ВЕДЬ ЭТО ТАК ПРОСТО!

2-е издание

Python и наука о данных

для ЧАЙНИКОВ®

Издательство ДИАЛЕКТИКА

Изучите программирование на языке Python в целях анализа данных и статистики

Пишите код в облаке, используя Google Colab

Обменивайтесь данными и визуализируйте информацию

Джон Пол Мюллер
Лука Массарон

Авторы книг *Machine Learning for Dummies* и *Искусственный интеллект для чайников*

Python
и наука о данных

для
чайников®

Python® for Data Science

by John Paul Mueller and Luca Massaron

Python и наука о данных

Джон Пол Мюллер и Лука Массарон

для чайников®

Диалектика

Москва ◆ Санкт-Петербург
2020

ББК 32.973.26-018.2.75
М98
УДК 681.3.07

ООО "Диалектика"

Зав. редакцией *С.Н. Тригуб*
Перевод с английского и редакция *В.А. Коваленко*

По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info.dialektika@gmail.com, http://www.dialektika.com

Мюллер, Джон Пол, **Массарон**, Лука.

М98 Python и наука о данных для чайников, 2-е изд. : Пер. с англ. — СПб. : ООО "Диалектика", 2020. — 512 с. : ил. — Парал. тит. англ.

ISBN 978-5-907203-47-1 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание
Джон Пол Мюллер, Лука Массарон

Python и наука о данных для чайников

2-е издание

Подписано в печать 28.07.2020. Формат 70x100/16.
Гарнитура Times.
Усл. печ. л. 41,28. Уч.-изд. л. 25,9.
Тираж 200 экз. Заказ № 5175.

Отпечатано в АО "Первая Образцовая типография"
Филиал "Чеховский Печатный Двор"
142300, Московская область, г. Чехов, ул. Полиграфистов, д. 1
Сайт: www.chpd.ru, E-mail: sales@chpd.ru, тел. 8 (499) 270-73-59

ООО "Диалектика", 195027, Санкт-Петербург, Магнитогорская ул., д. 30, лит. А, пом. 848

ISBN 978-5-907203-47-1 (рус.)

ISBN 978-1-119-54762-4 (англ.)

Оглавление

Содержание

Об авторе

Лука Массарон — аналитик данных и директор по маркетинговым исследованиям, специализирующийся на многомерном статистическом анализе, машинном обучении и изучении ожиданий потребителей. Он имеет более чем десятилетний опыт в решении реальных проблем и консультациях заинтересованных лиц на основании обоснованных доводов, статистики, интеллектуального анализа данных и алгоритмов. Предпочитая простоту ненужной изощренности, он верит, что в науке о данных можно многого достичь, поняв и применив ее основы.

Джон Мюллер — внештатный автор и технический редактор. Умение писать у него в крови, на настоящий момент он является автором 111 книг и более чем 600 статей. Разнообразие тем распространяется от работы с сетями до искусственного интеллекта и от управления базами данных до автоматического программирования. Некоторые из его книг затрагивают темы науки о данных, машинного обучения и алгоритмов. Его технические навыки редактирования помогли более чем 70 авторам усовершенствовать содержимое своих произведений. Джон оказывает услуги технического редактирования различным журналам, консультирует и пишет сертификационные экзамены. Блог Джона находится по адресу `http://blog.johnmuellerbooks.com/`. В Интернете он доступен по адресу `John@JohnMuellerBooks.com`. У него также есть веб-сайт по адресу `http://www.johnmuellerbooks.com/` и представительство на Amazon (`https://www.amazon.com/John-Mueller/`).

Посвящение Луки

Я хотел бы посвятить эту книгу своим родителям, Ренцо и Лиции, которые любят простые и хорошо объяснимые идеи. И теперь, прочитав эту книгу, которую мы написали, они поймут куда больше о моей ежедневной работе в науке о данных, а также то, как этой новой дисциплине предстоит изменить способ нашего восприятия мира и наших действий в нем.

Посвящение Джона

Эта книга посвящена действительно творческим людям мира — тем, кому не нужно мыслить вне рамок, поскольку рамок для них вовсе не существует.

Благодарности Луки

Я очень благодарен моей семье, Юкико, Амелии, за их поддержку и терпение.

Я также благодарю всех моих товарищей по Kagglers за их помощь и постоянный обмен идеями и мнениями. В частности, Альберто Бошетти (Alberto Boschetti), Джулиано Янсона (Giuliano Janson), Бастиана Сьярдина (Bastiaan Sjardin) и Захарию Вульгарис (Zacharias Voulgaris).

Благодарности Джона

Благодарю мою жену Ребекку. Хотя ее уже нет, ее дух в каждой написанной мною книге и в каждом слове на каждой странице. Она верила в меня, когда не верил никто.

Рус Маллен заслужил благодарность за техническое редактирование этой книги. Он привнес точность и глубину в изложенный материал. Рус работал исключительно усердно, помогая в исследованиях для этой книги, находя труднодоступные URL-адреса и внося множество предложений.

Мэтт Вагнер, мой агент, заслуживший благодарности за то, что помог мне получить первый контракт и заботился обо всех подробностях, которые большинство авторов не учитывают. Я всегда ценю его помощь. Приятно знать, что кто-то хочет тебе помочь.

Многие люди прочитали всю эту книгу или ее часть, чтобы помочь мне усовершенствовать подход, проверить примеры и вообще предоставить отзыв о том, что им не понравилось. Эти добровольные помощники помогли слишком во многом, и я не могу не упомянуть здесь всех. Я особенно ценю усилия Евы Битти, Гленна А. Рассела и Маттео Малосетти, которые внесли общий вклад, прочитали всю книгу и самоотверженно посвятили себя этому проекту.

И наконец, я хотел бы поблагодарить Кэти Мор, Сьюзен Кристоферсен и остальных сотрудников редакционного и технического коллектива.

Введение

Данные все чаще используются для всевозможных целей, многие из которых ускользают от внимания, но каждый раз, когда вы заходите в Интернет, вы создаете их еще больше. Наука о данных превращает этот огромный объем данных в нечто полезное — то, что вы используете каждый день, чтобы решить набор различных задач или получить услуги от кого-то.

На самом деле вы, вероятно, уже используете науку о данных, причем такими способами, о которых даже не подозреваете. Например, когда вы использовали свою любимую поисковую систему этим утром для поиска чего-либо, она предложила альтернативные поисковые термины. Эти термины предоставлены наукой о данных. Когда на прошлой неделе вы обратились к врачу и оказалось, что обнаруженная опухоль не является раком, доктор, вероятно, сделал этот прогноз с помощью науки о данных. Таким образом, вы можете работать с наукой о данных каждый день и даже не знать об этом. Данная книга не только поможет вам начать использовать науку данных для решения множества практических задач, но также поможет понять, в скольких областях используется наука о данных. Зная, как решать задачи науки о данных и где ее применять, вы получаете значительное преимущество перед всеми остальными, увеличивая свои шансы на продвижение по службе или получение новой работы, которую вы действительно хотите.

О книге

Главная цель книги — устранить пугающий фактор и показать, что наука о данных не только действительно интересна, но и вполне выполнима с использованием языка Python. Возможно, вы думаете, что нужно быть гением информатики, чтобы решать сложные задачи, обычно связанные с наукой о данных, но это далеко от истины. Язык Python поставляется с множеством полезных библиотек, которые выполняют всю тяжелую работу за вас. Вы даже не понимаете, что происходит, и вам не нужно об этом заботиться. Все, что вам действительно нужно знать, — это то, что вы хотите решать конкретные задачи, и язык Python делает эти задачи достаточно доступными.

Частично акцент в книге сделан на использование правильных инструментов. Вы начнете с продукта Anaconda, который включает в себя IPython и Jupyter Notebook — два инструмента, которые исключают необходимость

работы с Python. Вы поэкспериментируете с IPython в полностью интерактивной среде. Код, который вы помещаете в Jupyter Notebook (называемый в этой книге также просто Notebook), обладает качеством презентации, и вы можете смешать несколько элементов презентации прямо в вашем документе. Это совсем не похоже на использование среды разработки. Чтобы упростить использование этой книги на альтернативных платформах, мы также рассмотрим такое приложение интерактивной среды разработки (Interactive Development Environment — IDE) как Google Colab, способное взаимодействовать с большинством, но не со всеми, примерами в книге, используя ваш любимый планшет или (при условии, что вы сможете достаточно сощуриться) смартфон.

В книге вы также обнаружите некоторые интересные приемы. Например, вы сможете создавать графики всех ваших экспериментов с данными, используя MatPlotLib. Здесь также подробно описаны доступные ресурсы (такие, как пакеты), и как вы можете использовать пакет Scikit-learn для выполнения действительно интересных вычислений. Многие хотели бы знать, как осуществлять распознавание рукописного ввода, и если вы один из них, то в данной книге получите представление о процессе.

Конечно, вас все еще могут беспокоить проблемы среды программирования, и эта книга не оставит вас в неведении. В начале вы найдете полные инструкции по установке Anaconda, за которыми следует описание методов, необходимых для начала работы с наукой о данных с использованием Jupyter Notebook или Google Colab. Основной упор делается на то, чтобы вы как можно быстрее начали работать и чтобы примеры были простыми и понятными, чтобы код не стал камнем преткновения для обучения.

Во втором издании книги представлены обновленные примеры использования Python 3.x — самой современной версии языка Python. Кроме того, упрощены примеры и добавлен материал по глубокому обучению.

Соглашения, принятые в книге

Здесь используются соглашения, общепринятые в компьютерной литературе.

- Новые термины в тексте выделяются *курсивом*. Чтобы обратить внимание читателя на отдельные фрагменты текста, также применяется *курсив*.
- Текст программ, функций, переменных, URL веб-страниц и другой код представлен моноширинным шрифтом.
- Все, что придется вводить с клавиатуры, выделено **полужирным моноширинным** шрифтом.

- Знакоместо в описаниях синтаксиса выделено *курсивом*. Это указывает на необходимость заменить знакоместо фактическим именем переменной, параметром или другим элементом, который должен находиться на этом месте: `BINDSIZE=(`*`максимальная ширина колонки`*`)*(`*`номер колонки`*`)`.
- Пункты меню и названия диалоговых окон представлены следующим образом: Menu Option (Пункт меню).

Текст некоторых абзацев выделен специальным стилем. Это примечания, советы и предостережения, которые обращают внимание на наиболее важные моменты в изложении материала и помогают избежать ошибок в работе.


СОВЕТ

Советы хороши тем, что позволяют сэкономить время или упростить решение некой задачи. Советы также содержат указатели на ресурсы, с которыми имеет смысл ознакомиться, чтобы получить максимум пользы от изучения языка Python применительно к науке о данных.


ВНИМАНИЕ!

Мы не хотим походить на строгих родителей или каких-то маньяков, но вам не следует делать то, что отмечено данной пиктограммой. В противном случае вы можете обнаружить, что ваше приложение не работает должным образом, вы получите неправильные ответы из, казалось бы, "пуленепробиваемых" уравнений или (в худшем случае) потеряете данные.


ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Увидев эту пиктограмму, знайте, что это дополнительный совет (или методика). Вы могли бы найти его очень полезным или слишком скучным, но он может содержать решение, необходимое для запуска программы. Пропускайте эти разделы, если хотите.


ЗАПОМНИ!

Если вы не вынесли ничего из какой-то главы или раздела, то запомните хотя бы материал, отмеченный этой пиктограммой. Здесь обычно содержится наиболее важная информация, которую следует знать для работы с языком Python или успешного решения задач, связанных с наукой о данных.

Глупые предположения

Возможно, вам будет трудно поверить, что мы о вас что-то думали — мы ведь даже не встречались! Хотя большинство предположений действительно глупы, мы сделали их, чтобы обеспечить некую отправную точку для книги.

Вы должны быть знакомы с платформой, которую хотите использовать, поскольку данная книга не предлагает никаких рекомендаций в этом отношении. (Однако в главе 3 приведены инструкции по установке Anaconda, а в главе 4 вы познакомитесь с Google Colab.) Чтобы предоставить вам максимум информации о языке Python и его применении в науке о данных, в этой книге не обсуждаются никакие проблемы, специфичные для платформы. Вам нужно знать, как устанавливать и использовать приложения и вообще работать с выбранной вами платформой, прежде чем начинать работу с этой книгой.

Вы должны знать, как работать с языком Python. В этой редакции книги больше не содержится учебника по языку Python, поскольку в Интернете вы можете найти множество таких учебных пособий (см. примеры на `https://www.w3schools.com/python/` и `https://www.tutorialspoint.com/python/`).

Настоящая книга не является учебником по математике. Да, в ней много примеров сложной математики, но основной упор делается на то, чтобы помочь вам использовать язык Python и науку о данных для решения задач анализа, а не преподавания математической теории. Главы 1 и 2 дадут вам лучшее представление о том, что нужно знать для успешного использования этой книги.

Мы также предполагаем, что вы можете получить доступ к Интернету. И в книге повсюду разбросаны многочисленные ссылки на сетевые материалы, которые расширят ваш опыт обучения. Но эти дополнительные источники полезны только в том случае, если вы их найдете и будете использовать.

Источники дополнительной информации

Эта книга — не конец вашего изучения языка Python или науки о данных, а только начало. Чтобы она стала для вас максимально полезной, мы предоставляем дополнительные источники информации. Получая от вас письма по электронной почте, мы сможем ответить на возникшие у вас вопросы, а также подсказать, как обновления Python или связанных с ним надстроек влияют на содержание книги. Вы также можете использовать следующие замечательные источники.

- **Шпаргалка.** Вы помните, как использовали шпаргалки в школе, чтобы получить лучшие оценки по контрольной? Да, это разновидность шпаргалки. В ней содержится ряд заметок о малоизвестных задачах, решаемых с помощью Python, IPython, IPython Notebook и науки о данных, известных не каждому. Шпаргалка находится в конце книги. В ней содержатся действительно полезная информация: наиболее

распространенные ошибки программирования, вызывающие у людей затруднения при использовании языка Python.

- **Обновления.** Рано или поздно все изменяется. Например, мы могли не заметить грядущих изменений, когда смотрели в свои хрустальные шары во время написания этой книги. Когда-то это просто означало, что книга устарела и стала менее полезной, но теперь вы можете найти ее обновления по адресу `www.dummies.com`, если будете искать по английскому названию этой книги. Кроме этих обновлений, имеет смысл посетить блог автора по адресу `http://blog.johnmuellerbooks.com/`, содержащий ответы на вопросы читателей и связанные с книгой полезные материалы.
- **Сопутствующие файлы.** Кто действительно хочет набрать весь код в книге и восстановить все эти графики вручную? Большинство читателей предпочитают тратить свое время на работу с Python, решение задач по науке о данных и просмотр чего-то интересного, а не на набор текста. К счастью для вас, примеры, используемые в книге, доступны для скачивания, поэтому все, что вам нужно сделать, — это прочитать книгу, чтобы изучить использование язык Python для методов науки о данных. Вы можете найти эти файлы на сайте `www.dummies.com`. Ищите по английскому названию этой книги и прокрутите вниз появившуюся страницу до изображения обложки книги и щелкните на нем. Затем щелкните на кнопке More about This Book (Подробнее об этой книге) и на открывшейся странице перейдите на вкладку Downloads (Загрузки). Сопутствующие файлы можно также загрузить со страницы русского издания книги по адресу: `http://www.dialektika.com/books/978-5-907203-47-1.html`

Что дальше

Если вы абсолютный новичок в Python и его использовании для задач по науке о данных, вам следует начать с главы 1 и продвигаться по книге со скоростью, позволяющей освоить как можно больше материала.

Если вы новичок, который спешит начать работу с языком Python для науки о данных как можно быстрее, вы можете перейти к главе 3, но имейте в виду, что некоторые темы впоследствии могут оказаться немного запутанными. Переход к главе 5 — это нормально, если у вас уже установлена Anaconda (программный продукт, описанный в книге), но обязательно просмотрите хотя бы главу 3, чтобы вы знали, какие предположения мы сделали при написании этой книги. Если вы планируете использовать для работы с этой книгой планшет, то

обязательно ознакомьтесь с главой 4, чтобы знать об ограничениях, накладываемых Google Colab при выполнении примера кода (не все примеры работают в этой IDE). Обязательно установите Anaconda с соответствующей версией Python 3.6.5, чтобы получить наилучшие результаты из исходного кода книги.

Читатели, которые знакомы с языком Python и имеют установленную программу Anaconda, могут сэкономить время, перейдя непосредственно к главе 5. В случае необходимости у вас всегда есть возможность вернуться к предыдущим главам. Однако вы должны понимать, как работает каждая техника, прежде чем переходить к следующей. Иначе вы можете пропустить жизненно важный момент, если начнете пропускать слишком много информации.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам электронное письмо либо просто посетить наш веб-сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Актуальность ссылок не гарантируется.

Наши электронные адреса:

E-mail: `info.dialektika@gmail.com`

WWW: `http://www.dialektika.com`

1 Приступая к работе с наукой о данных и языком Python

В ЭТОЙ ЧАСТИ...

- Как Python может упростить науку о данных
- Средства языка Python, общепринятые для науки о данных
- Создание собственной конфигурации Python
- Работа с Google Colab на альтернативных устройствах

Глава 1

Взаимосвязь науки о данных с языком Python

В ЭТОЙ ГЛАВЕ...

- Чудеса науки о данных
- Как работает наука о данных
- Связь между языком Python и наукой о данных
- Приступая к работе с языком Python

Наука о данных может показаться одной из тех технологий, которые вы никогда не используете, но вы ошибаетесь. Да, наука о данных включает использование передовых математических методов, статистики и больших данных. Тем не менее она оказывает также помощь в принятии осознанных решений, выработке рекомендаций для разных вариантов на основе предыдущих вариантов, помогает роботам видеть объекты. Фактически люди используют науку о данных столь разными способами, что трудно найти область деятельности, не подвергающуюся ее воздействию. Короче говоря, наука о данных стоит на грани опыта и чудес технологии. Без науки о данных многое из того, что вы принимаете сегодня как должное и вполне ожидаемое, было бы невозможно. В этом причина того, почему профессия аналитика данных столь популярна.

ЗАПОМНИ!

Для решения задач по науке о данных можно использовать любой из множества инструментов, но Python уникально подходит для упрощения работы. Во-первых, язык Python предоставляет невероятное количество связанных с математикой библиотек, которые помогают решать задачи при неполном пониманием того, что именно происходит. Однако язык Python идет дальше, поддерживая несколько стилей программирования (парадигмы программирования) и делая другие вещи, облегчающие вашу работу. Конечно, вы можете использовать другие языки для написания приложений для обработки данных, но Python — это естественный выбор для тех, кто действительно предпочитает работать умно.

Эта глава знакомит с языком Python. Несмотря на то что данная книга не является полным руководством по языку Python, изучение некоторых основных аспектов этого языка сократит время, необходимое для его освоения. (Если вам нужен хороший учебник для начинающих, обратитесь к книге Эрика Фримана *Учимся программировать с примерами на Python* (серия Head First, пер. с англ., изд. "Диалектика", 2020 г.). В этой книге содержатся ссылки на учебные пособия и другие вспомогательные средства, необходимые для заполнения пробелов в знаниях языка Python.

ВЫБОР ЯЗЫКА ДЛЯ НАУКИ О ДАННЫХ

В мире есть множество языков программирования, и большинство из них разработано для решения задач определенным образом или даже для облегчения выполнения конкретных работ. Выбор правильного инструмента упрощает жизнь. Специалисты по данным обычно используют лишь несколько языков, поскольку они облегчают работу с данными. Имея это в виду, рассмотрим основные языки для работы с данными в порядке предпочтения.

- **Python (общего назначения)**. Многие аналитики данных предпочитают использовать язык Python потому, что он предоставляет множество библиотек, таких как NumPy, SciPy, MatPlotLib, pandas и Scikit-learn, чтобы значительно упростить задачи по обработке данных. Язык Python является также точным языком, позволяющим легко использовать многопроцессорную обработку больших наборов данных, сокращая время, необходимое для их анализа. Сообщество аналитиков данных расширило также свои возможности, используя специализированные IDE, такие как Anaconda, реализующие концепцию Jupyter Notebook и значительно упрощающие работу с вычислениями в области науки о данных (в главе 3 демонстрируется, как использовать Jupyter Notebook, поэтому не беспокойтесь пока об этом). Помимо всего этого, в пользу Python говорит также то, что это отличный

язык для создания связующего кода с такими языками, как C/C++ и Fortran. Документация Python демонстрирует, как создавать необходимые расширения. Большинство пользователей Python полагаются на этот язык, чтобы искать шаблоны, например, позволяя роботу видеть группу пикселей как объект. Это также подходит для всех видов научных задач.

- **R (специальный статистический)**. Во многих отношениях языки Python и R имеют одинаковые функциональные возможности, но реализуют их по-разному. Эти языки имеют примерно одинаковое количество сторонников, а некоторые люди используют Python и R взаимозаменяемо (а иногда и в тандеме). В отличие от Python, язык R предоставляет собственную среду, поэтому вам не нужен сторонний продукт, такой как Anaconda. Тем не менее язык R не смешивается с другими языками с той легкостью, которую обеспечивает Python.
- **SQL (управление базой данных)**. Самое важное, что следует помнить о *языке структурированных запросов* (Structured Query Language — SQL), — это то, что он фокусируется на данных, а не на задачах. Предприятия не могут работать без хорошего управления данными, данные — это бизнес. Для хранения своих данных большие организации используют некую реляционную базу данных, доступ к которой обычно обеспечивает SQL. Большинство продуктов *систем управления базами данных* (Database Management System — DBMS) полагаются на язык SQL в качестве основного языка, но в DBMS, как правило, имеется множество встроенных средств анализа данных и других средств науки о данных. Поскольку вы обращаетесь к данным естественным способом, зачастую наблюдается значительный выигрыш в скорости при выполнении задач науки о данных. *Администраторы баз данных* (Database Administrator — DBA) обычно используют язык SQL для управления или манипулирования данными, а не для проведения их детального анализа. Тем не менее аналитик данных может использовать язык SQL для различных задач по обработке данных, а также сделать полученные сценарии доступными для администраторов баз данных.
- **Java (общего назначения)**. Некоторые аналитики данных решают другие виды задач программирования, требующие универсального, высоко адаптированного и популярного языка. Помимо предоставления доступа к большому количеству библиотек (большинство из которых на самом деле не так уж и полезны для науки о данных, но годятся для других нужд), язык Java лучше поддерживает объектную ориентацию, чем любой другой язык из этого списка. Кроме того, он строго типизирован и работает довольно быстро. Поэтому некоторые предпочитают именно его для окончательного кода. Язык Java не является хорошим выбором для экспериментов или специальных запросов.
- **Scala (общего назначения)**. Поскольку язык Scala использует *виртуальную машину Java* (Java Virtual Machine — JVM), у него есть некоторые преиму-

ства и недостатки языка Java. Но, как и Python, язык Scala обеспечивает мощную поддержку парадигмы функционального программирования, в основе которой лежит лямбда-исчисление. Кроме того, Apache Spark написан на языке Scala, а значит, при использовании этого языка есть хорошая поддержка кластерных вычислений; — считайте, огромная поддержка наборов данных. К некоторым из недостатков использования языка Scala относится то, что его трудно правильно настроить, у него крутая кривая обучения и в нем отсутствует полный набор библиотек, специализирующихся на науке о данных.

Популярная профессия

Когда-то любой статистик рассматривался, как своего рода бухгалтер. Многие считают статистику и анализ данных скучными. Тем не менее в науке о данных чем больше учишься, тем больше хочешь учиться. Ответ на один вопрос зачастую порождает еще больше вопросов, которые даже еще интереснее, чем тот, на который вы только что ответили. В следующих разделах вы найдете более подробную информацию о том, почему наука о данных является такой удивительной областью изучения.

Появление науки о данных

Наука о данных (data science) — относительно новый термин. Уильям С. Кливленд (William S. Cleveland) придумал его в 2001 году, в монографии *Data Science: An Action Plan for Expanding the Technical Areas of the Field of Statistics*. Только через год Международный совет по науке фактически признал науку о данных и создал соответствующий комитет. Колумбийский университет в 2003 году начал публикацию *Journal of Data Science*.

Математическая основа науки о данных имеет многовековую историю, поскольку она, по сути, представляет собой метод анализа статистики и вероятности. Впервые термин "статистика" упоминается в 1749 году, но сама статистика, безусловно, намного старше. Люди использовали статистику для поиска закономерностей на протяжении тысячелетий. Например, историк Фукидид (в своей *Истории Пелопоннесской войны*) описывает, как афиняне рассчитывали высоту стены Платеи в V веке до н.э., считая кирпичи в непокрытом участке стены. Поскольку счет должен был быть точным, афиняне взяли среднее значение из подсчетов нескольких солдат.

Процесс количественного определения и понимания статистики является относительно новым, но сама наука довольно древняя. Первая попытка документировать важность статистики относится к IX веку, когда Аль-Кинди написал манускрипт *О дешифровке криптографических сообщений*. В этой книге Аль-Кинди описывает, как использовать комбинацию статистики и частотного анализа для расшифровки зашифрованных сообщений. Даже в начале статистика использовалась в научном применении к задачам, которые казались практически неразрешимыми. Наука о данных продолжает этот процесс, и некоторым людям это может показаться магией.

Основные компетенции аналитика данных

Подобно любому, кто совершает сегодня сложные сделки, аналитику данных требуются знания и широкий спектр навыков. На самом деле требуется так много разных навыков, что аналитики данных зачастую работают в командах. Кто-то, кто хорош в сборе данных, может объединиться с тем, кто одарен в представлении информации. Было бы трудно найти одного человека со всеми необходимыми навыками. Ниже описаны области, в которых аналитик данных мог бы преуспеть (хотя чем шире спектр навыков, тем лучше).

- **Сбор данных**. Неважно, какие у вас математические навыки, если вы не можете получить данные для анализа. Процесс сбора данных начинается с работы с источником данных, что требует навыков управления базами данных. Однако во многих ситуациях необработанные данные не особенно полезны — необходимо понимать предметную область, чтобы взглянуть на данные и сформулировать вопросы, которые нужно задать. Наконец, следует обладать навыками моделирования данных, чтобы понять, как данные связаны и структурированы ли они.
- **Анализ**. Получив данные для работы и понимания их сложности, вы можете приступить к анализу данных. Вы проводите некий анализ, используя базовые навыки статистики, очень похожие на те, которые почти каждый получает в колледже. Однако использование специализированных математических приемов и алгоритмов может сделать шаблоны в данных более очевидными или помочь вам сделать выводы, которые вы не можете сделать, просматривая одни только данные.
- **Представление**. Большинство людей плохо понимают цифры и не могут видеть шаблоны, которые видит аналитик данных. Важно обеспечить графическое представление этих шаблонов, чтобы помочь другим увидеть то, что означают цифры, и как их применять осмысленно. Что еще важней, презентация должна рассказать конкретную историю, чтобы влияние данных не было потеряно.

Связь между наукой о данных, большими данными и искусственным интеллектом

Интересно, что перемещение данных таким образом, чтобы кто-то мог выполнить их анализ, — это специальность ETL (Extract, Transformation, and Loading — *извлечение, преобразование и загрузка*). Специалист ETL использует языки программирования, такие как Python, для извлечения данных из нескольких источников. Корпорации, как правило, не хранят данные в одном легко доступном месте, поэтому поиск данных, необходимых для проведения анализа, занимает много времени. После того как специалист ETL найдет данные, язык программирования или другой инструмент преобразует их в единый формат, пригодный для анализа. Процесс загрузки имеет много форм, но эта книга для решения данной задачи опирается на язык Python. В сложных реальных ситуациях для решения этой задачи вы можете использовать такие инструменты, как Informatica, MS SSIS или Teradata.


ЗАПОМНИ!

Наука о данных не обязательно является средством достижения цели, она вполне может быть только шагом на этом пути. Поскольку аналитик данных работает с различными наборами данных и находит интересные факты, эти факты могут выступать в качестве исходных данных для других видов анализа и приложений искусственного интеллекта. Например, ваши покупательские привычки способны подсказать, какие книги вам могут понравиться или куда вы хотели бы поехать в отпуск. Покупки или другие привычки могут также помочь людям понять друг друга, а иногда и в менее благородных действиях. Книги *Machine Learning For Dummies* и *Искусственный интеллект для чайников*, написанные Джоном Мюллером и Лукой Массароном (Wiley), помогут вам понять эти и другие области применения науки о данных. А пока подумайте о том, что изученное вами в этой книге может оказать определенное влияние на ваш карьерный путь, который может найти множество других направлений.

Роль программирования

Аналитику данных, возможно, необходимо знать несколько языков программирования для достижения конкретных целей. Например, вам может потребоваться знание языка SQL для извлечения данных из реляционных баз данных. Язык Python поможет решать задачи загрузки, преобразования и анализа данных. Однако вы можете выбрать такой продукт, как MATLAB (у которого есть собственный язык программирования) или PowerPoint (который зависит

от VBA), чтобы предоставить информацию другим. (Если вам интересно сравнить MATLAB с Python, прочитайте мою книгу *MATLAB For Dummies*, опубликованную издательством John Wiley & Sons, Inc.). Огромные наборы данных, на которые опираются ученые, зачастую требуют нескольких уровней обработки для преобразования в полезные обработанные данные. Решение этих задач вручную занимает много времени и склонно к ошибкам, поэтому программирование представляет собой лучший способ создания согласованного, пригодного для использования источника данных.

Учитывая количество продуктов, которые используют большинство аналитиков данных, может оказаться невозможным использование только одного языка программирования. Да, Python способен загружать данные, преобразовывать их, анализировать и даже предоставлять их конечному пользователю, но он работает только тогда, когда язык обеспечивает требуемую функциональность. Возможно, вам придется выбрать другие языки, чтобы дополнить ваш инструментарий. Языки, которые вы выбираете, зависят от ряда критериев. Ниже приведено несколько факторов, которые следует учитывать.

- Как вы собираетесь использовать науку данных в своем коде (у вас есть ряд задач, таких как анализ данных, классификация и регрессия).
- Ваше знакомство с языком.
- Необходимость взаимодействия с другими языками.
- Наличие инструментов для улучшения среды разработки.
- Наличие API и библиотек, облегчающих решение задач.

Создание конвейера науки о данных

Наука о данных — это отчасти искусство, отчасти — инженерия. Нахождение закономерностей в данных, рассмотрение вопросов и определение того, какие алгоритмы работают лучше всего, — это часть искусства науки о данных. Но чтобы сделать часть искусства науки о данных реализуемой, инженерная часть должна опираться на определенный процесс достижения конкретных целей. Этот процесс представляет собой конвейер науки о данных, требующий от аналитика данных выполнения определенных действий при подготовке, анализе и представлении данных. Следующие разделы помогут вам лучше понять сам конвейер науки о данных, а также его применение в примерах этой книги.

Подготовка данных

Данные, к которым вы получаете доступ из различных источников, редко представлены в форме, уже готовой к анализу, скорее наоборот. Необработанные данные могут не только существенно различаться по формату, но вам может также потребоваться преобразовать их, чтобы сделать все источники данных связными и доступными для анализа. Преобразование может потребовать изменения типов данных, порядка следования данных и даже создания записей данных на основе информации, предоставленной существующими записями.

Предварительный анализ данных

Математические подходы, лежащие в основе анализа данных, основаны на инженерных принципах в том смысле, что результаты доказуемы и последовательны. Тем не менее наука о данных предоставляет доступ к множеству статистических методов и алгоритмов, которые помогут вам обнаружить закономерности в данных. Единый подход обычно не работает. Как правило, вы используете для обработки данных итеративный процесс, причем с разных точек зрения. Использование метода проб и ошибок является частью искусства науки о данных.

Изучение данных

По мере применения различных методов статистического анализа и алгоритмов для выявления закономерностей вы начинаете понимать данные. Данные могут рассказывать вовсе не ту историю, о которой вы первоначально думали, или могут рассказать много историй. Исследование является частью изучения данных. На самом деле это очень интересная часть науки о данных, поскольку вы никогда не можете знать заранее, что именно покажут вам данные.


ЗАПОМНИ!

Конечно, неточная природа данных и поиск в них, казалось бы, случайных закономерностей означает непредвзятость. Если у вас есть предвзятое представление о том, что содержат данные, вы не найдете ту информацию, которую они действительно содержат. Вы упускаете фазу изучения этого процесса, что приводит к упущенным возможностям как для вас, так и для людей, которые зависят от вас.

Визуализация

Визуализация означает просмотр шаблонов, найденных в данных, а также возможность реагировать на эти шаблоны. Это также означает возможность видеть, когда данные не являются частью шаблона. Считайте себя скульптором

данных — удаляя данные, которые лежат вне шаблонов (выбросы), вы позволяете другим увидеть шедевр информации. Да, вы можете видеть шедевр, но пока другие не увидят его, он останется только в вашем *в*идении.

Осознание сути и смысла данных

Может показаться, что аналитик данных просто ищет уникальные методы просмотра данных. Но процесс не завершится, пока у вас не будет четкого понимания того, что означают данные. Выводы, получаемые вами от манипулирования и анализа данных, помогают решать реальные задачи. Например, вы можете использовать результаты анализа для принятия бизнес-решения.

В некоторых случаях результат анализа автоматически создает ответ. Например, когда робот просматривает серию пикселей, полученных из камеры, формирующие объект пиксели имеют особое значение, и программирование робота может диктовать своего рода взаимодействие с этим объектом. Однако, до тех пор, пока специалист по данным не создаст приложение, которое сможет загружать, анализировать и визуализировать пиксели из камеры, робот вообще ничего не увидит.

Роль языка Python в науке о данных

При наличии правильных источников данных, требований к анализу и представлению вы можете использовать язык Python для любой части конвейера данных. В каждом примере книги используется язык Python, чтобы помочь вам понять другую часть уравнения науки о данных. Из всех языков, которые вы можете выбрать для решения задач по науке о данных, Python является наиболее гибким и удобным, поскольку поддерживает множество сторонних библиотек, предназначенных для этой задачи. Следующие разделы помогут вам лучше понять, почему язык Python является хорошим выбором для многих (если не для большинства) потребностей в науке о данных.

Смещение профиля аналитика данных

Некоторые люди считают аналитика данных неприступным ботаником, который совершает с данными чудеса при помощи математики. Аналитик данных — это человек за кулисами, похожий на волшебника Оз. Но эта точка зрения меняется. Во многих отношениях аналитика данных считают теперь дополнением к разработчику или как разработчика нового типа. Суть этого изменения заключается в том, что доминирующими теперь являются приложения способные обучаться. Чтобы приложение могло учиться, оно должно

иметь возможность манипулировать большими базами данных и обнаруживать в них новые шаблоны. Кроме того, приложение должно иметь возможность создавать новые данные, основанные на старых данных, делая своего рода информированное предсказание. Новые виды приложений влияют на людей таким образом, который казался научной фантастикой всего несколько лет назад. Конечно, наиболее заметные из этих приложений определяют поведение роботов, которые завтра будут взаимодействовать с людьми гораздо теснее, чем сегодня.

С точки зрения бизнеса необходимость объединения науки о данных и разработки приложений очевидна: предприятия должны выполнять различные виды анализа огромных баз данных, которые они собрали, чтобы понять и использовать информацию для прогнозирования будущего. Однако, по правде говоря, гораздо большее влияние объединения этих двух отраслей науки (науки о данных и разработки приложений) будет ощущаться при создании совершенно новых видов приложений, некоторые из которых даже невозможно представить сегодня. Например, новые приложения могут помочь учащимся лучше учиться, анализируя их тенденции в обучении, и создавать новые учебные методы, которые лучше подходят для конкретного ученика. Эта комбинация наук может также решить множество медицинских задач, которые сегодня невозможно решить, причем не только в борьбе с болезнями, но и в ходе решения таких задач, как создание действительно полезных протезов, которые выглядят и действуют как настоящие конечности.

Работа с многоцелевым, простым и эффективным языком

Существует много способов решения задач науки о данных, но в данной книге описан только один из них. Тем не менее язык Python представляет собой одно из немногих универсальных решений, которые вы можете использовать для решения сложных задач науки о данных. Вместо того чтобы использовать несколько инструментов для решения задачи, вы можете использовать только язык Python, чтобы выполнить всю работу. Отличие Python заключается в большом количестве научных и математических библиотек, созданных для него третьими лицами. Подключение этих библиотек значительно расширяет язык Python и позволяет ему легко решать задачи, которые могут решать и другие языки, но с большими трудностями.

СОВЕТ

Библиотеки Python являются его основным преимуществом. Но язык Python предлагает куда больше, чем повторное использование кода. Самое важное, что нужно знать о языке Python, — это то, что он поддерживает четыре разных стиля программирования.

- **Функциональный**. Рассматривает каждое утверждение как математическое уравнение и избегает любых форм состояния или изменяемых данных. Основным преимуществом этого подхода является отсутствие побочных эффектов. Кроме того, этот стиль программирования лучше, чем другие, подходит для параллельной обработки, поскольку не нужно учитывать состояния. Многие разработчики предпочитают этот стиль программирования для рекурсии и лямбда-исчисления.
- **Императивный**. Выполняет вычисления как прямое изменение состояния программы. Этот стиль особенно полезен при манипулировании структурами данных, он создает элегантный, но простой код.
- **Объектно-ориентированный**. Опирается на поля данных, которые рассматриваются как объекты и обрабатываются только с помощью заранее определенных методов. Язык Python не полностью поддерживает эту форму программирования, поскольку он не может реализовать такие функции, как сокрытие данных. Тем не менее это полезный стиль программирования для сложных приложений, поскольку он обеспечивает инкапсуляцию и полиморфизм. Этот стиль программирования также способствует повторному использованию кода.
- **Процедурный**. Рассматривает задачи как пошаговые итерации, в которых общие задачи помещаются в функции, вызываемые по мере необходимости. Этот стиль программирования поддерживает итерации, последовательность, выбор и модульность.

Быстро учимся использовать Python

Пришло время опробовать язык Python и увидеть конвейер данных в действии. В следующих разделах представлен краткий обзор процесса, который подробно исследуется в остальной части книги. На самом деле в следующих разделах вы не будете решать задачи. Вы не будете устанавливать Python до главы 3, поэтому пока только читайте. В этой книге используется определенная версия языка Python и IDE под названием Jupyter Notebook, поэтому, пожалуйста, подождите до главы 3, чтобы установить эти функции (или, если настаиваете, установите их сейчас). На данном этапе не беспокойтесь о понимании каждого аспекта процесса. Цель этих разделов — помочь понять, как использовать язык Python для решения задач науки о данных. Многие детали могут показаться вам трудными для понимания на данный момент, но остальная часть книги поможет вам понять их.

ЗАПОМНИ!

Примеры этой книги основаны на веб-приложении Jupyter Notebook. Снимки экрана, представленные в этой и других главах, отражают внешний вид Jupyter Notebook в браузере Firefox на операционной системе Windows 7. То, что вы увидите, будет содержать те же данные, но фактический интерфейс может немного отличаться в зависимости от используемой платформы (например, при использовании ноутбука вместо настольной системы), операционной системы и браузера. Не беспокойтесь, если увидите небольшие различия между вашим дисплеем и снимками экрана в книге.

СОВЕТ

Вам не нужно вводить исходный код, приведенный в этой главе, вручную. Намного проще, если вы используете загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле исходного кода `P4DS4D2_01_Quick_Overview.ipynb`.

Загрузка данных

Прежде чем вы сможете что-либо сделать, вам нужно загрузить некоторые данные. Книга демонстрирует все виды методов для решения этой задачи. В данном случае на рис. 1.1 показано, как загрузить набор данных Boston, содержащий цены на жилье и другие факты о недвижимости в районе Бостона. Код помещает весь набор данных в переменную `boston`, а затем помещает части этих данных в переменные с `X` и `y`. Считайте переменные емкостями для хранения. Переменные важны, поскольку они позволяют работать с данными.

Рис. 1.1. Загрузка данных в переменные, чтобы вы могли манипулировать ими

Обучение модели

Теперь, когда у вас есть данные для работы, вы можете с ними что-то сделать. Все виды алгоритмов встроены в Python. Рис. 1.2 демонстрирует модель линейной регрессии. Опять же, не волнуйтесь о том, как именно это работает, линейная регрессия подробно обсуждается в последующих главах. На рис. 1.2 важно отметить, что язык Python позволяет выполнять линейную регрессию, используя всего два оператора, и помещать результат в переменную `hypothesis`.

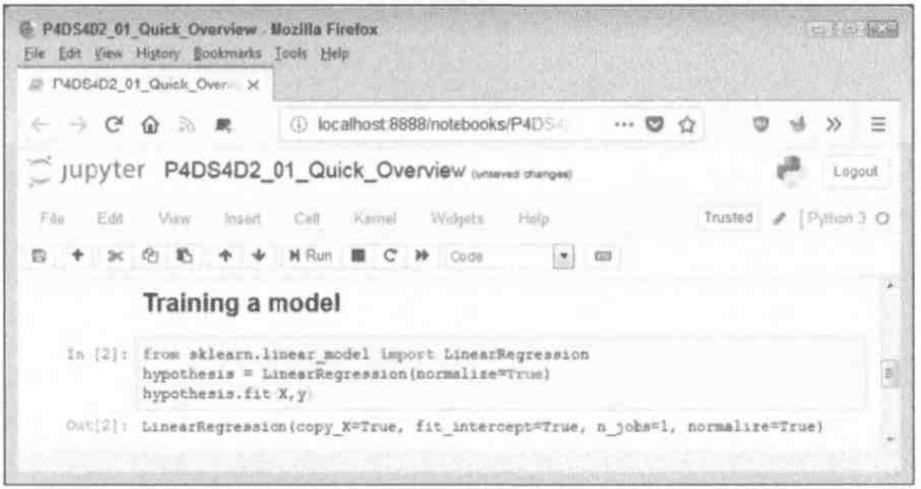


Рис. 1.2. Использование содержимого переменной для обучения модели линейной регрессии

Просмотр результата

Выполнение анализа любого вида не окупается, если вы не получаете какой-либо выгоды от него в виде результата. В этой книге показаны всевозможные способы просмотра результатов, но рис. 1.3 начинает с чего-то простого: вы видите коэффициент, полученный из анализа линейной регрессии.

СОВЕТ

Одна из причин, по которой в этой книге используется Jupyter Notebook, заключается в том, что этот продукт помогает создавать красиво отформатированные выходные данные. Снова посмотрите на рис. 1.3, и вы увидите отчет, который можно просто распечатать и предложить коллеге. Такой вывод не подходит для многих людей, но те, кто имеет опыт работы с Python и наукой о данных, найдут его весьма полезным и информативным.

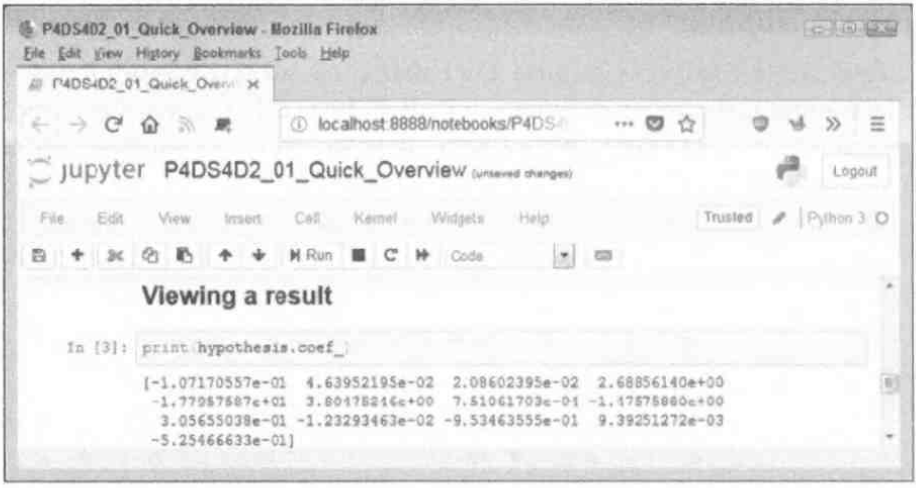


Рис. 1.3. Вывод результата как ответа модели

Глава 2

Возможности и чудеса языка Python

В ЭТОЙ ГЛАВЕ...

- Почему появился Python
- Быстрое начало работы с языком Python
- Особенности Python
- Возможности языка Python для аналитика данных

Все компьютеры работают только на одном языке — машинном коде. Но если вы не хотите научиться разговаривать, как компьютер, нулями и единицами, то машинный код не особенно полезен. Вы никак не захотите пытаться определять задачи науки о данных, используя машинный код. Потребовалась бы целая жизнь (возможно, не одна), чтобы определить одну задачу. Высокоуровневые языки позволяют писать много кода, который люди могут понять довольно быстро. Используемые этими языками инструменты позволяют преобразовать читаемый человеком код в машинный. Поэтому выбор языков зависит от потребностей человека, а не от потребности машины. С учетом этого данная глава знакомит вас с возможностями, предоставляемыми языком Python, которые делают его практическим выбором для аналитика данных. В конце концов, вы ведь хотите знать, почему в этой книге используется язык Python, а не другой язык, такой как Java или C++. Эти языки являются идеальным выбором для некоторых задач, но они не настолько подходят для удовлетворения потребностей аналитика данных.

Глава начинается с краткой истории языка Python, так что в первую очередь вы узнаете, почему разработчики создали его. Вы также увидите несколько

простых примеров на языке Python, чтобы получить представление о нем. В рамках изучения в этой главе вы обнаружите все виды интересных функций, которые предоставляет язык Python. Он предоставляет доступ к множеству библиотек, которые особенно подходят для потребностей аналитика данных. Фактически вы будете использовать несколько этих библиотек, когда будете работать с примерами кода. Знание возможностей этих библиотек поможет вам понять примеры кода, а также то, почему эта книга демонстрирует решение некоторых задач определенным образом.

Несмотря на то что в этой главе приведены примеры работы с языком Python, по-настоящему вы начнете использовать его только в главе 6. В данной главе представлен только обзор, чтобы вы могли лучше понять, на что способен язык Python. В главе 3 будет показано, как установить конкретную версию языка Python, используемую для этой книги. В главах 4 и 5 рассказывается об инструментах, которые вы можете использовать, а в главе 4 особое внимание уделяется Google Colab — альтернативной среде программирования. Короче говоря, если вы не совсем понимаете пример, приведенный в этой главе, не беспокойтесь: в последующих главах вы получите много дополнительной информации.

Почему Python?

Python — это в*и*дение одного человека, Гвидо ван Россума, который приступил к созданию этого языка в декабре 1989 года в качестве замены языка ABC. Доступно немного информации о точных целях языка Python, но он сохраняет способность ABC создавать приложения с использованием меньшего количества кода. В то же время, он намного превосходит возможности языка ABC по созданию приложений всех типов, и в отличие от ABC может похвастаться четырьмя стилями программирования. Короче говоря, Гвидо взял язык ABC в качестве отправной точки, обнаружил, что он ограничен, и создал новый язык без этих ограничений. Это пример создания нового языка, который действительно лучше его предшественника.

Язык Python прошел несколько итераций и в настоящее время имеет два пути развития. Путь 2.*x* обратно совместим с предыдущими версиями Python, а путь 3.*x* — нет. Проблема совместимости связана с тем, как наука о данных использует язык Python, поскольку некоторые библиотеки не будут работать с версией 3.*x*. Тем не менее эта проблема постепенно решается, и вы должны использовать версию 3.*x* для всех новых разработок, поскольку версии 2.*x*

скоро придет конец (см. подробнее на https://pythonclock.org/). Таким образом, в этом издании используется код версии 3.*x*. Кроме того, в некоторых версиях используется разное лицензирование, поскольку во время разработки языка Python Гвидо работал в разных компаниях. Вы можете увидеть список версий и соответствующих лицензий по адресу https://docs.python.org/3/license.html. Всеми текущими версиями языка Python владеет организация Python Software Foundation (PSF), поэтому, если вы не используете более старую версию, не беспокойтесь о проблеме лицензирования.

ИСПОЛЬЗОВАНИЕ ДЛЯ РАБОТЫ ПРАВИЛЬНОГО ЯЗЫКА

Компьютерные языки позволяют систематически и понятно записывать инструкции. На самом деле компьютеры не понимают компьютерные языки и для инструкций используют машинный код. Причина, по которой языки так важны, заключается в том, что люди не понимают машинный язык, поэтому переход от того, что понимают люди, к тому, что понимают машины, очень важен. Python предоставляет определенный набор функций, которые облегчают написание приложений науки о данных. Подобно любым другим языкам, он предоставляет набор инструментов, подходящих для одних ситуаций, но не для других. Используйте язык Python (или любой другой), если он предоставляет функции, необходимые для выполнения вашей задачи. Если вы начинаете находить возможности языка недостаточными, значит, пришло время выбрать другой язык, поскольку компьютер, в конце концов, не заботит, какой именно язык вы используете. Компьютерные языки — для людей, а не наоборот.

Базовая философия языка Python

Фактически Гвидо начал разработку языка Python как проект Skunk works. Основная идея заключалась в том, чтобы как можно быстрее создать язык Python, но создать гибкий язык, работающий на любой платформе и обеспечивающий значительный потенциал для расширения. Python предоставляет все эти функции и многое другое. Конечно, на дороге всегда есть ухабы, например, определение того, какую часть базовой системы следует предоставить. Вы можете прочитать больше о философии дизайна языка Python по адресу http://python-history.blogspot.com/2009/01/pythons-design-philosophy.html, а об истории языка по адресу http://python-history.blogspot.com/2009/01/introduction-and-overview.html также предоставляет некоторую полезную информацию.

Вклад в науку о данных

Поскольку эта книга о науке о данных, вы, вероятно, задаетесь вопросом: как язык Python способствует улучшению науки о данных и что на самом деле означает слово "*лучше*" в данном случае. Знание того, что многие организации используют язык Python, не поможет, поскольку в действительности это мало что говорит о том, как они его используют, и если вы хотите согласовать выбор языка со своими конкретными потребностями, то ответ на вопрос, как они используют язык Python, становится важным.

Один из таких примеров приведен по адресу `https://www.datasciencegraduateprograms.com/python/`. В этой статье рассказывается об интернет-ресурсе Forecastwatch.com (`https://forecastwatch.com/`), который следит за погодой и пытается сделать прогнозы лучше. Каждый день Forecastwatch.com сравнивает 36 000 прогнозов с фактической погодой, а затем использует результаты для создания лучших прогнозов. Попытка агрегировать и разобраться в данных о погоде для 800 городов США утомительна, поэтому Forecastwatch.com нуждался в языке, который мог бы делать то, что ему нужно, с наименьшей суетой. Ниже приведены причины, по которым ресурс Forecast.com выбрал язык Python.

- **Поддержка библиотек**. Язык Python обеспечивает поддержку множества библиотек, куда больше, чем когда-либо понадобится любой организации. Согласно `https://www.python.org/about/success/forecastwatch/`, Forecastwatch.com счел библиотеки регулярных выражений, потоков, сериализации объектов и сжатия данных особенно полезными.
- **Параллельная обработка**. Каждый из прогнозов обрабатывается как отдельный поток, чтобы система могла быстро обработать их. Данные потока включают в себя URL-адрес веб-страницы, которая содержит требуемый прогноз, а также информацию о категории, такую как название города.
- **Доступ к данным**. Этот огромный объем данных не может существовать в памяти, поэтому Forecast.com использует базу данных MySQL, доступ к которой осуществляется через библиотеку MySQLdb (`https://sourceforge.net/projects/mysql-python/`), которая является одной из немногих библиотек, еще не перешедших на Python 3.*x* Тем не менее соответствующий веб-сайт обещает необходимую поддержку в ближайшее время.
- **Отображение данных**. Первоначально вывод Forecastwatch.com создавал язык PHP. Но с помощью Quixote (`https://www.mems-exchange.org/software/quixote/`), представляющего собой структуру отображения, ресурс Forecastwatch.com смог перенести все на язык Python.

