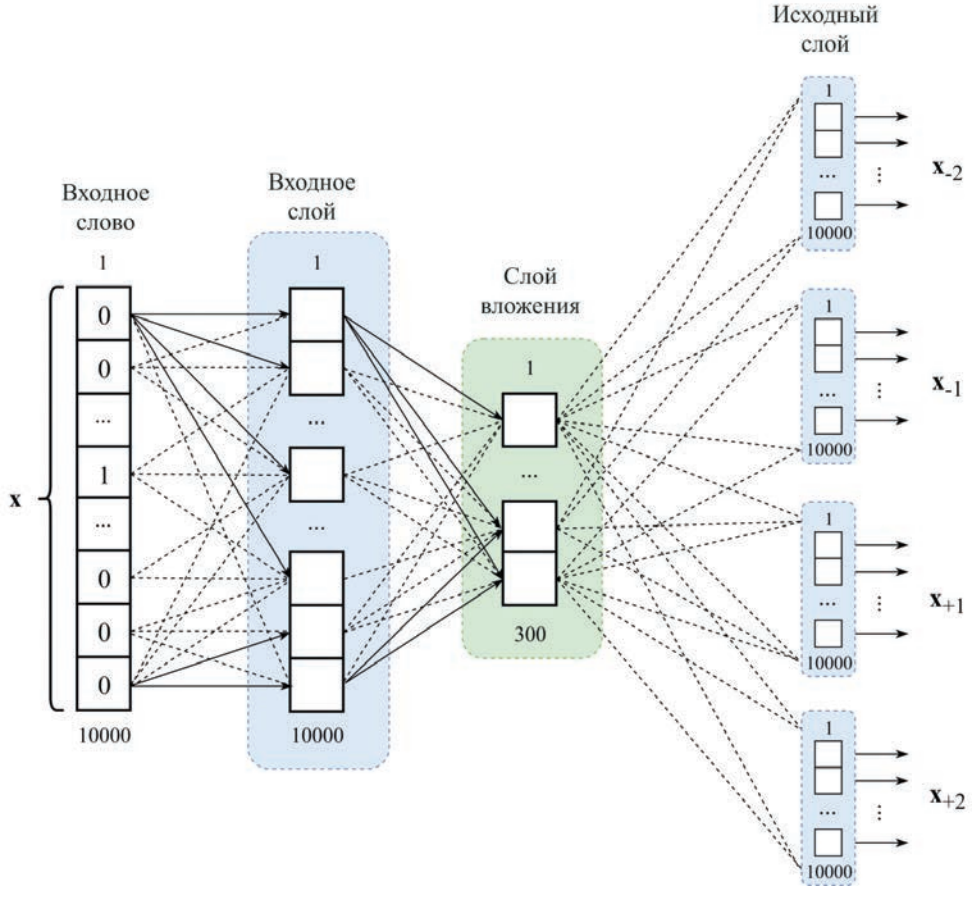


Рис. 10.2. Модель скипграммы с размером окна 5 и слоем вложения с 300 узлами

Модели word2vec имеют большое количество параметров, поэтому для повышения эффективности вычислений используются два метода: *иерархический softmax* (эффективный способ вычисления softmax, заключающийся в представлении выходных данных softmax в виде листьев бинарного дерева) и *отрицательная выборка* (идея которой состоит в обновлении в каждой итерации градиентного спуска случайной выборки из всех выходов). Знакомство с этими методами я оставляю вам для самостоятельного изучения.

11 Заключение

Ух ты, как быстро! Вы отлично поработали, если попали сюда и сумели понять бóльшую часть того, о чем рассказывалось в книге.

Книга получилась чуть больше, чем планировалось. В конце концов, если бы я хотел ограничить эту книгу, например, ровно сотней страниц, то мог бы уменьшить размер шрифта, поля и межстрочный интервал или удалить раздел об алгоритме UMAP и отправить вас к оригинальному источнику. Уверяю вас: вам едва ли понравилось бы самостоятельно знакомиться с оригинальной статьей об UMAP! (Шутка!)