Настоящие и будущие цели развития

Исходные цели разработки языка Python не совсем соответствуют тому, что произошло с языком впоследствии, с тех пор, как Гвидо изначально задумал его. Первоначально Гвидо планировал использовать язык Python в качестве второго языка для тех разработчиков, которым необходимо было создать одноразовый код, но которые не могли достичь своих целей, используя язык сценариев. Первоначальной целевой аудиторией для языка Python были разработчики на языке C. Вы можете прочитать об этих первоначальных целях в интервью на `http://www.artima.com/intv/pyscale.html`.

Сегодня можно найти множество приложений, написанных на языке Python, поэтому идея использовать его исключительно для написания сценариев не осуществилась. (См. списки приложений Python на `https://www.python.org/about/apps/` и `https://www.python.org/about/success/`.) Он продолжает развиваться, поскольку разработчики видят в этом один из лучших способов создания современных приложений, многие из которых опираются на науку о данных.

Естественно, что после всех этих успешных историй люди с энтузиазмом относятся к дополнениям к языку Python. Вы можете найти списки *предложений по улучшению Python* (Python Enhancement Proposals — PEP) по адресу `http://legacy.python.org/dev/peps/`. Эти списки PEP могут увидеть свет, а могут и не увидеть, но они доказывают, что Python — это живой, растущий язык, который будет по-прежнему предоставлять функции, необходимые разработчикам для создания великолепных приложений всех типов, а не только для науки о данных.

Работа с языком Python

Эта книга не содержит полного руководства по языку Python. (Представление об этом см. в книге Эрика Фримана *Учимся программировать с примерами на Python* (серия Head First). На данный момент полезно получить представление о языке Python и о том, как с ним взаимодействовать.

СОВЕТ

Вам не нужно вводить исходный код этой главы вручную. Намного проще, если вы будете использовать загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_02_Using_Python.ipynb`.

Знакомство с языком

Python предназначен для предоставления четких формулировок языка, причем делает он это в невероятно компактном пространстве. Одна строка кода Python может выполнять задачи, которые на другом языке обычно занимают несколько строк. Например, если вы хотите отобразить нечто на экране, вы указываете Python вывести это, например:

```
print("Hello There!")
```


СОВЕТ

Это пример функции 3.*x* `print()`. (Версия Python 2.*x* включает как форму функции `print`, для которой нужны круглые скобки, так и форму инструкции `print`, в которой круглые скобки отсутствуют.) Выше, в разделе "Почему Python?", упоминались некоторые различия между версиями 2.*x* и 3.*x*. Если вы используете функцию `print()` без скобок в версии 3.*x*, то получите сообщение об ошибке:

```
    File ">Jupyter-input-1-fe18535d9681>", line 1
    print "Hello There!"
                       ^
SyntaxError: Missing parentheses in call to 'print'. Did you
    mean print("Hello There!")?
```

Дело в том, что вы можете просто указать Python вывести текст, объект или что-то еще, используя простое выражение. Вам не нужно слишком много в плане передовых навыков программирования. Если хотите завершить сеанс, используя среду командной строки, такую как IDLE, достаточно ввести **`quit()`** и нажать клавишу <Enter>. В этой книге рассматривается среда Jupyter Notebook, которая действительно делает ваш код похожим на чей-то блокнот.

Необходимость отступа

Python использует отступы для создания различных языковых элементов, таких как условные операторы. Одна из наиболее распространенных ошибок, с которыми сталкиваются разработчики, — это отсутствие правильного отступа в коде. Вы увидите этот принцип в действии позже, а сейчас просто обращайте внимание на отступы при работе с примерами, приведенными в книге. Например, вот оператор `if` (который выполняет следующий за ним код, если условие истинно) с правильным отступом:

```
if 1 > 2:
    print("1 is less than 2")
```


ВНИМАНИЕ!

Оператор `print` должен отображаться с отступом под условным оператором. В противном случае условие не будет работать должным образом, и вы также можете увидеть сообщение об ошибке.

Работа в командной строке или в IDE

Anaconda — это продукт, упрощающий использование языка Python. Он поставляется с набором утилит, которые помогут работать с языком Python разнообразными способами. Эта книга в основном опирается на Jupyter Notebook, являющийся частью комплекта Anaconda, которую вы установите в главе 3. Вы видели, как этот редактор использовался в главе 1, и вы увидите его снова позже. Фактически в этой книге вообще не описываются какие-либо другие утилиты Anaconda. Тем не менее они существуют и иногда полезны в работе с Python. В следующих разделах представлен краткий обзор других утилит Anaconda для создания кода Python. Вы можете поэкспериментировать с ними, работая над различными методами программирования.

ПАКЕТ ANACONDA

В книге Anaconda рассматривается как единый продукт. Фактически вы устанавливаете и взаимодействуете с Anaconda, как и с любым другим продуктом. Но фактически Anaconda — это несколько приложений с открытым исходным кодом. Вы можете использовать эти приложения по отдельности или в комбинации для достижения конкретных задач программирования. В большинстве книг для решения задач используется одно приложение — Jupyter Notebook. Однако имеет смысл узнать и о других приложениях, входящих в состав Anaconda, чтобы максимально эффективно использовать этот продукт в целом.

Множество аналитиков данных полагаются на пакет продуктов Anaconda, поэтому он и используется в этой книге. Однако вы можете обнаружить, что при отдельной загрузке некоторые из продуктов с открытым исходным кодом поступают в новой форме. Например, Jupyter Notebook поставляется в более новой форме, чем в комплекте с Anaconda (`http://jupyter.org/`). Из-за различий в версиях Jupyter Notebook вам необходимо установить ту его версию, которая указана в этой книге, что означает использование пакета Anaconda, а не отдельной загрузки.

Создание новых сеансов с помощью командной строки Anaconda

Только одна из утилит Anaconda обеспечивает прямой доступ к командной строке — это Anaconda Prompt. После запуска этой утилиты вы увидите командную строку, в которой можете вводить команды. Главным преимуществом этой утилиты является то, что вы можете запустить Anaconda с любым из ее переключателей и изменить стандартную среду этой утилиты. Конечно, вы запускаете большинство утилит с помощью интерпретатора Python, к которому обращаетесь с помощью команды `python.exe`. (Если в вашей системе установлены Python 3.6 и Python 2.7 и вы открыли обычную командную строку или окно терминала, то может запуститься версия Python 2.7 вместо версии Python 3.6, поэтому всегда лучше открыть командную строку Anaconda для гарантии того, что вы получили правильную версию Python.) Таким образом, вы можете просто набрать **`python`** и нажать <Enter>, чтобы запустить копию интерпретатора Python, если захотите. На рис. 2.1 показан простой интерпретатор Python.

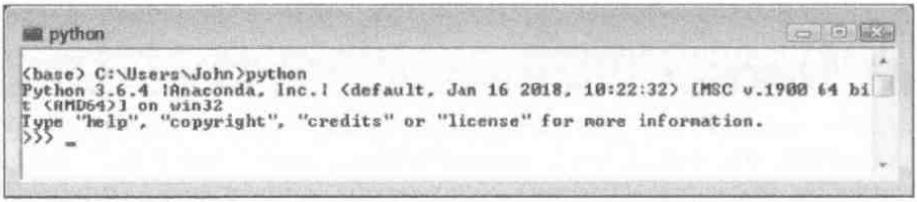


Рис. 2.1. Простой интерпретатор Python

Чтобы выйти из интерпретатора, введите **`quit()`** и нажмите клавишу <Enter>. Вернувшись в командную строку, набрав **`python -?`** и нажав <Enter>, вы можете найти список ключей командной строки для `python.exe`. На рис. 2.2 показаны только некоторые из способов изменения среды интерпретатора Python.

По желанию, запустив интерпретатор с правильным сценарием, вы можете создать измененную форму любой из утилит, предоставляемых Anaconda. Сценарии появляются в подкаталоге `scripts`. Например, введите **`python Anaconda3/scripts/Jupyter-script.py`** и нажмите клавишу <Enter>, чтобы запустить среду Jupyter без использования графической команды для вашей платформы. Вы также можете добавить аргументы командной строки для дальнейшего изменения поведения сценария. При работе с этим сценарием вы можете получить информацию о Jupyter, используя следующие аргументы командной строки.

Рис. 2.2. Интерпретатор Python включает в себя все виды параметров командной строки

- `--version`. Отображает версию используемого Jupyter Notebook.
- `--config-dir`. Отображает каталог конфигурации Jupyter Notebook (где хранится информация о конфигурации).
- `--data-dir`. Отображает место хранения данных приложения Jupyter, а не проектов. Проекты обычно появляются в вашей пользовательской папке.
- `--runtime-dir`. Отображает расположение файлов времени выполнения Jupyter, которые обычно хранятся в подкаталоге каталога данных.
- `--paths`. Создает список путей, по которым Jupyter настроен для использования.

Следовательно, если хотите получить список путей Jupyter, введите **python Anaconda3/scripts/Jupyter-script.py --paths** и нажмите клавишу <Enter>. Подкаталог `scripts` также содержит множество исполняемых файлов, которые нередко являются скомпилированными версиями сценариев, — так они могут выполняться быстрее. Если хотите запустить среду браузера Jupyter Notebook, либо введите **python Anaconda3/scripts/Jupyter-notebook-script.py** и нажмите клавишу <Enter>, чтобы использовать версию сценария, либо запустите `Jupyter-notebook.exe`. Результат будет одинаков в любом случае.

Вход в среду IPython

Среда Interactive Python (IPython) является улучшением стандартного интерпретатора Python. Для запуска этой среды в приглашении Anaconda используется команда `IPython`, а не стандартная команда `Python`. Основное назначение среды, показанной на рис. 2.3, — помочь использовать Python с наименьшими затратами труда. Обратите внимание, что версия Python — та же, что и при использовании команды `Python`, но версия IPython отличается, и вы увидите другое приглашение. Чтобы увидеть эти улучшения (см. рис. 2.3), введите **%quickref** и нажмите клавишу <Enter>.

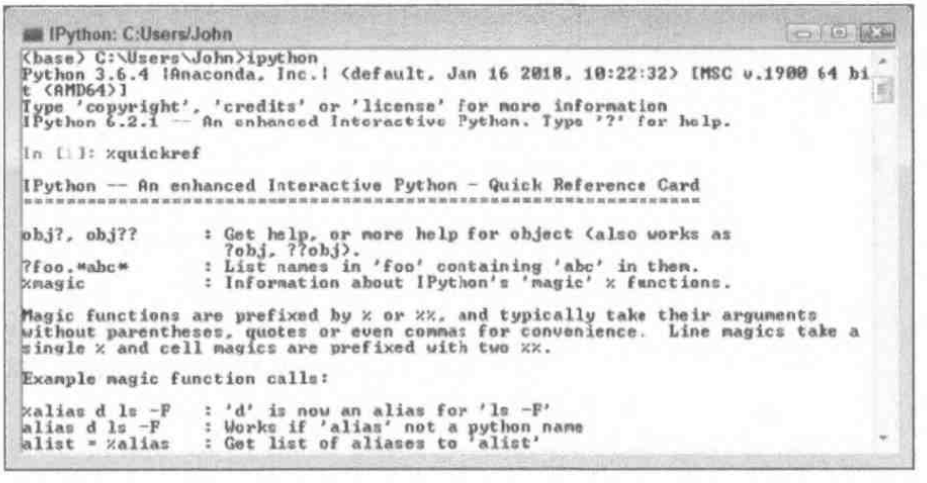


Рис. 2.3. Среда Jupyter проще в использовании, чем стандартный интерпретатор Python

Одним из наиболее интересных дополнений IPython является полнофункциональная команда очистки экрана (`cls`). Вы не можете легко очистить экран при работе в интерпретаторе Python, а это означает, что через некоторое время все будет заполнено. Здесь также можно осуществлять поиск переменных с использованием подстановочных знаков. Далее вы узнаете, как использовать

магические функции для решения таких задач, как фиксация времени, необходимого для выполнения задачи (для оптимизации).

Вход в среду Jupyter QTConsole

Трудно запомнить команды и функции Python, а запомнить расширенные дополнения Jupyter — еще сложнее. Кто-то даже может сказать, что это невозможно (и, вполне вероятно, что они правы). Jupyter QTConsole добавляет к Jupyter *графический пользовательский интерфейс* (GUI), что значительно упрощает использование улучшений, которые предоставляет Jupyter.


СОВЕТ

Если вы использовали предыдущие версии Anaconda, у вас может сложиться впечатление, что среда QTConsole отсутствует, но она все же есть, нет только метода прямого доступа. Чтобы запустить QTConsole, откройте приглашение Anaconda, введите **`Jupyter QTConsole`** и нажмите клавишу <Enter>. Вы увидите запуск QTConsole, как показано на рис. 2.4. Конечно, вы потеряете немного экранного пространства, чтобы получить эту функцию, и некоторым программистам не нравится идея использовать графический интерфейс, поэтому вам придется выбирать, с какой средой работать.

Некоторые из расширенных команд отображаются в меню в верхней части окна. Все, что вам нужно сделать, — это выбрать команду, которую вы хотите использовать. Например, чтобы перезапустить ядро, выберите пункт меню Kernel⇨Restart Current Kernel (Ядро⇨Перезапустить текущее ядро). У вас также есть такой же доступ к командам IPython. Например, введите **`%magic`** и нажмите клавишу <Enter>, чтобы увидеть список магических команд.

Редактирование сценариев с использованием Spyder

Spyder — это полнофункциональная *интегрированная среда разработки* (Integrated Development Environment — IDE), которая используется для загрузки и редактирования сценариев, запуска и выполнения отладки. На рис. 2.5 показана стандартная оконная среда.

IDE Spyder очень похожа на любую другую IDE, которую вы уже могли использовать. В левой части находится редактор, в котором вводится код. Любой созданный вами код помещается в файл сценария, и вы должны сохранить его перед запуском. В верхнем правом окне находятся различные вкладки для проверки объектов, изучения переменных и взаимодействия с файлами. В правом нижнем окне находятся консоль Python, журнал истории и консоль Jupyter. Вверху отображаются пункты меню для решения всех задач, которые вы обычно связываете с работой в IDE.

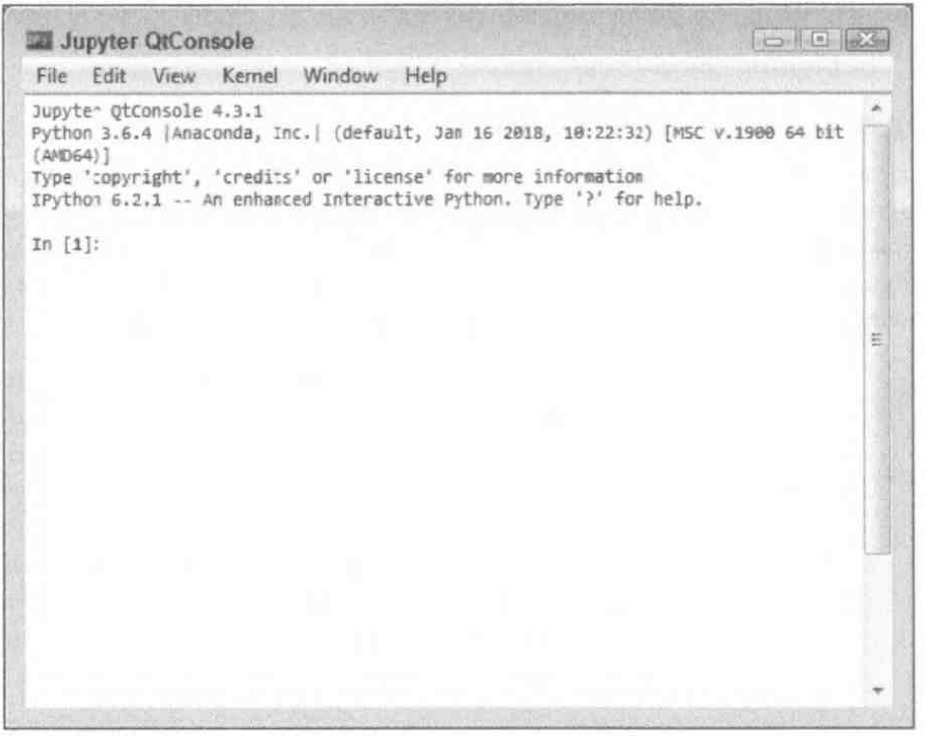


Рис. 2.4. Использование QTConsole для упрощения работы с Jupyter

Быстрое создание прототипа и эксперименты

Создание проекта приложения в коде без обязательного заполнения всех деталей — это *создание прототипа* (prototyping). Python использует меньше кода, чем другие языки, поэтому создание прототипа происходит быстрее. Многие из необходимых вам действий уже определены в составе библиотек загружаемых в память, поэтому осуществляются они очень быстро.

Наука о данных не опирается на статические решения. Возможно, вам придется опробовать несколько решений, чтобы найти то из них, которое работает лучше всего. После создания прототипа вы используете его для экспериментов с различными алгоритмами, чтобы определить, какой из них лучше всего работает в конкретной ситуации. Алгоритм, который вы используете, зависит от ответов, которые вы видите, и данных, которые используете, поэтому приходится учитывать слишком много переменных.

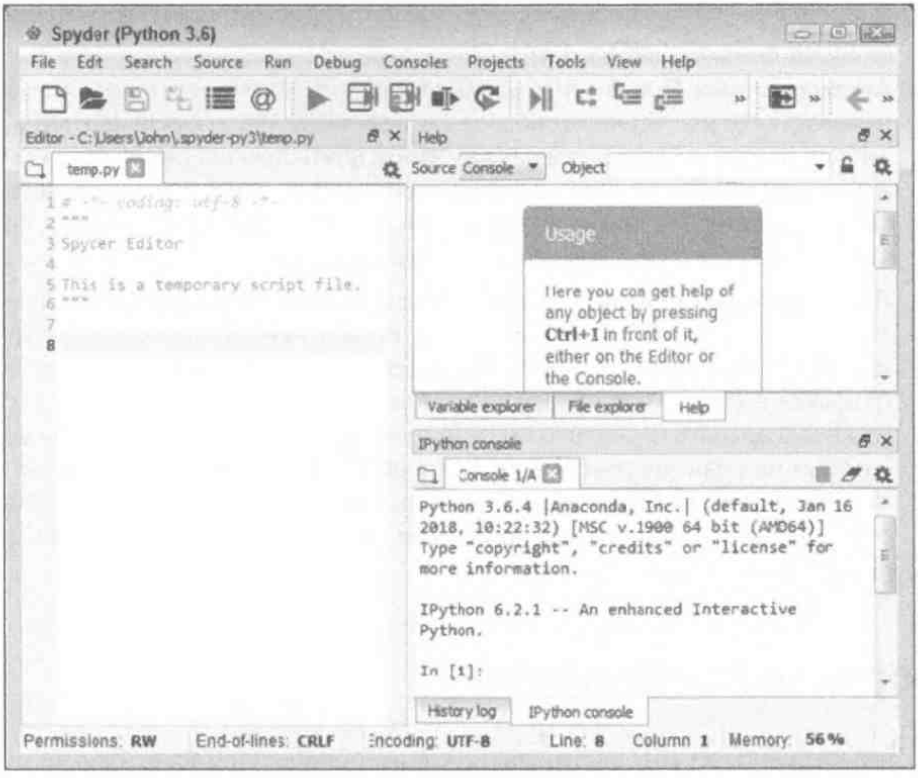


Рис. 2.5. IDE Spyder

ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Процесс создания прототипа и эксперименты с ним осуществляются в несколько этапов. Эти этапы имеют различное назначение и следуют в определенном порядке. В следующем списке они приведены в том порядке, в котором их обычно выполняют.

1. **Построение конвейера данных**. Для работы с данными необходимо создать конвейер. Некоторые данные можно загрузить в память. Но после того, как набор данных достигнет определенного размера, вам придется начать работать с ним на диске или использовать другие средства для взаимодействия с ним. Техника, которую вы используете для получения доступа к данным, важна, поскольку она влияет на то, как быстро вы получите результат.
2. **Форматирование**. Форма данных — это способ их отображения и характеристики (например, тип данных), они важны при проведении анализа. Чтобы сравнивать яблоки с яблоками, данные должны быть одинаковыми. Однако одного лишь форматирования данных недостаточно. Форма должна быть подходящей для алгоритмов, которые используются для анализа. Последующие

главы (начиная с главы 7) помогут вам понять необходимость формирования данных различными способами.

3. **Анализ данных**. При анализе данных вы редко используете один алгоритм и называете его достаточно хорошим. Вы не можете знать, какой алгоритм даст такие же результаты в самом начале. Чтобы найти лучший результат из своего набора данных, вы можете поэкспериментировать с ним, используя несколько алгоритмов. Эта практика подчеркивается в последующих главах книги, когда вы начнете проводить серьезный анализ данных.
4. **Представление результата**. Картинка стоит тысячи слов, по крайней мере, так говорят. Тем не менее вам нужно изображение, чтобы сказать правильные слова, или ваше сообщение потеряется. Используя функции построения графиков в стиле MATLAB, предоставляемую библиотекой `matplotlib`, вы можете создавать несколько презентаций одних и тех же данных, каждая из которых по-разному графически описывает данные. Чтобы убедиться в том, что смысл действительно не потерян, вы должны поэкспериментировать с различными методами представления и определить, какой из них работает лучше всего.

Скорость выполнения

Компьютеры известны своим мастерством обработки чисел. Тем не менее анализ требует значительной вычислительной мощи. Наборы данных настолько велики, что вы можете перегрузить даже невероятно мощную систему. В целом на скорость выполнения вашего приложения по обработке данных влияют следующие факторы.

- **Размер набора данных**. Наука о данных во многих случаях опирается на огромные наборы данных. Да, вы можете заставить робота видеть объекты, используя набор данных довольно скромного размера, но когда дело доходит до принятия бизнес-решений, как правило, лучше подойдет размер побольше. Размер вашего набора данных частично определяет тип приложения, но размер набора данных зависит также от размера исходных данных. Недооценка влияния размера набора данных смертельно опасна в приложениях науки о данных, особенно в тех, которые должны работать в режиме реального времени (например, автомобили с автоматическим управлением).
- **Метод загрузки**. Метод, который вы используете для загрузки анализируемых данных, имеет решающее значение, и вы всегда должны использовать самые быстрые средства, имеющиеся в вашем распоряжении, даже если это означает обновление вашего

оборудования. Работа с данными в памяти всегда быстрее, чем с данными, хранящимися на диске. Доступ к локальным данным всегда быстрее, чем к данным в сети. Решение задач по науке о данных, основанных на доступе к Интернету через веб-службы, — возможно, самый медленный метод из всех. В главе 6 методы загрузки рассматриваются более подробно. Позже вы также узнаете о влиянии техники загрузки.

- **Стиль программирования**. Некоторые люди, вероятно, попытаются сказать вам, что парадигмы программирования Python делают написание медленного приложения практически невозможным. Они не правы. Любой может создать медленное приложение, используя любой язык, если применять такие методы программирования, которые не позволяют использовать функциональные возможности языка программирования наилучшим образом. Чтобы создавать быстрые приложения для обработки данных, следует использовать лучшие методы программирования. Методы, продемонстрированные в этой книге, являются отличной отправной точкой.
- **Возможности машины**. Запуск приложений для обработки данных в системе с ограниченным объемом памяти или медленным процессором невозможен. Система, которую вы используете, должна иметь лучшее оборудование, которое вы можете себе позволить. Учитывая, что приложения для обработки данных связаны как с процессором, так и с диском, вы не сможете срезать углы в любой области и ожидать отличных результатов.
- **Алгоритм анализа**. Используемый вами алгоритм определяет тип получаемого результата и контролирует скорость выполнения. Многие из глав в последних частях этой книги демонстрируют несколько методов для достижения цели с использованием различных алгоритмов. Тем не менее вы все равно должны экспериментировать, чтобы найти лучший алгоритм для вашего конкретного набора данных.


ЗАПОМНИ!

В нескольких главах этой книги уделяется особое внимание производительности, прежде всего скорости и надежности, поскольку оба фактора имеют решающее значение для приложений, работающих с данными. Несмотря на то что приложения баз данных имеют тенденцию в некоторой степени подчеркивать необходимость в скорости и надежности, сочетание отличных возможностей доступа к набору данных (задачи диска) и анализа данных (задачи процессора) в приложениях науки о данных делает необходимость правильного выбора еще более важным.

Сила визуализации

Язык Python позволяет исследовать окружение науки о данных, не прибегая к использованию отладчика или кода отладки, как этого потребовали бы многие другие языки. Оператор `print` (или функция, в зависимости от используемой версии Python) и функция `dir()` позволяют интерактивно исследовать любой объект. Короче говоря, вы можете загрузить что-нибудь и поиграть с ним некоторое время, чтобы увидеть, как разработчик сделал это. Игра с данными, визуализация того, что это значит для вас лично, зачастую может помочь получить новое понимание и выработать новые идеи. Судя по многим разговорам в Интернете, игра с данными — это та часть науки о данных, которую специалисты в ней находят наиболее забавной.

Вы можете играть с данными, используя любой из инструментов в составе Anaconda, но одним из лучших инструментов для работы является IPython (см. раздел "Вход в среду IPython" ранее в этой главе), поскольку вам действительно не нужно слишком беспокоиться об окружении и ничего из сделанного вами не является постоянным. В конце концов, вы играете с данными. Таким образом, вы можете загрузить набор данных, чтобы увидеть, что он может предложить, как показано на рис. 2.6. Не беспокойтесь, если сейчас этот код выглядит непонятно. Начиная с главы 4, вы начнете больше играть с кодом, а различные разделы дадут больше деталей. Вы также можете обратиться к книге Джона Пола Мюллера *Beginning Programming with Python For Dummies*, 2-е издание (Wiley), если вам нужен более подробный учебник. На данный момент просто следуйте концепции игры с данными.

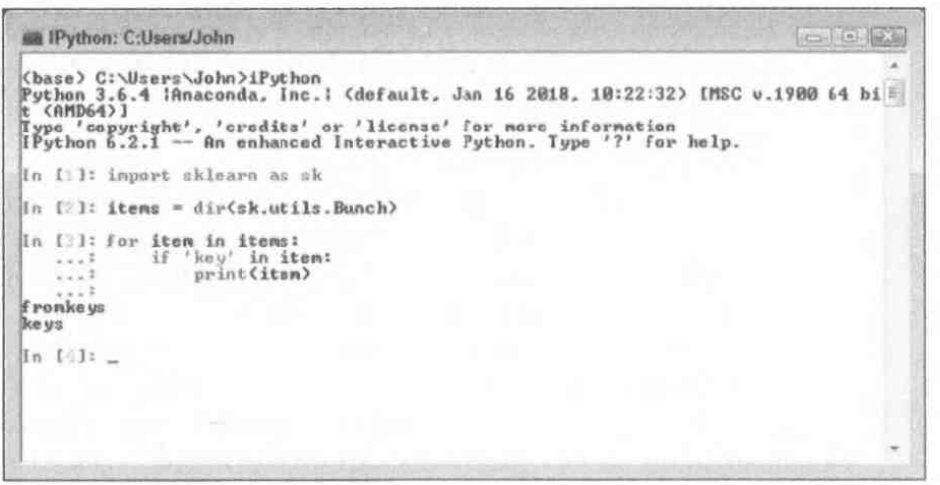


Рис. 2.6. Загрузите набор данных и поиграйте с ним немного

Наборы данных Scikit-learn появляются в *связках* (bunch) (связка — это некая структура данных). Когда вы импортируете набор данных, он будет иметь определенные функции, которые вы можете использовать с ним, они определяются кодом, используемым для определения структуры данных, это и есть связка. Этот код демонстрирует, какие функции работают с *ключами* (key) — идентификаторами данных для их *значений* (value) (один или несколько столбцов информации) в наборе данных. Каждая строка в наборе данных имеет уникальный ключ, даже если значения в этой строке повторяют другую строку. Вы можете использовать эти функции для выполнения некой работы с набором данных в составе вашего приложения.

Прежде чем вы сможете работать с набором данных, вы должны предоставить доступ к нему в локальной среде. На рис. 2.7 показан процесс импорта, а также использование функции `keys()` для отображения списка ключей, которые вы можете использовать для доступа к данным в наборе данных.

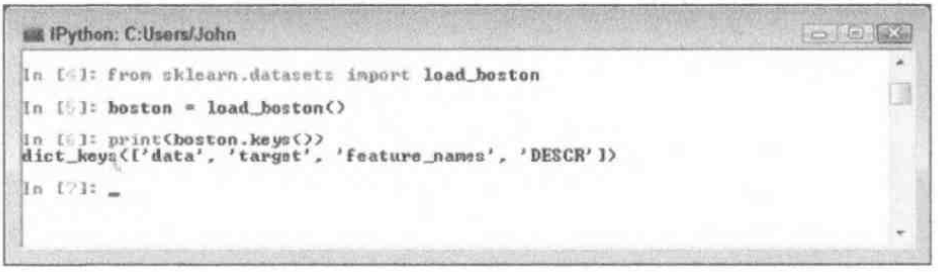


Рис. 2.7. Используйте функцию, чтобы получить больше информации

Когда у вас есть список ключей, доступных для использования, вы можете получать доступ к отдельным элементам данных. Например, на рис. 2.8 показан список всех имен объектов, содержащихся в наборе данных Boston. Python действительно позволяет узнать достаточно много о наборе данных, прежде чем вам придется работать с ним.

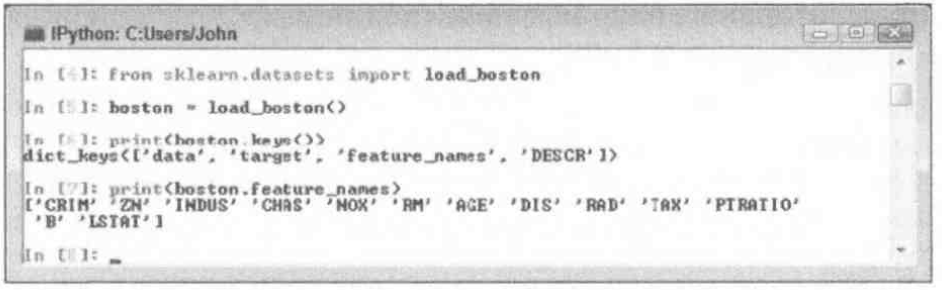


Рис. 2.8. Доступ к конкретным данным с помощью ключа

Использование экосистемы Python для науки о данных

Вы уже знаете о необходимости загрузки библиотек для выполнения задач науки о данных с использованием языка Python. В следующих разделах представлен обзор библиотек, которые используются для примеров задач в этой книге. Библиотеки в действии демонстрируют различные примеры книги.

Доступ к научным инструментам с помощью SciPy

Набор SciPy (`http://www.scipy.org/`) содержит множество других библиотек, которые вы также можете скачать по отдельности. Эти библиотеки обеспечивают поддержку для математики, науки и техники. Приобретая SciPy, вы получаете набор библиотек, предназначенных для совместной работы по созданию приложений различного рода. Вот эти библиотеки:

- NumPy
- SciPy
- matplotlib
- Jupyter
- Sympy
- pandas

Сама библиотека SciPy фокусируется на числовых подпрограммах, таких как подпрограммы для числовой интеграции и оптимизации. SciPy — это библиотека общего назначения, которая обеспечивает функции для нескольких проблемных доменов. Она обеспечивает также поддержку специфических для доменов библиотек, таких как Scikit-learn, Scikit-image и statsmodels.

Фундаментальные научные вычисления с использованием NumPy

Библиотека NumPy (`http://www.numpy.org/`) предоставляет средства для выполнения манипуляций с *n*-мерными массивами, что крайне важно для работы с данными. Набор данных Boston, используемый в примерах глав 1 и 2, является примером *n*-мерного массива, и вы не сможете легко получить к нему доступ без функций NumPy, которые включают поддержку линейной алгебры, преобразование Фурье и генерацию случайных чисел (см. список функций на `http://docs.scipy.org/doc/numpy/reference/routines.html`).

Анализа данных с использованием библиотеки pandas

Библиотека pandas (`http://pandas.pydata.org/`) предоставляет поддержку структур данных и инструменты анализа данных. Библиотека оптимизирована для быстрого и эффективного выполнения задач обработки данных. Основной принцип библиотеки pandas заключается в поддержке анализа и моделирования данных для языка Python, аналогичных другим языкам, таким как R.

Реализация машинного обучения с использованием Scikit-learn

Библиотека Scikit-learn (`http://scikit-learn.org/stable/`) является одной из ряда библиотек Scikit, которые основаны на возможностях, предоставляемых NumPy и SciPy, и позволяет разработчикам Python выполнять задачи, специфичные для домена. В данном случае библиотека фокусируется на интеллектуальном анализе данных и предоставляет доступ к следующим видам функций:

- классификация;
- регрессия;
- кластеризация;
- уменьшение размеров;
- выбор модели;
- предварительная обработка.

Набор этих функций представлен в виде названий глав в книге[1]. В результате вы можете предположить, что Scikit-learn — это самая важная библиотека для книги (хотя в ней используются и другие библиотеки).

Глубокое обучение с использованием Keras и TensorFlow

Keras (`https://keras.io/`) — это интерфейс прикладного программирования (API), который используется для моделей глубокого обучения. API зачастую определяет только модель для выполнения чего-либо, но не обеспечивают реализацию. Следовательно, для выполнения полезной работы нужна реализация Keras, и именно здесь вступает в игру TensorFlow (`https://www.tensorflow.org/`). Для реализации Keras вы также можете использовать Microsoft Cognitive Toolkit, CNTK (`https://docs.microsoft.com/en-us/cognitive-toolkit/`) или Theano (`https://github.com/Theano`), но эта книга посвящена TensorFlow.

[1] Во всяком случае, некоторые. — *Примеч. ред.*

Работая с API, вы ищете способы упростить нечто. Интерфейс Keras облегчает работу следующими способами.

- **Согласованный интерфейс**. Интерфейс Keras оптимизирован для общих случаев применения с акцентом на действенную обратную связь, для исправления ошибок пользователя.
- **Подход Lego**. Использование подхода "черного ящика" упрощает создание моделей, соединяя настраиваемые строительные блоки, лишь с несколькими ограничениями на то, как их можно соединять.
- **Расширяемость**. Вы легко можете добавлять собственные строительные блоки для выражения новых идей для исследований, которые включают новые слои, функции потерь и модели.
- **Параллельная обработка**. Чтобы быстро выполнять приложения, нужна хорошая поддержка параллельной обработки. Keras работает как на CPU, так и на GPU.
- **Прямая поддержка Python**. Вам не нужно делать ничего особенного, чтобы реализация Keras в TensorFlow работала с Python, что может стать главным камнем преткновения при работе с другими видами API.

Графический вывод данных с использованием matplotlib

Библиотека matplotlib (`http://matplotlib.org/`) предоставляет интерфейс, похожий на MATLAB, для создания презентаций данных, полученных в результате выполняемого анализа. В настоящее время библиотека ограничена двумерным выводом, но все же предоставляет средства для графического выражения шаблонов, которые вы видите в анализируемых вами данных. Без этой библиотеки вы не смогли бы создать вывод, который могли бы легко понять люди вне сообщества, занимающегося наукой о данных.

Создание графиков с помощью NetworkX

Для правильного изучения взаимосвязи между сложными данными в сетевой системе (например, той, которая используется вашей системой GPS для поиска маршрутов по улицам города) необходима библиотека для создания, обработки и изучения структуры сетевых данных различными способами. Кроме того, библиотека должна предоставлять средства для вывода полученного результата анализа в такой понятной людям форме, как графические данные. NetworkX (`https://networkx.github.io/`) позволяет выполнять такой анализ. Преимущество NetworkX в том, что узлы могут быть чем угодно (включая изображения), а ребра могут содержать произвольные данные. Эти функции

NetworkX позволяют выполнять гораздо более широкий спектр задач анализа, чем пользовательский код (а создание такого кода потребует много времени).

Анализ документов HTML с использованием Beautiful Soup

Загружаемая библиотека Beautiful Soup (`http://www.crummy.com/software/BeautifulSoup/`) находится по адресу `https://pypi.python.org/pypi/beautifulsoup4/4.3.2`. Она предоставляет средства для анализа данных HTML или XML способом, понятным Python, и позволяет работать с данными в древовидной форме.

ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Помимо предоставления средств для работы с древовидными данными, библиотека Beautiful Soup значительно облегчает работу с документами HTML. Например, она автоматически преобразует *кодировку* (способ хранения символов в документе) документов HTML из UTF-8 в Unicode. Разработчику Python обычно нужно беспокоиться о таких вещах, как кодировка, но с Beautiful Soup вы можете вместо этого сосредоточиться на своем коде.

Глава 3

Конфигурация Python для науки о данных

В ЭТОЙ ГЛАВЕ...

- Получение готового решения
- Установка Anaconda на Linux, Mac OS и Windows
- Получение и установка наборов данных, а также примера кода

Прежде чем вы сможете сделать хоть что-то с Python или использовать его для решения задач по науке о данных, вам нужна работоспособная конфигурация. Кроме того, нужен доступ к наборам данных и коду, используемому в этой книге. Загрузка примера кода и его установка в вашей системе — это наилучший способ извлечь хороший опыт из изучения данной книги. Эта глава поможет настроить вашу систему так, чтобы вы могли легко следовать примерам в остальной части книги.

Книга опирается на Jupyter Notebook версии 5.5.0, поставляемый со средой Anaconda 3 (версия 5.2.0), которая поддерживает Python версии 3.6.5, использованный для создания примеров кода. Чтобы примеры работали, вы должны использовать Python 3.6.5, а также ту версию пакетов, которая представлена в Anaconda 3 версии 5.2.0. В старых версиях Python и его пакетов, как правило, отсутствуют необходимые функции, а более новые версии претерпели серьезные изменения. Если вы используете какую-то другую версию Python, то примеры, скорее всего, не будут работать так, как задумано. Тем не менее вы можете найти другие инструменты разработки, которые можете предпочесть Jupyter Notebook. В этой главе рассматриваются также некоторые другие предложения, возможные в качестве инструментов для написания кода на языке

Python. Если вы выберете одно из этих предложений, ваши сценарии не будут соответствовать представленным в книге и вы не сможете выполнять процедуры. Но если выбранный вами пакет поддерживает Python 3.6.5, то код все равно должен работать так, как описано в книге.


СОВЕТ

Использование загруженного кода вовсе не мешает вам набирать примеры самостоятельно, исследовать их с помощью отладчика, дополнять их или работать с кодом всевозможными способами. Загружаемый код предназначен для того, чтобы помочь вам начать работу с данными и получить опыт при изучении Python. Увидев, как работает код, когда он правильно набран и настроен, вы можете попробовать создавать примеры самостоятельно. Если вы допустили ошибку, сравните то, что вы ввели, с загружаемым кодом и точно определите, где кроется ошибка. Загружаемый код этой главы находится в файлах `P4DS4D2_03_Sample.ipynb` и `P4DS4D2_03_Dataset_Load.ipynb`. (Где можно скачать исходный код этой книги, см. во введении.)

Готовые кросс-платформенные научные дистрибутивы

Вполне возможно получить универсальный экземпляр Python и добавить в него все библиотеки, необходимые для науки о данных. Процесс может быть сложным, поскольку для успеха вам нужно убедиться в наличии всех необходимых библиотек в правильных версиях. Кроме того, придется выполнить настройку, необходимую для обеспечения доступности библиотек, когда они вам будут нужны. К счастью, эти работы не являются обязательными, поскольку вам доступно несколько продуктов Python для обработки данных. Эти продукты предоставляют все необходимое, чтобы начать работу с проектами в области данных.


ЗАПОМНИ!

Для работы с примерами из этой книги вы можете использовать любой из пакетов, упомянутых в следующих разделах. Но исходный код книги и загружаемый исходный код опираются на Continuum Analytics Anaconda, поскольку этот конкретный пакет работает на всех платформах, для поддержки которых предназначена эта книга: Linux, Mac OS X и Windows. В последующих главах конкретный пакет не упоминается, но все снимки экрана демонстрируют вид при использовании Anaconda на операционной системе Windows.

Возможно, вам придется настроить код для использования другого пакета, и экраны будут выглядеть иначе, если вы используете Anaconda на какой-то другой платформе.

Получение пакета Anaconda от Continuum Analytics

Базовый пакет Anaconda можно загрузить бесплатно по адресу `https://www.anaconda.com/download/` (для получения версии 5.2.0, используемой в этой книге, может потребоваться перейти по адресу `https://repo.anaconda.com/archive/`, если на основном сайте доступна более новая версия продукта). Щелкните на одной из ссылок Python 3.6 Version, чтобы получить доступ к бесплатному продукту. Имя искомого файла начинается на `Anaconda3-5.2.0-`, за которым следует платформа и 32- или 64-разрядная версия, например `Anaconda3-5.2.0-Windows-x86_64.exe` для 64-разрядной версии Windows. Anaconda поддерживает следующие платформы.

- 32- и 64-разрядную версию Windows (программа установки может предложить вам только 64- или 32-разрядную версию, в зависимости от обнаруженной им версии Windows)
- 32- и 64-разрядную версии Linux
- 64-разрядную версию Mac OS X

Стандартная версия загрузки устанавливает Python версии 3.6, которая и используется в этой книге. Вы также можете установить Python 2.7, щелкнув на одной из ссылок. Программы установки для Windows и Mac OS X имеют графический интерфейс. При использовании Linux вы полагаетесь на утилиту `bash`.


СОВЕТ

Вполне возможно получить Anaconda и с более старыми версиями Python. Если хотите использовать более старую версию Python, используйте ссылку на архив программ установки, расположенную примерно на пол страницы ниже. Используйте более старую версию Python, только когда вам это необходимо.

Бесплатный продукт — это все, что вам нужно для этой книги. Однако, просматривая сайт, вы видите, что доступно много других дополнительных продуктов, которые могут помочь создавать рабочие приложения. Например, когда вы добавляете в комплект Accelerate, вы получаете возможность выполнять операции на нескольких ядрах и GPU. Использование этих дополнительных продуктов выходит за рамки описания данной книги, но подробные сведения об их использовании предоставляет сайт Anaconda.

Получение продукта Enthought Canopy Express

Enthought Canopy Express — это бесплатный продукт для создания как технических, так и научных приложений с использованием Python. Вы можете получить его на `https://www.enthought.com/canopy-express/`. Щелкните на Download на главной странице, чтобы отобразить список версий, которые можно скачать. Только Canopy Express предоставляется бесплатно, полный продукт Canopy предоставляется за отдельную плату. Canopy Express поддерживает следующие платформы:

- 32- и 64-разрядные версии Windows
- 32- и 64- разрядные версии Linux
- 32- и 64- разрядные версии Mac OS X


ВНИМАНИЕ!

На момент написания этой книги Canopy поддерживает Python 3.5. Вам нужен Python 3.6, чтобы примеры работали так, как ожидалось. Обязательно загрузите ту версию Canopy, которая поддерживает Python 3.6, или учтите, что некоторые примеры в книге могут не работать. На странице `https://www.enthought.com/product/canopy/#/package-index` перечислены пакеты, которые работают с Python 3.6.

Выберите платформу и версию, которую хотите скачать. Когда вы щелкнете на Download Canopy Express, вы увидите необязательную форму для предоставления информации о себе. Загрузка начинается автоматически, даже если вы не предоставите компании личную информацию.

Одним из преимуществ Canopy Express является то, что Enthought активно оказывает помощь как студентам, так и преподавателям. Люди также могут посещать занятия, в том числе онлайн-уроки, где обучают использованию Canopy Express различными способами (см. `https://training.enthought.com/courses`). Также предлагается живое обучение в классе, специально разработанное для аналитиков данных; прочитайте об этом обучении по адресу `https://www.enthought.com/services/training/data-science`.

Получение WinPython

Как следует из названия, WinPython — это продукт только для Windows, который вы можете найти по адресу `http://winpython.github.io/`. Этот сайт обеспечивает поддержку Python 3.5, 3.6 и 3.7. Данный продукт представляет собой развитие Python(x,y) (IDE, которая не использовалась и разработка которой прекращена в 2015 году; см. `http://python-xy.github.io/`) и не только предназначена для его замены. Наоборот, WinPython обеспечивает

более гибкий способ работы с Python(x,y). Вы можете прочитать о первоначальной мотивации при создании WinPython на `http://sourceforge.net/p/winpython/wiki/Roadmap/` и о его последнем плане развития на `https://github.com/winpython/winpython/wiki/Roadmap`.

Суть этого продукта в том, что вы получаете гибкость за счет дружелюбия и небольшой интеграции с платформой. Однако для разработчиков, которым необходимо поддерживать несколько версий IDE, WinPython может иметь существенное значение. При использовании WinPython с этой книгой обратите особое внимание на проблемы конфигурации, иначе вы обнаружите, что даже загружаемый код имеет мало шансов сработать.

Установка Anaconda на Windows

Anaconda поставляется с графическим приложением установки для Windows, поэтому хорошая установка означает использование мастера, как и при любой другой установке. Конечно, вам нужен экземпляр установочного файла, прежде чем вы начнете (см. выше раздел "Получение пакета Anaconda от Continuum Analytics"). Следующая процедура должна нормально сработать в любой системе Windows, независимо от того, используете ли вы 32- или 64-разрядную версию Anaconda.

1. **Найдите загруженный экземпляр Anaconda в вашей системе.**

   Эти файлы имеют разные имена, но обычно они выглядят как `Anaconda3-5.2.0-Windows-x86.exe` для 32-разрядных систем и `Anaconda3-5.2.0-Windowsx86_64.exe` для 64-разрядных. Номер версии встроен в имя файла. В данном случае имя файла относится к версии 5.2.0, используемой в этой книге. Если вы используете какую-то другую версию, у вас могут возникнуть проблемы с исходным кодом и вам придется вносить изменения при работе с ним.

2. **Дважды щелкните на установочном файле**.

   (Может появиться диалоговое окно Open File – Security Warning, в котором вас спросят, хотите ли вы запустить этот файл. Щелкните на Run (Выполнить), если появится это диалоговое окно.) Вы увидите диалоговое окно установки Anaconda 5.2.0, аналогичное показанному на рис. 3.1. Конкретное диалоговое окно, которое вы увидите, зависит от скачанной версии программы установки Anaconda. Если у вас 64-разрядная операционная система, всегда лучше использовать 64-разрядную версию Anaconda, чтобы добиться максимальной производительности. В этом первом диалоговом окне сообщается версия продукта (64-разрядная).

3. **Щелкните на кнопке** Next **(Далее)**.

   Мастер отобразит лицензионное соглашение. Обязательно ознакомьтесь с ним, чтобы знать условия использования.

4. **Щелкните на кнопке** I Agree **(Я согласен), если согласны с лицензионным соглашением**.

   Вас спросят, какой тип установки выполнять (рис. 3.2). Как правило, вы хотите установить продукт только для себя. Исключение составляют случаи, когда вашу систему используют несколько человек и им всем нужен доступ к Anaconda.

5. **Выберите один из типов установки и щелкните на кнопке** Next **(Далее)**.

   Мастер спросит, где установить Anaconda на диске, как показано на рис. 3.3. В книге предполагается, что вы используете стандартное расположение. Если вы выберете другое место, впоследствии вам, возможно, придется изменить некоторые процедуры.

6. **Выберите место установки (при необходимости) и щелкните на кнопке** Next **(Далее)**.

   Вы увидите дополнительные параметры установки, показанные на рис. 3.4. Эти параметры выбраны стандартно, и в большинстве случаев нет веских причин для их изменения. Возможно, вам придется изменить их, если Anaconda не предоставит настройки Python 3.6 стандартно. Однако в книге предполагается, что вы настроили Anaconda с использованием стандартных параметров.


СОВЕТ

Флажок Add Anaconda to My PATH Environment Variable (Добавить Anaconda в мою переменную среды PATH) стандартно сброшен, и вы должны оставить его без изменения. Добавление в переменную среды PATH позволяет находить файлы Anaconda с помощью стандартной командной строки, но если у вас установлено несколько версий Anaconda, то доступна будет только первая установленная версия. Вместо этого лучше использовать Anaconda Prompt, чтобы получать доступ к ожидаемой версии.

7. **Измените дополнительные параметры установки (при необходимости) и щелкните на кнопке** Install **(Установить)**.

   Откроется диалоговое окно установки с индикатором выполнения. Процесс установки может занять несколько минут, поэтому возьмите себе чашку кофе и немного почитайте комиксы. Когда процесс установки завершится, вы увидите кнопку Next (Далее).

8. **Щелкните на кнопке** Next **(Далее)**.

   Мастер сообщит, что установка завершена.

9. **Щелкните на кнопке** Next **(Далее)**.

   Anaconda предложит интегрировать поддержку кода Visual Studio. Для данной книги эта поддержка не нужна, и ее добавление может потенциально изменить работу инструментов Anaconda. Если вам абсолютно не нужна поддержка Visual Studio, имеет смысл поддержать чистоту среды Anaconda.

10. **Щелкните на кнопке** Skip **(Пропустить)**.

    Вы увидите экран завершения. Этот экран содержит параметры, позволяющие узнать больше об Anaconda Cloud и получить информацию о запуске вашего первого проекта Anaconda. Установка этих флажков (или их сброс) зависит от того, что вы хотите сделать далее; флажки не влияют на настройки Anaconda.

11. **Установите все необходимые параметры. Щелкните на кнопке** Finish **(Готово)**.

    Вы готовы начать использовать Anaconda.

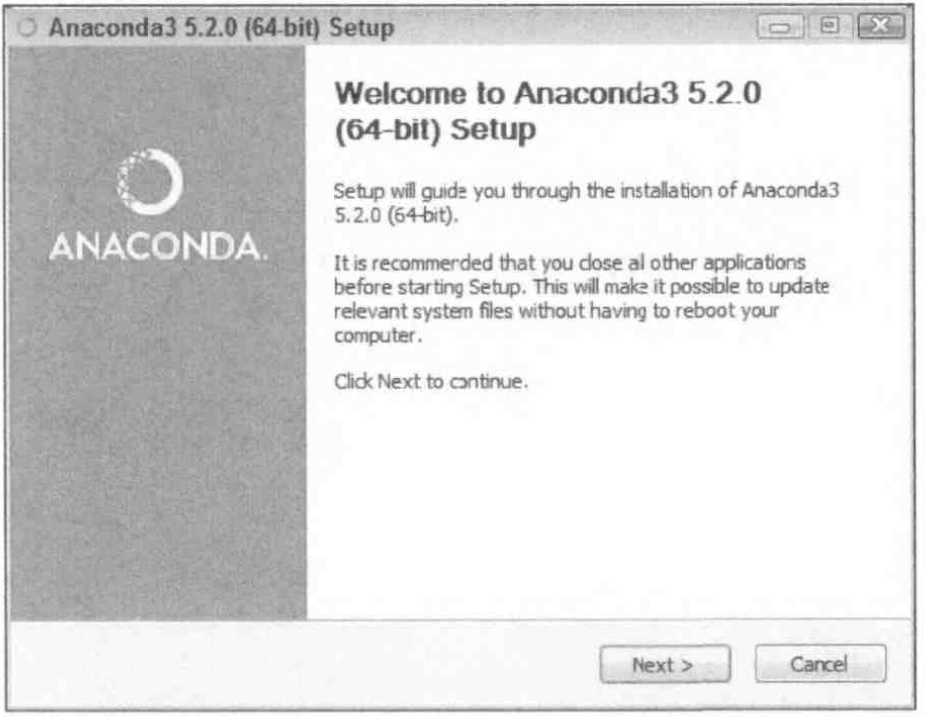


Рис. 3.1. Процесс установки начинается с сообщения о наличии 64-разрядной версии

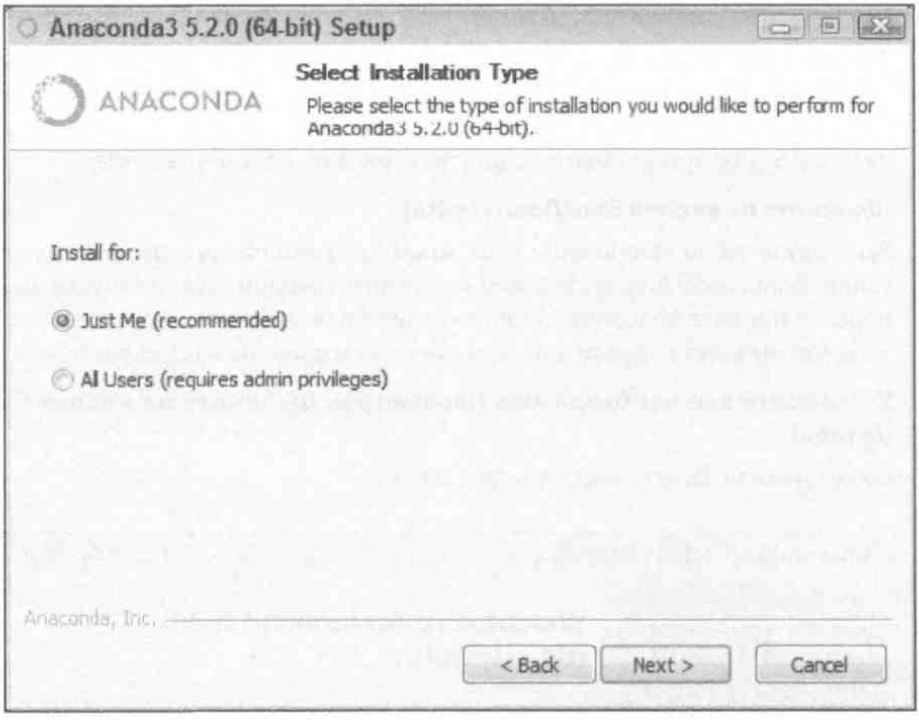


Рис. 3.2. Укажите мастеру, как установить Anaconda в вашей системе

Рис. 3.3. Укажите место установки

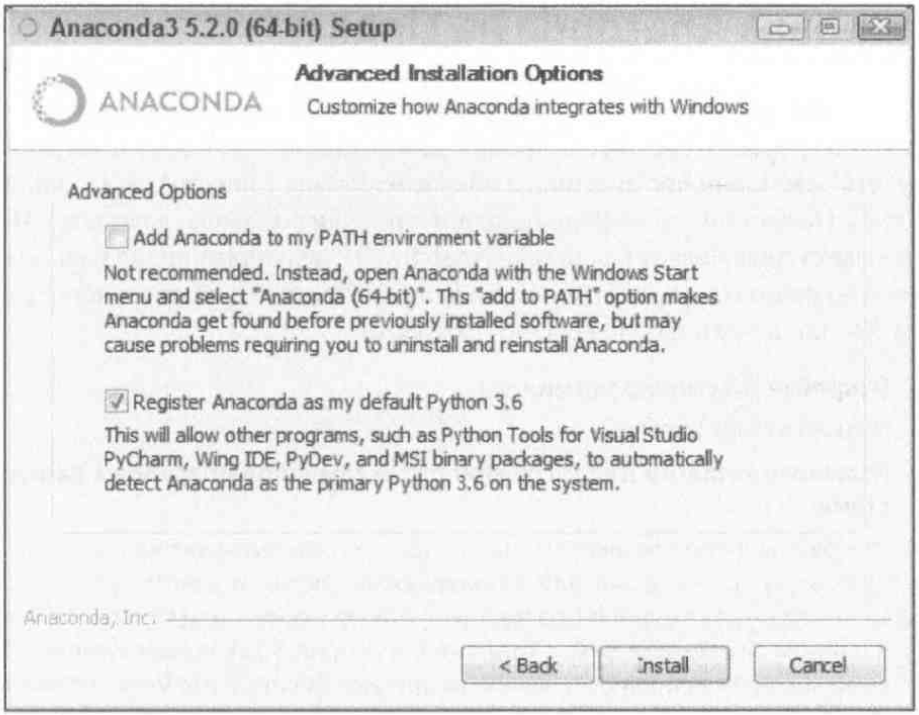


Рис. 3.4. Задайте дополнительные параметры установки

НЕСКОЛЬКО СЛОВ О СНИМКАХ ЭКРАНА

Работая с книгой, вы будете использовать IDE по вашему выбору, чтобы открывать файлы Python и Jupyter Notebook, содержащие исходный код книги. Каждый снимок экрана, который содержит специфичную для IDE информацию, основан на Anaconda, поскольку Anaconda работает на всех трех упоминаемых в книге платформах. Использование Anaconda не подразумевает, что это лучшая IDE или что авторы дают какие-либо рекомендации для нее, просто Anaconda отлично работает в качестве демонстрационного продукта.

Когда вы работаете с Anaconda, название графической среды (GUI), Jupyter Notebook, абсолютно одинаково для всех трех платформ, и вы даже не увидите существенного различия. Различия, которые вы видите, незначительны, и вы должны игнорировать их во время работы с книгой. С учетом этого, книга существенно зависит от сценариев Windows 7. Работая на Linux, Mac OS X или другой платформе Windows, вы должны ожидать некоторых отличий, но они не помешают вам работать с примерами.

Установка Anaconda на Linux

Для установки Anaconda на Linux вы используете командную строку, графический интерфейс отсутствует. Прежде чем выполнить установку, вы должны загрузить экземпляр программного обеспечения для Linux с сайта Continuum Analytics. Необходимую информацию для загрузки см. выше, в разделе "Получение пакета Anaconda от Continuum Analytics". Следующая процедура должна нормально работать на любой системе Linux, независимо от того, используете ли вы 32- или 64-разрядную версию Anaconda.

1. **Откройте экземпляр терминала.**

   Появится окно Terminal.

2. **Измените каталоги для загруженного экземпляра Anaconda в вашей системе.**

   Эти файлы имеют разные имена, но обычно они выглядят как `Anaconda3-5.2.0-Linux-x86.sh` для 32-разрядных систем и `Anaconda3-5.2.0-Linuxx86_64.sh` для 64-разрядных. Номер версии встроен в имя файла. В данном случае имя файла относится к версии 5.2.0, используемой в этой книге. Если вы используете какую-то другую версию, у вас могут возникнуть проблемы с исходным кодом и вам придется вносить изменения при работе с ним.

3. **Введите `bash Anaconda3-5.2.0-Linux-x86` (для 32-разрядной версии) или `Anaconda3-5.2.0-Linux-x86_64.sh` (для 64-разрядной версии) и нажмите клавишу <Enter>.**

   Запустится мастер установки, который попросит вас принять условия лицензии для использования Anaconda.

4. **Прочитайте лицензионное соглашение и примите условия, используя метод, необходимый для вашей версии Linux.**

   Мастер попросит вас указать место установки Anaconda. В книге предполагается, что вы используете стандартное местоположение `~/anaconda`. Если вы выберете другое место, впоследствии вам, возможно, придется изменить некоторые процедуры, чтобы работать с вашими настройками.

5. **Укажите место установки (при необходимости) и нажмите клавишу <Enter> (или щелкните на кнопке** Next **(Далее)).**

   Вы видите, что начинается процесс извлечения приложения. После завершения извлечения вы увидите сообщение о завершении.

6. **Добавьте путь установки в инструкцию `PATH`, используя метод, необходимый для вашей версии Linux.**

   Вы готовы начать использовать Anaconda.

Установка Anaconda на Mac OS X

Установка на Mac OS X возможна только в одной форме: 64-разрядной. Прежде чем выполнить установку, вы должны загрузить экземпляр программного обеспечения для Mac с сайта Continuum Analytics. Необходимую информацию для загрузки см. в разделе "Получение пакета Anaconda от Continuum Analytics". Следующие шаги помогут установить Anaconda 64-bit на системе Mac.

1. **Найдите загруженный экземпляр Anaconda в вашей системе.**

   Эти файлы имеют разные имена, но обычно они выглядят как `Anaconda3-5.2.0-MacOSX-x86_64.pkg`. Номер версии встроен в имя файла. В данном случае имя файла относится к версии 5.2.0, используемой в этой книге. Если вы используете какую-то другую версию, у вас могут возникнуть проблемы с исходным кодом и вам придется вносить изменения при работе с ним.

2. **Дважды щелкните на установочном файле.**

   Откроется начальное диалоговое окно.

3. **Щелкните на кнопке** Continue **(Продолжить).**

   Мастер спросит, хотите ли вы просмотреть материалы Read Me. Поскольку вы можете прочитать эти материалы позже, пропустите эту информацию.

4. **Щелкните на кнопке** Continue **(Продолжить).**

   Мастер отобразит лицензионное соглашение. Обязательно ознакомьтесь с ним, чтобы знать условия использования.

5. **Щелкните на кнопке** I Agree **(Я согласен), если согласны с лицензионным соглашением.**

   Мастер попросит вас указать место для установки. Это определяет, предназначена ли установка для отдельного пользователя или группы.

Может появиться сообщение об ошибке, в котором говорится, что вы не можете установить Anaconda в систему. Причиной сообщения является ошибка в установщике и не имеет ничего общего с вашей системой. Чтобы избавиться от сообщения об ошибке, выберите вариант Install Only for Me (Только для меня). Вы не можете установить Anaconda для группы пользователей в системе Mac.

6. **Щелкните на кнопке** Continue **(Продолжить).**

   Установщик отобразит диалоговое окно, содержащее возможности для изменения типа установки. Щелкните на Change Install Location (Изменить место установки), если хотите изменить место установки Anaconda в вашей системе (в книге предполагается, что вы используете стандартный путь `~/anaconda`).

Щелкните на кнопке Customize (Настроить), если хотите изменить работу установщика. Например, вы можете не добавлять Anaconda в свою инструкцию PATH. Однако в книге предполагается, что вы выбрали стандартные параметры установки, и нет веских причин менять их, если у вас не установлен другой экземпляр Python 2.7 в другом месте.

7. **Щелкните на кнопке** Install **(Установить)**.

   Вы видите начало установки. Индикатор выполнения показывает, как проходит этот процесс. После завершения установки откроется диалоговое окно завершения.

8. **Щелкните на кнопке** Continue **(Продолжить)**.

   Вы готовы начать использовать Anaconda.

Загрузка наборов данных и примеров кода

Эта книга об использовании языка Python для решения задач науки о данных. Конечно, вы можете потратить все свое время на создание примера кода с нуля, на его отладку и только потом узнать, как он связан с наукой о данных, или можете пойти простым путем и скачать уже написанный код, чтобы сразу приступить к работе. Аналогично создание наборов данных, достаточно больших для задач науки о данных, займет довольно много времени. К счастью, вы можете легко получить доступ к стандартизированным предварительно созданным наборам данных, используя функции, предоставляемые в некоторых библиотеках науки о данных. В следующих разделах вы узнаете, как загрузить и использовать примеры кода, а также наборы данных, чтобы сэкономить время и получить возможность работать с задачами, связанными с наукой о данных.

Использование Jupyter Notebook

Чтобы упростить работу с относительно сложным кодом этой книги, используйте Jupyter Notebook. Этот интерфейс позволяет легко создавать файлы блокнотов Python, способных содержать любое количество примеров, каждый из которых может запускаться отдельно. Программа работает в вашем браузере, поэтому не имеет значения, какую платформу вы используете для разработки; пока в ней есть браузер, у вас должно быть все в порядке.

Начинаем работу с Jupyter Notebook

На большинстве платформ имеется значок для доступа к Jupyter Notebook. Все, что вам нужно сделать для доступа к Jupyter Notebook, — это открыть этот значок. Например, в системе Windows вы выбираете Start⇨All

Programs⇨Anaconda3⇨Jupyter Notebook (Пуск⇨Все программы⇨Anaconda3⇨Jupyter Notebook). На рис. 3.5 показан внешний вид интерфейса в браузере Firefox. Конкретный внешний вид вашей системы зависит от используемого браузера и типа установленной платформы.

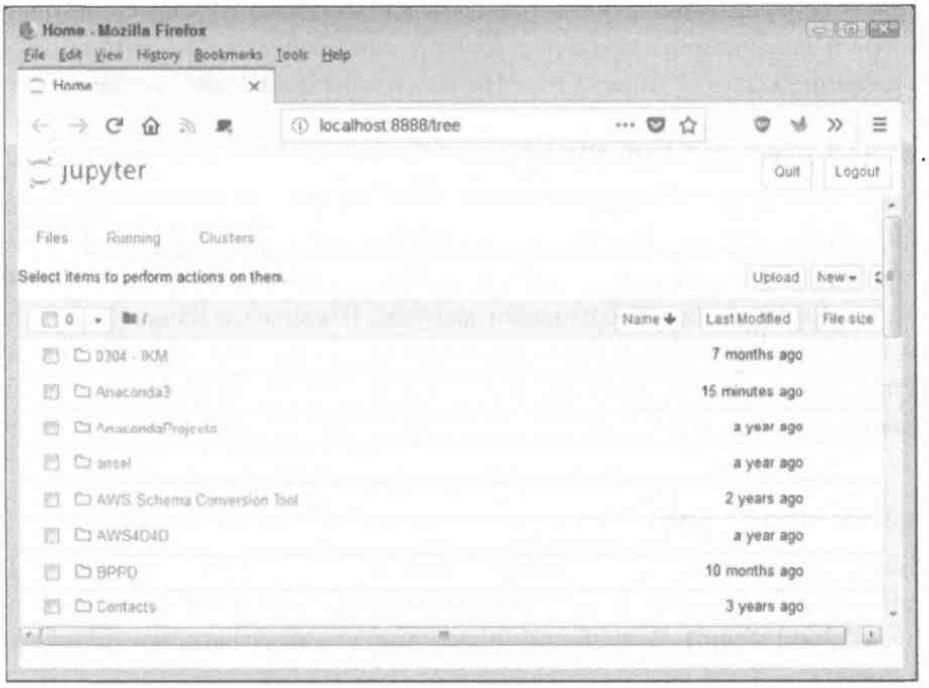


Рис. 3.5. Jupyter Notebook обеспечивает простой способ создания примеров науки о данных

Если ваша платформа предлагает простой доступ с помощью значка, используйте следующие этапы для доступа к Jupyter Notebook.

1. **Откройте Anaconda Prompt, Command Prompt или Terminal Window в вашей системе.**

   Откроется окно, в котором можно вводить команды.

2. **Перейдите в каталог** `\Anaconda3\Scripts` **на вашем компьютере.**

   Для этого большинство систем позволяют использовать команду CD.

3. **Введите** `..\python Jupyter-script.py notebook` **и нажмите клавишу <Enter>.**

   В вашем браузере откроется страница Jupyter Notebook.

Остановка сервера Jupyter Notebook

Независимо от того, как вы запускаете Jupyter Notebook (или просто Notebook, как он упоминается в оставшейся части книги), система обычно открывает командную строку или окно терминала для его размещения. Это окно содержит сервер, позволяющий работать приложению. После закрытия окна браузера и завершения сеанса перейдите в окно сервера и нажмите комбинацию клавиш <Ctrl + C> или <Ctrl + Break>, чтобы остановить сервер.

Определение хранилища кода

Код, который вы создаете и используете в этой книге, будет находиться в хранилище на вашем жестком диске. Считайте это *хранилище* неким шкафом для хранения документов, куда вы помещаете свой код. Notebook открывает ящик, вынимает папку и показывает код вам. Вы можете изменять его, запускать в папке отдельные примеры, добавлять новые примеры и просто взаимодействовать со своим кодом естественным образом. В следующих разделах вы познакомитесь с Notebook лучше и узнаете, как работает концепция хранилища.

Создание новой папки

Вы используете папки для хранения файлов кода конкретного проекта. Проект этой книги называется `P4DS4D2` (что означает *Python for Data Science For Dummies,* 2nd Edition). Следующие шаги помогут вам создать новую папку для этой книги.

1. **Выберите пункт меню** New⇨Folder **(Новая⇨Папка).**

   Блокнот создает новую папку. Имя папки может отличаться, но для пользователей Windows она указана как Untitled Folder (Папка без названия). Возможно, вам придется прокрутить вниз список доступных папок, чтобы найти нужную.

2. **Установите флажок в поле рядом с** Untitled Folder **(Папка без названия).**

3. **Щелкните** Rename **(Переименовать) в верхней части страницы.**

   Откроется диалоговое окно Rename Directory (Переименовать каталог), показанное на рис 3.6.

4. **Введите `P4DS4D2` и нажмите клавишу <Enter>.**

   Notebook переименует папку.

Создание нового блокнота

Каждый новый блокнот похож на папку с файлами. В папку с файлами вы можете поместить отдельные примеры, так же, как и листы бумаги в

физическую папку. Каждый пример появляется в своей ячейке. Вы также можете помещать в папку и другие вещи, по мере чтения книги вы увидите, как это работает. Используйте следующие шаги, чтобы создать новый блокнот.

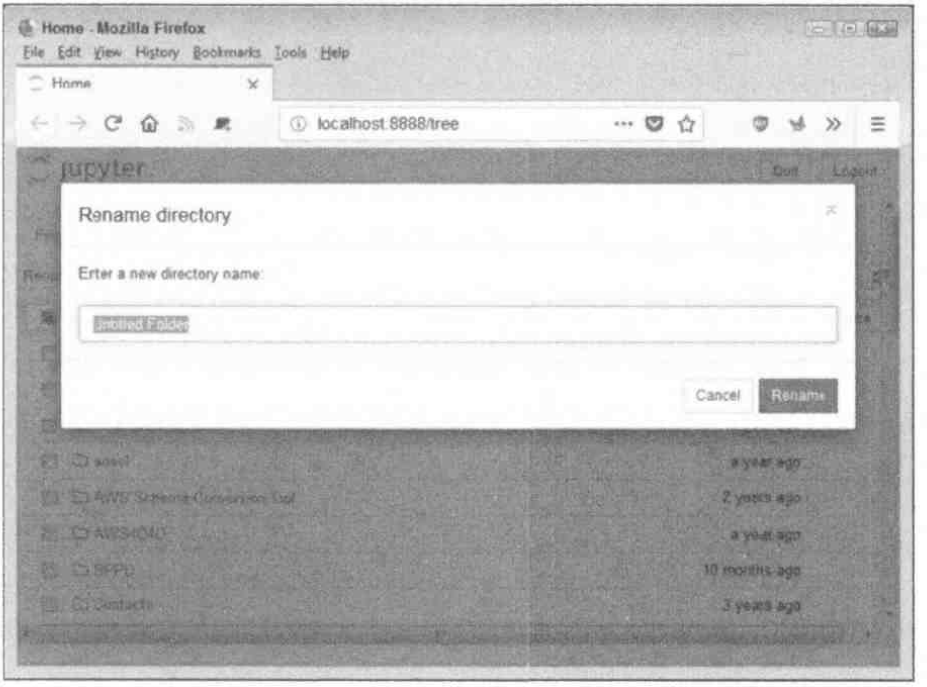


Рис. 3.6. Создание папки для хранения кода книги

1. **Щелкните на элементе** P4DS4D2 **на домашней странице.**

   Вы увидите содержимое папки проекта этой книги, которая будет пустой, если вы выполняете это упражнение впервые.

2. **Выберите пункт меню** New⇨Python 3.

   В браузере откроется новая вкладка с новым блокнотом, как показано на рис. 3.7. Обратите внимание, что блокнот содержит ячейку, и эта ячейка выделена, чтобы вы могли вводить в нее код. Название блокнота сейчас Untitled (Без названия), поэтому его нужно изменить.

3. **Щелкните на странице** Untitled **(Без названия)**.

   Notebook спросит, хотите ли вы использовать новое имя (рис. 3.8).

4. **Введите `P4DS4D2_03_Sample` и нажмите клавишу <Enter>.**

   Новое имя говорит вам, что это файл для примера главы 3 этой книги. Использование этого соглашения об именах позволит вам легко отличать эти файлы от других файлов в вашем хранилище.

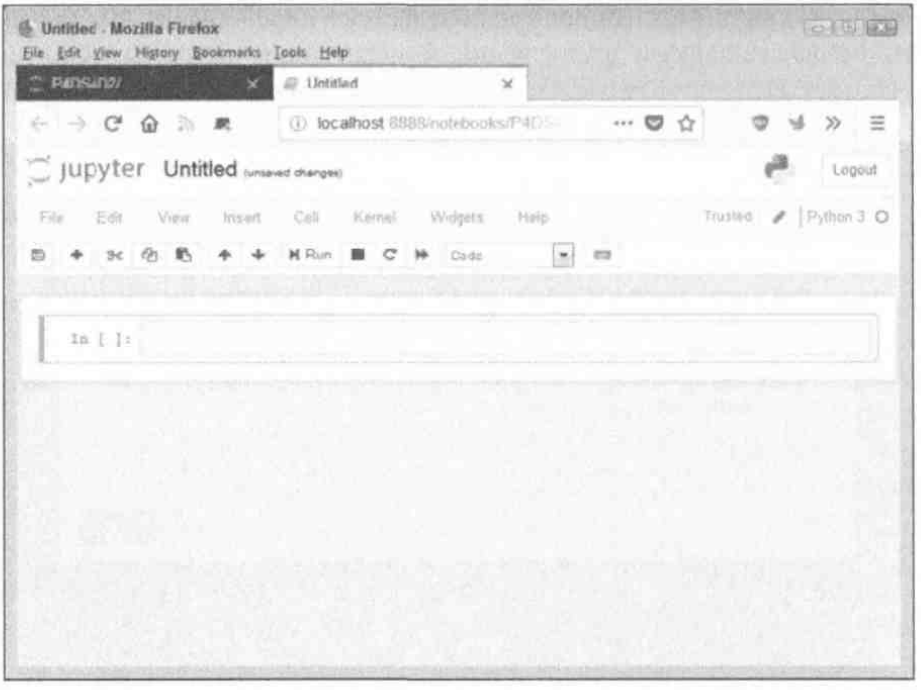


Рис. 3.7. Блокнот содержит ячейки, используемые для хранения кода

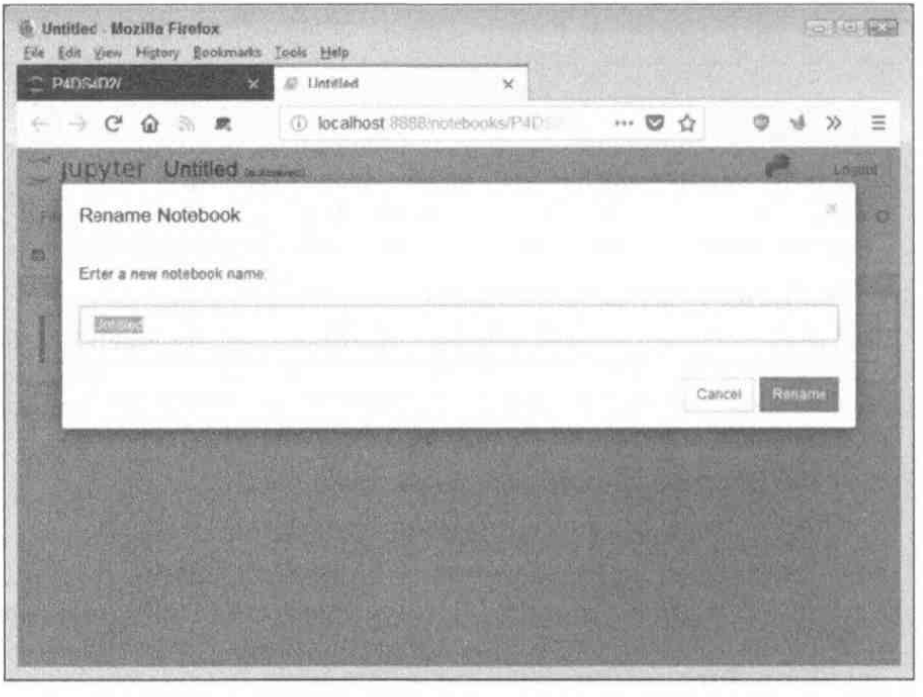


Рис. 3.8. Укажите новое имя блокнота

Добавление содержимого в блокнот

Конечно, новый блокнот ничего не содержит. В этой книге соблюдается соглашение об объединении файлов исходного кода, что упрощает их использование. Следующие шаги расскажут вам об этом соглашении.

1. **Выберите в раскрывающемся списке, который содержит слово** Code, **пункт** Markdown.

   Ячейка Markdown содержит текст документации. Вы можете поместить в эту ячейку что угодно, поскольку Notebook не будет интерпретировать ее содержимое. Используя ячейки Markdown, вы можете легко документировать то, что имеете в виду при написании кода.

2. **Введите** `# Downloading the Datasets and Example Code` **и щелкните на кнопке** Run **(Выполнить) (кнопка со стрелкой вправо на панели инструментов)**.

   Метка # создает заголовок. Один символ # создает заголовок первого уровня. Далее следует текст заголовка. Щелкните на кнопке Run (Выполнить), чтобы преобразовать форматированный текст в заголовок, как показано на рис. 3.9. Обратите внимание, что Notebook автоматически создает новую ячейку.

3. **Выберите пункт** Markdown **снова, введите** `## Defining the code repository` **и щелкните на кнопке** Run **(Выполнить)**.

   Notebook создаст заголовок второго уровня, который выглядит меньше заголовка первого уровня.

4. **Выберите пункт** Markdown**, введите** `### Adding notebook content` **и щелкните на кнопке** Run **(Выполнить)**.

   Notebook создаст заголовок третьего уровня. Ваши заголовки теперь соответствуют иерархии, которая начинается с заголовка первого уровня для данного раздела. Использование этого подхода поможет легко найти определенный фрагмент кода в загружаемом источнике. Как всегда, Notebook автоматически создает новую ячейку, и тип ячейки автоматически меняется на Code (Код), поэтому теперь вы готовы ввести некоторый код для этого примера.

5. **Введите** `print('Python is really cool!')` **и щелкните на кнопке** Run **(Выполнить)**.

   Обратите внимание, что код имеет цветовую кодировку, чтобы вы могли различать функцию (`print`) и связанные с ней данные (`'Python is really cool!'`). Результат показан на рис 3.10. Вывод является частью той же ячейки, что и код. Однако Notebook визуально отделяет вывод от кода, чтобы их можно было отличить друг от друга. Далее Notebook автоматически создает новую ячейку.

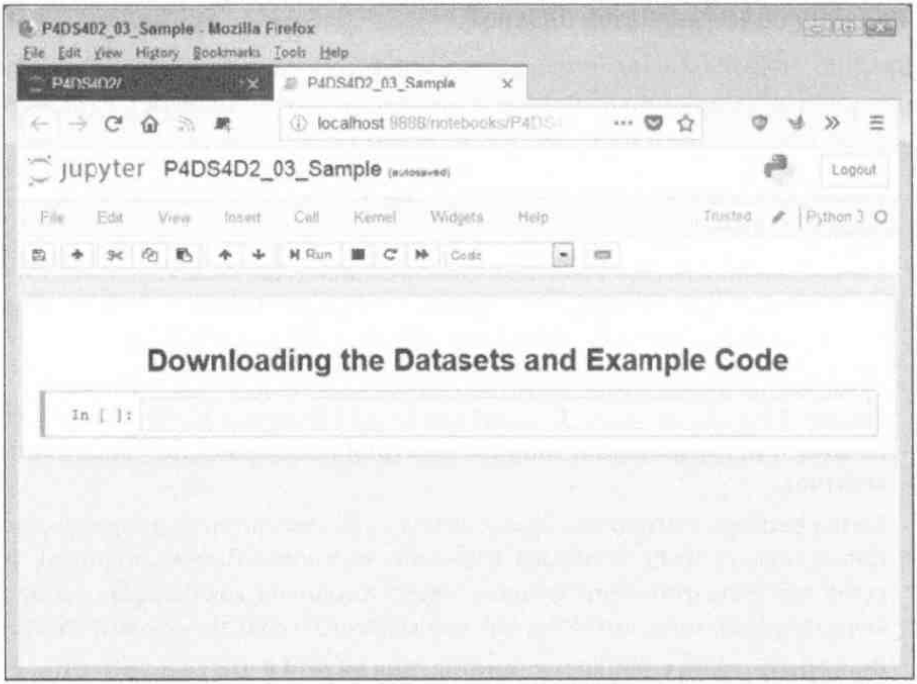


Рис. 3.9. Создание заголовка для документирования кода

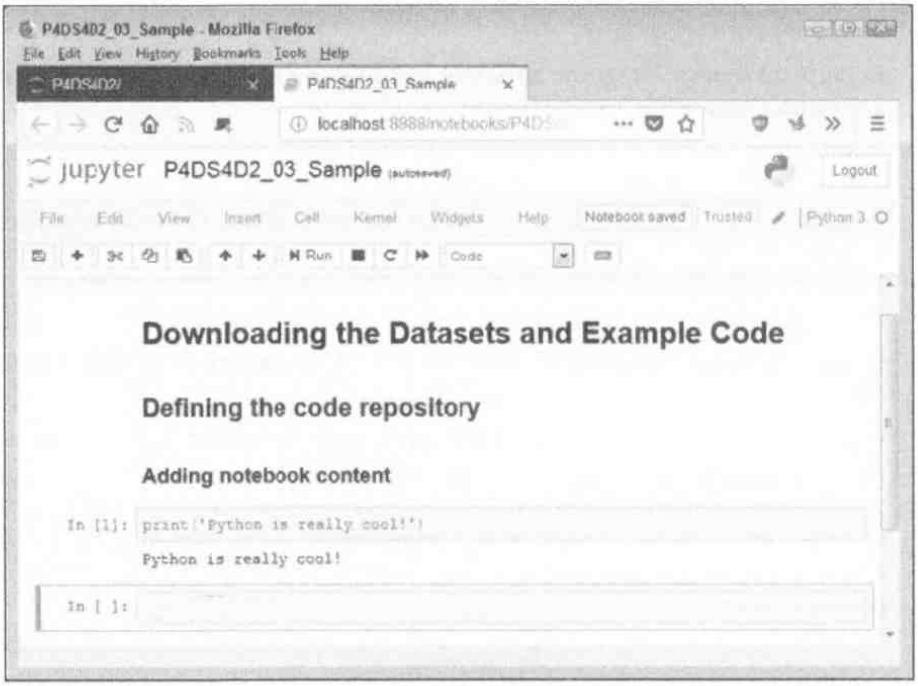


Рис. 3.10. Блокнот использует ячейки для хранения кода

Когда закончите работать с Notebook, закройте его, выбрав пункты меню File⇨Close and Halt (Файл⇨Закрыть и остановить). Вы вернетесь на страницу P4DS4D2, на которой увидите, что только что созданный блокнот добавлен в список (рис. 3.11).

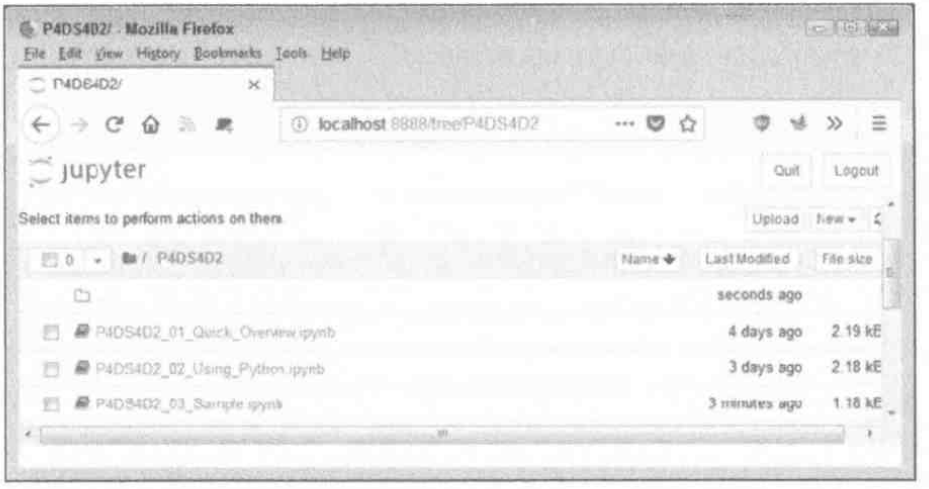


Рис. 3.11. Все блокноты, которые вы создадите, появятся в списке хранилища

Экспорт блокнота

В какой-то момент вы захотите поделиться блокнотами с другими людьми. Для этого вы должны экспортировать свой блокнот из хранилища в файл. Затем можете отправить файл кому-то другому, кто импортирует его в свое хранилище.

В предыдущем разделе было показано, как создать блокнот P4DS4D2_03_Sample. Вы можете открыть этот блокнот, щелкнув на его пункте в списке хранилища. Файл откроется, чтобы вы снова увидели свой код. Чтобы экспортировать этот код, выберите пункт меню File⇨Download As⇨Notebook (.ipynb) (Файл⇨Загрузить как⇨Блокнот (.ipynb)). То, что вы увидите далее, зависит от вашего браузера, но обычно это какое-то диалоговое окно для сохранения блокнота в виде файла. Для сохранения файла Notebook используйте тот же способ, что и для любого другого файла, сохраняемого с помощью браузера.

Удаление блокнота

Иногда блокноты устаревают, или вам просто не нужно больше с ними работать. Чтобы ваше хранилище не засорялось ненужными файлами, удалите ненужные блокноты из списка. Обратите внимание на флажок рядом с пунктом P4DS4D2_03_Sample.ipynb на рис. 3.11. Для удаления файла выполните следующие действия.

1. **Установите флажок рядом с пунктом** P4DS4D2_03_Sample.ipynb**.**
2. **Щелкните на значке** Delete **(Удалить) (мусорная корзина).**

   Появится предупреждение об удалении блокнота, подобное показанному на рис. 3.12.
3. **Щелкните на кнопке** Delete **(Удалить).**

   Блокнот удалит файл блокнота из списка.

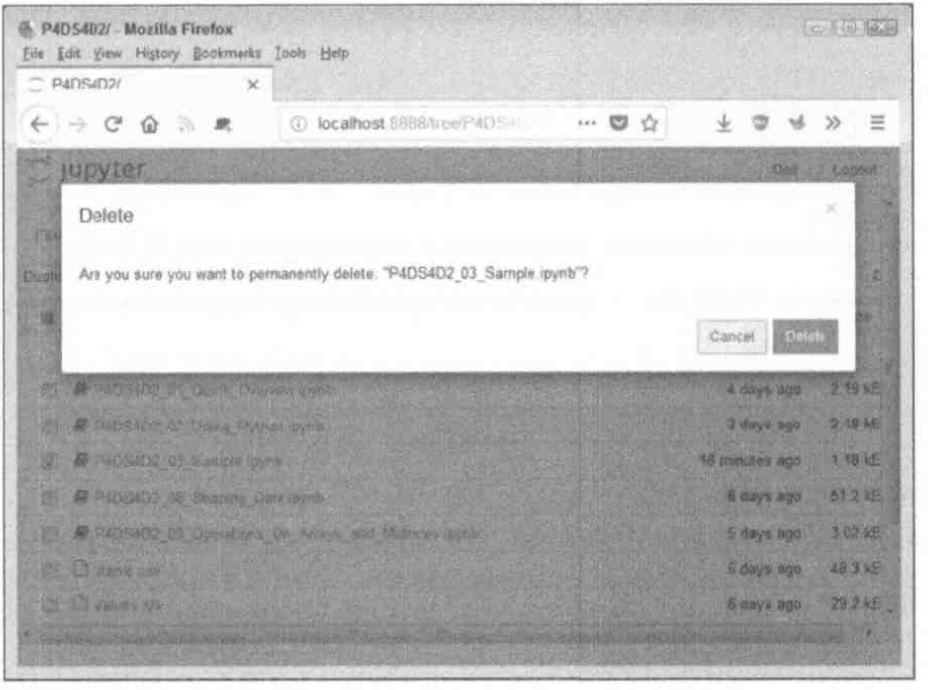


Рис. 3.12. Блокнот предупреждает перед удалением любых файлов из хранилища

Импорт блокнота

Чтобы использовать исходный код из этой книги, следует импортировать загруженные файлы в свое хранилище. Исходный код поставляется в архивном файле, содержимое которого вы извлекаете в папку на жестком диске. Архив содержит список файлов `.ipynb` (IPython Notebook), содержащих исходный код этой книги (подробности о загрузке исходного кода см. во введении). Ниже описано, как импортировать эти файлы в ваше хранилище.

1. **На странице Notebook P4DS4D2 щелкните на кнопке** Upload **(Загрузить).**

   То, что вы увидите, зависит от вашего браузера. В большинстве случаев это будет диалоговое окно загрузки файлов, которое предоставляет доступ к файлам на вашем жестком диске.

2. **Перейдите в каталог, содержащий файлы, которые хотите импортировать в Notebook.**

3. **Выделите один или несколько файлов для импорта и щелкните на кнопке** Open **(Открыть) (или на аналогичной), чтобы начать процесс загрузки.**

   Вы увидите файл, добавленный в список загрузки (рис. 3.13). Файл еще не является частью хранилища, вы просто выбрали его для загрузки.

4. **Щелкните на кнопке** Upload **(Загрузить).**

   Notebook поместит файл в хранилище, чтобы вы могли использовать его.

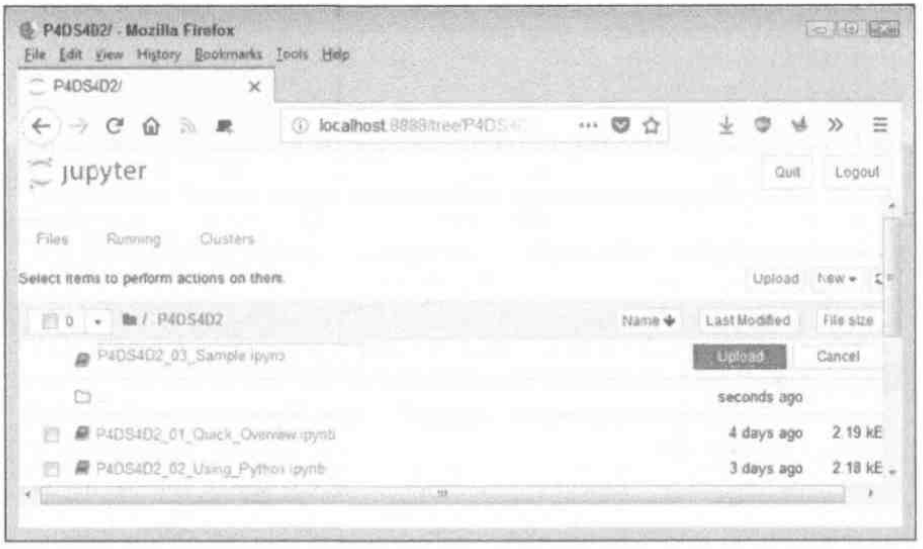


Рис.3.13. Файлы, которые вы хотите добавить в хранилище, появятся в списке загрузки

Наборы данных, используемые в этой книге

В этой книге используется несколько наборов данных, каждый из которых представлен в библиотеке Scikit-learn. Эти наборы данных демонстрируют различные способы взаимодействия с данными, и вы будете использовать их в примерах для решения различных задач. Ниже представлен краткий обзор функций, используемых для импорта каждого из наборов данных в код Python.

- `load_boston()`. Регрессионный анализ с набором данных по ценам на жилье в Бостоне.
- `load_iris()`. Классификация с набором данных Iris.
- `load_diabetes()`. Регрессионный анализ с набором данных diabetes.

- `load_digits([n_class])`. Классификация с набором данных digits.
- `fetch_20newsgroups(subset='train')`. Данные из 20 групп новостей.
- `fetch_olivetti_faces()`. Набор данных Olivetti faces от AT&T.

Техника загрузки каждого из этих наборов данных одинакова для всех примеров. В следующем примере показано, как загрузить набор данных о ценах на жилье в Бостоне. Вы можете найти этот код в блокноте `P4DS4D2_03_Dataset_Load.ipynb`.

```
from sklearn.datasets import load_boston
Boston = load_boston()
print(Boston.data.shape)
```

Чтобы увидеть, как работает код, щелкните на кнопке Run Cell (Запустить ячейку). Вывод функции `print` составляет `(506L, 13L)`. Результат приведен на рис. 3.14. (Будьте терпеливы, загрузка набора данных может занять несколько секунд.)

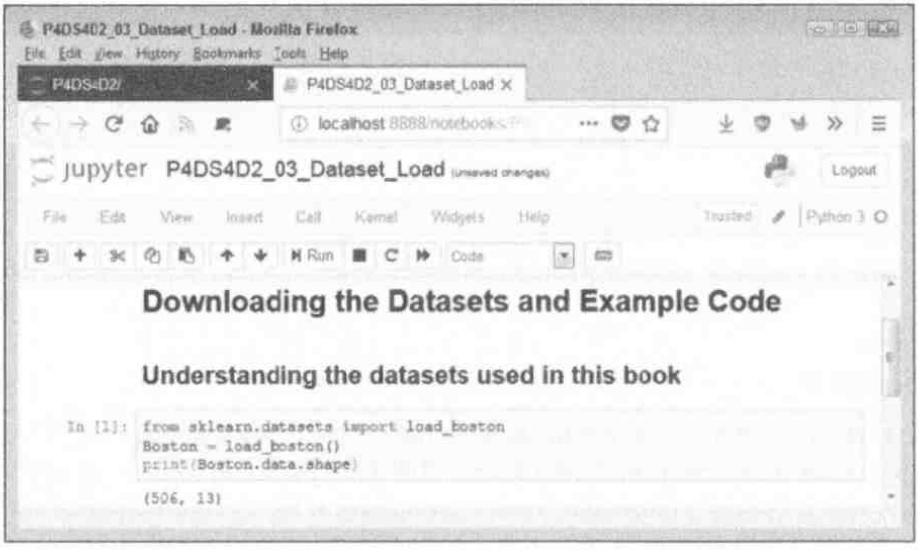


Рис. 3.14. Объект `Boston` *содержит загруженный набор данных*

Глава 4

Работа с Google Colab

В ЭТОЙ ГЛАВЕ...

- Понятие Google Colab
- Доступ к Google и Colab
- Выполнение основных задач Colab
- Получение дополнительной информации

Colab Laboratory (`https://colab.research.google.com/notebooks/welcome.ipynb`), или кратко Colab, представляет собой облачный служба Google, реплицирующий Jupyter Notebook в облаке. Для его использования не нужно ничего устанавливать в своей системе. В большинстве случаев Colab используется так же, как и настольная версия Jupyter Notebook (называемая в этой книге также Notebook). Эта глава включена в книгу в основном для тех читателей, которые для работы с примерами используют нечто отличное от стандартной настольной системы.


ЗАПОМНИ!

Поскольку вы, возможно, используете иные версии продуктов, представленных в этой книге, примеры исходного кода книги могут работать именно так, как описано в тексте, а могут и не работать. Кроме того, при использовании Colab вы можете не увидеть результатов, представленных в этой книге, из-за различий в аппаратном обеспечении между платформами. Во вводных разделах этой главы более подробно рассказывается и о Colab, и о том, что от него можно ожидать. Однако важнее всего помнить, что Colab не является заменой для Jupyter Notebook, и примеры не проверялись для работы конкретно с ним, но вы можете попробовать его на своем альтернативном устройстве, если хотите опробовать эти примеры.

Чтобы использовать Colab, необходима учетная запись Google, а также доступ к Colab с использованием вашей учетной записи. В противном случае большинство функций Colab не сработает. В следующем разделе вы познакомитесь с Google и узнаете, как получить доступ к Colab.

Как и Notebook, Colab можно использовать для решения определенных задач. В следующих разделах рассматриваются различные темы, связанные с заданиями, которые начинаются с использования Notebook. Если вы использовали Notebook в предыдущих главах, то заметите сильное сходство между ним и Colab. Конечно, вы будете также выполнять и другие виды задач, такие как создание различных типов ячеек и их использование для создания блокнотов, которые выглядят так же, как и те, которые создаются с помощью Notebook.

И наконец, в настоящей главе нельзя охватить все аспекты Colab, поэтому последний раздел главы служит удобным источником информации для поиска наиболее достоверной информации о нем.

Определение Google Colab

Google Colab — это облачная версия Notebook. Она даже использует файлы IPython (прежнее название Jupyter) Notebook (`.ipynb`). Так и есть, вы видите Notebook прямо в браузере. Несмотря на то что оба приложения похожи и оба используют файлы `.ipynb`, у них есть некоторые отличия, о которых нужно знать.

Что делает Google Colab

Вы можете использовать Colab для решения многих задач, но в этой книге вы будете использовать его для написания и выполнения кода, создания связанной с ним документации и отображения графики, как вы это делаете в Notebook. Методы, которые вы будете использовать, похожи на таковые в Notebook, но позже в этой главе вы обнаружите небольшие различия между ними. Несмотря на это, загружаемые файлы исходного кода этой книги должны работать без особых усилий с вашей стороны.

Notebook — это локализованное приложение, в котором используются локальные ресурсы. Вы можете использовать и другие источники, но в некоторых случаях это может оказаться неудобным или невозможным. Например, согласно `https://help.github.com/en/articles/working-with-jupyter-notebook-files-on-github`, при использовании хранилища GitHub файлы вашего блокнота будут отображаться в виде статических страниц HTML. А некоторые функции не будут работать вообще. Colab позволяет вам полностью взаимодействовать с файлами Notebook, используя GitHub в качестве хранилища.

Colab поддерживает несколько вариантов сетевых хранилищ, поэтому вы можете считать Colab своим сетевым партнером в создании кода Python.

Другая причина, по которой вам действительно нужно знать о Colab, заключается в том, что вы можете использовать его на ваших альтернативных устройствах. В процессе написания книги часть примеров кода была проверена на планшете с операционной системой Android (ASUS ZenPad 3S 10). На этом планшете был установлен браузер Chrome, и код выполняется достаточно хорошо, чтобы работать с примерам. Все это говорит о том, что вы, вероятно, не захотите пытаться писать код с помощью планшета такого размера (текст был невероятно маленьким), с другой стороны, отсутствие клавиатуры также может быть проблемой. Дело в том, что вам не обязательно иметь систему Windows, Linux или OS X, чтобы опробовать код, но альтернативы могут не обеспечивать ожидаемую производительность.

Google Colab обычно не работает с другими браузерами, кроме Chrome или Firefox. В большинстве случаев вы увидите сообщение об ошибке, если попытаетесь запустить Colab в браузере, который он не поддерживает. Для правильной работы ваш экземпляр Firefox может также нуждаться в некоторой настройке (подробнее об этом — далее, в разделе "Поддержка локальной среды выполнения"). Объем настройки зависит от того, какие функции Colab вы выбрали. Многие примеры прекрасно работают в Firefox и без каких-либо изменений.

НЕКОТОРЫЕ СТРАННОСТИ FIREFOX

Даже с помощью интерактивной справки вы можете обнаружить, что ваш экземпляр Firefox отображает сообщение об ошибке `SecurityError: The operation is insecure`. Первоначальное диалоговое окно сообщения об ошибке будет указывать на некоторую другую проблему, такую как файлы cookie, но вы увидите это сообщение об ошибке после щелчка на кнопке Details (Подробнее). Простое закрытие диалогового окна щелчком на кнопке OK заставит Colab работать, и он отобразит ваш код, но вы не увидите результатов запуска кода.

В качестве первого шага к решению этой проблемы убедитесь, что ваш экземпляр Firefox имеет современную версию; старые версии не обеспечивают необходимую поддержку. После того как обновите свою версию, установите параметр `network.websocket.allowInsecureFromHTTPS` в состояние `True`, используя `About:Config`, что должно решить проблему, но иногда это не работает. В этом случае убедитесь, что Firefox действительно разрешает исполь-

зование сторонних файлов cookie, установив параметр `Always for the Accept Third Party Cookies and Site Data` и выбрав пункт Remember History (Будет запомнить историю) в разделе History (История) на вкладке Privacy & Security (Приватность и защита) диалогового окна Options (Настройки). После каждого изменения перезапускайте Firefox, а затем снова попробуйте Colab. Если ни одно из этих исправлений не сработает, для работы с Colab на вашей системе придется использовать Chrome.

Особенности сетевого программирования

По большей части Colab используют так же, как и Notebook. Однако некоторые функции работают иначе. Например, чтобы выполнить код внутри ячейки, нужно выбрать эту ячейку и щелкнуть на кнопке запуска (стрелка вправо) для этой ячейки. Текущая ячейка остается выделенной, что означает, что фактически вы должны выбрать следующую ячейку в качестве отдельного действия. Кнопка рядом с выводом позволяет очистить только этот вывод, не затрагивая другие ячейки. Наведите указатель мыши на эту кнопку, чтобы узнать, когда некто запустил содержимое. В правой части ячейки вы увидите вертикальное многоточие, после щелчка на котором откроется меню параметров для этой ячейки. Результат такой же, как и при использовании Notebook, но процесс достижения результата иной.

ЗАПОМНИ

Фактический процесс работы с кодом также отличается от такового в Notebook. Да, вы все еще вводите код, как всегда, и полученный код выполняется без проблем, как в Notebook. Различие в том, как вы можете управлять кодом. При желании вы можете загрузить код с локального диска, а затем сохранить его на Google Drive или GitHub. Код становится доступным с любого устройства, получившего доступ к тем же исходным файлам. Все, что вам нужно сделать для доступа к Colab, — это загрузить его.

Если при работе с Colab вы используете Chrome и разрешаете синхронизацию своего экземпляра Chrome между различными устройствами, весь ваш код становится доступным на любом устройстве, с которым вы решили работать. Синхронизация передает ваш выбор на все ваши устройства, если они также настроены на синхронизацию настроек. Следовательно, вы можете написать код на своем рабочем столе, проверить его на планшете, а затем просмотреть на смартфоне. Это все тот же код, то же хранилище и те же настройки Chrome, просто устройства разные.

Однако вы можете обнаружить, что вся эта гибкость достигается ценой скорости и эргономики. При рассмотрении всех возможностей локальный

экземпляр Notebook выполняет код этой книги куда быстрее, чем копия Colab, использующая любую из доступных конфигураций (даже при работе с локальной копией файла `.ipynb`). Таким образом, работая с Colab, вы платите скоростью за гибкость. Кроме того, просмотр исходного кода на планшете затруднен; просматривать его на смартфоне практически невозможно. Если вы сделаете текст достаточно большим, чтобы увидеть его, то не увидите достаточно много кода, чтобы стало возможно любое осмысленное редактирование. В лучшем случае вы сможете просмотреть код по одной строке за раз, чтобы определить, как он работает.


СОВЕТ

Использование Notebook имеет и другие преимущества. Например, при работе с Colab есть возможность загружать исходные файлы только в виде файлов `.ipynb` или `.py`. Colab не поддерживает все остальные варианты загрузки, включая (но не ограничиваясь) HTML, LaTeX и PDF. Следовательно, ваши возможности создания презентаций из сетевого содержимого также в некоторой степени ограничены. Короче говоря, использование Colab и Notebook в некоторой степени обеспечивает различный опыт программирования. Однако они не являются взаимоисключающими, поскольку имеют общие форматы файлов. Теоретически переключение между ними двумя возможно при необходимости.

При использовании Notebook и Colab учитывайте следующее: оба продукта используют в основном одинаковую терминологию и множество одинаковых функций, но они совпадают не полностью. Методы, используемые для решения задач, и некоторые термины также различаются. Например, ячейка Markdown в Notebook — это ячейка Text в Colab. Ниже, в разделе "Выполнение общих задач" рассказывается о других различиях, которые необходимо учитывать.

Поддержка локальной среды выполнения

Единственный раз, когда вам действительно нужна поддержка локального выполнения, это когда вы хотите работать в командной среде и вам нужно преимущество в скорости или доступе к ресурсам, предлагаемое локальной средой выполнения. Использование локальной среды выполнения обычно дает лучшую скорость, чем вы получаете, полагаясь на облако. Кроме того, локальная среда выполнения позволяет вам получать доступ к файлам на вашем компьютере. Локальная среда выполнения предоставляет вам также контроль над версией Notebook, используемой для выполнения кода. Больше о поддержке локальной среды выполнения можно узнать по адресу `https://research.google.com/colaboratory/local-runtimes.html`.

ВНИМАНИЕ!

При определении необходимости поддержки локальной среды выполнения следует учитывать несколько вопросов. Наиболее очевидным является то, нужна ли вам локальная среда выполнения? А это означает, что данный параметр не будет работать с вашим ноутбуком или планшетом, если на нем не установлена операционная система Windows, Linux или OS X и соответствующая версия Notebook. Вашему ноутбуку или планшету также понадобится соответствующий браузер; Internet Explorer почти гарантированно вызывает проблемы, если он вообще сработает.