Я уверен, что вы получили все необходимое, чтобы стать отличным современным аналитиком или специалистом по машинному обучению. Это не значит, что я охватил все, что только можно, но то, что мне удалось описать, в других книгах растянуто на тысячи страниц. Многое из того, о чем я рассказал, вообще отсутствует в книгах: типичные книги по машинному обучению консервативны и академичны, я же сделал упор на алгоритмах и методах, которые пригодятся вам в повседневной работе.

О чем бы я рассказал в книге по машинному обучению в тысячу страниц?

11.1. Что не было затронуто

11.1.1. Тематическое моделирование

В текстовом анализе широко используется метод обучения без учителя, который называется тематическим моделированием. Представьте, что у вас есть коллекция

текстовых документов и вам нужно определить тему каждого документа. **Латентное размещение Дирихле** (Latent Dirichlet Allocation, LDA) — очень эффективный алгоритм определения темы. Вы сами решаете, сколько тем присутствует в коллекции документов, а алгоритм назначает тему каждому слову в этой коллекции. Затем, чтобы определить тему документа, вы просто подсчитываете количество слов в этом документе, соответствующих каждой теме.

11.1.2. Гауссовские процессы

Гауссовские процессы (Gaussian Processes, GP) — это метод обучения с учителем, конкурирующий с ядерной регрессией. Он имеет некоторые преимущества перед последней. Например, поддерживает доверительные интервалы для линии регрессии в каждой точке. Я решил не объяснять этот метод, потому что не смог придумать простой способ объяснить его, но вам определенно стоит потратить некоторое время, чтобы познакомиться с гауссовскими процессами поближе. Это не будет пустой тратой времени.

11.1.3. Обобщенные линейные модели

Обобщенная линейная модель (Generalized Linear Model, GLM) — это обобщение линейной регрессии для моделирования разного вида зависимостей между входным вектором признаков и целью. Например, логистическая регрессия является одной из форм GLM. Если вас интересует регрессия и вы ищете простые и объяснимые модели, обязательно познакомьтесь с GLM поближе.

11.1.4. Вероятностные графические модели

Я упомянул один пример **вероятностных графических моделей** (Probabilistic Graphical Models, PGM) в главе 7: условные случайные поля (Conditional Random Fields, CRF). С помощью CRF можно смоделировать входную последовательность слов и отношения между признаками и метками в этой последовательности в виде *последовательного графа зависимостей*. В общем случае PGM может быть любым графом. **Граф** — это структура, состоящая из набора узлов и ребер, соединяющих пары узлов. Каждый узел в PGM представляет некоторую случайную переменную (значения которой могут наблюдаться или не наблюдаться), а ребра представляют условную зависимость одной случайной переменной от другой. Например, случайная переменная «влажность тротуара» зависит от случайной переменной «погодные условия». Наблюдая значения некоторых случайных переменных, алгоритм оптимизации способен извлечь из данных зависимости между наблюдаемыми и ненаблюдаемыми переменными.

Вероятностные графические модели позволяют аналитику увидеть, как значения одного признака зависят от значений других признаков. Если ребра графа зависимостей ориентированы, появляется возможность вывести причинно-следственную связь. К сожалению, создание таких моделей вручную требует значительного опыта в предметной области и глубокого понимания теории вероятностей и математической статистики. Последнее требование часто оказывается невыполнимым для многих экспертов в предметной области. Некоторые алгоритмы могут определять структуру зависимостей на основе данных, но полученные таким способом модели часто трудно поддаются интерпретации и поэтому они малополезны для понимания сложных вероятностных процессов, которые генерируют данные. На сегодняшний день CRF является наиболее часто используемой вероятностно графической моделью, применяемой в основном для обработки текста и изображений. Однако в этих двух областях этот метод уступил пальму первенства нейронным сетям. Другая графическая модель, **скрытая марковская модель** (Hidden Markov Model, HMM), в прошлом часто использовалась в задачах распознавания речи, анализе временных рядов и других задачах определения временны́х зависимостей, но HMM точно так же проиграла нейронным сетям.

Если вы все же решите узнать больше о PGM, то знайте, что они также известны под названиями байесовские сети, сети доверия и вероятностные сети независимости.

11.1.5. Методы Монте-Карло с цепями Маркова

Если при работе с графическими моделями возникает необходимость выбрать данные из очень сложного распределения, определяемого графом зависимостей, можно попробовать использовать алгоритмы **Монте-Карло с цепями Маркова** (Markov Chain Monte Carlo, MCMC). MCMC — это целый класс алгоритмов для выборки из любого распределения вероятностей, определенного математически. Когда мы говорили об **автокодировщиках с шумоподавлением**, мы отбирали шум из нормального распределения. Выборка из стандартных распределений, таких как нормальное или равномерное, осуществляется относительно просто, потому что их свойства хорошо известны. Однако задача выборки значительно усложняется, когда распределение вероятностей может иметь произвольную форму, определяемую сложной формулой.

11.1.6. Генеративно-состязательные сети

Генеративно-состязательные сети (Generative Adversarial Networks, GAN) — это класс нейронных сетей, использующих обучение без учителя. Они реализуются как система двух нейронных сетей, состязающихся друг с другом в условиях игры

с нулевой суммой. Самое популярное применение GAN — обучение созданию фотографий, которые для людей-наблюдателей выглядят достоверно. Первая из двух сетей принимает случайный вход (обычно гауссовский шум) и учится генерировать изображение в виде матрицы пикселов. Вторая сеть принимает на входе два изображения: одно «реальное», взятое из некоторой коллекции, и второе — сгенерированное первой сетью. Она должна научиться распознавать, какое из двух изображений сгенерировано первой сетью. Первая сеть получает отрицательные потери, если вторая распознала «поддельное» изображение.

Вторая сеть, в свою очередь, получает штраф, если не может распознать, какое из двух изображений является поддельным.

11.1.7. Генетические алгоритмы

Генетические алгоритмы (Genetic Algorithms, GA) — это методика численной оптимизации, используемая для оптимизации недифференцируемых целевых функций. В ней для поиска глобального оптимума (минимума или максимума) используются понятия эволюционной биологии и имитируются эволюционные биологические процессы.

При использовании генетических алгоритмов сначала создается первоначальное поколение решений-кандидатов. Если мы хотим найти оптимальные значения параметров модели, сначала случайным образом генерируется несколько комбинаций таких значений. Затем каждая комбинация проверяется по отношению к целевой функции. Представьте себе каждую комбинацию значений параметров как точку в многомерном пространстве. Затем из предыдущего поколения генерируется последующее поколение точек, с применением таких понятий, как «селекция», «пересечение» и «мутация».

В итоге это приводит к тому, что в каждом новом поколении сохраняется больше точек, схожих с теми, которые имелись в предыдущем поколении и которые показали лучшие результаты по отношению к цели. Точки в новом поколении, которые показали худшие результаты в предыдущем поколении, заменяются «мутациями» и «пересечениями» точек, показавших лучшие результаты. Мутация точки получается случайным искажением некоторых атрибутов исходной точки. Пересечение — это определенная комбинация нескольких точек (например, среднее значение).

Генетические алгоритмы позволяют находить решения для любых измеримых критериев оптимизации. Например, GA можно использовать для оптимизации гиперпараметров алгоритма обучения. Они обычно намного медленнее методов оптимизации на основе градиента.

11.1.7. Обучение с подкреплением

Как уже говорилось, обучение с подкреплением (Reinforcement Learning, RL) решает очень специфическую задачу, когда решения принимаются последовательно. Обычно агент действует в неизвестном окружении. Каждое действие приносит вознаграждение и переносит агента в другое состояние окружения (обычно в результате какого-то случайного процесса с неизвестными свойствами). Цель агента — оптимизировать долгосрочное вознаграждение.

Алгоритмы обучения с подкреплением, такие как Q-обучение (Q-learning), а также его аналоги на основе нейронной сети используются при обучении видеоиграм, роботизированной навигации и координации, управлению запасами и цепочками поставок, оптимизации сложных электроэнергетических систем (электросетей) и изучении финансовых торговых стратегий.