Однако самое важное при использовании локальной среды выполнения — это то, что ваша машина теперь открыта для возможного заражения кодом Notebook. Вы должны доверять стороне, предоставляющей код. Однако выбор локальной среды выполнения не открывает вашу машину другим, с кем вы делитесь кодом; для выполнения кода они должны либо использовать свои собственные локальные среды выполнения, либо полагаться на облако.

СОВЕТ

Работая с Colab и используя локальную среду выполнения и Firefox, вы должны сделать некоторые специальные настройки. Обязательно прочитайте раздел Browser Specific Setups (Настройки браузера) на странице Local Runtimes (Локальные среды выполнения), чтобы убедиться, что Firefox настроен правильно. Всегда проверяйте ваши настройки. Firefox может работать правильно с Colab. Однако при выполнении задач возможны проблемы конфигурации, и Colab отображает сообщения об ошибках, в которых говорится, что код не выполняется (или что-то еще, не особенно полезное).

Получение учетной записи Google

Прежде чем вы сможете выполнять какие-либо задачи с помощью Colab, нужно получить учетную запись Google. Вы можете использовать одну и ту же учетную запись для любых целей, а не только для разработки. Например, учетная запись Google предоставляет доступ к Google Docs (`https://www.google.com/docs/about/`) — сетевой системе управления документами, аналогичной Office 365.

Создание учетной записи

Чтобы создать учетную запись Google, перейдите на страницу `https://account.google.com/` и щелкните на ссылке Create Your Google Account (Создать

учетную запись Google). На этой странице содержится также подробная информация о том, что может предоставить учетная запись Google. Щелкнув на Create Your Google Account (Создать учетную запись Google), вы увидите страницу, показанную на рис. 4.1. Процесс создания учетной записи занимает несколько страниц. Вы просто предоставляете необходимую информацию на каждой странице и щелкаете на кнопке Next (Далее).

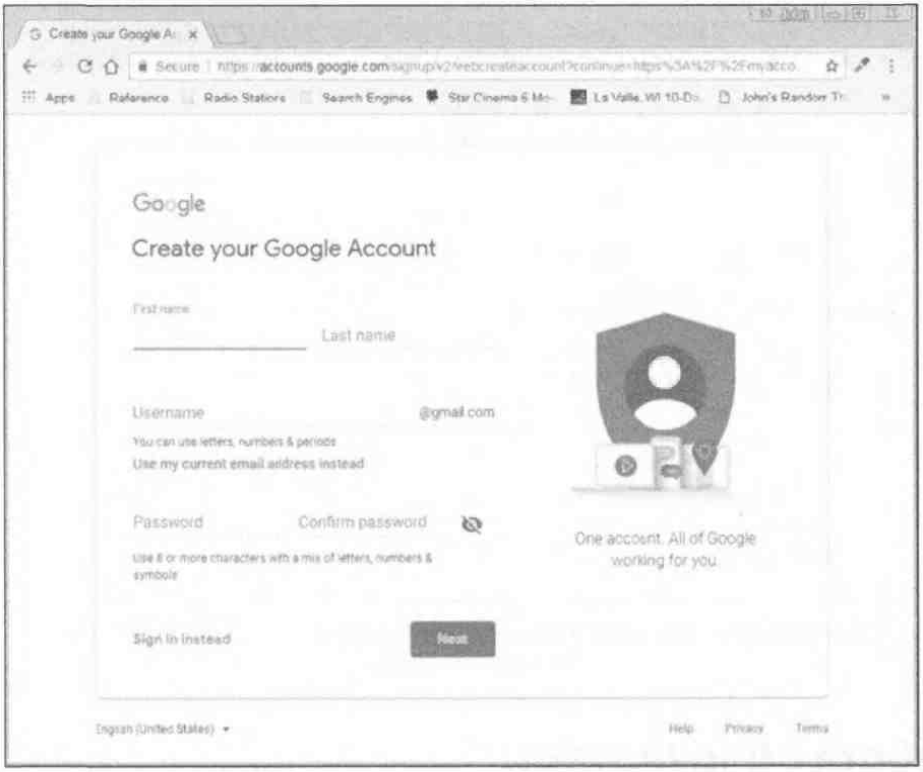


Рис. 4.1. Чтобы создать учетную запись Google, следуйте инструкциям

Вход в систему

После создания и подтверждения учетной записи, вы можете осуществить вход. Прежде чем вы сможете эффективно использовать Colab, необходим вход в учетную запись. Дело в том, что для определенных задач Colab полагается на ваш Google Drive. Вы также можете хранить блокноты в других местах, таких как GitHub, но наличие учетной записи Google гарантирует, что все будет работать, как запланировано. Чтобы войти в свою учетную запись, перейдите на страницу `https://accounts.google.com/`, укажите свой адрес электронной

почты и пароль, а затем щелкните на кнопке Next (Далее). Вы увидите страницу входа, показанную на рис. 4.2.

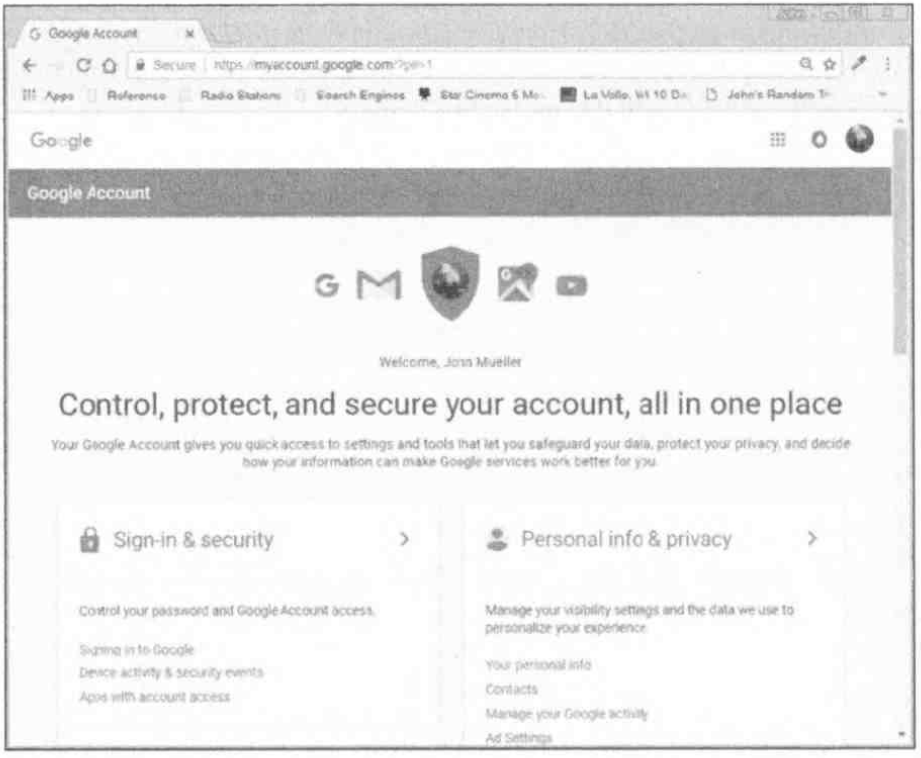


Рис. 4.2. Страница входа предоставляет доступ ко всем основным функциям, включая ваш диск

Работа с блокнотами

Как и в случае с Jupyter Notebook, блокнот служит основой взаимодействия с Colab. Если вы поместите указатель мыши в определенных местах страницы приветствия по адресу `https://colab.research.google.com/notebooks/welcome.ipynb`, то увидите возможности взаимодействия со страницей в результате добавления кода или текстовых записей (которые по необходимости можно использовать для заметок). Эти записи активны, поэтому вы можете взаимодействовать с ними. Вы также можете перемещать ячейки и копировать полученный материал на свой диск Google Drive. Цель этой главы — продемонстрировать, как взаимодействовать с блокнотом Colab. В следующих разделах описано, как с помощью Colab решать основные задачи, связанные с блокнотом.

Создание нового блокнота

Чтобы создать новый блокнот, выберите пункт меню File⇨New Python 3 Notebook (Файл⇨Новый блокнот Python 3). Вы увидите новый блокнот Python 3, подобный показанному на рис. 4.3. Новый блокнот выглядит похоже, но не точно так же, как в Notebook. Тем не менее все прежние функции остались. По желанию вы также можете создать новый блокнот Python 2, но эта тема в данной книге не обсуждается.

Рис. 4.3. Новый блокнот Python 3 создается точно так же, как и в обычном режиме

Блокнот, показанный на рис. 4.3, позволяет изменить имя файла щелчком на нем, точно так же, как и Notebook. Некоторые функции работают иначе, но дают одинаковые результаты. Например, чтобы запустить код в определенной ячейке, щелкните на стрелке вправо в левой части этой ячейки. В отличие от Notebook, фокус не переходит на следующую ячейку, поэтому вы должны выбрать следующую ячейку непосредственно или щелкнуть на кнопке Next Cell (Следующая ячейка) либо Previous Cell (Предыдущая ячейка) панели инструментов.

Открытие существующих блокнотов

Вы можете открывать существующие блокноты, находящиеся в локальном хранилище, на Google Drive или на GitHub. Вы также можете открыть любой из примеров Colab или загрузить файлы из таких источников, к которым у вас есть доступ, включая сетевой диск в вашей системе. В любом случае сначала выберите пункт меню File⇨Open Notebook (Файл⇨Открыть блокнот), чтобы открыть диалоговое окно, показанное на рис. 4.4.

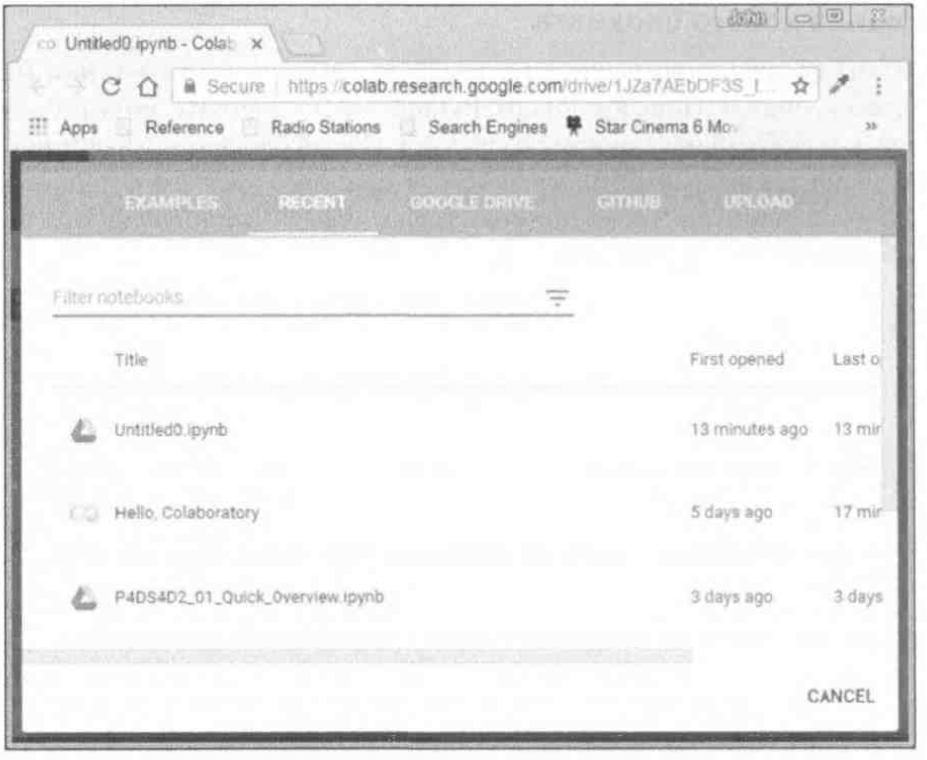


Рис. 4.4. Используйте это диалоговое окно, чтобы открывать существующие блокноты

В стандартном представлении отображаются все файлы, которые вы недавно открывали, независимо от их местоположения. Файлы отображаются в алфавитном порядке. Вы можете отфильтровать отображаемые элементы, введя соответствующую строку в поле Filter Notebooks (Фильтровать блокноты). Вверху находятся другие варианты открытия блокнотов.

СОВЕТ

Даже если вы не вошли в систему, вы все равно можете получить доступ к примерам проектов Colab. Эти проекты помогут вам изучить Colab, но не позволят что-либо делать со своими проектами. Тем не менее вы можете по-прежнему экспериментировать с Colab, не входя первоначально в Google. В следующих разделах эти варианты обсуждаются более подробно.

Использование Google Drive для существующих блокнотов

Google Drive является стандартным местоположением для многих операций в Colab, и вы всегда можете выбрать его в качестве пункта назначения. При работе с Google Drive вы увидите список файлов, похожий на показанный на

рис. 4.4. Чтобы открыть тот или иной файл, щелкните на его ссылке в диалоговом окне. Файл откроется в текущей вкладке вашего браузера.

Использование GitHub для существующих блокнотов

При работе с GitHub сначала необходимо указать местоположение файла исходного кода в сети, как показано на рис. 4.5. Указан должен быть открытый проект; вы не сможете использовать Colab для доступа к частным проектам.

Рис. 4.5. При использовании GitHub следует указать местоположение файла исходного кода

После подключения к GitHub отобразятся два списка: хранилища, которые являются контейнерами для кода, связанного с конкретным проектом; и ветви, конкретная реализация кода. При выборе хранилища и ветви отображается список файлов блокнотов, которые вы можете загрузить в Colab. Щелкните на нужной ссылке, и она загрузится, как если бы вы использовали Google Drive.

Использование локального хранилища для существующих блокнотов

Если хотите использовать загружаемый исходный код этой книги или любой другой локальный исходный код, выберите вкладку Upload (Загрузки) диалогового окна. В центре находится одна кнопка Choose File (Выбрать файл), после щелчка на которой откроется диалоговое окно File Open (Открыть файл) вашего браузера. Необходимый для загрузки файл вы находите как обычно при открытии любого файла.

ЗАПОМНИ!

Выбрав файл и щелкнув на кнопке Open (Открыть), вы загружаете файл на Google Drive. Если вы вносите изменения в файл, то они появятся на Google Drive, а не на локальном диске. В зависимости от вашего браузера, вы обычно увидите новое окно с загруженным кодом. Но может быть также отображено сообщение об успешном завершении, и в этом случае вы должны открыть файл, используя ту же технику, что и при использовании Google Drive. В некоторых случаях браузер спрашивает, хотите ли вы покинуть текущую страницу. Вы должны подтвердить это.

СОВЕТ

Команда File⇨Upload Notebook (Файл⇨Загрузить блокнот) загружает также файл на Google Drive. Фактически блокнот загружается так же, как и файл любого другого типа, и отображается то же самое диалоговое окно. Если вы хотите загрузить файлы других типов, то самый быстрый способ — использование команды File⇨Upload Notebook (Файл⇨Загрузить блокнот).

Сохранение блокнотов

Для сохранения блокнота Colab предоставляет несколько вариантов. Однако ни один из них не работает с локальным диском. После того как вы загрузите содержимое с локального диска на Google Drive или GitHub, Colab манипулирует содержимым в облаке, а не на локальном диске. Чтобы сохранить обновления на локальном диске, необходимо загрузить файл, используя методы, описанные в разделе "Загрузка блокнотов" далее в этой главе. В следующих разделах рассматриваются облачные варианты сохранения блокнотов.

Использование Drive для сохранения блокнотов

Стандартное расположение хранения ваших данных — Google Drive. Когда вы выбираете пункт меню File⇨Save (Файл⇨Сохранить), создаваемое вами содержимое попадает в корневой каталог диска Google Drive. Если вы хотите сохранить содержимое в другой папке, выберите ее на Google Drive (`https://drive.google.com/`).

ЗАПОМНИ!

При сохранении Colab отслеживает версии вашего проекта. Но по мере устаревания этих версий Colab удаляет их. Чтобы сохранить версию, которая не устарела, используйте пункт меню File⇨Save and Pin Revision (Файл⇨Сохранить и закрепить версию). Чтобы просмотреть версии своего проекта, выберите пункт меню File⇨Revision History (Файл⇨История изменений). Результат показан на рис. 4.6. Обратите внимание, что первая запись закреплена. Вы также можете

закрепить записи, отметив их в списке истории. Журнал изменений показывает также дату изменения, кто его внес и размер полученного файла. Вы можете использовать этот список, чтобы восстановить предыдущую версию или загрузить ее на локальный диск.

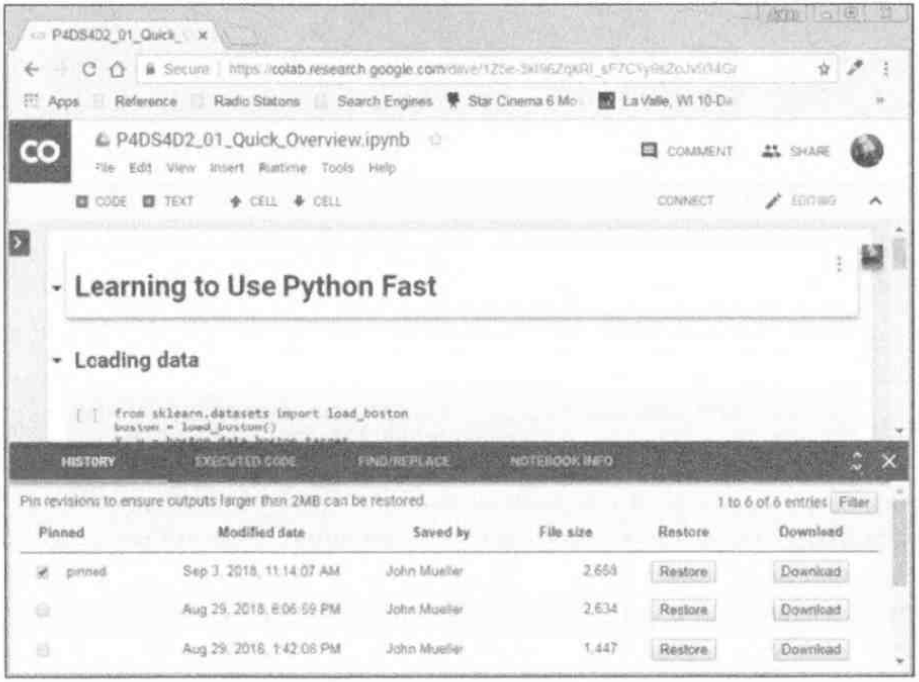


Рис. 4.6. Colab хранит историю изменений вашего проекта

Вы также можете сохранить копию своего проекта, выбрав пункт меню File⇨Save a Copy In Drive (Файл⇨Сохранить копию на диске). В имя копии добавляется слово *Copy*. Конечно, позже вы можете его переименовать. Colab сохраняет копию в текущей папке Google Drive.

Использование GitHub для сохранения блокнотов

GitHub представляет собой альтернативу Google Drive для хранения содержимого. Он предлагает организованный метод обмена кодом в целях обсуждения, просмотра и распространения. Вы можете найти GitHub по адресу `https://github.com/`.


ЗАПОМНИ!

При работе с GitHub от Colab вы можете использовать только общедоступные хранилища, хотя GitHub поддерживает также частные хранилища. Чтобы сохранить файл в GitHub, выберите пункт меню File⇨Save a Copy in GitHub (Файл⇨Сохранить копию в GitHub). Если

вы еще не вошли в GitHub, Colab отобразит окно, в котором запрашивается ваша информация для входа. После входа вы увидите диалоговое окно, похожее на показанное на рис. 4.7.

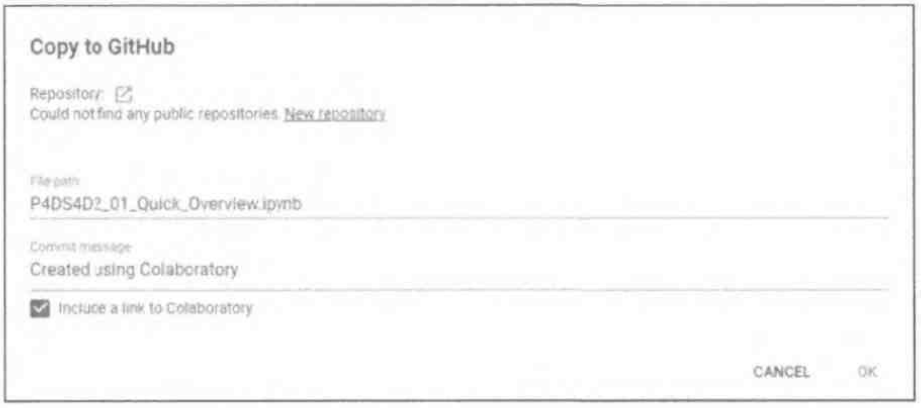


Рис. 4.7. Использование GitHub означает хранение ваших данных в хранилище

Обратите внимание, что на данный момент в этой учетной записи нет хранилища. Вы должны либо создать новое хранилище, либо выбрать существующее, в котором будут храниться ваши данные. После сохранения файла он появится в вашем хранилище GitHub по вашему выбору. Стандартно хранилище будет содержать ссылку для открытия данных в Colab, если только вы не откажетесь от этой функции.

Использование GitHub Gist для хранения блокнотов

GitHub Gist используется в качестве средства обмена с другими людьми отдельными файлами или другими ресурсами. Некоторые используют их и для полных проектов, но идея заключается в том, что у вас есть концепция, которой вы хотите поделиться, нечто не совсем сформированное и не представляющее пригодного для использования приложения. Вы можете прочитать больше о Gist на `https://help.github.com/articles/about-gists/`.

Как и GitHub, Gist представлены как в открытой, так и в засекреченной форме. Из Colab вы можете получить доступ как к открытым, так и к секретным спискам, но он автоматически сохраняет ваши файлы в секретной форме. Чтобы сохранить текущий проект как Gist, выберите пункт меню File⇨Save a Copy as a GitHub Gist (Файл⇨Сохранить копию как GitHub Gist). В отличие от GitHub, в этом случае вам не нужно создавать хранилище или делать что-то необычное. Файл сохраняется как Gist без каких-либо дополнительных усилий. Результирующая запись всегда содержит ссылку View in Colaboratory (рис. 4.8).

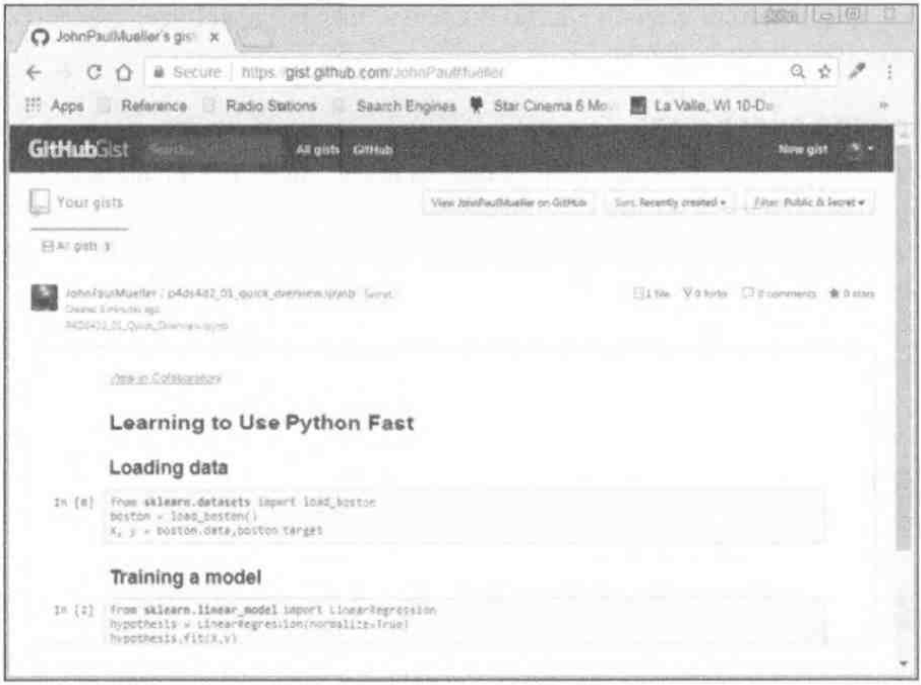


Рис. 4.8. Используйте Gist для хранения отдельных файлов или других ресурсов

Загрузка блокнотов

Colab поддерживает два метода загрузки блокнотов на локальный диск: как файлы `.ipynb` (пункт меню File⇨Download .ipynb (Файл⇨Загрузить .ipynb) и как файлы `.py` (пункт меню File⇨Download .py (Файл⇨Загрузить .py). В обоих случаях файл появляется в стандартном каталоге загрузки для браузера; Colab не предлагает метод для загрузки файла в определенный каталог.

Выполнение общих задач

Большинство задач в Colab выполняется аналогично таковым в Notebook. Например, вы можете создавать ячейки кода так же, как и в Notebook. Ячейки Markdown представлены в трех формах: текст, заголовок и оглавление. Они работают несколько иначе, чем ячейки Markdown в Notebook, но идея та же. Вы также можете редактировать и перемещать ячейки, как вы делаете это в Notebook. Одним из важнейших отличий является то, что нельзя изменить тип ячейки. Ячейка, которую вы создаете как заголовок, не может внезапно

трансформироваться в ячейку кода. В следующих разделах представлен краткий обзор различных функций.

Создание ячеек кода

Первая ячейка, которую Colab создает для вас, является ячейкой кода. Код, который вы создаете в Colab, использует все те же функции, что и Notebook. Но рядом с ячейкой отображается меню дополнительных функций, которые можно использовать в Colab и которых нет в Notebook (рис. 4.9).

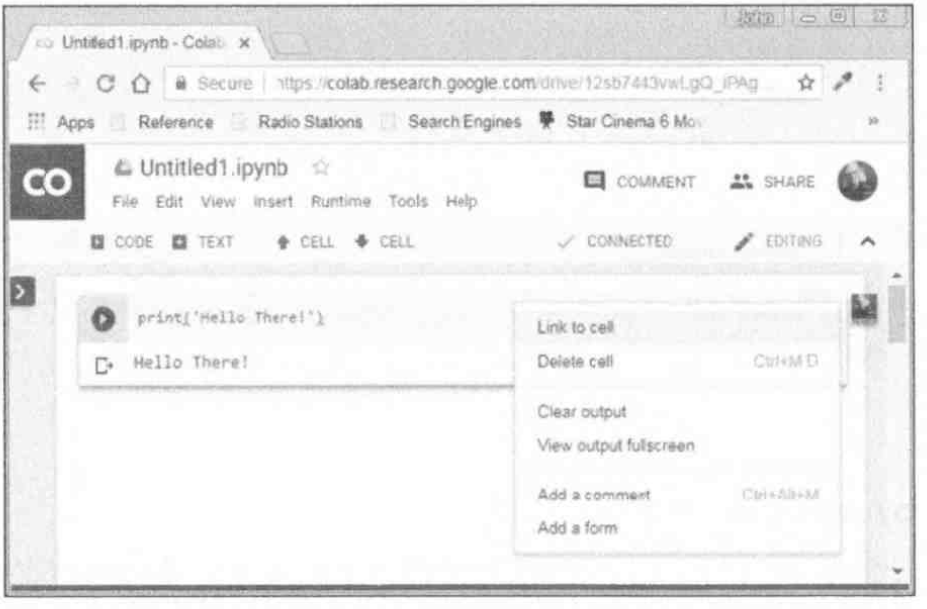


Рис. 4.9. Ячейки кода Colab содержат несколько дополнений, которых нет в Notebook

Возможности, показанные на рис. 4.9, используют для улучшения работы с кодом Colab. Ниже приведено краткое описание этих функций.

- Link to Cell (Ссылка на ячейку). Отображает диалоговое окно, содержащее ссылку, которая используется для доступа к определенной ячейке в блокноте. Вы можете встроить эту ссылку в любое место на веб-странице или в блокноте, чтобы кто-то другой мог получить доступ к этой конкретной ячейке. Собеседник видит весь блокнот, но ему уже не нужно искать ячейку, которую вы хотите обсудить.
- Delete Cell (Удалить ячейку). Удаляет ячейку из блокнота.
- Clear Output (Очистить вывод). Удаляет вывод из ячейки. Чтобы восстановить вывод, вы должны запустить код снова.

- View Output Fullscreen (Просмотр вывода в полноэкранном режиме). Отображает вывод (но не всю ячейку или любую другую часть блокнота) в полноэкранном режиме на устройстве. Эта возможность полезна при отображении значительного объема содержимого или когда подробное представление графики помогает объяснить тему. Чтобы выйти из полноэкранного режима, нажмите клавишу <Esc>.
- Add a Comment (Добавить комментарий). Создает всплывающую подсказку справа от ячейки. Это не то же самое, что комментарий к коду, который существует в соответствии с кодом, но распространяется на всю ячейку. Комментарии можно редактировать, удалять или разрешать. *Разрешенный комментарий* (resolved comment) — это тот комментарий, который привлекает внимание и более не применим.
- Add a Form (Добавить форму). Вставляет форму в ячейку справа от кода. Формы используются для графического ввода параметров. Формы не отображаются в Notebook, но они и не помешают вам запустить код. Подробнее о формах можно прочитать на странице `https://colab.research.google.com/notebooks/forms.ipynb`.

Ячейки кода также сообщают о коде и его исполнении. Если вы наведете курсор мыши на маленький значок рядом с выводом, отобразится информация о выполнении (рис. 4.10).

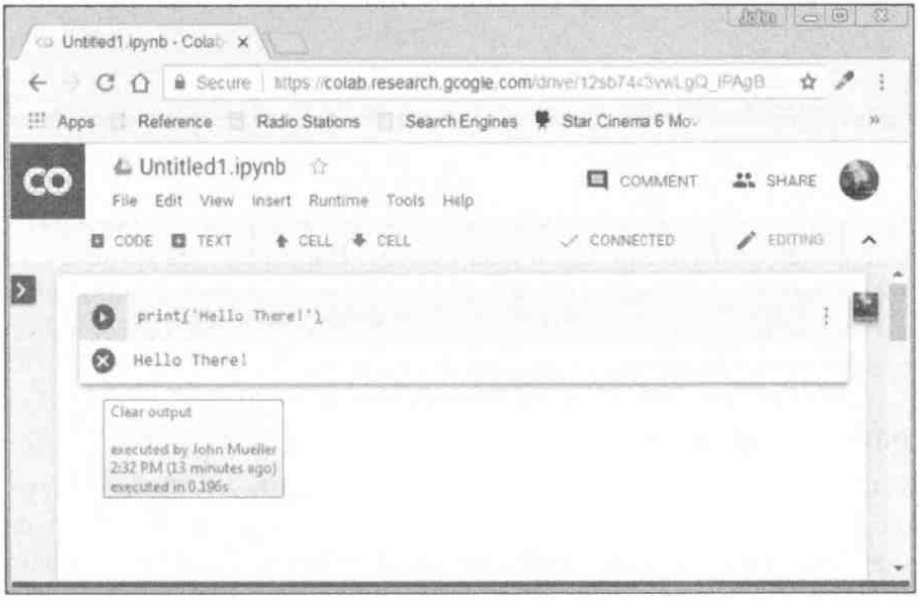


Рис. 4.10. Ячейки кода Colab содержат несколько дополнений, которых нет в Notebook

Создание текстовых ячеек

Текстовые ячейки работают так же, как ячейки Markup в Notebook. Однако на рис. 4.11 показано, что вы получаете дополнительную помощь в форматировании текста с использованием графического интерфейса. Разметка — такая же, но у вас есть возможность разрешить графический интерфейс для помощи в разметке. Например, в данном случае, чтобы создать знак # для заголовка, щелкните на значке двойной буквы T, расположенном первым в списке. Повторный щелчок на этом значке увеличит уровень заголовка. Справа показано, как будет выглядеть текст в блокноте.

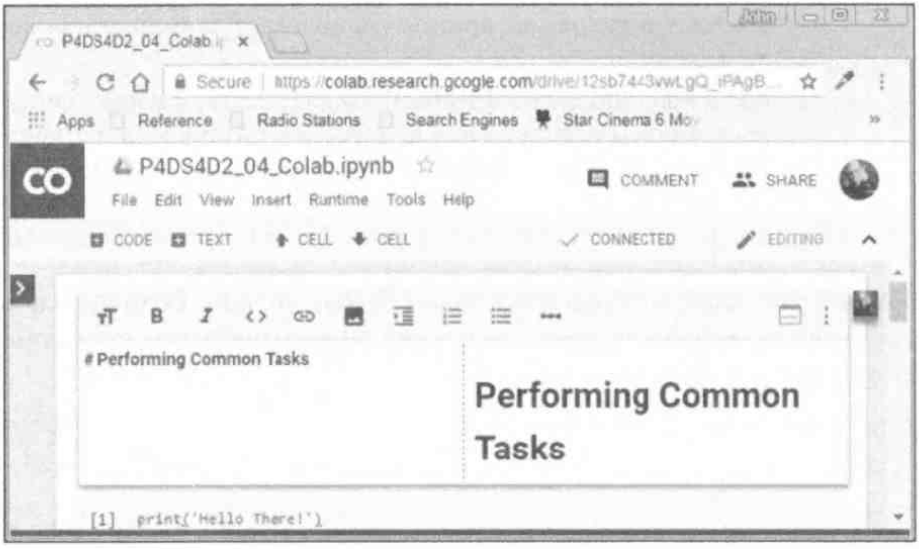


Рис. 4.11. Графический интерфейс упрощает форматирование текста

Обратите внимание на меню справа от текстовой ячейки: оно содержит некоторые из тех же пунктов, что и ячейка кода. Например, вы можете создать список ссылок, чтобы помочь людям получить доступ к определенным частям вашего блокнота через индекс. В отличие от Notebook, здесь нельзя запускать текстовые ячейки для просмотра разметки, которую они содержат.

Создание специальных ячеек

Специальные ячейки, которые предоставляет Colab, — это варианты текстовой ячейки. Эти специальные ячейки, доступ к которым осуществляется с помощью пункта меню Insert (Вставка), ускоряют создание необходимых ячеек. В следующих разделах описывается каждый из типов этих специальных ячеек.

Работа с заголовками

Когда вы выбираете пункт меню Insert⇨Section Header Cell (Вставка⇨Ячейка заголовка раздела), вы видите новую ячейку, созданную ниже текущей выбранной ячейки, у которой будет уровень заголовка 1. Вы можете увеличить уровень заголовка, щелкнув на кнопке с двойным значком буквы T. Графический интерфейс выглядит как на рис. 4.11, поэтому у вас есть все стандартные функции форматирования для текста.

Работа с оглавлением

Интересным дополнением Colab является автоматическое создание оглавления для блокнота. Чтобы использовать эту функцию, выберите пункт меню Insert⇨Table of Contents Cell (Вставка⇨Ячейка оглавления). На рис. 4.12 показан вывод данного конкретного примера.

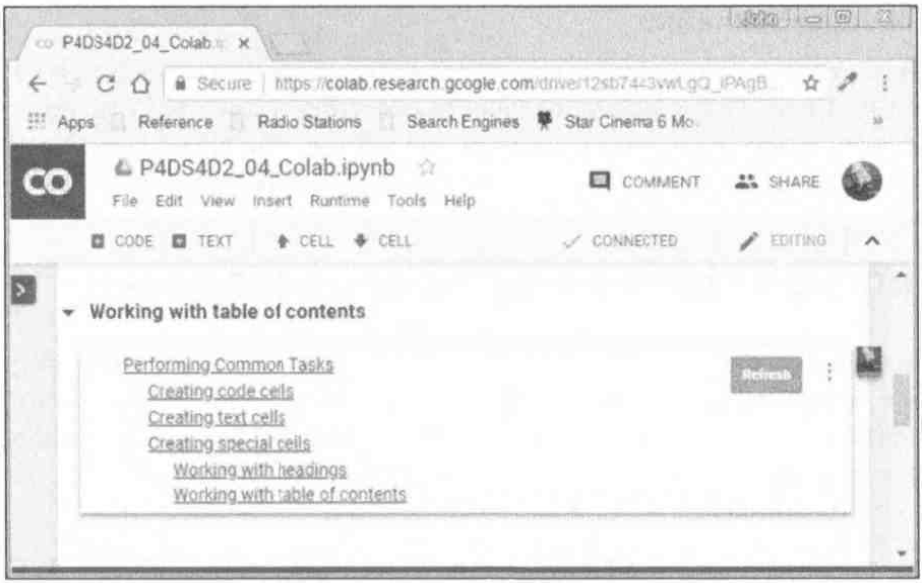


Рис. 4.12. Добавление оглавления в блокнот делает информацию более доступной

Ячейка оглавления содержит заголовок фактической ячейки, который вы можете изменить в своем блокноте так же, как любой другой заголовок. Оглавление появляется в выводе ячейки. Чтобы обновить оглавление, щелкните на кнопке Refresh (Обновить) справа от ячейки вывода. При работе с ячейкой оглавления вы также имеете доступ ко всем обычным функциям текстовых ячеек.

ВНИМАНИЕ!

Функция оглавления появилась и в Notebook, но ее нельзя использовать. Если вы планируете поделиться блокнотом с теми, кто использует Notebook, удалите оглавление, чтобы избежать путаницы.

Редактирование ячеек

Как и в Colab, в Notebook есть меню Edit (Правка), которые содержат все пункты вырезания, копирования и вставки ячейки. Но эти два продукта имеют некоторые интересные различия. Например, Notebook позволяет разделять и объединять ячейки. Colab содержит переключатель, позволяющий показать или скрыть код. Эти различия придают каждому продукту немного разные оттенки, но на самом деле не мешают использовать их для создания и изменения кода Python.

Движущиеся ячейки

Та же техника, которая используется для перемещения ячеек в Notebook, работает и с Colab. Единственное отличие заключается в том, что Colab полагается исключительно на кнопки панели инструментов, в то время как в Notebook есть также пункты перемещения ячеек в меню Edit (Правка).

Использование аппаратного ускорения

Код Colab выполняется на сервере Google. Все, что делает ваше компьютерное устройство, — это предоставляет браузер, который отображает код и результаты его выполнения. Следовательно, любое специальное оборудование на вашем вычислительном устройстве игнорируется, если вы не решите выполнять код локально.

СОВЕТ

К счастью, у вас есть еще один вариант при работе с Colab. Выберите пункт меню Edit⇨Notebook Settings (Правка⇨Настройки Notebook), чтобы открыть диалоговое окно Notebook Settings (Настройки Notebook), показанное на рис. 4.13. Это диалоговое окно позволяет выбрать среду выполнения Python, а также добавить графический интерфейс выполнения для кода. В статье, размещенной по адресу `https://medium.com/deep-learningturkey/google-colab-free-gpu-tutorial-e113627b9f5d`, приведены дополнительные сведения о работе этой функции. Доступность графического процессора не является приглашением для запуска больших вычислений с использованием Colab. На странице `https://research.google.com/`

colaboratory/faq.html#gpuavailability сообщается об ограничениях аппаратного ускорения Colab (в том числе о том, что оно может быть недоступно при необходимости).

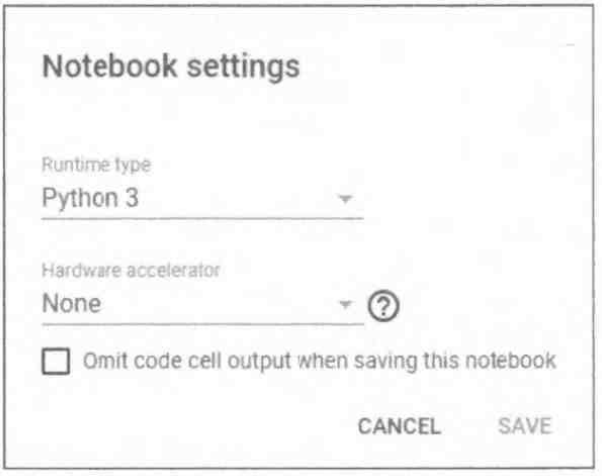


Рис. 4.13. Аппаратное ускорение выполнения кода

Диалоговое окно Notebook Settings (Настройки Notebook) позволяет также решить, включать ли вывод ячейки при сохранении блокнота. С учетом того, что, как правило, вы храните блокнот в облаке и загрузка больших файлов в браузер может занимать много времени, эта функция позволяет быстрее перезапустить сеанс. Конечно, за это приходится платить восстановлением всех нужных вам результатов.

Выполнение кода

Чтобы код был вам полезен, его нужно запустить. В предыдущих разделах упоминалась кнопка со стрелкой вправо, которая появляется в текущей ячейке. Щелчок на ней запускает только текущую ячейку. Конечно, у вас есть и другие варианты, кроме щелчка на стрелке вправо, и все эти возможности предоставляются в меню Runtime (Выполнение) и приведены ниже.

- **Запуск текущей ячейки**. Кроме щелчка на стрелке вправо, также можно выбрать пункт меню Runtime⇨Run the Focused Cell (Выполнение⇨Запуск ячейки в фокусе), чтобы выполнить код в текущей ячейке.
- **Запуск других ячеек**. В меню Runtime (Выполнение) Colab предоставляет пункты для выполнения кода в следующей ячейке,

в предыдущей или в выбранных ячейках. Выберите пункт, который соответствует ячейке или набору ячеек, которые хотите выполнить.

- **Запуск всех ячеек**. Иногда нужно выполнить весь код в блокноте. В этом случае выберите пункт меню Runtime⇨Run All (Выполнение⇨Запуск всего). Выполнение начинается в верхней части блокнота, в первой содержащей код ячейке, и продолжается до последней содержащей код ячейки блокнота. Остановить выполнение можно в любое время, выбрав пункт меню Runtime⇨Interrupt Execution (Выполнение⇨Прервать выполнение).


СОВЕТ

Выбор пункта меню Runtime⇨Manage Sessions (Выполнение⇨Управление сеансами) отображает диалоговое окно, содержащее список всех сеансов, которые в данный момент выполняются для вашей учетной записи в Colab. Это диалоговое окно можно использовать для выяснения, когда последний раз выполнялся код в этом блокноте и сколько памяти потребляет этот блокнот. Щелкните на кнопке Terminate (Завершить), чтобы завершить выполнение для указанного блокнота. Щелкните на кнопке Close (Закрыть), чтобы закрыть диалоговое окно и вернуться к текущему блокноту.

Просмотр блокнота

В левом поле блокнота есть стрелка вправо, после щелчка на которой отображается панель, содержащая вкладки с различной информацией о вашем блокноте. Вы также можете выбрать конкретную информацию для просмотра в меню View (Вид). Чтобы закрыть эту панель, щелкните на значке X в верхнем правом углу панели. В следующих разделах описывается каждая из этих частей информации.

Отображение оглавления

Выберите пункт меню View⇨Table of Contents (Вид⇨Оглавление), чтобы просмотреть оглавление вашего блокнота, как показано на рис. 4.14. Щелкнув на любой из записей, вы попадете в этот раздел блокнота.

В нижней части панели находится кнопка + Section (+ Раздел). Щелкните на ней, чтобы создать новую ячейку заголовка под выбранной ячейкой.

Получение информации о блокноте

Если выбрать пункт меню View⇨Notebook Information (Вид⇨Информация о блокноте), откроется панель в нижней части браузера (рис. 4.15). Эта панель

содержит информацию о размере блокнота, настройках и владельце. Обратите внимание, что на дисплее также отображается максимальный размер блокнота.

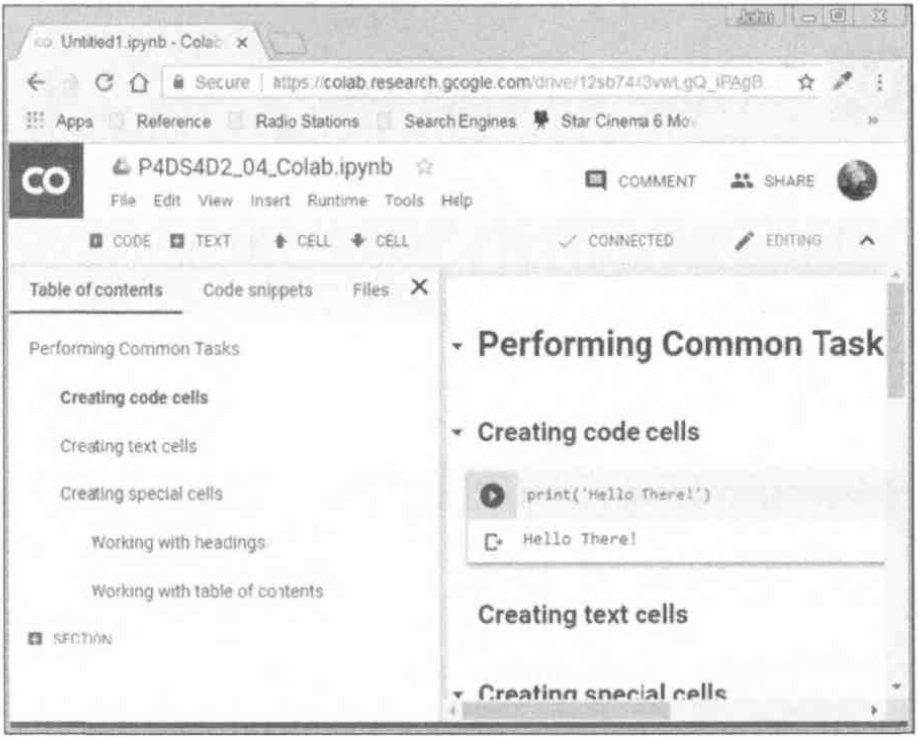


Рис. 4.14. Используйте оглавление для навигации по блокноту

На вкладке Notebook Info (Информация о блокноте) имеются также две ссылки Modify (Изменить). Обе ссылки отображают диалоговое окно Notebook Settings (Параметры блокнота), в котором вы можете выбрать тип среды выполнения и использовать аппаратное ускорение (см. выше раздел "Использование аппаратного ускорения").

Проверка выполнения кода

Colab отслеживает ваш код по мере его выполнения. Выберите пункт меню View⇨Executed Code History (Вид⇨История исполняемого кода), чтобы отобразить вкладку Executed Code (Выполненный код) в нижней части окна, как показано на рис. 4.16. Обратите также внимание, что номер, связанный с записью на вкладке Executed Code (Выполненный код), может не соответствовать номерам, связанным с соответствующими ячейками. Кроме того, каждый отдельный запуск кода получает отдельный номер.

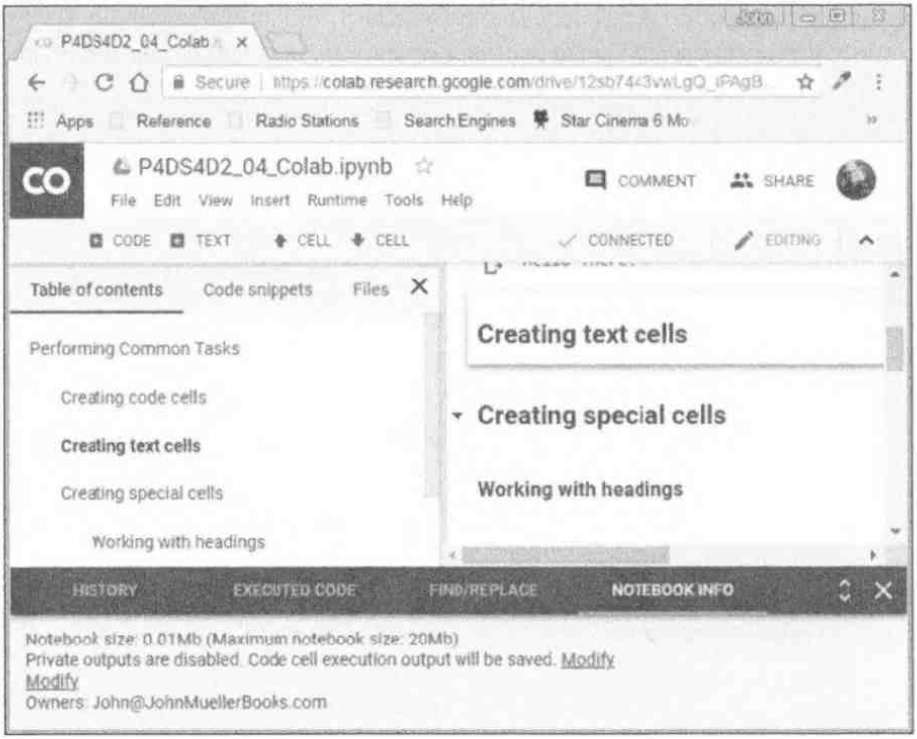


Рис. 4.15. Информация о блокноте включает в себя и размер, и настройки

Совместное использование блокнота

Вы можете поделиться своими блокнотами Colab несколькими способами. Например, сохранить его в GitHub или GitHub Gists. Но есть два наиболее простых метода:

- создать сообщение о совместном использовании и отправить его получателю;
- получить ссылку на код и отправить ее получателю.

В обоих случаях нужно щелкнуть на кнопке Share (Поделиться) в правом верхнем углу окна Colab, чтобы открыть диалоговое окно Share with Others (Поделиться с другими), представленное на рис. 4.17.

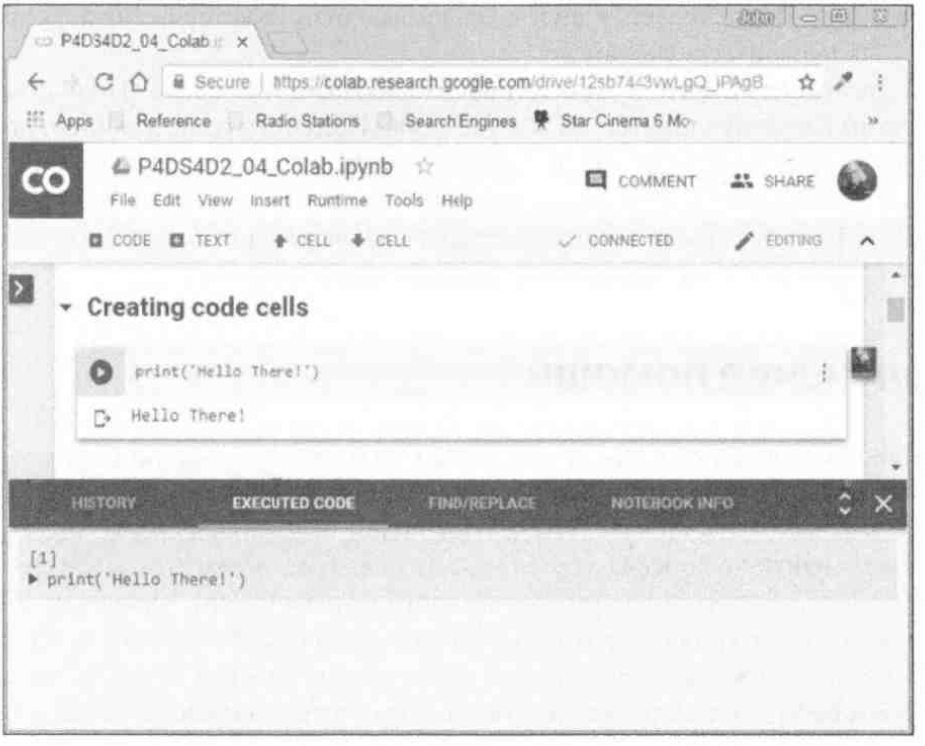


Рис. 4.16. Colab отслеживает, какой код вы выполняете и в каком порядке

Рис. 4.17. Отправьте сообщение или получите ссылку, чтобы поделиться своим блокнотом

Когда вы вводите одно или несколько имен в поле People (Люди), открывается дополнительное поле, в которое можно добавить сообщение для обмена. Вы можете ввести сообщение и щелкнуть на кнопке Send (Отправить), чтобы немедленно отправить ссылку. Если вместо этого щелкнуть на кнопке Advanced

(Дополнительно), откроется другое диалоговое окно, в котором можно определить, как поделиться блокнотом.

В правом верхнем углу диалогового окна Share with Others (Поделиться с другими) находится кнопка Get Shareable Link (Получить ссылку общего доступа). Щелкните на ней, чтобы отобразить диалоговое окно со ссылкой на ваш блокнот. После щечка на кнопке Copy Link (Копировать ссылку) URL-адрес помещается в буфер обмена вашего устройства, и вы можете вставить его в сообщения или иные формы общения с другими пользователями.

Получение помощи

Наиболее очевидное место, где можно получить помощь по Colab, — это меню справки Colab. Это меню содержит все обычные пункты для доступа к часто задаваемым вопросам (FAQ). В меню нет ссылки на общую справку, но ее можно найти по адресу `https://colab.research.google.com/notebooks/welcome.ipynb` (для этого необходимо зайти на сайт Colab). Меню также предоставляет пункты для отправки сообщения об ошибке и отправки отзыва.

Один из наиболее интригующих пунктов меню Help (Справка) — Search Code Snippets (Поиск фрагментов кода). Этот пункт открывает панель, показанную на рис. 4.18, в которой вы можете найти пример кода, который мог бы удовлетворить ваши потребности после небольшой модификации. После щелчка на кнопке Insert (Вставить) код вставляется в текущее местоположение курсора в ячейке, которая имеет фокус. Каждый из пунктов также демонстрирует пример кода.

Colab пользуется сильной поддержкой сообщества. Выбор пункта меню Help⇨Ask a Question on Stack Overflow (Справка⇨Задать вопрос на Stack Overflow) открывает новую вкладку браузера, где вы можете задавать вопросы другим пользователям. Вы увидите экран регистрации, если еще не зарегистрированы на Stack Overflow.

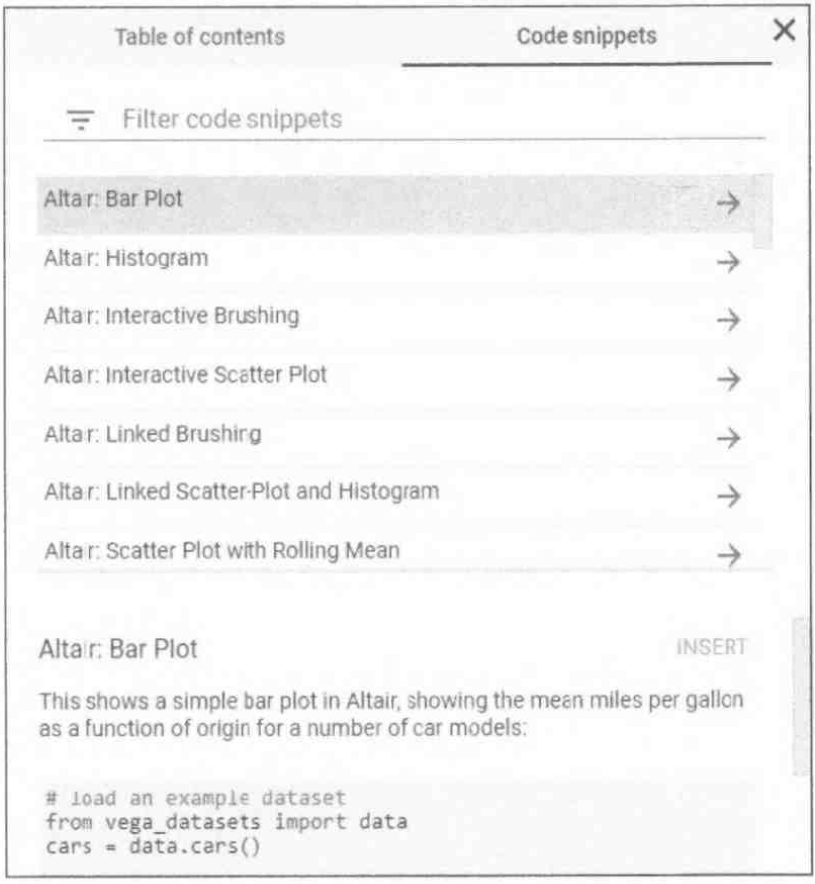


Рис. 4.18. Используйте фрагменты кода, чтобы писать свои приложения быстрее

2 Данные

В ЭТОЙ ЧАСТИ...

- Инструменты, входящие в комплект Jupyter Notebook
- Доступ к данным из различных источников и взаимодействие с ними
- Подготовка необходимых данных
- Формирование данных
- Разработка общего решения для обработки данных

Глава 5

Инструменты

В ЭТОЙ ГЛАВЕ...

- Работа с консолью Jupyter
- Работа с Jupyter Notebook
- Взаимодействие с мультимедиа и графикой

До этого момента в книге уделялось много времени работе с языком Python по выполнению задач обработки данных и фактически не задействовались инструменты, предоставляемые IDE Anaconda. Да, многое из того, что вы делаете, включает в себя ввод кода и просмотр того, что происходит. Однако, если вы на самом деле не знаете, как правильно использовать свои инструменты, вы упускаете возможность выполнять задачи проще и быстрее. Автоматизация является неотъемлемой частью решения задач по науке о данных на языке Python.

Эта глава посвящена работе с двумя основными инструментами IDE Anaconda: консолью Jupyter и Jupyter Notebook. Благодаря предыдущим главам вы получили некоторый опыт работы с обоими инструментами, но в этих главах каждый инструмент не рассматривался подробно, и для изучения последующих глав вам нужно знать эти инструменты гораздо лучше. Навыки, которые вы выработаете в этой главе, помогут выполнять задачи последующих глав быстрее и с меньшими усилиями.

В данной главе также рассматриваются задачи, которые вы можете решать с помощью приобретенных вами навыков. По мере изучения книги вы приобретете еще больше навыков, но эти задачи помогут оценить ваши новые навыки и понять, как их можно использовать для еще большего упрощения работы с Python.

СОВЕТ

Вам не нужно вводить исходный код этой главы вручную. Намного проще, если вы будете использовать загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле исходного кода `P4DS4D2_05_Understanding the Tools.ipynb`.

Использование консоли Jupyter

Консоль Python (доступная через Anaconda Prompt) позволяет интерактивно экспериментировать с наукой о данных. Вы можете нечто опробовать и сразу просмотреть результаты. Если допустите ошибку, закройте консоль и создайте новую. Консоль предназначена для игр и обдумывания возможностей. Следующие разделы помогут понять, что вы можете сделать для улучшения работы консоли Jupyter.

ЗАПОМНИ!

Стандартная консоль Python, поставляется с загружаемым экземпляром Python, а версия консоли Python из IDE Anaconda (доступ к которой осуществляется с помощью команды `ipython`) выглядит аналогично, и используя их, вы можете решать многие из задач. (С этого момента версия консоли Python из IDE Anaconda для простоты упоминается в тексте как просто консоль IPython.) Если вы уже знаете, как использовать стандартную консоль Python, у вас будет преимущество, когда дело дойдет до работы с консолью IPython. Тем не менее они имеют также и различия. Консоль IPython предоставляет усовершенствования, которых нет в стандартной консоли Python. Кроме того, решение некоторых задач, таких как вставка большого объема текста, осуществляется двумя консолями по-разному, поэтому, даже если вы знаете, как использовать стандартную консоль Python, будет полезно чтение следующих разделов.

Взаимодействие с текстом на экране

Когда вы впервые запустите консоль IPython, набрав **`ipython`** в приглашении Anaconda и нажав клавишу <Enter>, вы увидите экран, аналогичный показанному на рис. 5.1. Экран кажется загруженным текстом, но все это — полезная информация. В верхней строке отображается ваша версия Python и Anaconda. Ниже приведены три пункта справки (copyright (авторские права), credits (создатели) и license (лицензия)), которые вы вводите, чтобы получить больше информации о вашей версии этих двух продуктов. Например, когда вы

введете **credits** и нажмете клавишу <Enter>, отобразится список участников выпуска этой версии продукта.

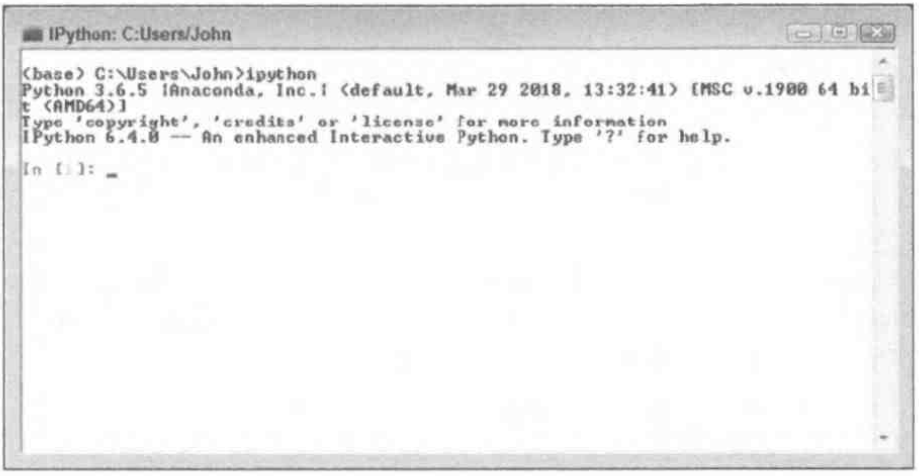


Рис. 5.1. На первом экране отображается информация о том, где можно получить дополнительную помощь

Обратите внимание, что это строка расширенной помощи. Если ввести **?** и нажать клавишу <Enter>, отобразятся пять команд, которые используются для следующих задач.

- **?**. Использование Jupyter для выполнения полезных задач.
- **object?**. Выяснение фактов о пакетах, объектах и методах, которые вы используете в Python для взаимодействия с данными.
- **object??**. Получение подробной информации о пакетах, объектах и методах (получаемые страницы могут быть неудобны для чтения).
- **%quickref**. Получение информации о магических функциях, которые предоставляет Jupyter.
- **help**. Информация о языке программирования Python (`help` и `?` — это не одно и то же, первое — для языка Python, второе — для функций IPython).

В зависимости от операционной системы вы должны иметь возможность щелкнуть правой кнопкой мыши в окне Anaconda Prompt, чтобы открыть контекстное меню, содержащее пункты для работы с текстом в окне. На рис. 5.2 показано контекстное меню для Windows. Это меню важно, поскольку оно позволяет взаимодействовать с текстом и скопировав результаты ваших экспериментов сохранять их в более постоянной форме.

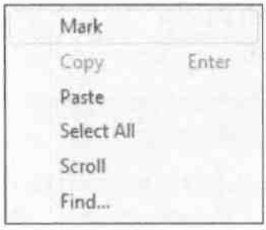


Рис. 5.2. Используя это контекстное меню, вы можете вырезать, копировать и вставлять текст

Вы можете получить доступ к тому же меню параметров, выбрав в системном меню (щелкните на значке в верхнем левом углу окна) меню Edit (Правка). Пункты, которые отобразятся, приведены ниже.

- Mark (Пометка). Выбирает конкретный текст, который вы хотите скопировать.
- Copy (Копировать). Помещает помеченный текст в буфер обмена (чтобы выполнить копирование, можете также нажать клавишу <Enter> после пометки текста).
- Paste (Вставка). Перемещает текст из буфера обмена в окно. К сожалению, с консолью IPython эта команда не работает для копирования нескольких строк текста. Вместо этого используйте магическую функцию `%paste`, которая копирует несколько строк текста.
- Select All (Выбрать все). Помечает весь текст, видимый в окне.
- Scroll (Прокрутка). Позволяет прокручивать окно с помощью клавиш со стрелками. Чтобы остановить прокрутку, нажмите клавишу <Enter>.
- Find (Найти). Отображает диалоговое окно Find (Найти), используемое для поиска текста в любом месте буфера экрана. Это исключительно полезная команда, поскольку вы можете быстро найти текст, который ввели ранее.


СОВЕТ

Одним из средств, предоставляемых консолью IPython и отсутствующих в стандартной консоли Python, является `cls`, или очистка экрана (clear screen). Чтобы очистить экран и упростить ввод новых команд, введите **`cls`** и нажмите клавишу <Enter>. Или используйте следующий код для сброса оболочки, что аналогично перезапуску ядра в Notebook:

```
import IPython
app = IPython.Application.instance()
app.shell.reset()
```

В этом случае нумерация начинается заново, что позволяет лучше видеть последовательность выполнения. Если ваша единственная цель — очистить переменные в памяти, используйте вместо этого магическую функцию `%reset`.

Изменение внешнего вида окна

Консоль Windows позволяет с легкостью изменять внешний вид окна Anaconda Prompt. В зависимости от используемой консоли и платформы вы можете обнаружить, что у вас есть и другие варианты. Если ваша платформа не предоставляет никакой гибкости в изменении внешнего вида окна Anaconda Prompt, вы все равно можете сделать это, используя магическую функцию, как описано в разделе "Использование магических функций" далее в этой главе. Чтобы изменить консоль Windows, щелкните на системном меню и выберите пункт Properties "Свойства". Откроется диалоговое окно, подобное показанному на рис. 5.3.

Рис. 5.3. Диалоговое окно свойств позволяет контролировать внешний вид окна

Каждая вкладка контролирует различные аспекты внешнего вида окна. Даже если вы работаете с IPython, базовая консоль по-прежнему влияет на то, что вы видите. Каждая из вкладок, показанных на рис. 5.3, описана ниже.

- **Options** (Параметры). Определяет размер курсора (большой курсор лучше работает при высокой яркости), количество запоминаемых окном команд и способ редактирования (например, режим вставки).
- **Font** (Шрифт). Задает шрифт, используемый для отображения текста в окне. Параметр Raster Fonts (Растровые шрифты) для большинства людей подходит лучше всего, но использование и других вариантов шрифтов может помочь лучше видеть текст при определенных условиях.
- **Layout** (Расположение). Определяет размер окна, его положение на экране и размер буфера, используемого для хранения информации, прокручиваемой вне поля зрения. Если вы обнаружите, что старые команды прокручиваются слишком быстро, может помочь увеличение размера окна. Аналогично, если обнаружите, что не можете найти более старые команды, может помочь увеличение размера буфера.
- **Colors** (Цвета). Определяет основные настройки цвета окна. Стандартная настройка с черным фоном и серым текстом трудна для чтения. Использование белого фона с черным текстом намного лучше. Поэтому нужно выбрать те настройки цвета, которые лучше всего подходят для вас. Эти цвета дополняются цветами, используемыми магической функцией `%colors`.

Получение справки по Python

Никто не может помнить абсолютно все о языке программирования. Даже лучшие программисты имеют провалы в памяти. Вот почему так важна помощь по языку. Без этой помощи программисты тратили бы много времени в Интернете на изучение пакетов, классов, методов и свойств.

Часть Python консоли IPython предоставляет два способа получения справки: режим справки (help mode) и интерактивная справка (interactive help). Используйте режим справки, когда хотите изучить язык и планируете потратить на это немного времени. Интерактивная справка лучше, когда вы точно знаете, в чем вам нужна помощь, и не хотите тратить много времени на просмотр других видов информации. В следующих разделах рассказывается, как получить помощь по языку Python, когда вам это нужно.

Вход в режим справки

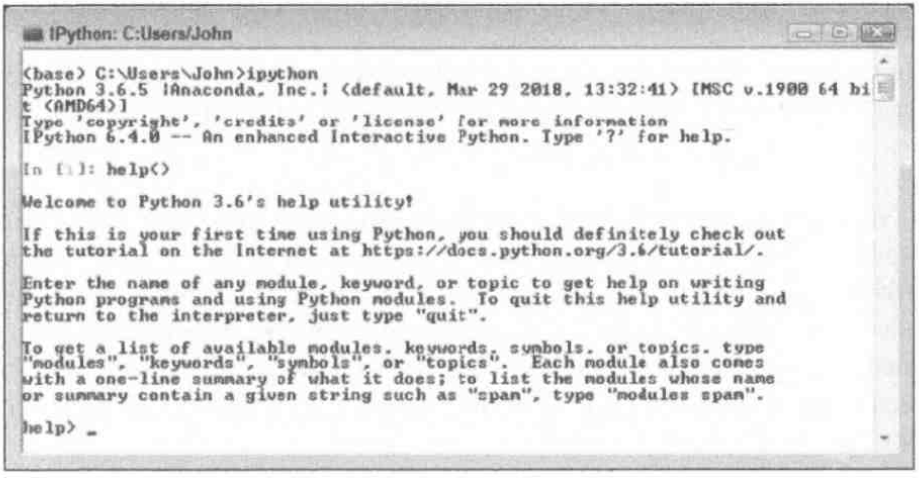


Рис. 5.4. Режим справки имеет специальное приглашение, `help>`

Чтобы получить справку о любом объекте или команде, введите имя объекта или команды и нажмите клавишу <Enter>. Вы также можете ввести любую из следующих команд, чтобы получить список других тем для обсуждения.

- **`modules`** (модули). Составляет и выводит список модулей, загруженных в данный момент. Этот список зависит от того, как настроен ваш экземпляр Python (базовый язык) в любой момент времени, поэтому список не будет тем же самым при каждом использовании этой команды. Выполнение команды может занять некоторое время, и выводимый список обычно довольно большой.
- **`keywords`** (ключевые слова). Выводит список ключевых слов Python, которые могут интересовать вас. Например, вы можете ввести ключевое слово **`assert`** и узнать о нем больше.
- **`symbols`** (символы). Выводит список символов, которые имеют особое значение в языке Python, например `*` для умножения и `<<` для смещения влево.
- **`topics`** (темы). Выводит список общих тем Python, таких как `CONVERSIONS`. Темы отображаются в верхнем, а не в нижнем регистре.

Запрос справки в режиме справки

Чтобы получить справку в режиме справки, введите имя модуля, ключевое слово, символ или тему, о которой вы хотите узнать больше, и нажмите клавишу <Enter>. Режим справки специфичен для языка Python, а значит, вы можете

сделать запрос о list, но не о созданном на его основе объекте mylist. Вы также не можете сделать запрос о специфичных для IPython функциях, таких как команда cls.

При работе с функциями, которые являются частью модуля, необходимо указать имя модуля. Например, если хотите узнать о методе version() в модуле sys, введите **sys.version** и нажмите клавишу <Enter> в приглашении, а не просто вводите **version**.

Если раздел справки слишком велик, чтобы отобразить его в виде единого экрана с информацией, внизу экрана вы увидите -- More --. Чтобы просмотреть справочную информацию по одной строке за раз, нажмите клавишу <Enter>, или пробел, чтобы переместить справочную информацию на один полный экран за раз. Вы не сможете вернуться назад в списке справки. Нажатие клавиши <Q> (или <q>) немедленно завершает вывод справочной информации.

Выход из режима справки

После того как закончите изучать справку, вам нужно вернуться к приглашению Python, чтобы вводить больше команд. Просто нажмите клавишу <Enter>, не вводя ничего в приглашении справки, или введите в приглашении справки ключевое слово **quit** (без скобок) и нажмите клавишу <Enter>.

Получение интерактивной справки

Чтобы не выходить из окна Python и получить справочную информацию, введите **help('<*тема*>')** и нажмите клавишу <Enter>. Например, чтобы получить справку по команде print, введите **help('print')** и нажмите клавишу <Enter>. Обратите внимание, что раздел справки заключен в одинарные кавычки. Если вы попытаетесь запросить помощь, не заключая тему в одинарные кавычки, то увидите сообщение об ошибке.


СОВЕТ

Интерактивная справка работает с любым модулем, ключевым словом или темой, которые поддерживает язык Python. Например, вы можете ввести **help('CONVERSIONS')** и нажать клавишу <Enter>, чтобы получить справку по теме CONVERSIONS. Важно отметить, что регистр символов по-прежнему важен при работе с интерактивной справкой. При вводе **help('conversions')** и нажатии клавиши <Enter> отображается сообщение о том, что справка недоступна.

Получение справки по IPython

Получение справки по IPython отличается от получения справки по Python. Когда вы получаете помощь по IPython, вы работаете со средой разработки, а

не с языком программирования. Чтобы получить справку по IPython, введите **?** и нажмите клавишу <Enter>. Вы увидите длинный список различных способов использования справки IPython.

Некоторые из наиболее важных форм помощи зависят от ввода ключевого слова со знаком вопроса. Например, если хотите узнать больше о команде `cls`, введите **`cls?`** или **`?cls`** и нажмите клавишу <Enter>. Где будет стоять знак вопроса, до или после команды, не имеет значения.


СОВЕТ

Если хотите получить более подробную информацию о команде или другой функции IPython, используйте два знака вопроса. Например, **`??cls`** отображает исходный код команды **`cls`**. Двойной знак вопроса (`??`) не всегда может давать дополнительную информацию, если искомой информации нет.

Если хотите прекратить отображать информацию IPython раньше, нажмите клавишу <Q>, чтобы выйти. В противном случае вы можете нажать пробел или клавишу <Enter>, чтобы отобразить каждый экран информации, пока справочная система не отобразит все доступное.

Использование магических функций

Удивительно, но вы действительно можете применить магию на своем компьютере! Jupyter предоставляет специальные *магические функции* (magic function), позволяющие вам выполнять с консолью Jupyter всевозможные удивительные задачи. Обзор магических функций представлен в следующих разделах. Некоторые из магических функций используются далее в книге. Однако имеет смысл уделить некоторое время их изучению.

Получение списка магических функций

Лучший способ начать работу с магическими функциями — это получить их список, введя **`%quickref`** и нажав клавишу <Enter>. Вы увидите экран справки, аналогичный показанному на рис. 5.5. Читать список может быть сложно, поэтому не торопитесь.

Работа с магическими функциями

Большинство магических функций начинаются либо с одного знака процента (`%`), либо с двух (`%%`). Функции, имеющие один знак процента, работают на уровне командной строки, в то время как имеющие два знака процента работают на уровне ячеек. Ячейки Jupyter Notebook обсуждаются далее в этой главе. На настоящий момент все, что вам действительно нужно знать, — это то, что

в консоли IPython обычно используют магические функции с одним знаком процента.

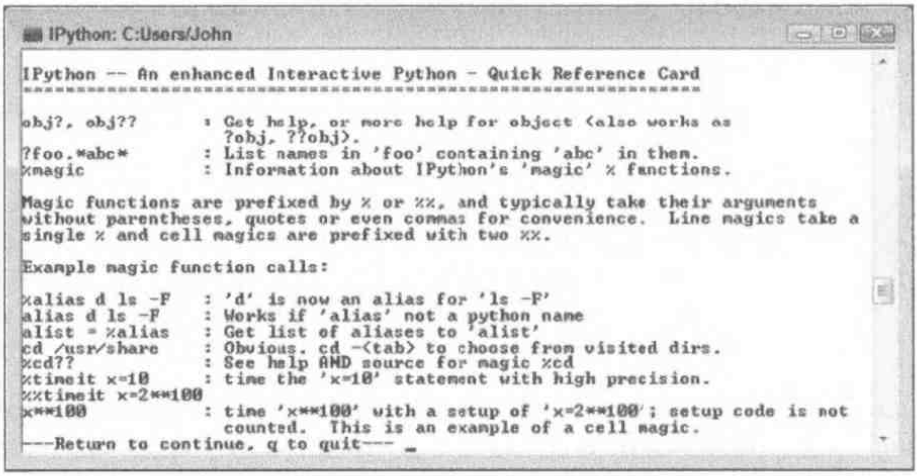


Рис. 5.5. Уделите время магической функции `help` — она предоставляет много информации


ЗАПОМНИ!

При использовании, большинство магических функций отображают информацию о состоянии. Например, когда вы вводите **`%cd`** и нажимаете клавишу <Enter>, вы видите текущий каталог. Чтобы сменить каталог, введите **`%cd`** и новое местоположение каталога в вашей системе. Но из этого правила есть исключения. Например, ввод **`%cls`** без параметров очищает только один экран.

Одна из наиболее интересных магических функций — это `%colors`. Вы можете использовать эту функцию для изменения цветов отображаемой на экране информации, что весьма полезно при использовании различных устройств. Доступны следующие варианты: `NoColor` (все в черно-белом режиме), `Linux` (стандартная настройка) и `LightBG` (когда используется сине-зеленая цветовая схема). Данная конкретная функция является еще одним исключением из правила. Ввод только **`%colors`** отображает не текущую цветовую схему, а сообщение об ошибке.

Обнаружение объектов

Python построен на объектах. На самом деле вы ничего не можете сделать в Python, не работая с какими-либо объектами. Имея это в виду, полезно знать, как именно определить, с каким объектом вы работаете и какие функции он предоставляет. Следующие разделы помогут вам обнаруживать объекты Python, которые вы используете при программировании.

Получение справки по объекту

В IPython вы можете запрашивать информацию о конкретных объектах, используя имя объекта и знак вопроса (?). Например, если вы хотите узнать больше об объекте `list` по имени `mylist`, введите **`mylist?`** и нажмите клавишу <Enter>. Отобразится тип `mylist`, содержимое в строковой форме, длина и строковая документация, содержащая краткий обзор `mylist`.

Если вам нужна подробная справка о `mylist`, вместо этого введите **`help(mylist)`** и нажмите клавишу <Enter>. Отобразится та же справка, что и при запросе информации о `list` в Python. Однако вы получите информацию, соответствующую конкретному объекту, по которому вам нужна помощь, вместо того, чтобы сначала выяснить тип объекта, а затем запрашивать информацию для этого объекта.

Получение специфики объекта

Функция `dir()` часто упускается из виду, но это важный способ узнать о специфике объекта. Чтобы получить список свойств и методов, связанных с любым объектом, используйте `dir(<object name>)`. Например, если вы создаете список под именем `mylist` и хотите узнать, что вы можете с ним делать, введите **`dir(mylist)`** и нажмите клавишу <Enter>. Консоль IPython отобразит список методов и свойств, специфичных для объекта `mylist`.

Использование справки по объекту IPython

Python предоставляет один уровень справки о ваших объектах, а IPython — другой. Если хотите узнать о вашем объекте больше информации, чем предоставляет Python, попробуйте использовать с ним знак вопроса. Например, при работе со списком по имени `mylist` введите **`mylist?`** и нажмите клавишу <Enter>, чтобы узнать тип объекта, его содержимое, длину и связанную строку документации. Строка документации (`docstring`) предоставляет краткий обзор информации об использовании этого типа, чего вполне достаточно для поиска более подробной информации об объекте.

Использование одного знака вопроса заставляет IPython обрезать слишком длинное содержимое. Если хотите получить полное содержимое об объекте, используйте двойной знак вопроса (??). Например, введите **`mylist??`** и нажмите клавишу <Enter>, чтобы увидеть все подробности (хотя их может и не быть). Когда это возможно, IPython предоставляет вам полный исходный код объекта (при условии, что он доступен).

С объектами вы также можете использовать магические функции. Эти функции упрощают вывод справки и предоставляют только необходимую информацию, как показано ниже.

- **%pdoc**. Выводит строку документации для объекта.
- **%pdef**. Показывает, как вызвать объект (при условии, что он может быть вызван).
- **%source**. Отображает исходный код объекта (при условии, что источник доступен).
- **%file**. Выводит имя файла, содержащего исходный код объекта.
- **%pinfo**. Отображает подробную информацию об объекте (зачастую больше, чем предоставлено только справкой).
- **%pinfo2**. Отображает подробную дополнительную информацию об объекте (при наличии).

Использование Jupyter Notebook

Консоль IPython, описанную в предыдущих разделах, обычно используют только для игр с кодом. Конечно, она прекрасно подходит для этой цели. Тем не менее, интегрированная среда разработки (IDE) Jupyter Notebook является еще одной частью набора инструментов Anaconda и может многое сделать для вас. В следующих разделах описаны некоторые интересные и полезные особенности Jupyter Notebook (или просто Notebook).

Работа со стилями

Notebook превосходит практически любую другую IDE тем, что он позволяет сделать вывод привлекательным. Вместо того чтобы загромождать экран кучей старого кода, вы можете использовать Notebook для создания разделов и добавления стилей, чтобы выводимые данные были хорошо оформлены. В итоге вы получите красивый отчет, который содержит исполняемый код. Такой улучшенный вывод обусловлен использованием стилей.

Когда вы вводите код в Notebook, вы помещаете его в ячейку. Каждый раздел кода, который вы создаете, помещается в отдельную ячейку. Когда вам нужно создать новую ячейку, вы щелкаете на кнопке Insert Cell Below (Вставить ячейку ниже) (кнопка со знаком "плюс") панели инструментов. Аналогично, если вы решили, что ячейка вам больше не нужна, выберите ее, а затем щелкните на кнопке Cut Cell (Вырезать ячейку) (кнопка с изображением ножниц), чтобы поместить удаленную ячейку в буфер обмена, или выберите пункт меню Edit⇨Delete Cell (Правка⇨Удалить ячейку), чтобы полностью удалить ее.

Стандартным стилем ячейки является Code (Код). Однако, если вы щелкнете на стрелке вниз рядом с пунктом Code (Код), появится список стилей, как показано на рис. 5.6.

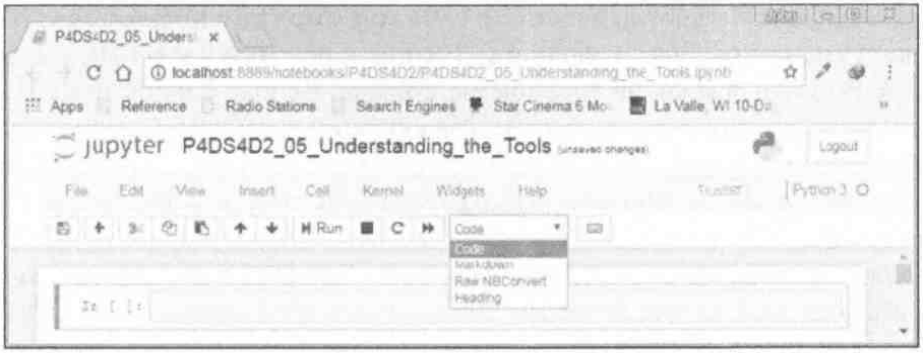


Рис. 5.6. Notebook упрощает применение стилей

Стили помогают оформлять содержимое различными способами. Стиль Markdown вполне определенно используется для разделения различных записей. Чтобы опробовать его, выберите в раскрывающемся списке стиль Markdown, введите заголовок раздела этой главы, **`# Using Jupyter Notebook`**, в первой ячейке, а затем щелкните на кнопке Run (Выполнить). Содержимое изменится на заголовок. Одиночный знак решетки (#) указывает Notebook, что это заголовок первого уровня. Обратите внимание, что щелчок на кнопке Run (Выполнить) автоматически добавляет новую ячейку и помещает в нее курсор. Чтобы добавить заголовок второго уровня, выберите стиль Markdown из раскрывающегося списка, введите **`## Working with styles`** и щелкните на кнопке Run (Выполнить). На рис. 5.7 показано, что две записи действительно являются заголовками и вторая запись меньше первой.

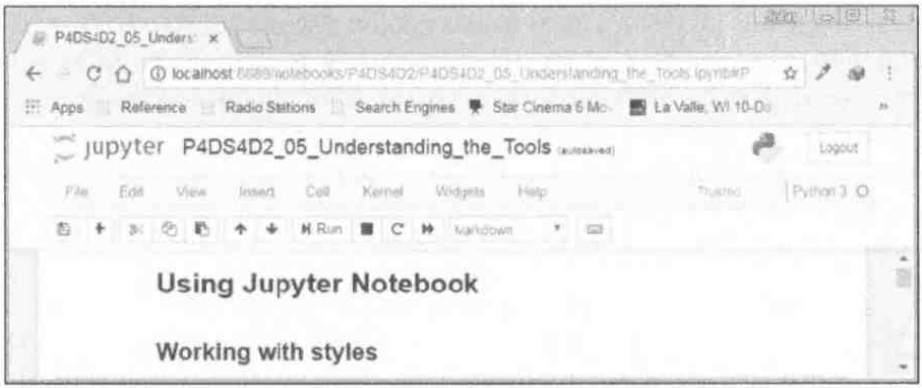


Рис. 5.7. Добавление заголовков облегчает разделение содержимого в записях

Стиль Markdown позволяет также добавлять содержимое HTML, способное содержать все, что содержит веб-страница в отношении стандартных тегов HTML. Другой способ создания заголовка первого уровня — определить

тип ячейки как Markdown, ввести **<h1>Using Jupyter Notebook</h1>** и щелкнуть на кнопке Run (Выполнить). В общем, используйте HTML для документирования и ссылок на внешние материалы. Использование тегов HTML позволяет включать такие элементы, как списки и даже изображения. Короче говоря, вы можете фактически включить фрагмент документа HTML в состав ячейки, что делает Notebook чем-то намного бо́льшим, чем просто средством записи кода.

Использование параметра форматирования Raw NBConvert выходит за рамки описания этой книги. Отметим лишь, что он предоставляет средства для включения такой информации, которую не следует изменять конвертером (NBConvert). Вы можете выводить содержимое Notebook в различных форматах, и NBConvert выполнит эту задачу сам. Вы можете прочитать об этой функции по адресу `https://nbconvert.readthedocs.io/en/latest/`. Стиль Raw NBConvert должен позволить вам включать специальное содержимое, такое как содержимое Lamport TeX (LaTeX). Система документов LaTeX не привязана к конкретному редактору — это просто средство кодирования научных документов.

Перезапуск ядра

Каждый раз, когда вы выполняете задачу в своем Notebook, вы создаете переменные, импортируете модули и выполняете множество других задач, затрудняющих рабочую среду. В какой-то момент все начинает работать не так, как должно. Чтобы решить эту проблему, щелкните на кнопке Restart Kernel (Перезапуск ядра) (кнопка с открытым кружком и стрелкой на одном конце) после сохранения документа щелчком на кнопке Save and Checkpoint (Сохранить и установить контрольную точку) (кнопка с символом дискеты). Затем можете снова запустить код, чтобы убедиться, что он работает так, как вы думали.

Иногда к отказу ядра приводит ошибка. Ваш документ начинает работать странно, медленно обновляется или демонстрирует другие признаки неполадок. Решение опять же заключается в перезапуске ядра, гарантирующем, что среда чистая и ядро работает так, как должно.

ВНИМАНИЕ!

Всякий раз, когда вы щелкаете на кнопке Restart Kernel (Перезапуск ядра), отображается предупреждение, показанное на рис. 5.8. Обратите внимание на это предупреждение, поскольку при перезапуске ядра можно потерять временные изменения. Всегда сохраняйте документ перед тем, как перезапустить ядро.

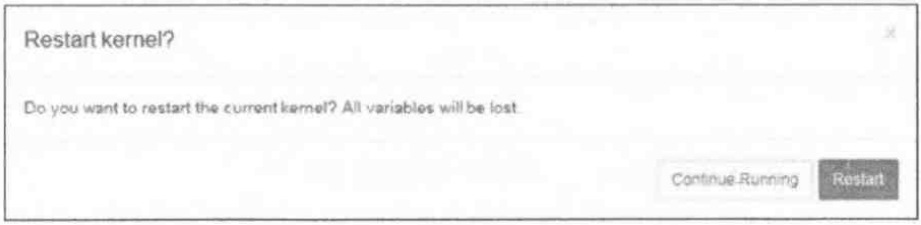


Рис. 5.8. Сохраните документ перед перезапуском ядра

Восстановление контрольной точки

В какой-то момент вы можете обнаружить, что допустили ошибку. Но поскольку в Notebook отсутствует кнопка Undo (Отменить), создавайте контрольные точки каждый раз, когда завершаете задачу. Создание контрольных точек при стабильной работе документа обеспечит быстрое восстановление после ошибок.

Чтобы восстановить настройки, установленные в контрольной точке, выберите пункт меню File⇨Revert to Checkpoint (Файл⇨Вернуться к контрольной точке). В появившемся списке доступных контрольных точек выберите ту, которую хотите использовать. Когда будете выбирать контрольную точку, отобразится предупреждающее сообщение (рис. 5.9). Щелкните на кнопке Revert (Восстановить), и любая прежняя информация исчезает, и текущей становится информация, находящаяся в контрольной точке.

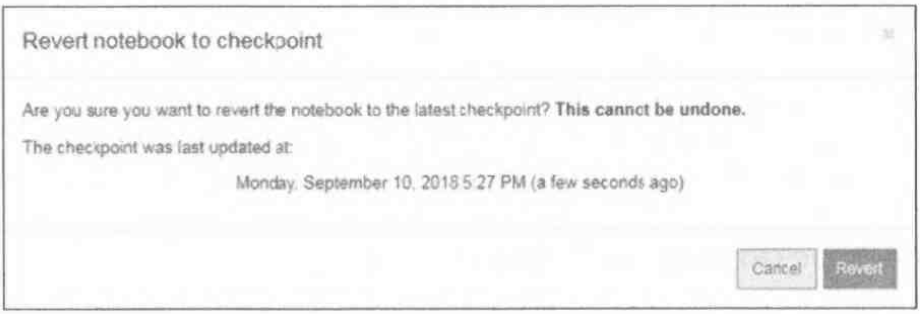


Рис. 5.9. Возврат к предыдущей настройке Notebook для исправления ошибки

Интеграция мультимедиа и графики

Изображения стоят тысяч слов и могут сказать очень многое (или, по крайней мере, они делают это с гораздо меньшими усилиями). Notebook является

платформой программирования и демонстрации. Вас может удивить то, на что она способна. В следующих разделах представлен краткий обзор некоторых наиболее интересных функций.

Встраивание графиков и других изображений

В какой-то момент вы, возможно, заметите ячейки Notebook со встроенными в него мультимедиа или графикой и поинтересуетесь, почему вы не видите таких же эффектов в ваших собственных файлах. Фактически все графические примеры в книге представлены как часть кода. К счастью, вы можете получить больше магии, использовать магическую функцию `%matplotlib`. Возможные значения для этой функции: `'gtk'`, `'gtk3'`, `'inline'`, `'nbagg'`, `'osx'`, `'qt'`, `'qt4'`, `'qt5'`, `'tk'` и `'wx'`, каждое из которых определяет свое содержимое для построения графиков (код, используемый для фактического отображения графика) на экране.

Когда вы запустите функцию `%matplotlib inline`, все созданные вами графики становятся частью документа. Так, на рис. 8.1 (см. раздел об основах использования NetworkX в главе 8) показан график, который созданный непосредственно под рассматриваемым кодом.

Загрузка примеров с сайтов в Интернете

Поскольку некоторые примеры, которые вы видите в Интернете, могут быть трудными для понимания, если не загрузить их в свою собственную систему, вам также следует знать о магической функции `%load`. Все, что вам нужно, — это URL примера, который вы хотите видеть в своей системе. Попробуйте, например, `https://matplotlib.org/_downloads/pyplot_text.py`. Щелкните на кнопке Run Cell (Выполнить ячейку), и Notebook загрузит пример прямо в ячейку, и закомментирует вызов `%load`. Затем запустите пример и просмотрите результаты его работы в своей системе.

Получение сетевой графики и мультимедиа

В `Jupyter.display` содержится множество функций, необходимых для выполнения специальной обработки мультимедиа и графики. Импортируя необходимый класс, вы можете выполнять такие задачи, как встраивание изображений в свой Notebook. Вот пример встраивания одной из картинок из блога автора для этой главы:

```
from IPython.display import Image
Embed = Image(
'http://blog.johnmuellerbooks.com/' +
```

```
'wp-content/uploads/2015/04/Layer-Hens.jpg')
Embed
```

Код начинается с импорта требуемого класса, `Image`, и продолжается использованием его функций, которые сначала определяют, что нужно встраивать, а затем фактически встраивают изображение. Вывод этого примера показан на рис. 5.10.

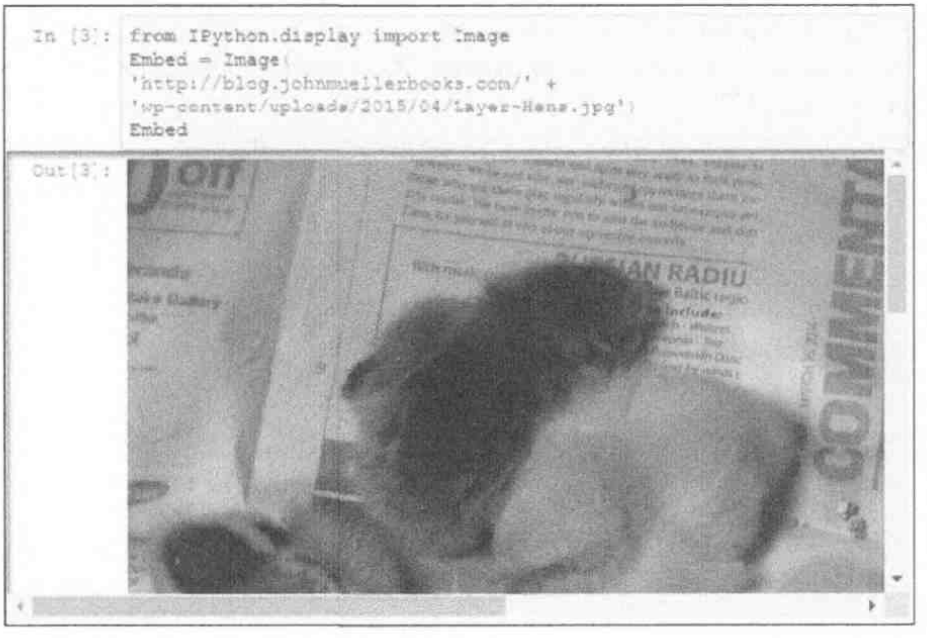


Рис. 5.10. Встраивание изображений может украсить вашу презентацию


СОВЕТ

Если вы ожидаете, что изображение со временем изменится, вместо встраивания создайте ссылку на него. Вы должны обновить ссылку, поскольку содержимое в Notebook является только ссылкой, а не фактическим изображением. Однако, когда изображение изменится, вы также увидите изменения в Notebook. Для этого вместо `Embed` используйте `SoftLinked = Image(url='http://blog.johnmuellerbooks.com/wpcontent/uploads/2015/04/Layer-Hens.jpg')`.

При регулярной работе со встроенными изображениями вы можете установить форму, в которую будут вставляться изображения. Например, можете встраивать их в файлы PDF. Для этого используйте код, подобный следующему:

```
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('pdf', 'svg')
```

При работе с Notebook у вас есть доступ к широкому разнообразию форматов. Обычно поддерживаются следующие форматы: `'png'`, `'retina'`, `'jpeg'`, `'svg'` и `'pdf'`.

Система отображения IPython удивительна, и в этом разделе мы даже не начали раскрывать ее возможности. Например, вы можете импортировать видео YouTube и поместить его непосредственно в свой Notebook как часть презентации, если хотите. Еще несколько функций отображения представлены на `http://nbviewer.jupyter.org/github/ipython/ipython/blob/1.x/examples/notebooks/%20Rich%20Display%20System.ipynb`.

Глава 6

Работа с реальными данными

В ЭТОЙ ГЛАВЕ...

- Управление потоками данных
- Работа с плоскими и неструктурированными файлами
- Взаимодействие с реляционными базами данных
- Использование NoSQL как источника данных
- Взаимодействие с данными в веб

Приложения науки о данных требуют данных по определению. Было бы неплохо, если бы вы могли просто пойти куда-нибудь в хранилище данных, купить необходимые данные в легко открываемом пакете, а затем написать приложение для доступа к ним. Тем не менее данные появляются в самых разных местах, в самых разных формах, и вы можете интерпретировать их по-разному. В каждой организации есть свой метод просмотра данных, которые также хранятся по-разному. Даже если система управления данными, используемая одной компанией, совпадает с системой управления данными, используемой другой компанией, маловероятно, что данные будут отображаться в том же формате или даже использовать те же типы данных. Короче говоря, прежде чем вы сможете выполнять какие-либо действия с данными, вы должны выяснить, как получить доступ к ним во всех их бесчисленных формах. Реальные данные требуют много предварительной работы, и, к счастью, Python вполне справляется с задачей манипулирования ими при необходимости.

В этой главе описаны методы, необходимые для доступа к данным в различных формах и местах. Например, потоки памяти (memory stream) представляют собой форму хранения данных, которую ваш компьютер поддерживает изначально; плоские файлы (flat file) существуют на вашем жестком диске; реляционные базы данных обычно находятся в сетях (хотя меньшие реляционные базы данных, такие как в Access, также могут появляться на вашем жестком диске); веб-данные обычно находятся в Интернете. Мы не будем рассматривать все доступные формы хранения данных (например, системы хранения в точках продаж (point-of-sale) или POS-системах). Вполне возможно, что целой книги по этой теме будет недостаточно, чтобы подробно рассмотреть тему форматов данных. Однако методы, описанные в этой главе, демонстрируют, как получить доступ к данным в форматах, с которыми вы чаще всего сталкиваетесь при работе с реальными данными.


СОВЕТ

Библиотека Scikit-learn включает в себя несколько *игрушечных* (toy) наборов данных (небольших наборов данных, предназначенных, чтобы вы играли с ними). Эти наборы данных достаточно сложные для выполнения ряда задач, таких как эксперименты в языке Python с задачами науки о данных. Поскольку эти данные легкодоступны, а создание слишком сложных для понимания примеров — это плохая идея, данная книга опирается на эти игрушечные наборы данных в качестве исходных данных для многих примеров. Несмотря на то что книга использует эти игрушечные наборы данных для упрощения примеров, методы, которые демонстрируются в ней, одинаково хорошо работают с реальными данными, к которым вы обращаетесь, используя методы, описанные в этой главе.


СОВЕТ

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле исходного кода `P4DS4D2_06_Dataset_Load.ipynb`.


ВНИМАНИЕ!

Важно, чтобы файлы `Colors.txt`, `Titanic.csv`, `Values.xls`, `Colorblk.jpg` и `XMLData.xml`, которые поставляются с загружаемым исходным кодом, находились в той же папке (каталоге), что и файлы Notebook. В противном случае примеры следующих разделов завершатся ошибкой ввода/вывода (input/output IO). Расположение файла зависит от используемой вами платформы. Например, в системе Windows вы найдете их в папке C `C:\Users\`*`ИмяПользователя`*`\P4DS4D2`, где *`ИмяПользователя`* — ваше имя пользователя. (В книге предполагается, что вы выбрали местоположение папки `P4DS4D2`

так, как описано в разделе "Определение хранилища кода" главы 3.) Чтобы примеры работали, скопируйте четыре файла из папки с загружаемыми исходными файлами в свою папку для Notebook.

Загрузка, потоковая передача и выборка данных

Хранение данных в локальной памяти компьютера представляет собой самый быстрый и надежный способ доступа к ним. Данные могут находиться где угодно. Однако на самом деле вы не взаимодействуете с данными в их хранилище. Вы загружаете данные в память из места хранения и затем взаимодействуете с ними в памяти.


ЗАПОМНИ!

Аналитики данных называют столбцы в базе данных *признаками* (feature) или *переменными* (variable). Строки — это *случаи* (case). Каждая строка представляет собой набор переменных, которые вы можете анализировать.

Загрузка небольших объемов данных в память

Самый удобный метод, который вы можете использовать для работы с данными, — это загрузить их непосредственно в память. Такой прием встречался в книге ранее несколько раз, но использован был игрушечный набор из библиотеки Scikit-learn. В этом разделе для ввода используется файл `Colors.txt`, показанный на рис. 6.1.

Рис. 6.1. Формат файла `Colors.txt`

В примере для решения задачи используются также встроенные (native) функции Python. Когда вы загружаете файл (любого типа), весь набор данных доступен постоянно, и процесс загрузки довольно короткий. Вот пример того, как работает эта техника:

```
with open("Colors.txt", 'r') as open_file:
    print('Colors.txt content:\n' + open_file.read())
```

Пример начинается с использования метода `open()` для получения объекта `file`. Функция `open()` получает имя файла и режим доступа. В данном случае режим доступа — чтение (`r`). Затем используется метод `read()` объекта `file` для чтения всех данных из файла. Если бы вы указали в методе `read()` аргумент размера, например `read(15)`, Python прочитал бы только то количество символов, которое вы указали, или остановился бы, достигнув конца файла (End Of File — `EOF`). Когда вы запустите этот пример, отобразится следующий вывод:

```
Colors.txt content:
Color     Value
Red       1
Orange    2
Yellow    3
Green     4
Blue      5
Purple    6
Black     7
White     8
```

ВНИМАНИЕ!

Весь набор данных загружается из библиотеки в свободную память. Конечно, процесс загрузки не удастся, если вашей системе не хватит памяти для хранения всего набора данных. Если возникнет эта проблема, рассмотрите другие методы работы с набором данных, такие как потоковая передача или выборка. Короче говоря, прежде чем использовать эту технику, вы должны убедиться, что набор данных действительно поместится в памяти. Но обычно при работе с игрушечными наборами данных из библиотеки Scikit-learn не возникает никаких проблем.

Загрузка в память большого количества данных

Некоторые наборы данных будут настолько большими, что вы не сможете разместить их полностью в памяти за один раз. Кроме того, вы можете обнаружить, что некоторые наборы данных загружаются очень медленно, поскольку они находятся на дистанционном сайте. Потоковая передача отвечает обеим потребностям, позволяя работать с данными постепенно. Загружая отдельные

фрагменты, вы получаете возможность работать только с частью данных и по мере их получения, а не ждать загрузки всего набора данных. Вот пример того, как вы можете загружать данные с помощью Python:

```
with open("Colors.txt", 'r') as open_file:
    for observation in open_file:
        print('Reading Data: ' + observation)
```

В этом примере используется файл `Colors.txt`, который содержит заголовок, а затем ряд записей, которые связывают название цвета с его значением. Файловый объект `open_file` содержит указатель на открытый файл.

Поскольку код выполняет чтение данных в цикле `for`, указатель файла перемещается к следующей записи. В `observation` каждая запись появляется по одной. Код выводит значение в `observation`, используя оператор `print`. Вы должны получить такой вывод:

```
Reading Data:  Color   Value

Reading Data:  Red     1

Reading Data:  Orange  2

Reading Data:  Yellow  3

Reading Data:  Green   4

Reading Data:  Blue    5

Reading Data:  Purple  6

Reading Data:  Black   7

Reading Data:  White   8
```

Python передает из источника каждую запись. Это означает, что вы должны выполнить чтение для каждой записи.

Генерация вариаций в данных изображения

Иногда вам нужно будет импортировать и анализировать данные изображений. Источник и тип изображения имеют значение. В книге приведено несколько примеров работы с изображениями. Однако хорошей отправной точкой будет чтение локально хранимого изображения, получение статистики о нем и отображение изображения на экране, как показано в следующем коде:

```
import matplotlib.image as img
import matplotlib.pyplot as plt
%matplotlib inline
```

```
image = img.imread("Colorblk.jpg")
print(image.shape)
print(image.size)
plt.imshow(image)
plt.show()
```

Пример начинается с импорта двух библиотек `matplotlib`: `image` и `pyplot`. Библиотека `image` считывает изображение в память, а библиотека `pyplot` отображает его на экране.

После того как код считает файл, он начинает с отображения свойства `shape` изображения — количества горизонтальных и вертикальных пикселей, а также их глубины. На рис. 6.2 показано, что изображение имеет размер 100 × 100 × 3 пикселя. Свойство `size` изображения является комбинацией этих трех элементов и составляет 30 000 байтов.