На этом книга заканчивается. Не забывайте время от времени посещать вики-страницу книги, чтобы быть в курсе событий в областях машинного обучения, рассмотренных в книге. Как я уже говорил в предисловии, благодаря постоянно обновляемой вики-странице эта книга, как хорошее вино, со временем становится только лучше.

11.2. Благодарности

Высокое качество этой книги было бы невозможно без редакторов-волонтеров. Я особенно благодарен следующим читателям за их систематический вклад: Мартейну ван Аттекуму (Martijn van Attekum), Даниэлю Мараини (Daniel Maraini), Али Азизу (Ali Aziz), Рэйчел Мак (Rachel Mak), Кельвину Сундли (Kelvin Sundli) и Джону Робинсону (John Robinson).

Также я очень благодарен за помощь: Кнуту Свердрупу (Knut Sverdrup), Фредди Дреннану (Freddy Drennan), Карлу У. Хэндлину (Carl W. Handlin), Абхиджиту Кумару (Abhijit Kumar), Лаззе Веддбарду (Lazze Veddbärd), Рикардо Рейсу (Ricardo Reis), Даниэлю Гроссу (Daniel Gross), Иогану Фаузи (Johann Faouzi), Акашу Агравалу (Akash Agrawal), Натанаэлю Вайлю (Nathanael Weill), Филипу Джекичу (Filip Jekic), Абишеку Бабуджи (Abhishek Babuji), Луану Виейре (Luan Vieira), Саяку Полу (Sayak Paul), Вахейду Уоллетсу (Vaheid Wallets), Лоренцо Буффони (Lorenzo Buffoni), Эли Фридман (Eli Friedman), Лукашу Мёдри (Łukasz Madry), Хаолану Циню (Haolan Qin), Бибеку Бехере (Bibek Behera), Дженнифер Купер (Jennifer Cooper), Нишанту Тяги (Nishant Tyagi), Денису Ахиярову (Denis Akhiyarov), Арону Джанарву (Aron Janarv), Александру Овчаренко, Рикардо Риосу (Ricardo Rios), Майклу Муллену (Michael Mullen), Мэтью Эдвардсу (Matthew Edwards), Дэвиду

Этлину (David Etlin), Манодж Баладжи Дж. (Manoj Balaji J), Давиду Руа (David Roy), Луису Феликсу (Luiz Felix), Ананду Мохану (Anand Mohan), Хади Сотуде (Hadi Sotudeh), Чарли Ньюи (Charlie Newey), Замиру Акимбекову, Хесусу Ренеро (Jesus Renero), Карану Гадия (Karan Gadiya), Мустафе Анил Дербенту (Mustafa Anıl Derbent), Джейкью Веенстра (JQ Veenstra), Жолту Крезису (Zsolt Kreisz), Яну Келли (Ian Kelly), Лукашу Заваде (Lukasz Zawada), Роберту Уэрхэму (Robert Wareham), Томасу Босману (Thomas Bosman), Льву Стивену (Lv Steven), Ариэлю Россаниго (Ariel Rossanigo), Майклу Лумпкинсу (Michael Lumpkins), Сесил Созуер (Secil Sozuer), Борису Куамбо (Boris Kouambo), И Джеону (Yi Jayeon), Тиму Флоку (Tim Flocke), Мохамеду Бехери (Mohamed Behery), Ане Фотине (Ana Fotina), Самину Иштяку (Samin Ishtiaq), Алексею Шматову, Кристиану Йеншу (Christian Jaensch) и Лучано Сегуре (Luciano Segura).

Алфавитный указатель

Л

М

Н

О

П

Андрей Бурков

Машинное обучение без лишних слов

Перевел с английского *А. Киселев*

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*К. Тульцева*
Литературный редактор	*А. Руденко*
Художественный редактор	*В. Мостипан*
Корректоры	*С. Беляева, М. Молчанова*
Верстка	*Л. Егорова*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 02.2020. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 23.01.20. Формат 70×100/16. Бумага офсетная. Усл. п. л. 15,480. Тираж 1700. Заказ 0000.