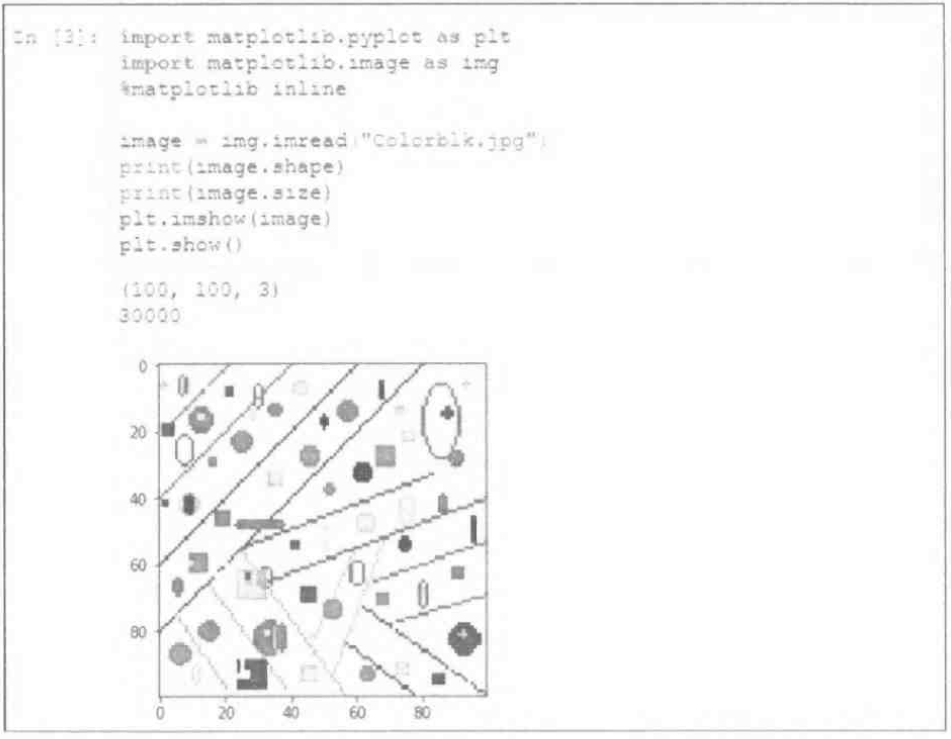


Рис. 6.2. Тестовое изображение высотой 100 пикселей и длиной 100 пикселей

Следующим шагом является загрузка изображения для отображения с помощью метода `imshow()`. Последний вызов метода `plt.show()` отображает изображение на экране, как показано на рис. 6.2. Этот метод представляет собой лишь один из нескольких способов взаимодействия с изображениями с использованием языка Python, чтобы вы могли каким-то образом их анализировать.

Выборка данных разными способами

Поток данных получает все записи из источника данных. Если вам нужны не все записи, вы можете сэкономить время и ресурсы, сделав выборку данных. Это означает извлечение определенного количества записей, например, каждую пятую запись или выборку случайных записей. Следующий код демонстрирует, как получить все остальные записи в файле `Colors.txt`:

```
n = 2
with open("Colors.txt", 'r') as open_file:
    for j, observation in enumerate(open_file):
        if j % n==0:
            print('Reading Line: ' + str(j) +
            ' Content: ' + observation)
```

Основная идея выборки такая же, как у потоковой передачи. Однако в этом случае приложение использует для получения номера строки метод `enumerate()`. Когда `j % n == 0`, строка является именно той, которую вы хотите сохранить, и приложение выводит информацию. В данном случае вы увидите следующий вывод:

```
Reading Line:  0 Content:  Color   Value

Reading Line:  2 Content:  Orange  2

Reading Line:  4 Content:  Green   4

Reading Line:  6 Content:  Purple  6

Reading Line:  8 Content:  White   8
```

Значение `n` важно при определении того, какие записи будут частью набора данных. Попробуйте изменить значение `n` на `3`. Выходные данные изменятся так, чтобы выбрать только заголовок, а также строки 3 и 6.

Вы также можете осуществить случайную выборку. И все, что вам нужно, — это сделать выбор случайным, например, так:

```
from random import random
sample_size = 0.25
with open("Colors.txt", 'r') as open_file:
    for j, observation in enumerate(open_file):
        if random()<=sample_size:
            print('Reading Line: ' + str(j) +
            ' Content: ' + observation)
```

Чтобы эта форма выборки работала, импортируйте класс `random`. Метод `random()` выводит значение в диапазоне от 0 до 1. Однако Python делает вывод случайным, чтобы вы не знали, какое значение получите. Переменная

`sample_size` содержит число от 0 до 1 для определения размера выборки. Например, значение `0.25` выбирает 25% элементов в файле.

Вывод будет по-прежнему отображаться в порядке номеров. Например, вы не увидите зеленый прежде оранжевого. Однако выбранные элементы будут случайными, и вы не всегда будете получать одинаковое количество возвращаемых значений. Промежутки между возвращаемыми значениями также будут различны. Ниже приведен пример того, что вы можете увидеть как вывод (хотя ваш вывод, вероятно, будет отличаться).

```
Reading Line:  1 Content:  Red    1

Reading Line:  4 Content:  Green  4

Reading Line:  8 Content:  White  8
```

Доступ к данным в форме структурированного плоского файла

Во многих случаях данные, с которыми вам нужно работать, не будут отображаться в библиотеке, например, наборы игрушечных данных в библиотеке Scikit-learn. Реальные данные обычно находятся в файле некоего типа. *Плоский файл* (flat file) представляет собой самый простой вид файла для работы. Данные отображаются в виде простого списка пунктов, которые вы можете читать в память по одному, если хотите. В зависимости от требований вашего проекта вы можете прочитать весь файл или его часть.

Проблема использования встроенных техник Python заключается в том, что ввод не является интеллектуальным. Например, когда файл содержит заголовок, Python просто читает его как очередные данные для обработки, а не как заголовок. Вы не можете просто выбрать определенный столбец данных. Библиотека `pandas` (панды), используемая в следующих разделах, значительно упрощает чтение и понимание данных плоских файлов. Классы и методы в библиотеке `pandas` интерпретируют (анализируют) данные плоского файла, чтобы ими было легче манипулировать.


ЗАПОМНИ!

Наименее отформатированный, а следовательно, самый простой для чтения формат плоского файла — это *текстовый файл* (text file). Но текстовый файл рассматривает все данные как строки, поэтому, как правило, приходится преобразовывать числовые данные в другие формы. *Файл со значениями, разделенными запятыми* (Comma-Separated Value — CSV), обеспечивает больше возможностей форматирования информации, но для его чтения требуется немного

больше усилий. Наиболее сложными являются *пользовательские форматы* (custom data) данных, такие как файлы Excel, содержащие расширенное форматирование и способные включать несколько наборов данных в один файл.

Эти три уровня наборов данных в плоских файлах рассматриваются в следующих разделах наряду с описанием их использования. В этих разделах предполагается, что файл каким-то образом структурирует данные. Например, файл CSV использует для разделения полей данных запятые. Текстовый файл может использовать для разделения полей данных табуляцию. В файле Excel для разделения полей данных и предоставления обширной информации о каждом поле используется комплексный метод. Вы можете работать и с неструктурированными данными, но работать со структурированными данными намного проще, поскольку вы знаете, где начинается и заканчивается каждое поле.

Чтение из текстового файла

Текстовые файлы могут использовать различные форматы хранения. Тем не менее общий формат должен иметь строку заголовка, которая документирует назначение каждого поля, а затем еще одну строку для каждой записи в файле. Файл разделяет поля с помощью знаков табуляции. Файл `Colors.txt`, используемый для примера в этом разделе, приведен на рис. 6.1.

Базовый язык Python предоставляет широкий спектр методов, которые можно использовать для чтения такого файла. Но в данном случае гораздо проще использовать для решения задачи библиотеку `pandas`. В библиотеке `pandas` содержится набор *синтаксических анализаторов* (parser), код которых используется для чтения отдельных фрагментов данных и определения назначения каждого фрагмента в соответствии с форматом всего файла. Использование правильного анализатора необходимо, если вы хотите разобраться в содержимом файла. В данном случае для решения задачи мы используем метод `read_table()`, как показано в следующем коде:

```
import pandas as pd
color_table = pd.io.parsers.read_table("Colors.txt")
print(color_table)
```

Код импортирует библиотеку `pandas`, использует метод `read_table()` для чтения файла `Colors.txt` в переменную `color_table`, а затем отображает полученные данные на экране с помощью функции `print`. Вот результат, который вы можете увидеть в этом примере:

```
    Color  Value
0     Red      1
1  Orange      2
```

```
2  Yellow      3
3   Green      4
4    Blue      5
5  Purple      6
6   Black      7
7   White      8
```

Обратите внимание, что анализатор правильно интерпретирует первую строку как состоящую из названий полей. Он нумерует записи от 0 до 7. Используя аргументы метода `read_table()`, вы можете настроить интерпретацию анализатором входного файла, но стандартные настройки обычно работают лучше всего. Вы можете прочитать больше об аргументах метода `read_table()` по адресу `http://pandas.pydata.org/pandasdocs/version/0.23.0/generated/pandas.read_table.html`.

Чтение формата CSV с разделителями

Файл CSV обеспечивает больше возможностей форматирования, чем простой текстовый файл. На самом деле файлы CSV могут быть довольно сложными. Существует стандарт, который определяет формат файлов CSV (его можно увидеть его по адресу `https://tools.ietf.org/html/rfc4180`). Файл CSV, используемый для этого примера, довольно прост:

- заголовок определяет каждое из полей;
- поля разделены запятыми;
- записи разделены символами новой строки;
- строки заключены в двойные кавычки;
- целые и действительные числа отображаются без двойных кавычек.

На рис. 6.3 файл `Titanic.csv`, используемый в этом примере, показан в необработанном формате. Содержимое файла в необработанном виде можно увидеть с помощью любого текстового редактора.

Такие приложения, как Excel, могут импортировать и форматировать файлы CSV, чтобы их было легче читать. На рис. 6.4 показан тот же файл в Excel.

Excel фактически распознает заголовок как заголовок. Если бы вы использовали такие функции, как сортировка данных, то могли бы выбрать столбцы заголовка, чтобы получить желаемый результат. К счастью, библиотека `pandas` также позволяет работать с файлом CSV в виде отформатированных данных, как показано в следующем примере:

```
import pandas as pd
titanic = pd.io.parsers.read_csv("Titanic.csv")
X = titanic[['age']]
print(X)
```

Рис. 6.3. В необработанном виде файл CSV все еще остается текстовым и вполне читабельным

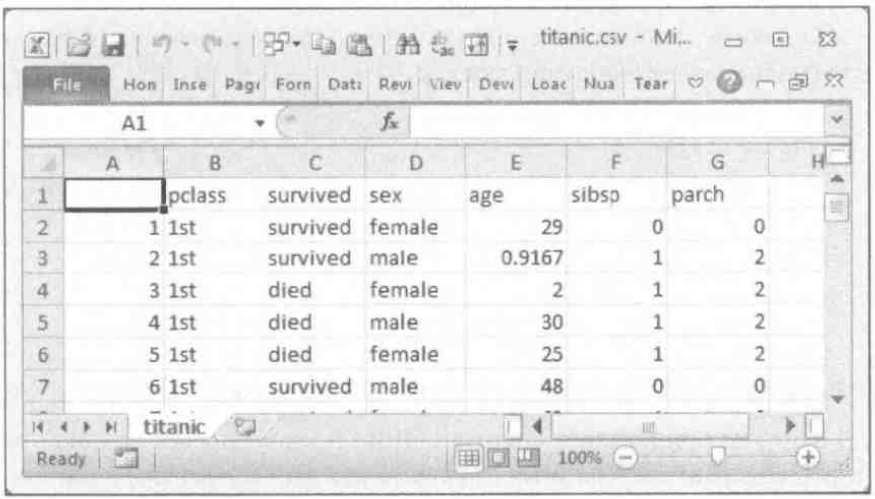


Рис. 6.4. Используйте приложения, например Excel, для создания отформатированной презентации CSV

Обратите внимание, что на этот раз предпочтительным анализатором является `read_csv()`, который понимает файлы CSV и предоставляет новые возможности для работы с ним. (Вы можете прочитать больше об этом анализаторе по адресу `http://pandas.pydata.org/pandasdocs/version/0.23.0/generated/pandas.read_csv.html`.) Выбор определенного поля довольно

прост — вы лишь указываете имя поля, как показано. Вывод этого примера выглядит следующим образом (некоторые значения опущены для экономии места):

```
age
0       29.0000
1        0.9167
2        2.0000
3       30.0000
4       25.0000
5       48.0000
...
1304    14.5000
1305  9999.0000
1306    26.5000
1307    27.0000
1308    29.0000
[1309 rows x 1 columns]
```

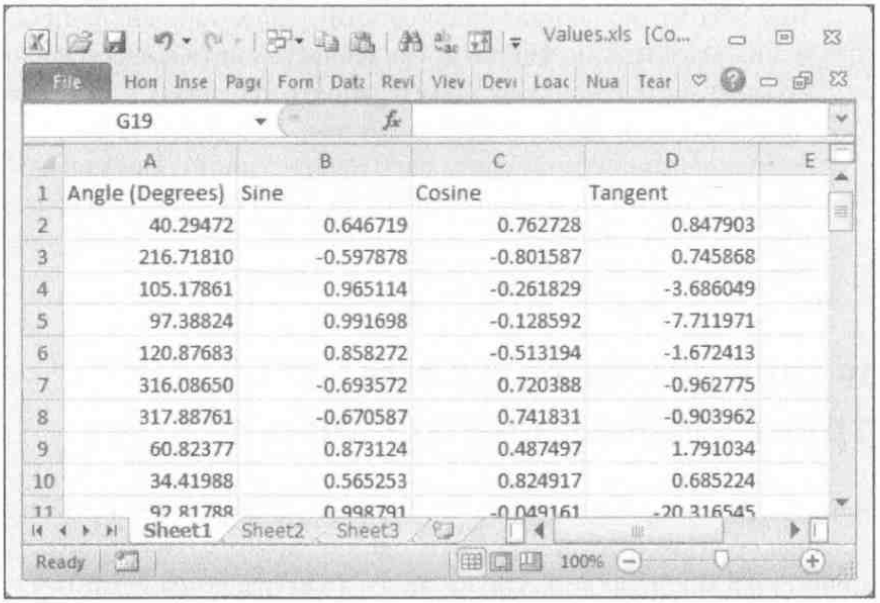
Конечно, удобочитаемый вывод, подобный этому, удобен при работе с примером, но вам также может понадобиться вывод в виде списка. Чтобы создать вывод в виде списка, просто измените третью строку кода на `X = titanic[['age']].values`. Обратите внимание на добавление свойства `values`. Выходные данные изменятся на что-то вроде следующего (некоторые значения опущены для экономии места):

```
[[ 29. ]
[ 0.91670001]
[ 2. ]
...
[ 26.5 ]
[ 27. ]
[ 29. ]]
```

Чтение файлов Excel и других файлов Microsoft Office

Excel и другие приложения Microsoft Office предоставляют сильно отформатированное содержимое. Вы можете указать каждый аспект информации, которую содержат эти файлы. Файл `Values.xls`, используемый для этого примера, предоставляет список значений синуса, косинуса и тангенса для случайного списка углов (рис. 6.5).

Работая с Excel или другими продуктами Microsoft Office, вы можете испытывать некоторые сложности. Например, файл Excel может содержать более одного рабочего листа (worksheet), поэтому вам нужно указать библиотеке `pandas`, какой рабочий лист обрабатывать. На самом деле вы можете при



	A	B	C	D	E
1	Angle (Degrees)	Sine	Cosine	Tangent	
2	40.29472	0.646719	0.762728	0.847903	
3	216.71810	-0.597878	-0.801587	0.745868	
4	105.17861	0.965114	-0.261829	-3.686049	
5	97.38824	0.991698	-0.128592	-7.711971	
6	120.87683	0.858272	-0.513194	-1.672413	
7	316.08650	-0.693572	0.720388	-0.962775	
8	317.88761	-0.670587	0.741831	-0.903962	
9	60.82377	0.873124	0.487497	1.791034	
10	34.41988	0.565253	0.824917	0.685224	
11	92.81788	0.998791	-0.049161	-20.316545	

Рис. 6.5. Файл Excel имеет сильное форматирование и может содержать информацию различных типов

желании обрабатывать несколько рабочих листов. При работе с другими продуктами Office необходимо четко указывать, что именно нужно обрабатывать. Просто указать библиотеке `pandas`, что она должна что-то обработать, недостаточно. Вот пример работы с файлом `Values.xls`:

```
import pandas as pd
xls = pd.ExcelFile("Values.xls")
trig_values = xls.parse('Sheet1', index_col=None,
                        na_values=['NA'])
print(trig_values)
```

Код начинается с импорта библиотеки `pandas` в обычном режиме. Затем он создает указатель на файл Excel с помощью конструктора `ExcelFile()`. Этот указатель, `xls`, позволяет получить доступ к рабочему листу, определить столбец индекса и указать, как интерпретировать пустые значения. *Столбец индекса* (index column) — это тот столбец, который используется для индексации записей. Использование значения `None` означает, что библиотека `pandas` должна сама создать индекс. Метод `parse()` получает запрошенные вами значения. (Больше о параметрах анализатора Excel см. по адресу `https://pandas.pydata.org/pandasdocs/stable/generated/pandas.ExcelFile.parse.html`.)

СОВЕТ

Вам абсолютно не нужно использовать двухэтапный процесс получения указателя на файл и последующего синтаксического анализа содержимого. Вы вполне можете решить эту задачу за один шаг, подобный следующему: `trig_values = pd.read_excel("Values.xls", 'Sheet1', index_col=None, na_values=['NA'])`. Поскольку файлы Excel являются довольно сложными, использование двухэтапного процесса зачастую удобнее и эффективнее, вам не нужно повторно открывать файл для каждого случая чтения данных.

Передача данных в форме неструктурированного файла

Файлы неструктурированных данных состоят из последовательности битов. Файл никоим образом не отделяет биты один от другого. Вы не можете просто заглянуть в файл и увидеть какую-либо структуру, потому что там ничего нет. Неструктурированные форматы файлов полагаются на пользователя файла, который должен знать, как интерпретировать данные. Например, каждый пиксель файла изображения может состоять из трех 32-битовых полей. Знание того, что каждое поле 32-битовое, лежит на вашей ответственности. Заголовок в начале файла может дать подсказки об интерпретации файла, но даже в этом случае вы должны знать, как взаимодействовать с файлом.

Пример в этом разделе демонстрирует работу с изображением в виде неструктурированного файла. Пример изображения — это изображение дня с сайта `http://commons.wikimedia.org/wiki/Main_Page`. Для работы с изображениями необходим доступ к *библиотеке Scikit-image* (`http://scikit-image.org/`), которая представляет собой бесплатную коллекцию алгоритмов, используемых для обработки изображений. Вы можете найти учебник по этой библиотеке по адресу `http://scipylectures.github.io/packages/scikit-image/`. Первая задача — показать изображение на экране, используя следующий код. (Для запуска этого кода может потребоваться некоторое время. Изображение готово, когда индикатор прогресса исчезнет с вкладки Notebook.)

```
from skimage.io import imread
from skimage.transform import resize
from matplotlib import pyplot as plt
import matplotlib.cm as cm

example_file = ("http://upload.wikimedia.org/" +
    "wikipedia/commons/7/7d/Dog_face.png")
image = imread(example_file, as_grey=True)
```

```
plt.imshow(image, cmap=cm.gray)
plt.show()
```

Код начинается с импорта нескольких библиотек. Затем он создает строку, которая указывает на файл примера в Интернете, и помещает его в файл `example_file`. Эта строка является частью вызова метода `imread()` вместе с аргументом `as_grey`, для которого установлено значение `True`. Аргумент `as_grey` указывает Python преобразовать любые цветные изображения в оттенки серого. Любые изображения, которые уже находятся в оттенках серого, остаются без изменения.

Теперь, когда у вас есть загруженное изображение, следует визуализировать его (подготовить к отображению на экране). Функция `imshow()` выполняет рендеринг и использует цветовую карту оттенков серого. Функция `show()` фактически отображает изображение, как показано на рис. 6.6.

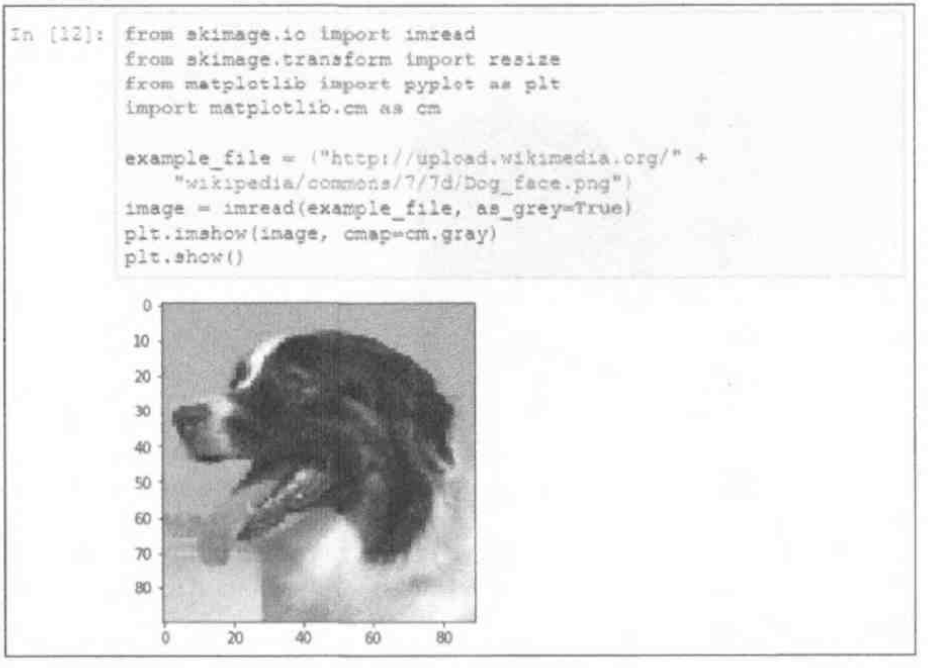


Рис. 6.6. Изображение появляется на экране после рендеринга и отображения

Теперь в памяти есть изображение, и вы можете узнать о нем больше. Когда вы запустите следующий код, вы получите тип и размер изображения:

```
print("data type: %s, shape: %s" %
    (type(image), image.shape))
```

Вывод этого кода сообщает, что типом изображения является `numpy.ndarray` и что размер изображения составляет 90 × 90 пикселей. Изображение на самом

деле представляет собой массив пикселей, которыми вы можете манипулировать различными способами. Например, если хотите обрезать изображение, используйте следующий код для управления массивом изображений:

```
image2 = image[5:70,0:70]
plt.imshow(image2, cmap=cm.gray)
plt.show()
```

Значение `numpy.ndarray` в `image2` меньше, чем в `image`, поэтому вывод также меньше. На рис. 6.7 представлены типичные результаты. Цель обрезки изображения — придать ему определенный размер. Оба изображения должны быть одного размера, чтобы вы могли их проанализировать. Обрезка является одним из способов обеспечения правильного размера изображений для анализа.

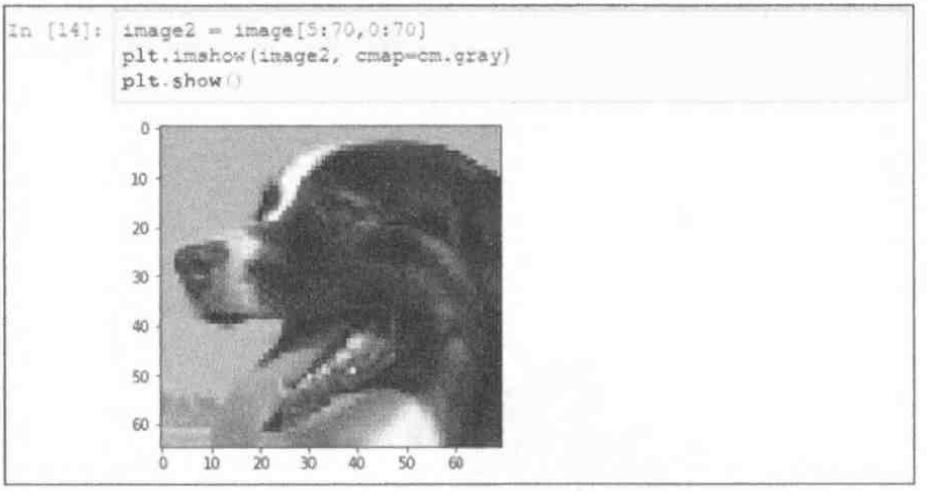


Рис. 6.7. Обрезка изображения делает его меньше

Другой метод, который можно использовать для изменения размера изображения, — это изменение его масштаба. Следующий код изменяет изображение до определенного размера:

```
image3 = resize(image2, (30, 30), mode='symmetric')
plt.imshow(image3, cmap=cm.gray)
print("data type: %s, shape: %s" %
    (type(image3), image3.shape))
```

Вывод функции `print()` сообщает, что размер изображения теперь составляет 30 × 30 пикселей. Вы можете сравнить его с любым изображением такого же размера.

После того как вы получите все изображения нужного размера, вам нужно сделать их плоскими (flatten). Строка набора данных всегда представляет

собой одно измерение, а не два. В настоящее время изображение представляет собой массив размером 30 × 30 пикселей, поэтому вы не можете сделать его частью набора данных. Следующий код преобразует `image3` так, что оно становится массивом из 900 элементов, который хранится в `image_row`:

```
image_row = image3.flatten()
print("data type: %s, shape: %s" %
    (type(image_row), image_row.shape))
```

Обратите внимание, что типом по-прежнему является `numpy.ndarray`. Вы можете добавить этот массив в набор данных, а затем использовать набор данных для анализа. Размер составляет 900 элементов, как и ожидалось.

Работа с данными из реляционных баз данных

Базы данных бывают разных форматов. Например, база AskSam (`http://asksam.en.softonic.com/`) является разновидностью текстовой базы данных в свободной форме. Однако подавляющее большинство данных, используемых организациями, полагаются на реляционные базы данных, поскольку они предоставляют средства для организации огромных объемов сложных данных организованным образом, что облегчает манипулирование данными. Задача менеджера баз данных — сделать данные простыми в использовании. Основное внимание в большинстве хранилищ данных уделяется облегчению поиска в них.

Реляционные базы данных позволяют манипулировать данными и извлекать их с относительной легкостью. Тем не менее, для широкого спектра вычислительных платформ существует множество различных продуктов реляционных баз данных. Фактически для аналитиков данных распространение различных *систем управления базами данных* (Database Management Systems — DBMS), использующих различные схемы размещения данных, является одной из основных проблем, с которой они сталкиваются при создании всеобъемлющего набора данных для анализа.

Общим для большинства реляционных баз данных является то, что все они полагаются на формат одного и того же языка манипулирования данными, что облегчает работу аналитика. *Язык структурированных запросов* (Structured Query Language — SQL) позволяет выполнять все виды задач управления реляционной базой данных, извлекать данные по мере необходимости и даже

форматировать их особым образом, так что необходимости в дополнительном формировании нет.

Создание соединения с базой данных может быть сложным делом. Прежде всего вам нужно знать, как подключиться к данной конкретной базе данных. Первый шаг — получить доступ к базе данных. Для этого используйте две строки кода, аналогичные следующему коду (однако представленный здесь код не предназначен для запуска и выполнения задачи):

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///:memory:')
```

Получив доступ к *ядру* (engine), вы можете использовать его для выполнения задач, специфичных для этой DBMS. Результатом вызова метода чтения всегда является объект `DataFrame`, который содержит запрошенные данные. Для записи данных необходимо создать объект `DataFrame` или использовать уже существующий. Обычно для выполнения большинства задач используют следующие методы.

- **`read_sql_table()`**. Читает данные из таблицы SQL в объект `DataFrame`.
- **`read_sql_query()`**. Читает данные из базы с помощью запроса SQL в объект `DataFrame`.
- **`read_sql()`**. Читает данные из таблицы SQL или запроса в объект `DataFrame`.
- **`DataFrame.to_sql()`**. Записывает содержимое объекта `DataFrame` в указанные таблицы базы данных.

Библиотека `sqlalchemy` поддерживает широкий спектр баз данных SQL. Ниже приведены только некоторые из них.

- SQLite.
- MySQL.
- PostgreSQL.
- SQL Server.
- Другие реляционные базы данных, к которым вы можете подключиться с помощью *открытого интерфейса доступа к базам данных* (Open Database Connectivity — ODBC).

Больше о работе с базами данных см. по адресу `https://docs.sqlalchemy.org/en/latest/core/engines.html`. Методы, которые вы изучите в этой книге при работе с игрушечными базами данных, также работают с реляционными базами данных.

Взаимодействие с данными из баз NoSQL

В дополнение к стандартным реляционным базам данных, основанным на языке SQL, вы найдете множество баз данных всех видов, которым не нужно полагаться на SQL (NoSQL). Эти базы данных используются для хранения больших данных, в которых реляционная модель может стать слишком сложной или ненадежной. Такие базы данных обычно не используют реляционную модель. Конечно, вы найдете меньше таких DBMS, используемых в корпоративной среде, поскольку они требуют специальной обработки и обучения. Тем не менее некоторые распространенные DBMS используются, поскольку они предоставляют специальные функции или отвечают уникальным требованиям. Процесс использования баз данных NoSQL по существу такой же для, как и для реляционных баз данных.

1. Импорт необходимых функций ядра базы данных.
2. Создание ядра базы данных.
3. Выполнение любых необходимых запросов, использование ядра базы данных и функций, поддерживаемых DBMS.

Детали различаются довольно сильно, и вам нужно знать, какую библиотеку использовать с вашим конкретным продуктом базы данных. Например, при работе с MongoDB (`https://www.mongodb.org/`) вы должны получить копию библиотеки PyMongo (`https://api.mongodb.org/python/current/`) и использовать класс `MongoClient`, чтобы создать необходимое ядро. При поиске данных ядро MongoDB в значительной степени опирается на функцию `find()`. Ниже приведен пример псевдокода сеанса MongoDB. (Вы не сможете выполнить этот код в Notebook; он показан только в качестве примера.)

```
import pymongo
import pandas as pd
from pymongo import Connection
connection = Connection()
db = connection.database_name
input_data = db.collection_name
data = pd.DataFrame(list(input_data.find()))
```

Доступ к данным из Интернета

Сегодня было бы невероятно трудно (скорее невозможно) найти организацию, которая не опиралась бы на какие-либо веб-данные. Большинство организаций используют веб-службы определенного типа. *Веб-службы* (web

service) — это своего рода веб-приложение, позволяющее задавать вопросы и получать ответы. Веб-службы обычно поддерживают несколько типов ввода. Фактически конкретный веб-служба может содержать целые группы входных запросов.

Другой тип системы запросов — это *микрослужба* (microservice). В отличие от веб-службы, микрослужбы имеют особую направленность и обеспечивают только один конкретный вид ввода и вывода. Использование микрослужб имеет определенные преимущества, описание которых выходят за рамки этой книги, но по сути они работают как крошечные веб-службы и упоминаются здесь.

API И ДРУГИЕ ВЕБ-СУЩНОСТИ

У аналитика данных может быть причина полагаться на различные *интерфейсы прикладных программ* (Application Programming Interface — API) веб-приложений для доступа к данным и манипулирования ими. Фактически фокусом анализа могут быть сами API. В этой книге интерфейсы API подробно не обсуждаются, поскольку каждый API уникален. Например, при работе с веб-приложением вы можете использовать такой продукт, как jQuery (`http://jquery.com/`), для доступа к данным и манипулирования ими различными способами. Тем не менее методы для этого больше похожи на написание приложения, чем на использование техники науки о данных.

Важно понимать, что API могут быть источниками данных, и вам может понадобиться использовать их для некоторых целей ввода или формирования данных. На самом деле вы найдете много сущностей данных, которые похожи на API, но не обсуждаются в этой книге. Разработчики на Windows могут создавать приложения *модели компонентных объектов* (Component Object Model — COM), которые выводят данные в сеть и которые вы могли бы использовать для анализа. На самом деле количество потенциальных источников практически бесконечно. Эта книга посвящена источникам, которые вы будете использовать чаще всего и наиболее общепринятым образом. Однако следует знать и о других возможностях — это всегда хорошая идея.

Одним из наиболее полезных методов доступа к данным, о котором нужно знать при работе с веб-данными, является доступ к XML. Все виды типов содержимого основаны на языке XML, даже некоторые веб-страницы. Работа с веб-службами и микрослужбами означает работу с языком XML (в большинстве случаев). С учетом этого, пример в текущем разделе работает с данными в формате XML, находящимися в файле `XMLData.xml` (рис. 6.8). В данном случае файл прост и использует только пару уровней. Код XML является иерархическим и может стать достаточно глубоким.

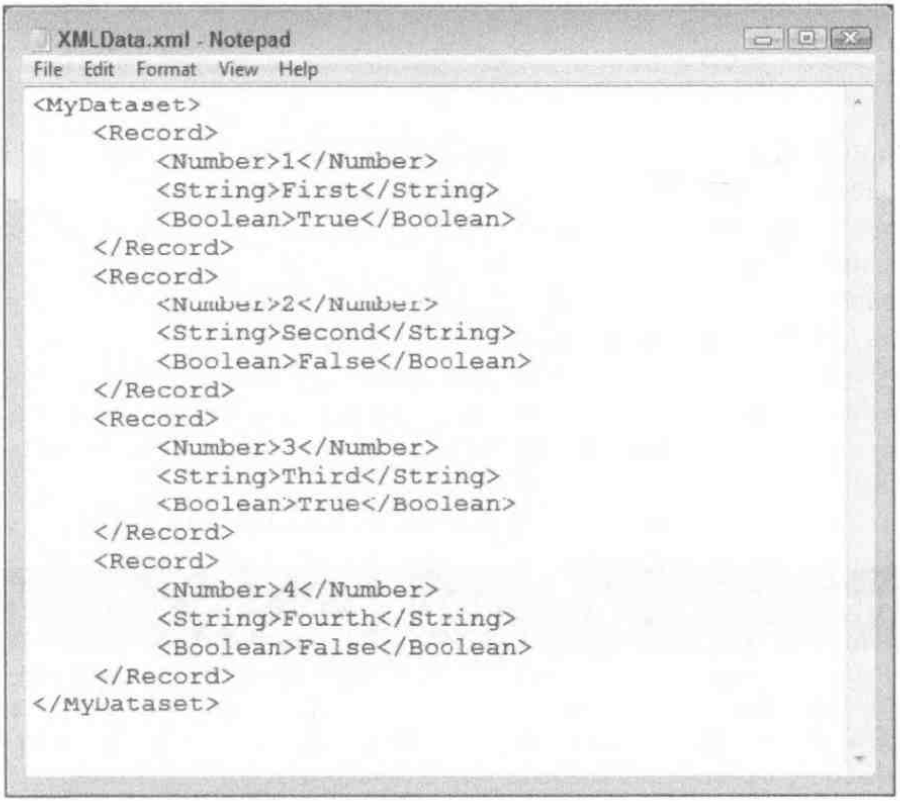


Рис. 6.8. XML — это иерархический формат, который может стать довольно сложным

Техника для работы с кодом XML, даже простым, может быть немного сложнее, чем все, с чем вы работали до сих пор. Ниже приведен код этого примера.

```
from lxml import objectify
import pandas as pd

xml = objectify.parse(open('XMLData.xml'))
root = xml.getroot()

df = pd.DataFrame(columns=('Number', 'String', 'Boolean'))

for i in range(0,4):
    obj = root.getchildren()[i].getchildren()
    row = dict(zip(['Number', 'String', 'Boolean'],
                   [obj[0].text, obj[1].text,
                    obj[2].text]))
    row_s = pd.Series(row)
```

```
    row_s.name = i
    df = df.append(row_s)

print(df)
```

Пример начинается с импорта библиотек и анализа файла данных с использованием метода `objectify.parse()`. Каждый документ XML должен содержать корневой узел, в данном случае — `<MyDataset>`. Корневой узел инкапсулирует остальную часть содержимого, и каждый узел под ним является дочерним. Чтобы сделать что-либо практическое с документом, вы должны получить доступ к корневому узлу с помощью метода `getroot()`.

Следующим шагом является создание пустого объекта `DataFrame`, который содержит правильные имена столбцов для каждой записи: `Number`, `String` и `Boolean`. Как и в случае любой другой обработки данных библиотекой `pandas`, обработка данных XML основана на объекте `DataFrame`. Цикл `for` заполняет объект `DataFrame` четырьмя записями из файла XML (каждая в узле `<Record>`).

ИСПОЛЬЗОВАНИЕ АЛЬТЕРНАТИВЫ JSON

Вы должны понимать, что не все данные, с которыми вы работаете в Интернете, представлены в формате XML. Возможно, вам придется рассмотреть другие популярные альтернативы как часть ваших планов развития. Одна из самых популярных — это JavaScript Object Notation (JSON) (`http://www.json.org/`). Ее сторонники утверждают, что JSON занимает меньше места, быстрее в использовании и с ней проще работать, чем с форматом XML (см. детали на `https://www.w3schools.com/js/js_json_xml.asp` и `https://blog.cloudelements.com/json-better-xml`). Следовательно, вы можете обнаружить, что при работе с определенными веб-службами и микрослужбами ваш следующий проект опирается на вывод JSON, а не XML.

Если ваш выбор форматирования данных состоял только из XML и JSON, вам может показаться, что взаимодействие с данными вполне управляемо. Тем не менее, существует немало других идей о форматировании данных так, чтобы вы могли легко и быстро анализировать их. Кроме того, разработчики теперь уделяют больше внимания понятию потока данных, поэтому некоторые методы форматирования обеспечивают и удобочитаемость для человека. Вы можете прочитать о некоторых из этих альтернатив на `https://insights.dice.com/2018/01/05/5-xml-alternatives-to-consider-in-2018/`. Одна из наиболее важных из этих альтернатив — это *еще один язык разметки* (Yet Another Markup Language — YAML) или *не язык разметки* (YAML Ain't Markup Language), в зависимости от того, с кем вы общаетесь и какими ресурсами пользуетесь (`http://yaml.org/spec/1.2/spec.html`).

Процесс выглядит сложным, но следует логическому порядку. Переменная `obj` содержит все дочерние элементы узла `<Record>`. Эти дочерние элементы загружаются в объект словаря, в котором используются ключи `Number`, `String` и `Boolean` для соответствия столбцам `DataFrame`.

Теперь есть объект словаря, который содержит данные строки. Далее код создает фактическую строку для объекта `DataFrame`. Это придает строке значение текущей итерации цикла `for`. Затем он добавляет строку в объект `DataFrame`. Чтобы увидеть, что все работает как ожидалось, код выводит результат, который выглядит следующим образом:

```
   Number  String  Boolean
0       1   First     True
1       2  Second    False
2       3   Third     True
3       4  Fourth    False
```

Глава 7

Подготовка данных

В ЭТОЙ ГЛАВЕ...

- Работа с библиотеками NumPy и pandas
- Работа с символическими переменными
- Учет влияния дат
- Исправление пропущенных данных
- Разделение, объединение и изменение элементов данных

Такие характеристики, как содержимое, тип и другие элементы, которые полностью определяют данные, являются *формой* (shape) данных. Форма данных определяет виды задач, которые вы можете решать с их помощью. Чтобы сделать данные доступными для определенных видов анализа, вы должны преобразовать их в другую форму. Считайте данные глиной, а себя гончаром. Но вместо того, чтобы использовать для формования данных свои руки, вы полагаетесь на функции и алгоритмы. В этой главе описаны инструменты, имеющиеся для придания данным формы.

Кроме того, в этой главе мы рассмотрим проблемы, связанные с формированием. Например, вам нужно знать, что делать, если в наборе данных отсутствуют данные. Важно правильно сформировать данные, иначе вы получите анализ, который просто не имеет смысла. Аналогично некоторые типы данных, такие как даты, могут представлять проблемы. Опять же, вам нужно действовать осторожно, чтобы получить желаемый результат, — набор данных должен стать более полезным и пригодным для анализов различных видов.


ЗАПОМНИ!

Целью некоторых типов формирования данных является создание бо́льшего набора данных. Во многих случаях необходимые для анализа данные не отражаются ни в одной базе данных, ни в какой

определенной форме. Вам нужно сформировать данные, а затем объединить их так, чтобы получить единый набор данных в известном формате, прежде чем вы сможете начать его анализ. Успешное объединение данных может быть искусством, поскольку данные зачастую не поддаются простому анализу или быстрым исправлениям.


СОВЕТ

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле исходного кода `P4DS4D2_07_Getting_Your_Data_in_Shape.ipynb`.


ВНИМАНИЕ!

Убедитесь, что файл `XMLData2.xml`, который поставляется с загружаемым исходным кодом, находится в той же папке (каталоге), что и файлы Notebook. В противном случае примеры в следующих разделах завершатся ошибкой ввода/вывода. Расположение файла зависит от используемой вами платформы. Например, в системе Windows они хранятся в папке `C:\Users\`*`ИмяПользователя`*`\P4DS4D2`, где *`ИмяПользователя`* — это ваше имя пользователя. (В книге предполагается, что вы выбрали местоположение папки `P4DS4D2` так, как описано в разделе "Определение хранилища кода" главы 3.) Чтобы примеры работали, скопируйте файл из папки с загружаемыми исходными файлами в свою папку для Notebook. Инструкцию по загрузке исходного кода см. во введении.

Баланс между NumPy и pandas

Нет сомнений, что библиотека `NumPy` вам нужна всегда. Библиотека `pandas` на самом деле построена поверх библиотеки `NumPy`. Тем не менее при решении задач, вам нужно сделать выбор между `NumPy` и `pandas`. Для некоторых задач вам потребуется низкоуровневая функциональность `NumPy`, но `pandas` упрощает работу, поэтому вы будете использовать ее чаще. В следующих разделах более подробно описано, когда использовать каждую библиотеку.

Когда использовать NumPy

Разработчики создали библиотеку `pandas` поверх `NumPy`. В результате каждая задача, которую вы решаете с помощью библиотеки `pandas`, также проходит через библиотеку `NumPy`. За преимущества `pandas` приходится платить производительностью, которая, по словам некоторых специалистов, в 100 раз медленнее, чем у `NumPy`, при решении аналогичных задач (см. `http://penandpants.`

com/2014/09/05/performance-of-pandas-seriesvs-numpy-arrays/). Учитывая, что современное компьютерное оборудование способно компенсировать значительные различия в производительности, проблема скорости может не вызывать беспокойства, но когда скорость важна, NumPy всегда является лучшим выбором.

Когда использовать pandas

Библиотека pandas используется для ускорения и упрощения написания кода. Поскольку pandas выполняет бо́льшую часть работы за вас, вы можете сказать, что ее использование снижает также вероятность ошибок программирования. Однако необходимо учитывать, что библиотека pandas предоставляет богатые функциональные возможности для временны́х рядов, выравнивания данных, статистики NA, методов группового объединения, слияния и объединения. Обычно при использовании NumPy необходимо программировать эти функции, а значит, вы продолжаете изобретать велосипед.

По мере изучения книги вы обнаружите, насколько полезной может быть библиотека pandas при решении таких задач, как *группировка* (binning) (метод предварительной обработки данных, призванный уменьшить влияние ошибок наблюдения) и работа с *фреймом данны́х* (dataframe) (двумерная структура данных с метками, столбцы которых потенциально могут содержать разные типы данных), так что вы можете рассчитать статистику по нему. Например, в главе 9 вы узнаете, как выполнять дискретизацию и группировку. В главе 13 приведены реальные примеры группировки, такие как получение частоты для каждой категориальной переменной набора данных. Фактически многие примеры главы 13 не работают без группировки. Другими словами, не стоит сейчас сильно беспокоиться о том, чтобы точно знать, что такое группировка и почему ее нужно использовать, — примеры приведены в книге позже, при подробном обсуждении этой темы. Все, что вам действительно нужно знать, — это то, что библиотека pandas значительно облегчает работу.

ВСЕ ДЕЛО В ПОДГОТОВКЕ

Может показаться, что в этой книге уделяется много времени подготовке данных и мало времени их анализу. Однако бо́льшая часть времени аналитика на самом деле тратится на подготовку данных, поскольку данные редко бывают в каком-либо порядке до выполнения анализа. Чтобы подготовить данные для использования, аналитик должен:

- получить данные;
- агрегировать данные;

- создать подмножества данных;
- очистить данные;
- разработать единый набор данных, объединив различные наборы данных.

К счастью, использование языка Python и различных библиотек, которые он предоставляет, делает эту задачу намного проще, быстрее и эффективнее, что придает смысл всей этой трате времени на кажущиеся очевидными темы первых глав. Чем лучше вы знаете, как использовать Python для ускорения решения этих рутинных задач, тем раньше вы начнете получать удовольствие от выполнения различных видов анализа данных.

Проверка данных

Когда дело доходит до данных, никто действительно не знает, что содержит большая база данных. Да, все видели ее по частям, но если учесть размер некоторых баз данных, ее просмотр целиком был бы физически невозможен. Поскольку вы не знаете, что там находится, вы не можете быть уверены, что ваш анализ действительно будет работать так, как нужно, и даст корректные результаты. Короче говоря, вы должны проверить свои данные, прежде чем использовать их, чтобы убедиться в том, что данные, по крайней мере, близки к ожидаемым. Это означает решение таких задач, как удаление дубликатов записей перед использованием данных для любого вида анализа.


ЗАПОМНИ!

Вам нужно подумать о том, что на самом деле дает проверка. Она не свидетельствует, что данные верны или что не будет значений вне ожидаемого диапазона. Последующие главы помогут вам понять методы решения подобных проблем. Что делает проверка, так это гарантирует, что вы можете выполнять анализ данных и резонно ожидать, что он будет успешным. Позже вам нужно будет выполнить дополнительную обработку данных, чтобы получить результаты, необходимые для решения задачи, которую вы намеревались выполнить в первую очередь.

Выяснение содержимого данных

Важно выяснить, что содержат данные, поскольку проверка данных вручную иногда просто невозможна из-за количества наблюдений и переменных. Кроме того, ручная проверка содержимого занимает много времени, подвержена ошибкам и, что наиболее важно, действительно скучна. Поиск дубликатов важен, поскольку вы в конечном итоге

- тратите больше вычислительного времени на обработку дубликатов, что замедляет ваши алгоритмы;
- получите ложные результаты, поскольку дубликаты неявно перевешивают результаты. Поскольку некоторые записи встречаются более одного раза, алгоритм считает их более важными.

Как аналитик данных вы хотите, чтобы данные приводили вас в восторг, поэтому пришло время, чтобы они поговорили с вами — не в прямом смысле, конечно, а через чудеса библиотеки `pandas`, как показано в следующем примере:

```
from lxml import objectify
import pandas as pd

xml = objectify.parse(open('XMLData2.xml'))
root = xml.getroot()
df = pd.DataFrame(columns=('Number', 'String', 'Boolean'))

for i in range(0,4):
    obj = root.getchildren()[i].getchildren()
    row = dict(zip(['Number', 'String', 'Boolean'],
                   [obj[0].text, obj[1].text,
                    obj[2].text]))
    row_s = pd.Series(row)
    row_s.name = i
    df = df.append(row_s)

search = pd.DataFrame.duplicated(df)
print(df)
print()
print(search[search == True])
```

В этом примере показано, как найти повторяющиеся строки. Он основан на модифицированной версии файла `XMLData.xml`, файле `XMLData2.xml`, который содержит простую повторяющуюся строку. Файл реальных данных содержит тысячи (или более) записей и, возможно, сотни повторов, но этот простой пример делает свою работу. Пример начинается с чтения файла данных в память, используя тот же метод, который вы рассмотрели в главе 6. Затем он помещает данные в объект `DataFrame`.

На этом этапе ваши данные некорректны, поскольку содержат повторяющуюся строку. Тем не менее вы можете избавиться от дубликатов строки, выполнив поиск. Первая задача — создать объект поиска, содержащий список дублированных строк, с помощью вызова метода `pd.DataFrame.duplicated()`. Дублированные строки содержат значение `True` рядом с номером строки.

Конечно, теперь у вас есть неупорядоченный список строк, которые не дублируются. Самый простой способ определить, какие строки дублируют-

ся, — это создать индекс, где в качестве выражения используется `search == True`. Ниже приведен вывод этого примера. Обратите внимание, что строка 3 дублируется в выводе данных `DataFrame` и также используется в результирующем вызове `search`:

```
  Number  String  Boolean
0      1   First     True
1      2  Second    False
2      3   Third     True
3      3   Third     True

3    True
dtype:  bool
```

Удаление дубликатов

Чтобы получить чистый набор данных, из него нужно удалить дубликаты. К счастью, вам не нужно писать какой-то странный код, чтобы выполнить эту работу, — `pandas` сделает это за вас, как показано в следующем примере:

```
from lxml import objectify
import pandas as pd

xml = objectify.parse(open('XMLData2.xml'))
root = xml.getroot()
df = pd.DataFrame(columns=('Number', 'String', 'Boolean'))
for i in range(0,4):
    obj = root.getchildren()[i].getchildren()
    row = dict(zip(['Number', 'String', 'Boolean'],
                   [obj[0].text, obj[1].text,
                    obj[2].text]))
    row_s = pd.Series(row)
    row_s.name = i
    df = df.append(row_s)

print(df.drop_duplicates())
```

Как и в предыдущем примере, вы начинаете с создания объекта `DataFrame`, который содержит дубликат записи. Чтобы удалить ошибочную запись, все, что вам нужно сделать, — это вызвать метод `drop_duplicates()`. Вот результат, который вы получите:

```
   Number  String  Boolean
0       1   First     True
1       2  Second    False
2       3   Third     True
```

Создание карты и плана данных

Вам нужно знать о вашем наборе данных, т.е. как он выглядит статически. *Карта данных* (data map) — это обзор набора данных. Вы используете его для выявления потенциальных проблем в ваших данных, таких как

- избыточные переменные;
- возможные ошибки;
- недостающие значения;
- преобразования переменных.

Проверка на наличие этих проблем входит в *план данных* (data plan), который представляет собой список задач, которые вы должны выполнить для обеспечения целостности ваших данных. В следующем примере показана карта данных A с двумя наборами данных B и C:

```
import pandas as pd
pd.set_option('display.width', 55)

df = pd.DataFrame({'A': [0,0,0,0,0,1,1],
                   'B': [1,2,3,5,4,2,5],
                   'C': [5,3,4,1,1,2,3]})

a_group_desc = df.groupby('A').describe()
print(a_group_desc)
```

В данном случае карта данных использует нули для первой серии и единицы — для второй. Функция `groupby()` помещает наборы данных B и C в группы. Чтобы определить, является ли карта данных корректной, вы получаете статистику с помощью функции `description()`. В результате вы получите набор данных B, серии 0 и 1, и набор данных C, серии 0 и 1, как показано в следующем выводе:

```
      B                                                \
  count mean       std  min   25%  50%   75%  max
A
0    5.0 3.0  1.581139  1.0  2.00  3.0  4.00  5.0
1    2.0 3.5  2.121320  2.0  2.75  3.5  4.25  5.0

      C
  count mean       std  min   25%  50%   75%  max
A
0    5.0 2.8  1.788854  1.0  1.00  3.0  4.00  5.0
1    2.0 2.5  0.707107  2.0  2.25  2.5  2.75  3.0
```

Эта статистика расскажет вам о двух сериях наборов данных. Разделение двух наборов данных с использованием конкретных случаев — это и есть *план*

данных (data plan). Как видите, статистика говорит вам, что этот план данных может быть не корректен, поскольку некоторые статистические данные отстоят относительно далеко друг от друга.

СОВЕТ

Стандартный вывод метода `description()` показывает, что данные разбросаны. К сожалению, разбросанные данные могут выводиться с неудачным разрывом, что затрудняет их чтение. Чтобы этого не происходило, установите ширину, которую хотите использовать для данных, вызвав метод `pd.set_option('display.width', 55)`. Таким образом, вы можете установить ряд параметров `pandas`, используя информацию, доступную по адресу `https://pandas.pydata.org/pandasdocs/stable/generated/pandas.set_option.html`.

Хотя разбросанные (unstacked) данные относительно легко читать и сравнивать, вы можете предпочесть более компактное представление. В этом случае вы можете собрать данные, используя следующий код:

```
stacked = a_group_desc.stack()
print(stacked)
```

Использование функции `stack()` создает новую презентацию. Ниже приведен вывод, показанный в компактной форме.

```
                B         C
A
0 count  5.000000  5.000000
  mean   3.000000  2.800000
  std    1.581139  1.788854
  min    1.000000  1.000000
  25%    2.000000  1.000000
  50%    3.000000  3.000000
  75%    4.000000  4.000000
  max    5.000000  5.000000
1 count  2.000000  2.000000
  mean   3.500000  2.500000
  std    2.121320  0.707107
  min    2.000000  2.000000
  25%    2.750000  2.250000
  50%    3.500000  2.500000
  75%    4.250000  2.750000
  max    5.000000  3.000000
```

Конечно, вам могут не понадобиться все данные, предоставляемые `describe()`. Возможно, вы действительно просто хотите увидеть количество элементов в каждой серии и их среднее значение. Вот как следует уменьшить размер вывода информации:

```
print(a_group_desc.loc[:,(slice(None),['count','mean']),])
```

Использование `loc` позволяет получить определенные столбцы. Окончательный результат примера, показывающий только ту информацию, которая вам абсолютно необходима для принятия решения, приведен ниже.

```
      B          C
  count mean count mean
A
0     5.0 3.0     5.0 2.8
1     2.0 3.5     2.0 2.5
```

Манипулирование категориальными переменными

Категориальная переменная (categorical variable) в науке о данных — это переменная, имеющая конкретное значение из ограниченного набора значений. Количество значений обычно фиксировано. Многие разработчики будут знать категориальные переменные по *перечислениям* (enumeration) имен. Каждое из потенциальных значений, которые может принимать категориальная переменная, является *уровнем* (level).

Чтобы понять, как работают категориальные переменные, предположим, что у вас есть переменная, выражающая цвет объекта, такого как автомобиль, и что пользователь может выбрать синий, красный или зеленый. Чтобы выразить цвет автомобиля способом, который компьютеры могут представить и эффективно вычислять, приложение присваивает каждому цвету числовое значение, поэтому синий цвет равен 1, красный — 2, зеленый — 3. Обычно при выводе каждого цвета вы видите значение, а не цвет.

Если вы используете объект `pandas.DataFrame` (`http://pandas.pydata.org/pandasdocs/dev/generated/pandas.DataFrame.html`), вы все равно можете увидеть символическое значение (синий, красный и зеленый), даже если компьютер сохраняет его как числовое значение. Иногда вам нужно переименовать и объединить эти именованные значения для создания новых символов. *Символьные переменные* (symbolic variable) — это просто удобный способ представления и хранения качественных данных.

При использовании категориальных переменных для машинного обучения важно учитывать алгоритм, используемый для манипулирования переменными. Некоторые алгоритмы, такие как деревья и ансамбли деревьев, могут работать непосредственно с числовыми переменными, а не символами. Другие алгоритмы, такие как линейная и логистическая регрессия и SVM, требуют

кодирования категориальных значений в двоичные переменные. Например, если у вас есть три уровня для цветовой переменной (синий, красный и зеленый), вы должны создать три двоичные переменные:

- одна — для синего (1, когда значение синего, 0 — если нет);
- одна — для красного (1, когда значение красного, 0 — если нет);
- одна — для зеленого (1, когда значение зеленого, 0 — если нет).

ПРОВЕРКА ВАШЕЙ ВЕРСИИ PANDAS

Примеры категориальных переменных в этом разделе зависят от установленной в вашей системе версии библиотеки `pandas`. Минимально допустимая версия — 0.23.0. Однако в вашей версии Anaconda может быть установлена предыдущая версия `pandas`. Используйте следующий код для проверки версии `pandas`:

```
import pandas as pd
print(pd.__version__)
```

Вы увидите номер версии установленной библиотеки `pandas`. Еще один способ проверки версии — открыть Anaconda Prompt, ввести **`pip show pandas`** и нажать клавишу <Enter>. Если у вас более старая версия, откройте приглашение Anaconda, введите **`pip install pandas --upgrade`** и нажмите клавишу <Enter>. Процесс обновления будет происходить автоматически вместе с проверкой связанных пакетов. При работе с Windows вам может потребоваться открыть Anaconda Prompt от имени администратора (щелкните правой кнопкой мыши на пункте Anaconda Prompt в меню Run (Пуск) и выберите из контекстного меню пункт Run as Administrator (Запуск от имени администратора)).

Создание категориальных переменных

Категориальные переменные имеют определенное количество значений, что делает их невероятно ценными при решении ряда задач по науке о данных. Представьте, например, что вы пытаетесь выявить значения, находящиеся за пределами диапазона в огромном наборе данных. В этом примере вы видите один метод создания категориальной переменной, а затем ее использования для проверки того, попадают ли некие данные в указанные пределы:

```
import pandas as pd

car_colors = pd.Series(['Blue', 'Red', 'Green'],
                       dtype='category')

car_data = pd.Series(
```

```
    pd.Categorical(
        ['Yellow', 'Green', 'Red', 'Blue', 'Purple'],
            categories=car_colors, ordered=False))

find_entries = pd.isnull(car_data)
print(car_colors)
print()
print(car_data)
print()
print(find_entries[find_entries == True])
```

Пример начинается с создания категориальной переменной `car_colors`. Переменная содержит значения `Blue`, `Red` и `Green` в качестве цветов, приемлемых для автомобиля. Обратите внимание, что вы должны указать для свойства `dtype` значение `category`.

Следующий шаг — создание еще одной серии. В качестве входных данных используется список реальных цветов автомобилей по имени `car_data`. Не все цвета автомобиля соответствуют предопределенным допустимым значениям. Когда возникает эта проблема, библиотека `pandas` выводит вместо цвета автомобиля значение `NaN` (Not a Number — не число).

Конечно, вы можете искать в списке некорректные автомобили вручную, но самый простой способ — поручить эту работу библиотеке `pandas`. В этом случае с помощью функции `isnull()` вы запрашиваете `pandas`, какие записи являются нулевыми, и помещаете их в `find_entries`. Затем вы можете вывести только те записи, которые на самом деле являются нулевыми. Вот результат примера, который вы видите:

```
0     Blue
1      Red
2    Green
dtype: category
Categories (3, object): [Blue, Green, Red]

0      NaN
1    Green
2      Red
3     Blue
4      NaN
dtype: category
Categories (3, object): [Blue, Green, Red]

0    True
4    True
dtype: bool
```

В списке выходных данных `car_data` вы видите, что записи 0 и 4 равны `NaN`. Вывод `find_entries` подтверждает этот факт. Если бы это был большой набор

данных, вы могли бы быстро найти и исправить ошибочные записи в наборе данных перед выполнением анализа.

Переименование уровней

Бывают случаи, когда наименования категорий, которые вы используете, неудобны или неправильны для конкретной цели. К счастью, вы можете переименовывать категории по мере необходимости, используя технику, показанную в следующем примере.

```
import pandas as pd

car_colors = pd.Series(['Blue', 'Red', 'Green'],
                       dtype='category')
car_data = pd.Series(
    pd.Categorical(
        ['Blue', 'Green', 'Red', 'Blue', 'Red'],
        categories=car_colors, ordered=False))

car_colors.cat.categories = ["Purple", "Yellow", "Mauve"]
car_data.cat.categories = car_colors

print(car_data)
```

Как показано, все, что вам действительно нужно сделать, — это установить для свойства `cat.categories` новое значение. Вот вывод этого примера:

```
0    Purple
1    Yellow
2     Mauve
3    Purple
4     Mauve
dtype: category
Categories (3, object): [Purple, Yellow, Mauve]
```

Объединение уровней

Некий категориальный уровень может быть слишком низок, чтобы предлагать существенные данные для анализа. Возможно, есть только несколько значений, которых может быть недостаточно для создания статистической разницы. В этом случае объединение нескольких небольших категорий может предложить лучшие результаты для анализа. В следующем примере показано, как объединять категории:

```
import pandas as pd

car_colors = pd.Series(['Blue', 'Red', 'Green'],
    dtype='category')
```

```
car_data = pd.Series(
    pd.Categorical(
        ['Blue', 'Green', 'Red', 'Green', 'Red', 'Green'],
        categories=car_colors, ordered=False))

car_data = car_data.cat.set_categories(
    ["Blue", "Red", "Green", "Blue_Red"])
print(car_data.loc[car_data.isin(['Red'])])
car_data.loc[car_data.isin(['Red'])] = 'Blue_Red'
car_data.loc[car_data.isin(['Blue'])] = 'Blue_Red'
car_data = car_data.cat.set_categories(
    ["Green", "Blue_Red"])

print()
print(car_data)
```

Этот пример демонстрирует, что есть только одна запись `Blue`, две `Red`, но три `Green`, что переводит `Green` в большинство. Объединение `Blue` и `Red` вместе — это двухэтапный процесс. Сначала категория `Blue_Red` добавляется к `car_data`. Затем записи `Red` и `Blue` меняются на `Blue_Red`, что создает объединенную категорию. В качестве последнего шага можно удалить ненужные категории.

Но прежде чем удастся изменить записи `Red` на `Blue_Red`, их нужно найти. Вот где комбинация вызовов `isin()`, находящая записи `Red`, и `loc[]`, получающих их индекс, обеспечивает именно то, что нужно. Первый оператор `print` демонстрирует результат использования этой комбинации. Вот вывод этого примера.

```
2     Red
4     Red
dtype: category
Categories (4, object): [Blue, Red, Green, Blue_Red]

0    Blue_Red
1       Green
2    Blue_Red
3       Green
4    Blue_Red
5       Green
dtype: category
Categories (2, object): [Green, Blue_Red]
```

Обратите внимание, что теперь есть три записи `Blue_Red` и три записи `Green`. Категории `Blue` и `Red` больше не используются. В результате уровни теперь объединены, как и ожидалось.

Работа с датами в данных

Даты в данных могут создавать проблемы. С одной стороны, даты хранятся в виде числовых значений. Однако точное значение числа зависит от представления для конкретной платформы и может даже зависеть от предпочтений пользователей. Например, пользователи Excel могут выбрать начало дат с 1900 или 1904 года (`https://support.microsoft.com/enus/help/214330/differences-between-the-1900-and-the-1904-datesystem-in-excel`). Числовая кодировка для каждого отличается, поэтому одна и та же дата может иметь два числовых значения в зависимости от даты начала.

Помимо проблем с представлением, вам также необходимо обдумать, как работать со значениями времени. Создать формат значения времени, представляющий значение, понятное пользователю, сложно. Например, вам может потребоваться использовать *среднее время по Гринвичу* (Greenwich Mean Time — GMT) в одних ситуациях, но местный часовой пояс — в других. Преобразование между различными форматами времени также проблематично. Имея это в виду, в следующих разделах представлены подробные сведения о том, как справляться с проблемами времени.

Форматирование значений даты и времени

Получение правильного представления даты и времени может значительно облегчить выполнение анализа. Например, вам часто приходится менять представление, чтобы получить правильную сортировку значений. Python предоставляет два распространенных метода форматирования даты и времени. Первый метод заключается в вызове функции `str()`, которая просто превращает значение `datetime` в строку без какого-либо форматирования. Функция `strftime()` требует больше работы, поскольку вы должны определить, как должно отображаться значение `datetime` после преобразования. При использовании функции `strftime()` вы должны предоставить строку, содержащую специальные директивы, которые определяют форматирование. Вы можете найти список этих директив по адресу `http://strftime.org/`.

Теперь, когда у вас есть представление о том, как работает преобразование времени и даты, пришло время рассмотреть пример. В следующем примере создается объект `datetime`, а затем его значение преобразуется в строку с использованием двух разных подходов:

```
import datetime as dt

now = dt.datetime.now()
```

```
print(str(now))
print(now.strftime('%a, %d %B %Y'))
```

Здесь вы можете увидеть, что использование функции `str()` — это самый простой подход. Однако, как показано в следующем выводе, он может не обеспечить нужного вам вывода. Использование функции `strftime()` куда более гибко.

```
2018-09-21 11:39:49.698891
Fri, 21 September 2018
```

Правильное преобразование времени

Часовые пояса и различия в местном времени могут вызвать всевозможные проблемы при выполнении анализа. В связи с этим некоторые типы расчетов просто требуют сдвига во времени, чтобы получить правильные результаты. Независимо от причины, в какой-то момент вам может потребоваться преобразовать одно время в другое. В следующих примерах показаны некоторые методы, которые можно использовать для решения этой задачи:

```
import datetime as dt

now = dt.datetime.now()
timevalue = now + dt.timedelta(hours=2)

print(now.strftime('%H:%M:%S'))
print(timevalue.strftime('%H:%M:%S'))
print(timevalue - now)
```

Функция `timedelta()` упрощает преобразование времени. Для изменения значения времени и даты с функцией `timedelta()` вы можете использовать любое из следующих имен параметров:

- `days`
- `seconds`
- `microseconds`
- `milliseconds`
- `minutes`
- `hours`
- `weeks`

Вы также можете манипулировать временем, выполняя сложение или вычитание его значений. Вы даже можете вычесть два значения времени, чтобы определить разницу между ними. Вот вывод этого примера:

```
11:42:22
13:42:22
2:00:00
```

Обратите внимание, что `now` — это местное время, значение `timevalue` — это два часовых пояса, отличных от данного, а разница между ними составляет два часа. Используя эти методы, вы можете осуществлять все виды преобразований, чтобы гарантировать, что ваш анализ всегда задействует точно ориентированные по времени значения, которые вам нужны.

Борьба с отсутствием данных

Иногда в данных, которые вы получаете, отсутствует информация в определенных полях. Например, в записи клиента может отсутствовать возраст. Если значения отсутствуют в достаточном количестве записей, любой выполняемый вами анализ будет искажен, а результаты анализа будут непредсказуемо перекошены. Наличие стратегии для борьбы с отсутствующими данными очень важно. Следующие разделы дадут некоторое представление о том, как решать эти проблемы и добиваться лучших результатов.

Нахождение недостающих данных

Поиск недостающих данных в наборе очень важен, так как он исключает получение неверных результатов анализа. Следующий код показывает, как можно получить список пропущенных значений без особых усилий:

```
import pandas as pd
import numpy as np

s = pd.Series([1, 2, 3, np.NaN, 5, 6, None])

print(s.isnull())

print()
print(s[s.isnull()])
```

Набор данных может представлять отсутствующие данные несколькими способами. В этом примере отсутствующие данные представлены как значения Python `np.NaN` и `None`.

Для обнаружения пропущенных значений используйте метод `isnull()`. Когда значение отсутствует, вывод показывает значение `True`. Добавляя индекс в набор данных, вы получаете только недостающие записи. Пример дает следующий вывод:

```
0    False
1    False
2    False
3     True
4    False
5    False
6     True
dtype: bool
3   NaN
6   NaN
dtype: float64
```

Отсутствие в коде

После того как вы выясните, что в вашем наборе данных отсутствует информация, вам нужно подумать, что с этим делать. Есть три возможности — игнорировать проблему, заполнить недостающие элементы или удалить (отбросить) недостающие записи из набора данных. Игнорирование проблемы может привести к всевозможным трудностям при вашем анализе, поэтому такой вариант используется реже всего. В следующем примере показан один метод заполнения недостающих данных или удаления ошибочных записей из набора данных:

```
import pandas as pd
import numpy as np

s = pd.Series([1, 2, 3, np.NaN, 5, 6, None])

print(s.fillna(int(s.mean())))
print()
print(s.dropna())
```

Интерес представляют два метода: `fillna()`, который заполняет пропущенные записи, и `dropna()`, который отбрасывает пропущенные записи. При использовании метода `fillna()` следует указать значение для пропущенных данных. В этом примере используется среднее значение всех значений, но вы можете выбрать ряд других подходов. Ниже приведен вывод этого примера.

```
0    1.0
1    2.0
2    3.0
3    3.0
4    5.0
5    6.0
6    3.0
dtype: float64
```

```
0    1.0
1    2.0
2    3.0
4    5.0
5    6.0
dtype: float64
```

Работать с сериями просто потому, что набор данных очень прост. Однако при работе с `DataFrame` проблема становится значительно более сложной. У вас все еще есть возможность отбросить весь ряд. Если столбец заполнен редко, вместо рядов можно удалить сам столбец. Заполнение данных также становится более сложным, поскольку необходимо учитывать набор данных в целом, в дополнение к потребностям отдельных функций.

Добавление недостающих данных

Предыдущий раздел упоминает процесс дополнения отсутствующих данных (дописывание характеристик на основе того, как эти данные используются). Используемая вами техника зависит от типа данных, с которыми вы работаете. Например, при работе с ансамблем деревьев (обсуждение деревьев приведено в разделе "Иерархическая кластеризация" главы 15 и в разделе "Простое дерево решений" главы 20) вы можете заменить отсутствующие значения на –1 и полагаться на *алгоритм imputer* (imputer) (алгоритм преобразования, используемый для заполнения пропущенных значений), чтобы определить наилучшее возможное значение для пропущенных данных. В следующем примере показан метод, который можно использовать для вычисления отсутствующих значений:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import Imputer

s = [[1, 2, 3, np.NaN, 5, 6, None]]

imp = Imputer(missing_values='NaN',
              strategy='mean', axis=0)

imp.fit([[1, 2, 3, 4, 5, 6, 7]])

x = pd.Series(imp.transform(s).tolist()[0])

print(x)
```

В этом примере в `s` отсутствуют некоторые значения. Для замены этих пропущенных значений код создает `Imputer`. Параметр `missing_values`

определяет, что искать, т.е. NaN. Вы устанавливаете параметр axis в 0 для вставки по столбцам и в 1 для вставки по строкам. Параметр strategy определяет, как заменить отсутствующие значения. (Подробнее о параметрах Imputer можно узнать на странице https://scikitlearn.org/stable/modules/generated/sklearn.preprocessing.Imputer.html.

- **mean**. Заменяет значения, используя среднее значение по оси.
- **median**. Заменяет значения, используя медиану вдоль оси.
- **most_frequent**. Заменяет значения, используя наиболее частые значения вдоль оси.

Прежде чем вы сможете что-либо вставить, нужно предоставить статистику для Imputer, чтобы вызвать метод fit(). Затем код вызывает метод transform() для s, чтобы заполнить пропущенные значения. В данном примере необходимо отобразить результат в виде ряда. Чтобы создать серию, вы должны преобразовать вывод Imputer в список и использовать полученный список в качестве входных данных для Series(). Результат процесса с заполненными пропущенными значениями приведен ниже.

```
0   1
1   2
2   3
3   4
4   5
5   6
6   7
dtype: float64
```

Разделение и дробление: фильтрация и выбор данных

Возможно, вам не понадобится работать со всеми данными в наборе данных. Фактически может быть полезен просмотр только одного конкретного столбца, такого как возраст, или набора строк со значительным количеством информации. Чтобы получить только те данные, которые вам необходимы для решения конкретной задачи, выполните два шага.

1. Отфильтруйте строки, чтобы создать субъект данных, который соответствует выбранному критерию (например, все дети в возрасте от 5 до 10 лет).
2. Выберите столбцы данных, содержащие данные, которые необходимо проанализировать. Например, вам, вероятно, не понадобятся имена людей, если вы не хотите провести некий анализ на основе имени.

Акт *разделение и дробление* (slicing and dicing) данных дает подмножество данных, подходящих для анализа. В следующих разделах описываются различные способы получения определенных фрагментов данных для удовлетворения конкретных потребностей.

Разделение строк

При работе с данными *разделение* (slicing) может происходить несколькими способами, но метод, представляющий интерес в этом разделе, заключается в усечении данных из ряда двух- или трехмерных данных. Двухмерный массив может содержать температуры (ось x) в течение определенного периода времени (ось y). Разделение ряда означает, что вы видите температуру в определенное время. В некоторых случаях вы можете связать строки с наблюдениями в наборе данных.

Трехмерный массив может включать в себя ось для места (ось x), продукта (ось y) и времени (ось z), чтобы можно было видеть продажи товаров с течением времени. Возможно, вы хотите отследить, растут ли продажи товара и, в частности, где они растут. Разделение строки будет означать просмотр всех продаж одного конкретного продукта для всех мест в любое время. Решение этой задачи демонстрирует следующий пример:

```
x = np.array([[[1, 2, 3], [4, 5, 6], [7, 8, 9],],
              [[11,12,13], [14,15,16], [17,18,19],],
              [[21,22,23], [24,25,26], [27,28,29]]])
x[1]
```

Пример создает трехмерный массив. Затем пример разделяет первую строку этого массива для получения следующего вывода:

```
array([[11, 12, 13],
       [14, 15, 16],
       [17, 18, 19]])
```

Разделение столбцов

Используя примеры из предыдущего раздела, разделение столбцов позволит получать данные из строк под углом 90 градусов. Другими словами, при работе с двухмерным массивом вы хотели бы видеть время, в которое были достигнуты определенные температуры. Аналогично, при работе с трехмерным массивом, вы можете в любой момент увидеть продажи всех продуктов для определенного местоположения. В некоторых случаях можно связать столбцы с признаками в наборе данных. В следующем примере показано, как решить эту задачу, используя тот же массив, что и в предыдущем разделе:

```
x = np.array([[[1, 2, 3], [4, 5, 6],  [7, 8, 9],],
              [[11,12,13], [14,15,16], [17,18,19],],
              [[21,22,23], [24,25,26], [27,28,29]]])
x[:,1]
```

Обратите внимание, что индексирование теперь происходит на двух уровнях. Первый индекс относится к строке. Применение двоеточия (:) для строки означает использование всех строк. Второй индекс относится к столбцу. В этом случае вывод будет содержать столбец 1. Вот вывод, который вы увидите:

```
array([[ 4, 5, 6],
       [14, 15, 16],
       [24, 25, 26]])
```

Этот массив трехмерный. Следовательно, каждый из столбцов содержит все элементы оси z. Вы видите каждую строку от 0 до 2 для столбца 1 с каждым элементом оси z от 0 до 2 для этого столбца.

Дробление

Дробление (dicing) набора данных означает разделение строк и столбцов таким образом, что в итоге получается клин данных. Например, при работе с трехмерным массивом может потребоваться в любое время увидеть продажи определенного продукта в определенном месте. В следующем примере показано, как решить эту задачу, используя тот же массив, что и в предыдущих двух разделах:

```
x = np.array([[[1, 2, 3],  [4, 5, 6],  [7, 8, 9],],
              [[11,12,13], [14,15,16], [17,18,19],],
              [[21,22,23], [24,25,26], [27,28,29]]])
print(x[1,1])
print(x[:,1,1])
print(x[1,:,1])
print()
print(x[1:2, 1:2])
```

В этом примере массив разделяется четырьмя способами. Вначале вы получаете строку 1, столбец 1. Конечно, вам может понадобиться столбец 1, ось z, 1. Если это не совсем правильно, вместо этого всегда можно запросить строку 1, ось z. С другой стороны, могут понадобиться строки 1 и 2 столбцов 1 и 2. Вот вывод всех четырех запросов:

```
[14 15 16]
[ 5 15 25]
[12 15 18]

[[[14 15 16]]]
```

Конкатенация и преобразование

Данные, используемые для науки о данных, редко бывают аккуратными. Возможно, вам придется работать с несколькими базами данных в разных местах, каждая из которых имеет свой собственный формат данных. Невозможно выполнить анализ таких разрозненных источников информации с какой-либо точностью. Чтобы сделать данные полезными, необходимо создать единый набор данных (в результате *конкатенации* (concatenating) или объединения данных из различных источников).

Частью процесса является обеспечение одинаковых характеристик для каждого поля, которое вы создаете для комбинированного набора данных. Например, поле возраста в одной базе данных может отображаться в виде строки, но в другой базе данных может использоваться целое число для того же поля. Чтобы поля работали вместе, они должны отображаться как информация одного типа.

В следующих разделах описано, как с помощью объединения и преобразования данных из различных источников создать единый набор. После получения единого набора данных из этих источников вы можете приступить к решению таких задач, как анализ данных. Конечно, хитрость заключается в создании единого набора данных, который действительно представляет данные во всех этих разрозненных наборах данных, — изменение данных приведет к искаженным результатам.

Добавление новых переменных и случаев

Нередко возникает необходимость комбинировать наборы данных различными способами или добавлять новую информацию для анализа. Результатом является комбинированный набор данных, который включает либо новые наблюдения, либо переменные. В следующем примере показаны методы решения обеих задач:

```
import pandas as pd

df = pd.DataFrame({'A': [2,3,1],
                   'B': [1,2,3],
                   'C': [5,3,4]})

df1 = pd.DataFrame({'A': [4],
                    'B': [4],
                    'C': [4]})

df = df.append(df1)
df = df.reset_index(drop=True)
```

```
print(df)

df.loc[df.last_valid_index() + 1] = [5, 5, 5]
print()
print(df)

df2 = pd.DataFrame({'D': [1, 2, 3, 4, 5]})

df = pd.DataFrame.join(df, df2)
print()
print(df)
```

Самый простой способ добавления большего количества данных в существующий объект `DataFrame` — это использование метода `append()`. Вы также можете использовать метод `concat()` (он описан в главе 13). В `df` находятся три случая (case), добавляемые к одному случаю, находящемуся в `df1`. Чтобы обеспечить добавление данных в соответствии с ожиданиями, столбцы в `df` и `df1` должны совпадать. Когда вы складываете два объекта `DataFrame` таким способом, новый объект `DataFrame` содержит старые значения индекса. Используйте метод `reset_index()`, чтобы создать новый индекс и упростить доступ к случаям (case).

Вы также можете добавить другой случай (case) в существующий объект `DataFrame`, создав новый случай напрямую. Каждый раз, когда вы добавляете новую запись в позицию, которая на одну позицию больше, чем `last_valid_index()`, вы получаете в результате новый случай (case).

Иногда вам нужно добавить новую переменную (столбец) в `DataFrame`. В этом случае вы полагаетесь на метод `join()`. Полученный объект `DataFrame` будет сопоставлять случаи с одинаковым значением индекса, поэтому индексация важна. Кроме того, если вам не нужны пустые значения, количество наблюдений в обоих объектах `DataFrame` должно совпадать. Вот вывод этого примера:

```
   A  B  C
0  2  1  5
1  3  2  3
2  1  3  4
3  4  4  4

   A  B  C
0  2  1  5
1  3  2  3
2  1  3  4
3  4  4  4
4  5  5  5
```

```
   A  B  C  D
0  2  1  5  1
1  3  2  3  2
2  1  3  4  3
3  4  4  4  4
4  5  5  5  5
```

Удаление данных

В какой-то момент вам может потребоваться удалить наблюдения или переменные из набора данных, поскольку они не требуются для анализа. В обоих случаях для выполнения этой задачи используйте метод `drop()`. Разница в удалении случаев (case) или переменных заключается в том, как вы описываете, что нужно удалить:

```
import pandas as pd

df = pd.DataFrame({'A': [2,3,1],
                   'B': [1,2,3],
                   'C': [5,3,4]})

df = df.drop(df.index[[1]])
print(df)

df = df.drop('B', 1)
print()
print(df)
```

Пример начинается с удаления случая из `df`. Обратите внимание, что код использует индекс для описания того, что нужно удалить. Вы можете удалить только один случай (как показано), диапазоны случаев или отдельные случаи, разделенные запятыми. Основная задача — убедиться, что у вас есть правильные номера индексов для случаев, которые вы хотите удалить.

Удаление столбца отличается. В этом примере показано, как удалить столбец, используя его имя. Вы также можете удалить столбец с помощью индекса. В обоих случаях вы должны указать ось как часть процесса удаления (обычно 1). Вот вывод этого примера:

```
   A  B  C
0  2  1  5
2  1  3  4

   A  C
0  2  5
2  1  4
```

Сортировка и перетасовка

Сортировка и *перетасовка* (shuffling) — это две крайности одной и той же цели — управлять порядком данных. В первом случае вы упорядочиваете данные, а во втором — удаляете из порядка систематические шаблоны. Как правило, вы не сортируете наборы данных для анализа, поскольку это может привести к получению неверных результатов. Однако можно отсортировать данные для презентации. В следующем примере показаны сортировка и перетасовка:

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'A': [2,1,2,3,3,5,4],
                   'B': [1,2,3,5,4,2,5],
                   'C': [5,3,4,1,1,2,3]})

df = df.sort_values(by=['A', 'B'], ascending=[True, True])
df = df.reset_index(drop=True)
print(df)

index = df.index.tolist()
np.random.shuffle(index)
df = df.loc[df.index[index]]
df = df.reset_index(drop=True)
print()
print(df)
```

Оказывается, сортировка данных немного проще, чем их перетасовка. Чтобы отсортировать данные, нужно использовать метод `sort_values()` и определить, какие столбцы использовать для индексации. Можно также определить, находится ли индекс в порядке возрастания или убывания. По завершении всегда вызывайте метод `reset_index()`, чтобы появился индекс.

Чтобы перетасовать данные, сначала получите текущий индекс с помощью метода `df.index.tolist()` и передайте его в `index`. Вызов метода `random.shuffle()` создает новый порядок для индекса. Затем примените новый порядок к `df`, используя `loc[]`. Как всегда, вызывайте метод `reset_index()`, чтобы сохранить новый порядок. Вывод этого примера (но обратите внимание, что второй вывод может не соответствовать вашему выводу, поскольку он был перетасован) приведен ниже.

```
   A  B  C
0  1  2  3
1  2  1  5
2  2  3  4
3  3  4  1
4  3  5  1
```

```
5   4   5   3
6   5   2   2

    A   B   C
0   2   1   5
1   2   3   4
2   3   4   1
3   1   2   3
4   3   5   1
5   4   5   3
6   5   2   2
```

Агрегирование данных на любом уровне

Агрегация (aggregation) — это процесс объединения или группировки данных в набор, пакет или список. Данные могут быть или не быть похожими. Однако в большинстве случаев функция агрегации статистически объединяет несколько строк, используя такие алгоритмы, как среднее значение, количество, максимум, медиану, минимум, моду или сумму. Причины для агрегирования данных приведены ниже.

- Упростить анализ.
- Ограничить способность любого выводить из набора личные данные человека по соображениям конфиденциальности или по другим причинам.
- Создать объединенный элемент данных из одного источника данных, который соответствует объединенному элементу данных в другом источнике.

Наиболее важным применением агрегации данных является обеспечение анонимности для решения правовых или иных задач. Иногда оказывается, что даже те данные, которые должны быть анонимными, позволяют идентифицировать личность с использованием надлежащих методов анализа. Например, исследователи обнаружили, что возможна идентификация лиц на основании только трех покупок с помощью кредитной карты (см. `https://www.computerworld.com/article/2877935/how-three-smallcredit-card-transactions-could-reveal-your-identity.html`). Вот пример выполнения агрегации:

```
import pandas as pd

df = pd.DataFrame({'Map': [0,0,0,1,1,2,2],
                   'Values': [1,2,3,5,4,2,5]})

df['S'] = df.groupby('Map')['Values'].transform(np.sum)
```

```
df['M'] = df.groupby('Map')['Values'].transform(np.mean)
df['V'] = df.groupby('Map')['Values'].transform(np.var)

print(df)
```

В этом случае у вас есть две начальные функции для этого объекта `DataFrame`. Значения в `Map` определяют, какие элементы в `Values` принадлежат друг другу. Например, при расчете суммы для `Map` индекса 0 вы используете `Values` 1, 2 и 3.

Чтобы выполнить агрегирование, сначала нужно вызвать функцию `groupby()` для группировки значений `Map`. Затем вы индексируете `Values` и полагаетесь на функцию `transform()` для создания агрегированных данных, используя один из нескольких алгоритмов библиотеки `NumPy`, например `np.sum`. Ниже приведены результаты этого расчета.

```
   Map  Values  S    M    V
0    0       1  6  2.0  1.0
1    0       2  6  2.0  1.0
2    0       3  6  2.0  1.0
3    1       5  9  4.5  0.5
4    1       4  9  4.5  0.5
5    2       2  7  3.5  4.5
6    2       5  7  3.5  4.5
```

Глава 8

Формирование данных

В ЭТОЙ ГЛАВЕ...

- Манипулирование данными HTML
- Манипулирование необработанным текстом
- Модель набора слов и другие методы
- Манипулирование графическими данными

В главе 7 описаны методы работы с данными как с сущностью, т.е. как с чем-то, с чем вы работаете в Python. Тем не менее данные существуют не в вакууме. Они не просто внезапно появляются в Python без всякой причины. Как было показано в главе 6, данные загружают. Но загрузки может быть недостаточно — возможно, придется формировать данные в ходе их загрузки. В этой главе вы узнаете, как работать с различными типами контейнеров таким образом, чтобы можно было загружать данные из ряда контейнеров сложных типов, таких как страницы HTML. На самом деле, вы можете работать даже с графикой, изображениями и звуком.


ЗАПОМНИ!

По мере изучения книги вы обнаружите, что данные принимают все виды форм. Что касается компьютера, то данные состоят из нулей и единиц. Люди придают данным смысл, форматируя, храня и интерпретируя их определенным образом. Одна и та же группа из 0 и 1 может быть числом, датой или текстом, в зависимости от интерпретации. Контейнер данных предоставляет подсказки о том, как их интерпретировать, поэтому данная глава так важна при использовании

языка Python для обнаружения шаблонов в данных. Вы обнаружите, что можете находить шаблоны в самых неожиданных местах.


СОВЕТ

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_08_Shaping_Data.ipynb`.

Работа со страницами HTML

Страницы HTML содержат данные в иерархическом формате. Содержимое HTML обычно находится строго в формате HTML или в XML. Формат HTML может представлять проблемы, поскольку он не всегда строго следует правилам форматирования. XML действительно строго следует правилам форматирования из-за стандартов, используемых для его определения, что облегчает анализ. Однако в обоих случаях для анализа страницы используют похожие методы. В первом разделе данной главы речь пойдет о том, как анализировать страницы HTML в целом.

Иногда требуются не все данные страницы — нужны только конкретные данные, и именно здесь вступает в игру XPath. Вы можете использовать XPath для поиска конкретных данных на странице HTML и извлечения их для неких потребностей.

Анализ XML и HTML

Простого извлечения данных из файла XML (см. главу 6) может быть недостаточно. Данные могут быть в неправильном формате. Используя подход, описанный в главе 6, вы получите объект `DataFrame`, содержащий три столбца типа `str`. Очевидно, что со строковыми данными вы можете выполнять не много операций. В следующем примере формируются данные XML из главы 6 для создания нового объекта `DataFrame`, содержащего только элементы `<Number>` и `<Boolean>` в правильном формате.

```
from lxml import objectify
import pandas as pd
from distutils import util

xml = objectify.parse(open('XMLData.xml'))
root = xml.getroot()
df = pd.DataFrame(columns=('Number', 'Boolean'))

for i in range(0, 4):
    obj = root.getchildren()[i].getchildren()
```

```
    row = dict(zip(['Number', 'Boolean'],
                   [obj[0].pyval,
                    bool(util.strtobool(obj[2].text))]))
    row_s = pd.Series(row)
    row_s.name = obj[1].text
    df = df.append(row_s)

print(type(df.loc['First']['Number']))
print(type(df.loc['First']['Boolean']))
```

Объект `df` типа `DataFrame` изначально создается пустым, но когда код проходит по дочерним узлам корневого узла, он извлекает список, который код использует для инкремента `df`, содержащий следующее:

- элемент `<Number>` (выражается как `int`);
- элемент порядкового номера (`string`);
- элемент `<Boolean>` (выражается как `string`).

Фактически код опирается на элемент порядкового номера в качестве метки индекса и создает новую отдельную строку для добавления к существующему `DataFrame`. Эта операция программно преобразует информацию, содержащуюся в дереве XML, в правильный тип данных для помещения в существующие переменные объекта `df`. Теперь элементы `<Number>` уже доступны как тип `int`; преобразование элемента `<Boolean>` немного сложнее. Вы должны преобразовать строку в числовое значение, используя функцию `strtobool()` из `distutils.util`. Выходными данными являются `0` для значений `False` и `1` для значений `True`. Но это уже не логическое значение. Чтобы получить логическое значение, вы должны преобразовать 0 или 1, используя функцию `bool()`.


СОВЕТ

В этом примере показано также получение доступа к отдельным значениям в `DataFrame`. Обратите внимание, что свойство `name` использует теперь значение элемента `<String>` для облегчения доступа. Вы предоставляете значение индекса, используя `loc`, а затем получаете доступ к отдельным признакам, используя второй индекс. Вывод этого примера таков:

```
<class 'int'>
<class 'bool'>
```

Использование XPath для извлечения данных

Использование XPath для извлечения данных из вашего набора данных может значительно снизить сложность вашего кода и, возможно, ускорить его. В следующем примере показана версия примера XPath из предыдущего

раздела. Обратите внимание, что эта версия короче и не требует использования цикла for.

```
from lxml import objectify
import pandas as pd
from distutils import util

xml = objectify.parse(open('XMLData.xml'))
root = xml.getroot()

map_number = map(int, root.xpath('Record/Number'))
map_bool = map(str, root.xpath('Record/Boolean'))
map_bool = map(util.strtobool, map_bool)
map_bool = map(bool, map_bool)
map_string = map(str, root.xpath('Record/String'))

data = list(zip(map_number, map_bool))

df = pd.DataFrame(data,
                  columns=('Number', 'Boolean'),
                  index = list(map_string))

print(df)
print(type(df.loc['First']['Number']))
print(type(df.loc['First']['Boolean']))
```

Пример начинается так же, как и предыдущий, с импорта данных и получения корневого узла. На этом этапе пример создает объект данных, содержащий пары номеров записей и логических значений. Поскольку все записи файла XML являются строками, необходимо использовать функцию `map()` для преобразования строк в соответствующие значения. Работать с номером записи просто, достаточно сопоставить его с `int`. Функция `xpath()` получает путь от корневого узла к нужным вам данным, в данном случае это `'Record/Number'`.

Сопоставление логического значения немного сложнее. Как и в предыдущем разделе, вы должны использовать функцию `util.strtobool()` для преобразования строковых логических значений в число, которое функция `bool()` может преобразовать в логический эквивалент. Однако, если вы попытаетесь выполнить только двойное сопоставление, появится сообщение об ошибке, в котором говорится, что списки не содержат обязательной функции `tolower()`. Чтобы преодолеть это препятствие, выполните тройное сопоставление и конвертируйте данные сначала в строку, используя функцию `str()`.

Создание объекта `DataFrame` тоже отличается. Вместо добавления отдельных строк добавьте все строки одновременно, используя `data`. Настройка имен столбцов такая же, как и ранее. Однако теперь нужен какой-то способ добавления имен строк, как в предыдущем примере. Эта задача решается за счет

установки в параметре index сопоставленной версии вывода функции xpath() для пути 'Record/String'. Ниже приведен результат, который вы можете ожидать.

```
       Number  Boolean
First       1     True
Second      2    False
Third       3     True
Fourth      4    False
<type 'numpy.int64'>
<type 'numpy.bool_'>
```

Работа с необработанным текстом

Даже если вам кажется, что необработанный текст не представляет проблемы при анализе, поскольку он не содержит какого-либо специального форматирования, вам нужно подумать о том, как хранится текст и содержит ли он специальные слова. Многочисленные формы Unicode также могут представлять проблемы с интерпретацией, которые необходимо учитывать при работе с текстом. Использование регулярных выражений может помочь найти конкретную информацию в текстовом файле. Вы можете использовать регулярные выражения для очистки данных и сопоставления с образцом. В следующих разделах описаны методы, используемые для формирования необработанных текстовых файлов.

Работа с Unicode

Текстовые файлы — это просто текст. Способ кодирования текста может быть разным. Например, символ может использовать семь, восемь или более битов для кодирования. Использование специальных символов также может отличаться. Короче говоря, интерпретация битов, используемых для создания символов, отличается от кодировки к кодировке. Множество кодировок представлено по адресу http://www.i18nguy.com/unicode/codepages.html.

Иногда вам придется работать с кодировками, отличными от стандартной, установленной в среде Python. При работе с Python 3.*x* вы должны полагаться на *8-битовый формат преобразования Юникода* (Universal Transformation Format 8-bit — UTF-8) в качестве кодировки, используемой для чтения и записи файлов. Эта среда всегда установлена на UTF-8, и попытка изменить ее вызывает сообщение об ошибке. Статья по адресу https://docs.python.org/3/howto/

unicode.html дает некоторое представление о том, как обойти проблемы Unicode в Python.


ВНИМАНИЕ!

Неправильное обращение с кодировкой может помешать вам выполнить такие задачи, как импорт модулей или обработка текста. Обязательно тщательно и полностью протестируйте свой код, чтобы любая проблема с кодировкой не повлияла на запуск приложения. Хорошие дополнительные статьи по этой теме находятся по адресам http://blog.notdot.net/2010/07/Getting-unicode-right-in-Python и http://web.archive.org/web/20120722170929/http://boodebr.org/main/python/all-about-python-and-unicode.

Морфологический поиск и удаление стоп-слов

Морфологический поиск (stemming) — это процесс приведения слов к их *основе* (stem) (или *корневому* (root)) слову. Эта задача отлична от выяснения того, что некие слова происходят от латинских или других корней, она делает слова похожими друг на друга в целях сравнения или замены. Например, слова *cats*, *catty* и *catlike* имеют основу *cat* (кот). Морфологический поиск помогает анализировать предложения с помощью более эффективного лексического анализатора, поскольку алгоритм машинного обучения должен изучать информацию об основе *cat*, а не обо всех ее вариантах.

Однако удаление суффиксов для создания основ слов и, как правило, лексический анализ предложений — это только две части процесса создания чего-то вроде интерфейса на естественном языке. Языки включают в себя большое количество связующих слов, таких как *a, as, the, that* и т.д. в английском языке, которые мало что значат для компьютера, но имеют важное значение для людей. Эти короткие, малополезные слова являются *стоп-словами* (stop word). Для людей предложения не имеют смысла без них, но для компьютера они могут стать средством прекращения анализа предложений.

Морфологический поиск и удаления стоп-слов упрощает текст и уменьшает количество текстовых элементов, так что остаются только основные элементы. Кроме того, вы сохраняете только те термины, которые более всего соответствуют истинному смыслу фразы. Сокращая фразы таким способом, вычислительный алгоритм может работать быстрее и обрабатывать текст более эффективно.


ВНИМАНИЕ!

В этом примере требуется использование комплекта Natural Language Toolkit (NLTK), который Anaconda (подробнее об Anaconda см. в главе 3) не устанавливает стандартно. Чтобы использовать этот пример, загрузите и установите NLTK, следуя инструкциям на

http://www.nltk.org/install.html для вашей платформы. Убедитесь, что вы устанавливаете NLTK для той версии Python, которую используете в этой книге, если в вашей системе установлено несколько версий Python. После установки NLTK вы также должны установить связанные с ним пакеты. В инструкциях, представленных по адресу http://www.nltk.org/data.html, рассказывается, как выполнить эту задачу (установите все пакеты, чтобы гарантировать, что у вас есть все).

В следующем примере демонстрируется, как выполнять морфологический поиск и удаление стоп-слов из предложения. Он начинается с подготовки алгоритма для выполнения необходимого анализа с использованием тестового предложения. После этого проверяется второе предложение на наличие слов, которые встречаются в первом.

```
from sklearn.feature_extraction.text import *
from nltk import word_tokenize
from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()

def stem_tokens(tokens, stemmer):
    stemmed = []
    for item in tokens:
        stemmed.append(stemmer.stem(item))
    return stemmed

def tokenize(text):
    tokens = word_tokenize(text)
    stems = stem_tokens(tokens, stemmer)
    return stems

vocab = ['Sam loves swimming so he swims all the time']
vect = CountVectorizer(tokenizer=tokenize,
                       stop_words='english')
vec = vect.fit(vocab)

sentence1 = vec.transform(['George loves swimming too!'])

print(vec.get_feature_names())
print(sentence1.toarray())
```

В начале примера создается словарь с помощью тестового предложения, а затем помещается в vocab. Затем создается CountVectorizer, vect для хранения списка основ слов, но исключая стоп-слова. Параметр tokenizer определяет функцию, используемую для определения основ слов. Параметр

`stop_words` ссылается на выбранный файл, который содержит стоп-слова для определенного языка, в данном случае английского. Есть также файлы для других языков, таких как французский и немецкий. (Другие параметры `CountVectorizer()` см. по адресу `https://scikitlearn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer`.) Словарь добавлен в другой `CountVectorizer`, `vec`, который используется для выполнения фактического преобразования в тестовом предложении с использованием функции `transform()`. Вот вывод этого примера:

```
['love', 'sam', 'swim', 'time']
[[1 0 1 0]]
```

Первый вывод показывает основы слов. Обратите внимание, что список содержит только *swim*, но не *swimming* и *swims*. Все стоп-слова также отсутствуют. Например, вы не видите слов *so, he, all* или *the*.

Во втором выводе показано, сколько раз каждое основание слов встречается в тестовом предложении. В этом случае вариант *love* появляется один раз и вариант *swim* появляется один раз. Слова *sam* и *time* не встречаются во втором предложении, поэтому их значения установлены в нуль.

Знакомство с регулярными выражениями

Регулярные выражения (regular expression) представляют аналитику данных интересный набор инструментов для анализа необработанного текста. Поначалу может быть сложно понять, как именно работают регулярные выражения. Однако такие сайты, как `https://regexr.com/`, позволяют экспериментировать с регулярными выражениями, чтобы вы могли увидеть, что использование различных выражений осуществляет определенные типы сопоставления с шаблоном. Конечно, в первую очередь следует выявить *соответствие шаблону* (pattern matching), т.е. использовать специальные символы, чтобы сообщить механизму синтаксического анализа, что нужно искать в необработанном текстовом файле. В табл. 8.1 приведены символы шаблонов и указано, как их использовать.

Таблица 8.1. Символы шаблонов, используемые в Python

Символ	Интерпретация
`(re)`	Группирует регулярные выражения и запоминает соответствующий текст
`(?: re)`	Группирует регулярные выражения без запоминания соответствующего текста

Продолжение табл. 8.1

Символ	Интерпретация
`(?#...)`	Указывает необрабатываемый комментарий
`re?`	Соответствует 0 или 1 вхождению предыдущего выражения (но не более 0 или 1 вхождению)
`re*`	Соответствует нулю или более вхождений предыдущего выражения
`re+`	Соответствует 1 или более вхождений предыдущего выражения
`(?> re)`	Соответствует независимому шаблону без возврата
`.`	Соответствует любому отдельному символу, кроме символа новой строки (`\n`) (параметр `m` соответствует также символу новой строки)
`[^...]`	Соответствует любому отдельному символу или диапазону символов, не находящихся в скобках
`[...]`	Соответствует любому отдельному символу или диапазону символов, находящихся в скобках
`re{ n, m}`	Соответствует не менее `n` и не более `m` вхождений предыдущего выражения
`\n, \t, etc.`	Соответствует управляющим символам, таким как символ новой строки (`\n`), возврат каретки (`\r`) и табуляция (`\t`)
`\d`	Соответствует цифрам (что эквивалентно использованию `[0-9]`)
`a|b`	Соответствует либо `a`, либо `b`
`re{ n}`	Точно соответствует числу вхождений предыдущего выражения, указанного в `n`
`re{ n,}`	Соответствует `n` или более вхождений предыдущего выражения
`\D`	Соответствует не цифрам
`\S`	Соответствует не пробелам
`\B`	Соответствует границам не слов
`\W`	Соответствует символам не слов
`\1...\9`	Соответствует *n*-му сгруппированному подвыражению

Символ	Интерпретация
`\10`	Соответствует *n*-му сгруппированному подвыражению, если оно уже найдено (в противном случае шаблон ссылается на восьмеричное представление кода символа)
`\A`	Соответствует началу строки
`^`	Соответствует началу линии
`\z`	Соответствует концу строки
`\Z`	Соответствует концу строки (когда есть символ новой строки, соответствует позиции непосредственно перед ним)
`$`	Соответствует концу линии
`\G`	Соответствует точке, где закончилось прошлое соответствие
`\s`	Соответствует пробелу (что эквивалентно использованию `[\t\n\r\f]`)
`\b`	Соответствует границам слов вне скобок; соответствует символу Backspace (0x08) внутри скобок
`\w`	Соответствует символам слова
`(?=re)`	Определяет позицию, используя шаблон (этот шаблон не имеет диапазона)
`(?!re)`	Определяет позицию, используя инверсию шаблона (у этого шаблона нет диапазона)
`(?-imx)`	Временно переключает параметры `i`, `m` или `x` в регулярном выражении (когда этот шаблон отображается в скобках, затрагивается только область в скобках)
`(?imx)`	Переключает параметры `i`, `m` или `x` в регулярном выражении (когда этот шаблон отображается в скобках, затрагивается только область в скобках)
`(?-imx:re)`	Временно отключает параметры `i`, `m` или `x` в скобках
`(?imx:re)`	Временно включает параметры `i`, `m` или `x` в скобках

Использование регулярных выражений помогает манипулировать сложным текстом, не прибегая к использованию других методов, описанных в этой

главе. В следующем примере показано, как извлечь номер телефона из предложения, независимо от того, где он указан. Подобные манипуляции полезны, когда приходится работать с текстом различного происхождения и в неправильном формате. Вы можете увидеть некоторые дополнительные процедуры манипулирования телефонными номерами по адресу `http://www.diveintopython.net/regular_expressions/phone_numbers.html`. Важно то, что этот пример помогает понять, как извлечь любой фрагмент из текста, который не нужен.

```
import re

data1 = 'My phone number is: 800-555-1212.'
data2 = '800-555-1234 is my phone number.'

pattern = re.compile(r'(\d{3})-(\d{3})-(\d{4})')

dmatch1 = pattern.search(data1).groups()
dmatch2 = pattern.search(data2).groups()

print(dmatch1)
print(dmatch2)
```

Пример начинается с двух телефонных номеров, помещенных в разных местах предложения. Прежде чем вы сможете что-то сделать, нужно создать шаблон. Всегда читайте шаблон слева направо. В этом случае шаблон ищет три цифры, затем тире, еще три цифры, затем еще одно тире и, наконец, четыре цифры.

Чтобы сделать процесс быстрее и проще, код вызывает функцию `compile()` для создания скомпилированной версии шаблона, чтобы Python не воссоздавал шаблон каждый раз, когда он вам нужен. Скомпилированный шаблон появляется в `pattern`.

Функция `search()` ищет шаблон в каждом тестовом предложении. Затем она помещает любой найденный текст в группы и выводит кортеж в одну из двух переменных. Ниже приведен вывод этого примера.

```
('800', '555', '1212')
('800', '555', '1234')
```

Использование модели наборов слов

Целью большинства случаев импорта данных является выполнение определенного типа анализа. Прежде чем вы сможете выполнить анализ текстовых данных, нужно лексически проанализировать каждое слово в наборе. Лексический анализ слов создает *набор слов* (bag of words). Затем вы можете использовать пакет слов для обучения *классификаторов* (classifier), алгоритмов особого

вида, используемых для разделения слов по категориям. В следующем разделе представлены дополнительные сведения о модели набора слов и показано, как с ним работать.

ПОЛУЧЕНИЕ 20 НАБОРОВ ДАННЫХ ИЗ ГРУПП НОВОСТЕЙ

В примерах, приведенных в следующих разделах, используется набор данных из 20 групп новостей (`http://qwone6.com/~jason/20Newsgroups/`), который является частью пакета Scikit-learn. Содержащий группы новостей сайт предоставляет дополнительную информацию о наборе данных, но по сути это хороший набор данных, который можно использовать для демонстрации различных видов анализа текста.

Вам не нужно делать ничего особенного, чтобы работать с этим набором данных, поскольку об этом позаботится библиотека Scikit-learn. Однако при запуске первого примера вы увидите сообщение `"WARNING:sklearn.datasets.twenty_newsgroups:Downloading dataset from http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz (14MB)"`. Это означает, что вам нужно дождаться завершения загрузки данных. Посмотрите на левую часть ячейки кода в IPython Notebook, и вы увидите знакомую запись `In [*]:`. Когда эта запись изменится и будет показан номер, загрузка завершена. Сообщение не исчезнет до следующего запуска ячейки.

Понятие модели "набор слов"

Как уже упоминалось во введении, для текстового анализа различных видов вам необходимо сначала лексически проанализировать слова и создать из них набор слов. Набор слов использует числа для обозначения слов, их частот и местоположения, которыми вы можете манипулировать математически, чтобы увидеть закономерности структурирования и использования слов. Модель набора слов игнорирует грамматику и даже порядок слов — основное внимание уделяется упрощению текста, чтобы вы могли легко анализировать его.

Создание набора слов опирается на *обработку текстов на естественном языке* (Natural Language Processing — NLP) и *поиск информации* (Information Retrieval — IR). Перед выполнением такого рода обработки вы обычно удаляете любые специальные символы (например, форматирование HTML из веб-источника), а также стоп-слова и, возможно, выполняете морфологический поиск (см. выше раздел "Морфологический поиск и удаление стоп-слов"). В этом примере вы непосредственно используете набор данных из 20 групп новостей. Вот пример того, как вы можете получить текстовый ввод и создать из него набор слов:

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import *

categories = ['comp.graphics', 'misc.forsale',
              'rec.autos', 'sci.space']
twenty_train = fetch_20newsgroups(subset='train',
                                  categories=categories,
                                  shuffle=True,
                                  random_state=42)

count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(
   twenty_train.data)

print("BOW shape:", X_train_counts.shape)
caltech_idx = count_vect.vocabulary_['caltech']
print('"Caltech": %i' % X_train_counts[0, caltech_idx])
```

В ряде примеров, которые вы видите в Интернете, неясно, откуда берется список используемых ими категорий. Сайт по адресу `http://qwone.com/~jason/20Newsgroups/` предоставляет список категорий, которые вы можете использовать. Список категорий взялся не из волшебной шляпы, многие сетевые примеры просто не удосуживаются документировать некоторые источники информации. Всегда обращайтесь к этому сайту, если у вас есть вопросы по таким темам, как категории наборов данных.