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР®
WWW.PITER.COM

ИЗДАТЕЛЬСКИЙ ДОМ «ПИТЕР» предлагает профессиональную, популярную и детскую развивающую литературу

Заказать книги оптом можно в наших представительствах

РОССИЯ

Санкт-Петербург: м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-83, 703-73-72; e-mail: sales@piter.com

Москва: м. «Электрозаводская», Семеновская наб., д. 2/1, стр. 1, 6 этаж
тел./факс: (495) 234-38-15; e-mail: sales@msk.piter.com

Воронеж: тел.: 8 951 861-72-70; e-mail: hitsenko@piter.com

Екатеринбург: ул. Толедова, д. 43а; тел./факс: (343) 378-98-41, 378-98-42;
e-mail: office@ekat.piter.com; skype: ekat.manager2

Нижний Новгород: тел.: 8 930 712-75-13; e-mail: yashny@yandex.ru; skype: yashny1

Ростов-на-Дону: ул. Ульяновская, д. 26
тел./факс: (863) 269-91-22, 269-91-30; e-mail: piter-ug@rostov.piter.com

Самара: ул. Молодогвардейская, д. 33а, офис 223
тел./факс: (846) 277-89-79, 277-89-66; e-mail: pitvolga@mail.ru,
pitvolga@samara-ttk.ru

БЕЛАРУСЬ

Минск: ул. Розы Люксембург, д. 163; тел./факс: +37 517 208-80-01, 208-81-25;
e-mail: og@minsk.piter.com

Издательский дом «Питер» приглашает к сотрудничеству авторов:
тел./факс: (812) 703-73-72, (495) 234-38-15; e-mail: ivanovaa@piter.com
Подробная информация здесь: http://www.piter.com/page/avtoru

Издательский дом «Питер» приглашает к сотрудничеству зарубежных торговых партнеров или посредников, имеющих выход на зарубежный рынок: *тел./факс: (812) 703-73-73; e-mail: sales@piter.com*

Заказ книг для вузов и библиотек:
тел./факс: (812) 703-73-73, доб. 6243; e-mail: uchebnik@piter.com

Заказ книг по почте: *на сайте www.piter.com; тел.: (812) 703-73-74, доб. 6216; e-mail: books@piter.com*

Вопросы по продаже электронных книг: *тел.: (812) 703-73-74, доб. 6217; e-mail: kuznetsov@piter.com*

КНИГА-ПОЧТОЙ

ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР» МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ:

- на нашем сайте: **www.piter.com**
- по электронной почте: **books@piter.com**
- по телефону: **(812) 703-73-74**

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ:

- Наложенным платежом с оплатой при получении в ближайшем почтовом отделении.
- С помощью банковской карты. Во время заказа вы будете перенаправлены на защищенный сервер нашего оператора, где сможете ввести свои данные для оплаты.
- Электронными деньгами. Мы принимаем к оплате Яндекс.Деньги, Webmoney и Qiwi-кошелек.
- В любом банке, распечатав квитанцию, которая формируется автоматически после совершения вами заказа.

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ДОСТАВКИ:

- Посылки отправляются через «Почту России». Отработанная система позволяет нам организовывать доставку ваших покупок максимально быстро. Дату отправления вашей покупки и дату доставки вам сообщат по e-mail.
- Вы можете оформить курьерскую доставку своего заказа (более подробную информацию можно получить на нашем сайте www.piter.com).
- Можно оформить доставку заказа через почтоматы,(адреса почтоматов можно узнать на нашем сайте www.piter.com).

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ:

- фамилию, имя, отчество, телефон, e-mail;
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру;
- название книги, автора, количество заказываемых экземпляров.

БЕСПЛАТНАЯ ДОСТАВКА:

- курьером по Москве и Санкт-Петербургу при заказе на сумму **от 2000 руб.**
- почтой России при предварительной оплате заказа на сумму **от 2000 руб.**