Вызов `fetch_20newsgroups()` загружает набор данных в память. Вы видите результирующий учебный объект, `twenty_train`, описанный как *связка* (bunch). На данный момент у вас есть объект, который содержит список категорий и связанных с ними данных, но приложение не анализировало данные, а алгоритм, используемый для работы с данными, не обучен.

Теперь, когда у вас есть связка данных для использования, вы можете создать набор слов. Процесс создания набора слов начинается с присвоения целочисленного значения (индекса вида) каждому уникальному слову в обучающем наборе. Кроме того, каждый документ получает целочисленное значение. Следующим шагом является подсчет каждого вхождения этих слов в каждом документе и создание списка пар документов и счета, чтобы вы знали, насколько часто встречается каждое слово в каждом документе.

Естественно, некоторые слова из основного списка не используются в некоторых документах, что приводит к созданию *многомерного разреженного набора данных* (high-dimensional sparse dataset). Матрица `scipy.sparse` представляет собой структуру данных, которая позволяет хранить только ненулевые элементы списка для экономии памяти. Когда код выполняет вызов

функции `count_vect.fit_transform()`, он помещает полученный набор слов в `X_train_counts`. Вы можете увидеть итоговое количество записей, обратившись к свойству `shape` и счету слов `"Caltech"` в первом документе:

```
BOW shape: (2356, 34750)
"Caltech": 3
```

Работа с n-граммами

N-грамма (n-gram) — это непрерывная последовательность элементов в тексте, которую вы хотите проанализировать. Это фонемы, слоги, буквы, слова или пары оснований. *N* в термине *n-грамма* относится к размеру. Например, *n*-грамма размером один — это *униграмма* (unigram). Пример в этом разделе использует размер три, образуя триграмму. Вы используете n-граммы вероятностным способом для выполнения таких задач, как прогнозирование следующей последовательности в серии, что будет не очень полезно, пока речь не пойдет о таких приложениях, как поисковые системы, которые пытаются предсказать слово, которое вы хотите ввести на основе предыдущих. Тем не менее эта техника имеет множество применений, таких как секвенирование ДНК и сжатие данных. В следующем примере показано, как создать *n*-граммы из набора данных 20 групп новостей:

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import *

categories = ['sci.space']

twenty_train = fetch_20newsgroups(subset='train',
                                  categories=categories,
                                  remove=('headers',
                                          'footers',
                                          'quotes'),
                                  shuffle=True,
                                  random_state=42)

count_chars = CountVectorizer(analyzer='char_wb',
                              ngram_range=(3,3),
                              max_features=10)

count_chars.fit(twenty_train['data'])

count_words = CountVectorizer(analyzer='word',
                              ngram_range=(2,2),
                              max_features=10,
                              stop_words='english')

count_words.fit(twenty_train['data'])
```

```
X = count_chars.transform(twenty_train.data)

print(count_chars.get_feature_names())
print(X[1].todense())
print(count_words.get_feature_names())
```

Код в начале такой же, как и в предыдущем разделе. Вы опять начинаете с выборки набора данных и помещаете его в связку. Но в этом случае процесс векторизации приобретает новый смысл. Аргументы обрабатывают данные особым образом.

В этом случае параметр `analyzer` определяет, как приложение создает *n*-граммы. Вы можете выбрать слова (`word`), символы (`char`) или символы в границах слова (`char_wb`). Параметру `ngram_range` требуется два ввода в виде кортежа: первый определяет минимальный размер *n*-грамм, второй — максимальный. Третий аргумент, `max_features`, определяет, сколько функций возвращает *векторизатор* (vectorizer). Во втором вызове векторизатора аргумент `stop_words` удаляет термины, содержащиеся в английском тексте (см. выше раздел "Морфологический поиск и удаление стоп-слов"). На этом этапе приложение подбирает данные к алгоритму преобразования.

В примере представлены три вывода. Первый показывает первые десять триграмм для символов из документа. Второй — это *n*-грамма для первого документа. Он показывает частоту первых десяти триграмм. Третий — это десятка лучших триграмм для слов. Вот вывод этого примера:

```
[' an', ' in', ' of', ' th', ' to', 'he ', 'ing', 'ion',
 'nd ', 'the']
[[0 0 2 5 1 4 2 2 0 5]]
['anonymous ftp', 'commercial space', 'gamma ray',
'nasa gov', 'national space', 'remote sensing',
'sci space', 'space shuttle', 'space station',
'washington dc']
```

Реализация преобразований TF-IDF

Преобразование TF-IDF (term frequency–inverse document frequency — частота термина–инверсная частота в документе) — это метод, используемый для компенсации слов, сравнительно часто встречающихся в разных документах, что затрудняет различие между документами, поскольку они слишком распространены (хороший пример — стоп-слова). Это преобразование действительно говорит о важности того или иного слова для уникальности документа. Чем больше частота слова в документе, тем важнее оно для этого документа. Однако измерение компенсируется размером документа — общим количеством слов, содержащихся в документе, — и частотой появления этого слова в других документах.

Даже если слово встречается в документе много раз, это не означает, что оно важно для понимания самого документа; во многих документах вы найдете

стоп-слова с той же периодичностью, что и слова, относящиеся к общим темам документа. Например, если вы анализируете документы, связанные с научной фантастикой (например, в наборе данных из 20 групп новостей), вы можете обнаружить, что многие из них имеют дело с *НЛО* (UFO); поэтому аббревиатура НЛО не может представлять различие между разными документами. Более того, документы большей длинны содержат больше слов, чем более короткие, и повторяющиеся слова легко найти, если текста достаточно много.


ЗАПОМНИ!

Фактически слово, встретившееся несколько раз в одном документе (или, возможно, в нескольких других), может оказаться весьма характерным и полезным при определении типа документа. Если вы работаете с документами, в которых обсуждаются научная фантастика и продажи автомобилей, то термин НЛО может быть отличительным, поскольку он легко разделяет два типа тем в документах.

Поисковым системам нередко нужно взвешивать слова в документе таким образом, чтобы определить, когда слово важно в тексте. Вы используете слова с более высоким весом для индексации документа, чтобы при запросе этих слов поисковая система получала данный документ. По этой причине преобразование TD-IDF довольно часто используется в приложениях поисковых систем.

Более подробно часть TF уравнения TF-IDF определяет, как часто термин появляется в документе, в то время как часть уравнения IDF определяет важность термина, поскольку он представляет обратную частоту этого слова среди всех документов. Большой IDF подразумевает редко встречающееся слово, и вес TF-IDF также будет больше. Небольшой IDF означает, что слово является распространенным, и это приведет к небольшому весу TF-IDF. Некоторые фактические расчеты этой конкретной меры по адресу `http://www.tfidf.com/`. Ниже приведен пример того, как рассчитать TF-IDF с использованием Python.

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import *

categories = ['comp.graphics', 'misc.forsale', 'rec.autos', 'sci.space']
twenty_train = fetch_20newsgroups(subset='train',
                                  categories=categories,
                                  shuffle=True,
                                  random_state=42)

count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(
    twenty_train.data)

tfidf = TfidfTransformer().fit(X_train_counts)
X_train_tfidf = tfidf.transform(X_train_counts)
```

```
caltech_idx = count_vect.vocabulary_['caltech']
print('"Caltech" scored in a BOW:')
print('count: %0.3f' % X_train_counts[0, caltech_idx])
print('TF-IDF: %0.3f' % X_train_tfidf[0, caltech_idx])
```

Этот пример начинается так же, как и другие примеры в этом разделе: с получения набора данных из 20 групп новостей. Затем создается набор слов, очень похожий на представленный в примере в разделе "Понятие модели "набор слов" ранее в этой главе. Тем не менее теперь вы видите то, что можете сделать с набором слов.

В данном случае код вызывает функцию `TfidfTransformer()` для преобразования необработанных документов группы новостей в матрицу функций TF-IDF. `Use_idf` контролирует перевешивание инверсной частоты в документе, которое он включил в данном случае. Векторизованные данные подбираются к алгоритму преобразования. Следующий шаг, вызов `tfidf.transform()`, выполняет фактический процесс преобразования. Вот результат, который вы получите из этого примера:

```
"Caltech" scored in a BOW:
count: 3.000
TF-IDF: 0.123
```

Обратите внимание, что слово *Caltech* теперь имеет меньшее значение в первом документе по сравнению с примером в предыдущем абзаце, где подсчет вхождений для одного и того же слова в том же документе получил значение 3. Чтобы понять, как подсчет вхождений относится к TF-IDF, вычислите среднее количество слов и средний TF-IDF:

```
import numpy as np
count = np.mean(X_train_counts[X_train_counts>0])
tfif = np.mean(X_train_tfidf[X_train_tfidf>0])
print('mean count: %0.3f' % np.mean(count))
print('mean TF-IDF: %0.3f' % np.mean(tfif))
```

Результаты свидетельствуют, что, независимо от того, как вы подсчитываете вхождения слова *Caltech* в первом документе или используете его TF-IDF, значение всегда вдвое больше среднего слова, а значит это ключевое слово для моделирования текста:

```
mean count: 1.698
mean TF-IDF: 0.064
```

ЗАПОМНИ!

TF-IDF помогает найти наиболее важные слова или *n*-граммы и исключить наименее важные из них. Это также очень полезно в качестве входных данных для линейных моделей, поскольку они лучше работают с показателями TF-IDF, чем с количеством слов. На этом этапе вы обычно обучаете классификатор и выполняете различные

виды анализа. Пока не беспокойтесь об этой следующей части процесса. В главах 12 и 15 вы познакомитесь с классификаторами, а в главе 17 начнете работать с ними.

Работа с данными графов

Представьте себе точки данных, которые связаны с другими точками данных, например, как одна веб-страница связана с другой веб-страницей с помощью гиперссылок. Каждая из этих точек данных является *узлом* (node). Узлы соединяются друг с другом *связями* (link). Но не каждый узел связывается с каждым другим узлом, поэтому соединения узлов становятся важными. Анализируя узлы и их связи, вы можете решать всевозможные интересные задачи науки о данных, такие как определение наилучшего маршрута от работы до дома. В следующих разделах описывается, как работают графы и как выполнять с ними простые задачи.

Понятие матрицы смежности

Матрица смежности (adjacency matrix) представляет связи между узлами графа. Когда существует соединение между двумя узлами, матрица указывает его как значение, большее нуля. Точное представление соединений в матрице зависит от того, направлен ли граф (где имеет значение направление соединения) или не направлен.

Проблема многих сетевых примеров заключается в том, что авторы делают их слишком упрощают. Однако реальные графы зачастую бывают огромными и не поддаются простому анализу в ходе визуализации. Подумайте только о количестве узлов, которое может иметь даже небольшой город с учетом перекрестков улиц (причем связи — это его улицы). Многие другие графы намного больше, и простой взгляд на них никогда не покажет каких-либо интересных закономерностей. Специалисты по данным называют проблему представления любого сложного графа с помощью матрицы смежности *комком шерсти* (hairball).

Одним из ключей к анализу матриц смежности является их сортировка определенным образом. Например, вы можете выбрать сортировку данных по свойствам, отличным от фактических соединений. Граф соединений улиц может включать в себя дату, когда улица была в последний раз вымощена, что позволяет искать шаблоны, которые направляют кого-то на основе улиц, которые находятся в лучшем состоянии с точки зрения ремонта. Короче говоря, для того, чтобы сделать данные графов полезными, нужно манипулировать организацией этих данных особым образом.

Использование основ NetworkX

Работа с графами может стать сложной, если вам придется писать весь код с нуля. К счастью, пакет NetworkX для Python позволяет легко создавать, управлять и изучать структуру, динамику и функции сложных сетей (или графов). Несмотря на то что эта книга охватывает только графы, вы также можете использовать этот пакет для работы с орграфами и мультиграфами.

Основное внимание в пакете NetworkX уделяется всем проблемам с "комком шерсти". Использование простых вызовов скрывает бо́льшую часть сложности работы с графами и матрицами смежности. В следующем примере показано, как создать простую матрицу смежности из одного из графов, предоставляемых NetworkX:

```
import networkx as nx

G = nx.cycle_graph(10)
A = nx.adjacency_matrix(G)

print(A.todense())
```

Пример начинается с импорта необходимого пакета. Затем создается граф с использованием шаблона `cycle_graph()`. Граф содержит десять узлов. Вызов `adjacency_matrix()` создает матрицу смежности из графа. Последний шаг — отобразить вывод в виде матрицы, как показано ниже.

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

Для проверки вам не нужно создавать собственный граф с нуля. Сайт NetworkX документирует ряд стандартных типов графов, которые вы можете использовать, и все они доступны в IPython. Список отображается по адресу `https://networkx.github.io/documentation/latest/reference/generators.html`.

Интересно посмотреть, как выглядит граф после его генерации. Следующий код отображает граф (рис. 8.1):

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
nx.draw_networkx(G)
plt.show()
```

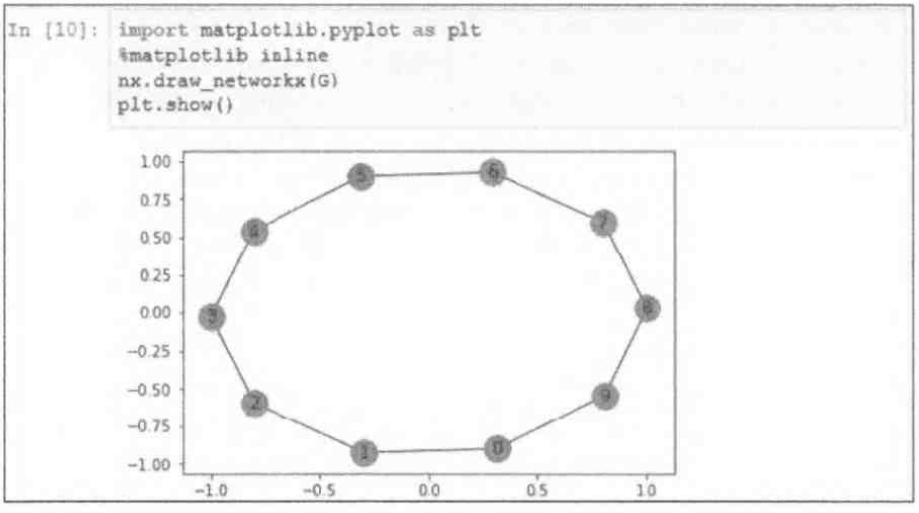


Рис. 8.1. Построение простого графа

На графе показано, что вы можете добавить ребро между узлами 1 и 5. Вот код, необходимый для выполнения этой задачи с помощью функции `add_edge()` (рис. 8.2):

```
G.add_edge(1,5)
nx.draw_networkx(G)
plt.show()
```

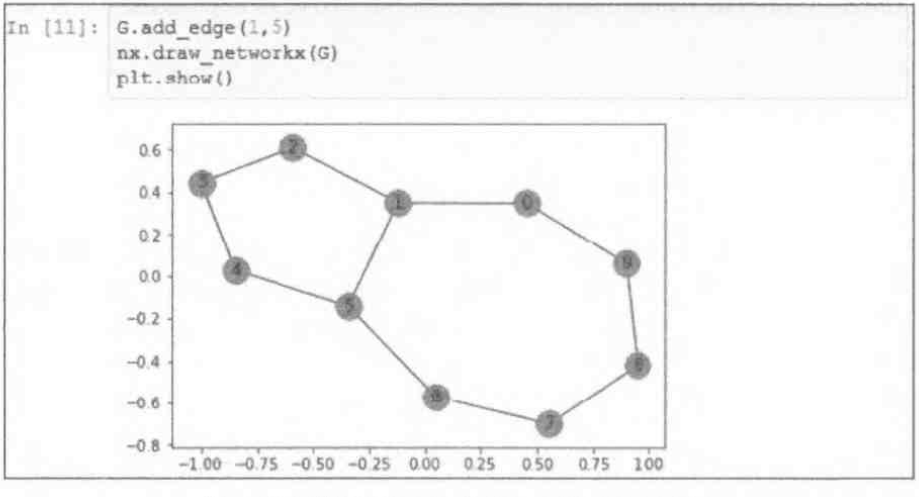


Рис. 8.2. Построение дополненного графа

Глава 9

Применение знаний на практике

В ЭТОЙ ГЛАВЕ...

- Представление задач науки о данных и данных в перспективе
- Создание и использование признаков
- Работа с массивами

Предыдущие главы носили подготовительный характер. Вы узнали, как решать важные задачи науки о данных с использованием языка Python. Кроме того, вы уделили время работе с различными инструментами, которые предоставляет Python, чтобы упростить задачи по обработке данных. Вся эта информация важна, но она не поможет вам увидеть общую картину. В этой главе показано, как использовать методы, которые вы изучали в предыдущих главах, для решения реальных задач науки о данных.


ЗАПОМНИ!

Эта глава не конец пути — это его начало. Подумайте о предыдущих главах так же, как вы думаете об упаковке ваших вещей, бронировании и создании маршрута перед поездкой. Эта глава — поездка в аэропорт, во время которой вы начинаете видеть, что все собралось вместе.

Глава начинается с рассмотрения аспектов, которые вы обычно должны учитывать при попытке решить задачу науки о данных. Вы не можете просто начать выполнять анализ; сначала вы должны уяснить задачу, а также рассмотреть ресурсы (в виде данных, алгоритмов, вычислительных ресурсов) для

ее решения. Помещение задачи в контекст, настройка своего рода, помогает понять задачу и определить, как к ней относятся данные. Контекст важен, поскольку, как и язык, он изменяет смысл как задачи, так и связанных с ней данных. Например, когда вы говорите своему другу: "У меня есть красная роза", значение, стоящее за предложением, имеет одну коннотацию. Если вы скажете то же самое садовнику, значение будет другим. Красная роза — это своего рода данные, а человек, с которым вы разговариваете, — это контекст. Нет смысла говорить: "У меня есть красная роза", если вы не знаете контекст, в котором делается это заявление. Аналогично данные не имеют смысла, они не отвечают ни на один вопрос, пока вы не узнаете контекст, в котором используются данные. Слова "у меня есть данные" вызывают вопрос: "Что означают данные?"

В конце концов, понадобится один или несколько наборов данных. Двумерные таблицы данных (наборы данных) состоят из *случаев* (cases) или наблюдений (это строки) и *признаков* (features) (это столбцы). Вы также можете упоминать признаки как *переменные* (variables) при использовании статистической терминологии. Признаки, которые вы решите использовать для любого заданного набора данных, определяют виды анализа, которые вы можете выполнять, способы, с помощью которых можете манипулировать данными, и в конечном итоге виды получаемых вами результатов. Определение того, какие признаки вы можете создать из исходных данных и как вы должны преобразовать данные, чтобы убедиться, что они пригодны для анализа, который хотите выполнить, является важной частью выработки решения для науки о данных.

Получив представление о задаче, о ресурсах, которые вам нужны для ее решения, вы готовы выполнить какую-то реальную работу. В последнем разделе этой главы показано, как эффективно решать простые задачи. Обычно вы можете решать задачи, используя более одной методологии, но при работе с большими данными, чем быстрее, тем лучше. Работая с массивами и матрицами для выполнения конкретных задач, вы заметите, что некоторые операции могут занять много времени, если вы не используете специальные вычислительные приемы.

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_09_Operations_On_Arrays_and_Matrices.ipynb`.

Помещение в контекст задач и данных

Помещение задачи в правильный контекст является важной частью выработки решения науки о данных для любой конкретной задачи и связанных

с ней данных. Наука о данных — это, безусловно, прикладная наука, и абстрактные ручные подходы могут не очень хорошо работать в конкретной ситуации. Запуск кластера Hadoop или построение глубокой нейронной сети может показаться вашим коллегам выдающимся достижением и заставить вас почувствовать, что вы двигаете великие проекты науки о данных, но они могут не обеспечить того, что нужно для решения вашей задачи. Постановка задачи в правильном контексте — это не просто вопрос об использовании определенного алгоритма или необходимости преобразования данных определенным образом, это искусство критического изучения задачи и доступных ресурсов, а также создание среды для решения задачи и получения желаемого решения.

Ключевым моментом здесь является *желаемое* решение, вы могли бы найти решения, которые нежелательны, поскольку они не говорят о том, что вам нужно узнать, или даже когда оно действительно говорит то, что вам нужно знать, на это уходит слишком много времени и ресурсов. В следующих разделах описан процесс помещения в контекст как задач, так и данных.

Оценка задачи науки о данных

Работая с задачей науки о данных, вы должны начать с рассмотрения своей цели и ресурсов, которые у вас есть для ее достижения. Ресурсы — это данные, вычислительные ресурсы, такие как доступная память, процессоры и дисковое пространство. В реальном мире никто не передаст вам готовые данные и не скажет, как выполнять определенный анализ этих данных. В большинстве случаев вам придется сталкиваться с совершенно новыми задачами, и вы должны строить свое решение с нуля. Во время первой оценки задачи науки о данных необходимо учесть следующее.

- **Имеющиеся данные с точки зрения доступности, количества и качества.** Вы должны рассматривать данные с точки зрения возможных отклонений, способных повлиять или даже исказить их характеристики и содержимое. Данные никогда не содержат абсолютных истин, только относительные истины, которые дают более или менее полное представление о задаче (подробнее об этом — ниже, в разделе "УЧЕТ ПЯТИ НЕДОСТОВЕРНОСТЕЙ В ДАННЫХ"). Всегда помните о достоверности данных и применяйте критические рассуждения в рамках своего анализа.
- **Методы, которые можно использовать для анализа набора данных.** Подумайте, являются ли методы простыми или сложными. Вы также должны решить, насколько хорошо знаете конкретную методологию. Начните с использования простых подходов и никогда не

останавливайтесь на какой-либо конкретной технике. В науке о данных нет ни бесплатных обедов, ни Святого Грааля.

» **Вопросы, на которые вы хотите ответить, выполнив анализ, и как вы можете количественно измерить, достигнут ли удовлетворительный ответ**. "Если вы не можете это измерить, то не можете это улучшить," — заявил лорд Кельвин (см. `https://zapatopi.net/kelvin/quotes/`). Если вы можете измерить производительность, то можете определить влияние вашей работы и даже сделать ее денежную оценку. Заинтересованные стороны будут рады узнать, что вы выяснили, что делать и какие преимущества принесет ваш проект по науке о данных.

УЧЕТ ПЯТИ НЕДОСТОВЕРНОСТЕЙ В ДАННЫХ

Люди привыкли видеть данные такими, какими они есть во многих случаях. На самом деле в некоторых случаях люди искажают данные до такой степени, что они становятся бесполезными, — это *недостоверность* (mistruth). Компьютер не может определить разницу между достоверными и недостоверными данными; все, что он видит, — это данные. Следовательно, когда вы выполняете анализ данных, вы должны учитывать их истинную ценность как часть вашего анализа. Лучшее, на что вы можете надеяться, — это увидеть ошибочные данные как выбросы, а затем отфильтровать их, но этот метод не обязательно решит проблему, поскольку человек все равно будет использовать данные и пытаться определить правду, основываясь на содержащейся в них неправде. Ниже приведены пять недостоверностей, которые обычно находят в данных (в качестве иллюстрации используем процесс составления протокола о дорожном происшествии).

- **Усердие** (commission). Недостоверность усердия отражает прямую попытку заменить правдивую информацию неправдивой. Например, при заполнении протокола о происшествии кто-то мог утверждать, что солнце на мгновение ослепило его, не позволив увидеть того, кого сбили. В действительности, возможно, человек был отвлечен чем-то другим или вообще не думал о вождении (возможно, вспоминал хороший ужин). Если никто не сможет опровергнуть эту теорию, человек может обойтись меньшим наказанием. Но дело в том, что данные будут искажены.
- **Умолчание** (omission). Недостоверность умолчания возникает, когда человек говорит правду в каждом установленном факте, но упускает важный факт, который меняет восприятие инцидента в целом. Предположим, что некто сбил оленя, нанеся значительный ущерб своей машине. Он правдиво говорит, что дорога была мокрой, были уже почти сумерки, поэтому света была недостаточно; он немного опоздал нажать на тормоз, а олень просто выбежал из зарослей на обочине дороги. Можно сделать вывод, что инци-

дент — просто случайность. Тем не менее человек упустил важный факт. В то время он переписывался. Если бы правоохранительные органы знали о текстовых сообщениях, это изменило бы причину аварии на невнимательное вождение. Водитель мог бы быть оштрафован, а страховой агент внести другую причину происшествия в базу данных.

- **Точка зрения** (perspective). Недостоверность точки зрения — это когда несколько сторон рассматривают инцидент с нескольких точек зрения. Например, при рассмотрении несчастного случая с участием пострадавшего пешехода, человека, управляющего автомобилем, сбившим человека, и свидетеля этого события. Офицер, составляющий протокол, получит от каждого человека, по понятным причинам, разные факты, даже если предположить, что каждый из них говорит правду, поскольку каждый знает ее. Фактически опыт показывает, что это почти всегда так, и то, что офицер заносит в протокол, является средним уровнем от того, что заявляет каждый из участников, дополненное его личным опытом. Другими словами, протокол будет близок к истине, но не полностью правдив. Чтобы справиться с недостоверностью точки зрения, важно учитывать позицию наблюдателя. Водитель машины видит приборную панель и знает состояние автомобиля на момент аварии. Это информация, которой нет у двух других сторон. Точно так же человек, которого сбила машина, лучше всего видит выражение лица (намерения) водителя. Посторонний может оказаться в лучшем положении, чтобы увидеть, предпринял ли водитель попытку остановиться, и оценить такие обстоятельства, как возможность водителя свернуть. Каждая сторона должна будет составить отчет на основе только увиденных данных без сокрытых данных.
- **Предубежденность** (bias). Недостоверность предубежденности возникает, когда некто способен увидеть правду, но личные опасения или убеждения искажают увиденное. Например, в случае дорожного происшествия водитель может полностью сосредоточить внимание на середине дороги так, что олени на обочине стали для него практически невидимыми. Следовательно, водитель не успевает среагировать, когда олень внезапно решает выскочить на середину дороги, пытаясь пересечь ее. Проблема с предубежденностью заключается в том, что ее может быть невероятно сложно классифицировать. Например, водитель, который не увидел оленя, мог попасть в настоящий несчастный случай, поскольку олень был скрыт от глаз кустарником. Тем не менее водитель мог быть виновен в невнимательном вождении. Водитель мог также испытать кратковременное отвлечение. Короче говоря, тот факт, что водитель не видел оленей, не является вопросом; вопрос в том, почему водитель не видел оленя. Во многих случаях выявление источника предубежденности становится важным при создании алгоритма, предназначенного для его исключения.
- **Недопонимание** (frame of reference). Из пяти недостоверностей эта не обязательно должна быть результатом какой-либо ошибки, а скорее является

проблемой понимания. Недостоверность недопонимания возникает, когда одна сторона описывает нечто, например, событие, подобное несчастному случаю, а отсутствие опыта у другой стороны об этом событии запутывает детали или делает все абсолютно непонятыми. На недостоверности недопонимания основаны многие комедийные сюжеты. Один из известных примеров — это телепостановка *Кто на первой базе?* (Who's On First?) от Эбботта и Костелло, доступная по адресу `https://www.youtube.com/watch?v=kTcRRaXV-fg`. Заставить одного человека понять то, что говорит другой, может быть невозможно, если у первого человека отсутствуют соответствующие знания.

Исследовательские решения

Наука о данных — это сложная система знаний на стыке информатики, математики, статистики и бизнеса. Очень немногие люди могут знать об этом все, и, если кто-то уже сталкивался с той же проблемой или дилеммами, с которыми сталкиваетесь вы, повторное изобретение колеса имеет мало смысла. Теперь, когда вы нашли контекст своего проекта, вы знаете, что ищете, но искать можно по-разному.

- **Просмотрите документацию Python**. Возможно, вы найдете примеры, которые предлагают возможное решение. Библиотеки NumPy (`https://docs.scipy.org/doc/numpy/user/`), SciPy (`https://docs.scipy.org/doc/`), pandas (`https://pandas.pydata.org/pandas-docs/version/0.23.2/`) и особенно Scikit-learn (`https://scikitlearn.org/stable/user_guide.html`) имеют подробную встроенную и сетевую документацию с большим количеством примеров, связанных с наукой о данных.
- **Ищите сетевые статьи и блоги, которые намекают на то, как другие специалисты решали подобные задачи**. Веб-сайты вопросов и ответов, такие как Quora (`https://www.quora.com/`), Stack Overflow (`https://stackoverflow.com/`) и Cross Validated (`https://stats.stackexchange.com/`), могут предоставить множество ответов на подобные вопросы.
- **Обратитесь к академическим работам**. Например, вы можете запросить вашу проблему в Google Scholar по адресу `https://scholar.google.it/` или в Microsoft Academic Search по адресу `https://academic.microsoft.com/`. Вы можете найти серию научных работ, способных рассказать о подготовке данных или подробно описать алгоритмы, которые лучше подходят для конкретной задачи.

ЗАПОМНИ!

Это может показаться тривиальным, но решения, которые вы создаете, должны отражать задачу, которую вы пытаетесь решить. Исследуя решения, вы можете обнаружить, что некоторые из них поначалу кажутся многообещающими, но затем вы не можете успешно применить их к вашей задаче, поскольку что-то в их контексте отличается. Например, ваш набор данных может быть неполным или не предоставлять достаточно информации для решения задачи. Кроме того, выбранная вами модель анализа может фактически не дать нужного ответа, или ответ может оказаться неточным. Когда вы решаете задачу, не бойтесь проводить исследования несколько раз, поскольку вы ищете, проверяете и оцениваете возможные решения, которые могли бы применить, учитывая имеющиеся ресурсы и ваши фактические ограничения.

Формулировка гипотезы

В какой-то момент у вас есть все, что, на ваш взгляд, вам нужно для решения задачи. Конечно, будет ошибкой полагать, что решения, которые вы создаете, действительно могут решить задачу. У вас есть гипотеза, а не решение, поскольку вы должны продемонстрировать эффективность потенциального решения научным способом. Чтобы сформировать и проверить гипотезу, вы должны обучить модель с использованием обучающего набора данных, а затем протестировать ее с использованием совершенно другого набора данных. В последующих главах много времени отводится обучению и тестированию алгоритмов, используемых для анализа, поэтому не беспокойтесь, если сейчас не понимаете этот аспект процесса.

Подготовка данных

После того как у вас появилось представление о задаче и ее решении, вы знаете, какие данные необходимы для работы алгоритма. К сожалению, ваши данные, вероятно, присутствуют в нескольких формах, вы получаете их из нескольких источников, а некоторые данные полностью отсутствуют. Более того, разработчики функций, предоставляемых существующими источниками данных, возможно, разработали их для других целей (например, для бухгалтерского учета или маркетинга), а не для вас, и вам придется трансформировать их так, чтобы вы могли использовать свой алгоритм в полную силу. Чтобы алгоритм работал, вы должны подготовить данные. Это означает проверку отсутствующих данных, создание новых функций по мере необходимости и, возможно, манипулирование набором данных, чтобы привести его в форму, которую ваш алгоритм может фактически использовать для прогнозирования.

Искусство создания признаков

Признаки связаны со столбцами в вашем наборе данных. Конечно, вам нужно определить, что должны содержать эти столбцы. Они могут не выглядеть точно так же, как данные в исходном источнике данных. Исходный источник данных может представлять данные в форме, которая приводит к неточному анализу или даже мешает вам получить желаемый результат, поскольку он не полностью соответствует вашему алгоритму или вашим целям. Например, данные могут содержать слишком много информации в нескольких переменных, что является проблемой, называемой *многомерной корреляцией* (multivariate correlation). Необходимо заставить колонки работать наилучшим образом для анализа данных — это задача *создания признаков* (feature creation) (или *конструирование признаков* (feature engineering)). Следующие разделы помогут вам понять создание признаков и почему оно так важно. (В следующих главах приводятся всевозможные примеры того, как вы на самом деле используете создание признаков для выполнения анализа.)

Определение создания признака

Некоторым создание признаков может показаться чем-то вроде волшебства или странной науки, но на самом деле оно имеет прочную основу в математике. Задача заключается в том, чтобы взять существующие данные и преобразовать их во что-то, с чем вы можете работать, выполняя анализ. Например, числовые данные могут отображаться в виде строк в исходном источнике данных. Чтобы выполнить анализ, во многих случаях необходимо преобразовать строковые данные в числовые значения. Непосредственная цель создания признака заключается в достижении лучшей производительности от алгоритмов, используемых для выполнения анализа, чем при использовании исходных данных.

Во многих случаях преобразование более чем непросто. Возможно, вам придется каким-то образом комбинировать значения или выполнять над ними математические операции. Информация, к которой вы можете получить доступ, способна отображаться во всевозможных формах, а процесс преобразования позволит работать с данными по-новому, чтобы вы могли увидеть в них шаблоны. Рассмотрим, например, популярный конкурс Kaggle: `https://www.kaggle.com/c/march-machine-learning-mania-2015`. Цель заключается в том, чтобы использовать все виды статистики и определить, кто победит в баскетбольном турнире NCAA. Представьте, что вы пытаетесь извлечь несопоставимые показатели из общедоступной информации о матче, такой как географическое местоположение команды, или отсутствие ключевых игроков, и вы возможно начнете понимать необходимость создания признаков в наборе данных.

ЗАПОМНИ!

Как вы можете себе представить, создание признаков действительно является искусством, и у каждого есть мнение, как именно его осуществлять. Эта книга предоставляет хорошую базовую информацию о создании признаков, а также ряд примеров, но передовые методы она оставляет для экспериментов и испытаний. Как сказал профессор Вашингтонского университета Педро Домингос в своей статье по науке о данных "A Few Useful Things to Know about Machine Learning" (см. `https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf`), разработка признаков является "самым важным фактором" в определении успеха или неудачи проекта машинного обучения, и ничто не может заменить "ум, который вы вкладываете в разработку признаков".

Объединение переменных

Данные нередко поступают в форме, которая вообще не подходит для алгоритма. Рассмотрим простую ситуацию из реальной жизни, в которой вам нужно определить, может ли один человек поднять доску на лесопилке. Вы получаете две таблицы данных. Первая содержит высоту, ширину, толщину и типы древесных плит. Вторая содержит список типов древесины и значение их веса на фут доски (кусок дерева 112" × 12" × 1" (2,8448м × 0,3048м × 0,0254м)). Каждый тип древесины имеет различные размеры, и некоторые партии поставляются без маркировки, так что вы не знаете, с каким деревом работаете. Цель заключается в создании прогноза, чтобы компания знала, сколько людей отправлять для работы с поставкой.

В этом случае нужно создать двумерный набор данных, объединив переменные. Полученный набор данных содержит только два признака. Первый признак содержит только длину доски. Разумно ожидать, что один человек будет нести доску длиной до десяти футов (3,048 м), но вы хотите, чтобы два человека несли доску в десять или более футов. Второй признак — вес доски. Доска длиной 10 футов, шириной 12 дюймов (30,48 см) и толщиной 2 дюйма (5,08 см) содержит 20 досочных футов[1] (0,047 м^3). Если доска из желтой сосны (pinus ponderosa) (со значением *досочного фута* (Board Foot Rating — BFR) 2,67), то общий вес составит 53,4 фунта (24,22183 кг), значит, один человек, вероятно, сможет поднять ее. Однако, если доска из гикори (carya) (со значением BFR 4,25), то общий вес составит 85 фунтов (38,5554 кг). Если у вас не работает

[1] Единица измерения пиломатериалов, равная доске, длиной в 1 фут, шириной 12 дюймов и толщиной 1 дюйм или их кубическим эквивалентам. — *Примеч. ред.*

Халк[2], вам действительно нужны два человека для подъема этой доски, хотя доска достаточно короткая, чтобы ее, казалось бы, мог поднять один человек.

Получить первый признак для вашего набора данных легко. Все, что вам нужно, — это длина каждой из досок, которые вы получаете. Однако второй признак требует объединения переменных из обеих таблиц:

```
Length (feet) * Width (feet) * Thickness (inches) * BFR
```

Полученный набор данных будет содержать вес для каждой длины каждого вида древесины, которую вы запасаете. Наличие этой информации означает, что вы можете создать модель, прогнозирующую количество человек, необходимых для выполнения конкретной задачи: один, два или даже три.

Понятие группирования и дискретизации

Чтобы выполнять некоторые виды анализа, нужно разделить числовые значения на классы. Например, есть набор данных, который включает записи для людей в возрасте от 0 до 80 лет. Чтобы получить статистику, которая работает в этом случае (например, запустить алгоритм наивного байесовского классификатора (Naïve Bayes)), вы можете захотеть просмотреть переменную в виде последовательности уровней с десятилетним приращением. Процесс разделения набора данных на эти десятилетние приращения является *группированием* (binning). Каждая группа представляет собой числовую категорию, которую вы можете использовать.

Группирование может повысить точность прогнозирующих моделей за счет снижения шума или за счет помощи в нелинейности. Кроме того, оно позволяет легко идентифицировать *выбросы* (outlier) (значения вне ожидаемого диапазона) и недопустимые или отсутствующие значения числовых переменных.

Группирование работает исключительно с единичными числовыми признаками. *Дискретизация* (discretization) — это более сложный процесс, когда комбинации значений из разных признаков помещают в сегмент, ограничивая количество состояний в каждом конкретном блоке. В отличие от группирования, дискретизация работает как с числовыми, так и со строковыми значениями. Это более обобщенный метод создания категорий. Например, вы можете получить дискретизацию как побочный продукт кластерного анализа.

Использование индикаторных переменных

Индикаторные переменные (indicator variable) — это признаки, способные принимать значение 0 или 1. Другое название индикаторных переменных — *фиктивные переменные* (dummy variable). Независимо от того, как вы

[2] Вымышленный персонаж, супергерой комиксов. — *Примеч. ред.*

их называете, эти переменные служат важной цели — облегчению работы с данными. Например, если вы хотите создать набор данных, в котором лица в возрасте до 25 лет обрабатываются одним способом, а лица в возрасте 25 лет и старше — другим, можно заменить признак возраста индикаторной переменной, содержащей значение 0, если лицо моложе 25 лет, или 1, когда человеку 25 лет и более.


СОВЕТ

Использование индикаторной переменной позволяет выполнять анализ быстрее и классифицировать случаи с большей точностью, чем без нее. Индикаторная переменная удаляет оттенки серого из набора данных. Кто-то моложе 25 лет, кто-то — нет, среднего не дано. Поскольку данные упрощены, алгоритм может выполнять свою задачу быстрее, и вам придется бороться с меньшей неопределенностью.

Преобразование распределений

Распределение (distribution) — это расположение значений переменной, которое показывает частоту, с которой встречаются различные значения. Узнав, как распределяются значения, вы лучше поймете данные. Существуют разные виды распределений (см. галерею распределений на `https://www.itl.nist.gov/div898/handbook/eda/section3/eda366.htm`), и большинство алгоритмов могут легко справиться с ними. Но вы должны сопоставить алгоритм с распределением.


ВНИМАНИЕ!

Обратите особое внимание на равномерное (uniform) и неравномерное (skewed) распределения. С ними довольно сложно иметь дело по разным причинам. Но колоколообразная кривая нормального распределения — это всегда хорошо. Когда вы видите распределение, отличающееся от колоколообразного, вам следует подумать о преобразовании.

При работе с распределениями вы можете обнаружить, что распределение значений каким-то образом искажено и что из-за перекоса значений любой алгоритм, примененный к набору значений, создает выходные данные, которые не будут соответствовать вашим ожиданиям. Преобразование распределения означает применение некоторой функции к значениям для достижения конкретных целей, таких как исправление перекоса данных, чтобы вывод вашего алгоритма был ближе к ожидаемому. Кроме того, преобразование помогает сделать распределение более дружественным, например, когда вы преобразуете набор данных в нормальное распределение. Ниже приведены преобразования, о возможности применения которых к своим числовым признакам вы всегда должны помнить.

- Логарифмическое `np.log(x)` и экспоненциальное `np.exp(x)`.
- Обратное `1/x`, квадратный корень `np.sqrt(x)` и кубический корень `x**(1.0/3.0)`.
- Полиномиальные преобразования, такие как `x**2`, `x**3` и т.д.

Операции над массивами

Основной формой манипулирования данными является их помещение в массив или матрицу и последующее использование стандартных математических методов для изменения их формы. Используя этот подход, вы помещаете данные в форму, удобную для выполнения других операций, осуществляемых на уровне каждого отдельного наблюдения, например, в итерациях, поскольку они могут использовать архитектуру вашего компьютера и некоторые высокооптимизированные процедуры числовой линейной алгебры, присутствующие в процессорах. Эти подпрограммы могут вызываться из любой операционной системы. Чем больше данных и вычислений, тем больше времени вы можете сэкономить. Кроме того, использование этих методов избавит вас также от написания длинного и сложного кода Python. В следующих разделах описано, как работать с массивами для задач науки о данных.

Использование векторизации

Компьютер предоставляет мощные подпрограммы для вычислений, и вы можете использовать их, если данные в нужном формате. Библиотека NumPy предоставляет `ndarray` — многомерную структуру хранения данных, которую можно использовать в качестве многомерной таблицы данных. Фактически вы можете использовать ее в качестве куба или даже гиперкуба, если существует более трех измерений.

Использование структуры `ndarray` упрощает и ускоряет вычисления. В следующем примере создается набор данных из трех наблюдений с семью признаками для каждого. В этом случае пример получает максимальное значение для каждого наблюдения и вычитает его из минимального значения, чтобы получить диапазон значений для каждого наблюдения.

```
import numpy as np
dataset = np.array([[2, 4, 6, 8, 3, 2, 5],
                    [7, 5, 3, 1, 6, 8, 0],
                    [1, 3, 2, 1, 0, 0, 8]])
print(np.max(dataset, axis=1) - np.min(dataset, axis=1))
```

Оператор `print` получает максимальное значение из каждого наблюдения, используя метод `np.max()`, а затем вычитает его из минимального значения, используя метод `np.min()`. Максимальное значение в каждом наблюдении составляет `[8 8 8]`, минимальное значение составляет `[2 0 0]`. В результате вы получите следующий вывод:

```
[6 8 8]
```

Простые арифметические действия с векторами и матрицами

Большинство операций и функций библиотеки NumPy, которые применяются к массивам, используют векторизацию, поэтому они быстры и эффективны — гораздо более эффективны, чем любое другое решение или код, созданный самостоятельно. Даже самые простые операции, такие как сложение или деление, могут использовать векторизацию.

Например, во многих случаях форма данных в наборе данных не совсем соответствует той, которая вам нужна. Список чисел может представлять проценты в виде целых чисел, когда они вам действительно нужны в виде дробных значений. В этом случае для решения задачи обычно выполняют некоторую простую математическую обработку:

```
import numpy as np
a = np.array([15.0, 20.0, 22.0, 75.0, 40.0, 35.0])
a = a*.01
print(a)
```

В этом примере создается массив, заполняется целочисленными процентами, а затем, используя значение 0,01 в качестве множителя, создает дробные проценты. Теперь можно умножить эти дробные значения на другие числа и определить, как процент влияет на это число. Вывод этого примера таков:

```
[0.15 0.2 0.22 0.75 0.4 0.35]
```

Матричное векторное умножение

Наиболее эффективными операциями векторизации являются матричные манипуляции, в которых вы добавляете и умножаете несколько значений на несколько других значений. Библиотека NumPy упрощает выполнение умножения вектора на матрицу, что весьма удобно, если вам необходимо оценить значение для каждого наблюдения в виде взвешенной суммы признаков. Вот пример этой техники:

```
import numpy as np
a = np.array([2, 4, 6, 8])
b = np.array([[1, 2, 3, 4],
```

```
                [2, 3, 4, 5],
                [3, 4, 5, 6],
                [4, 5, 6, 7]])
c = np.dot(a, b)
print(c)
```

Обратите внимание, что при умножении массив, отформатированный как вектор, должен следовать перед массивом, отформатированным как матрица, иначе произойдет ошибка. Пример выводит следующие значения:

```
[60 80 100 120]
```

Чтобы получить показанные значения, следует умножить каждое значение в массиве (`array`) на соответствующий столбец (`column`) в матрице, т.е. вы умножаете первое значение в массиве на первый столбец, первую строку матрицы (`matrix`). Например, первое значение в выводе составит 2 * 1 + 4 * 2 + 6 * 3 + 8 * 4, что равно 60.

Умножение матриц

Также можно умножить одну матрицу на другую. В этом случае выходные данные являются результатом умножения строк в первой матрице на столбцы во второй матрице. Ниже приведен пример умножения одной матрицы NumPy на другую:

```
import numpy as np

a = np.array([[2, 4, 6, 8],
              [1, 3, 5, 7]])
b = np.array ([[1, 2],
               [2, 3],
               [3, 4],
               [4, 5]])
c = np.dot(a, b)
print(c)
```

В качестве вывода в данном случае будет получена матрица 2 × 2. Вот значения, которые вы должны увидеть при запуске приложения:

```
[[60 80]
 [50 66]]
```

Каждая строка в первой матрице умножается на каждый столбец второй матрицы. Например, чтобы получить значение 50, показанное в столбце 1 строки 2 выходных данных, нужно сопоставить значения во второй строке матрицы `a` со столбцом 1 матрицы `b`, например: 1 * 1 + 3 * 2 + 5 * 3 + 7 * 4.

3 Визуализация информации

В ЭТОЙ ЧАСТИ...

- Установка и использование MatPlotLib
- Определение частей графического вывода
- Создание и использование различных видов презентаций данных
- Работа с географическими данными

Глава 10

Ускоренный курс по MatPlotLib

В ЭТОЙ ГЛАВЕ...

- Создание простого графика
- Добавление шкал на график
- Украшение графика стилями и цветом
- Документирование графика метками, аннотациями и легендами

Большинство людей лучше воспринимают информацию, когда видят ее в графическом виде, а не в текстовом. Графика помогает видеть отношения и проводить сравнения с большей легкостью. Даже если вы с легкостью справитесь с абстракцией текстовых данных, весь результат анализа данных связан с коммуникацией. Если вы не можете донести свои идеи до других людей, процесс получения, формирования и анализа данных не имеет особого значения, кроме как для ваших личных потребностей. К счастью, Python делает задачу преобразования ваших текстовых данных в графику относительно простой, благодаря библиотеке MatPlotLib, являющейся фактически симуляцией приложения MATLAB. Сравнение этих двух библиотек приведено по адресу `https://pyzo.org/python_vs_matlab.html`.

СОВЕТ

Если вы уже знаете, как использовать приложение MATLAB (в противном случае, если хотите научиться этому, см. книгу *MATLAB For Dummies*, Джона Пола Мюллера (издательство Wiley)), переход к MatPlotLib относительно прост, поскольку оба они используют для

выполнения своей задачи одинаковую машину состояний, и у них есть аналогичный метод определения графических элементов. Многие считают, что MatPlotLib превосходит MATLAB, поскольку вы можете выполнять такие же задачи, используя меньше кода при работе с MatPlotLib, чем при использовании MATLAB (см. `https://phillipmfeldman.org/Python/Advantages_of_Python_Over_Matlab.html`). Другие отметили, что переход с MATLAB на MatPlotLib относительно прост (см. `https://vnoel.wordpress.com/2008/05/03/bye-matlab-hellopython-thanks-sage/`). Однако самое главное то, что думаете вы. Если вам нравится экспериментировать с данными, используя MATLAB, а затем создавать приложения на основе ваших результатов, используя Python и MatPlotLib, то это вопрос только вкуса, а не абсолютная истина.

В этой главе описано, как быстро освоить библиотеку MatPlotLib. Вы будете использовать библиотеку MatPlotLib несколько раз далее в книге, поэтому данный краткий обзор очень важен, даже если вы уже знаете, как работать с MATLAB. Обязательно вернитесь к этой главе, когда начнете работать с MatPlotLib далее в этой книге.

СОВЕТ

Вам не нужно вводить исходный код этой главы вручную — вы используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_10_Getting_a_Crash_Course_in_MatPlotLib.ipynb`.

Начнем с графика

График или диаграмма — это просто визуальное представление числовых данных. Библиотека MatPlotLib предлагает большое количество типов графиков и диаграмм, и вы можете выбрать любой из распространенных типов графиков, таких как гистограммы, линейные графики или круговые диаграммы. Как и в случае с MATLAB, у вас также есть доступ к огромному количеству графиков статистических типов, таких как коробчатая диаграмма, диаграмма погрешностей и гистограмма. Галерея различных типов графиков, поддерживаемых MatPlotLib, представлена по адресу `https://matplotlib.org/gallery.html`. Но следует помнить, что вы можете комбинировать графические элементы практически бесконечным количеством способов для создания собственного представления данных, независимо от того, насколько сложными могут быть эти данные. В следующих разделах описывается, как создать

простой график, но у вас есть доступ к гораздо большему количеству функций, чем вписано в этих разделах.

Определение сюжета графика

Графики изображают то, что вы определили численно. Чтобы определить график, нужны некоторые значения, модуль `matplotlib.pyplot` и представление о том, что именно вы хотите отобразить, как показано в следующем коде:

```
import matplotlib.pyplot as plt
%matplotlib inline

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
plt.plot(range(1,11), values)
plt.show()
```

В данном случае код указывает функции `plt.plot()` создать график, используя значения по оси x от 1 до 11 и значения по оси y, которые отображаются в значениях. Вызов функции `plot.show()` отображает график в отдельном диалоговом окне, как показано на рис. 10.1. Обратите внимание, что вывод представляет собой линейный график. В главе 11 показано, как создавать другие типы диаграмм и графиков.

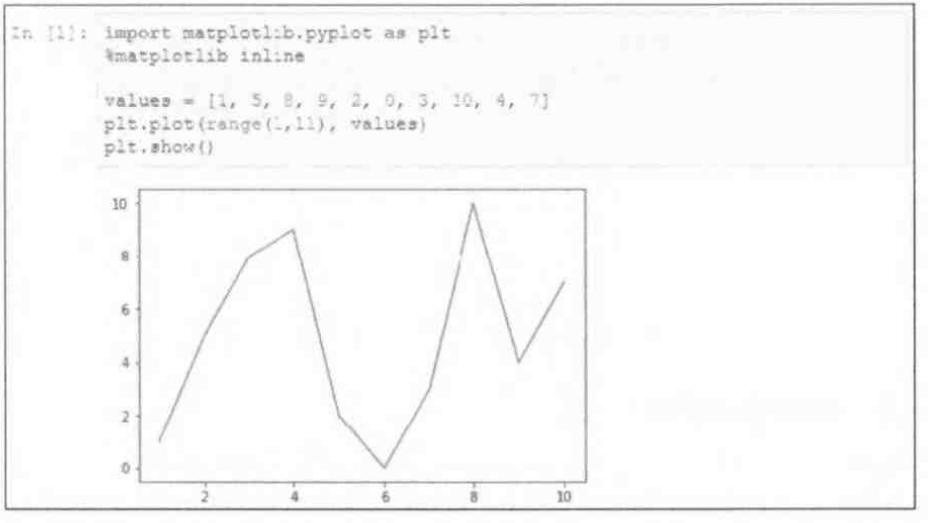


Рис. 10.1. Создание простого графика, демонстрирующего только одну линию

Рисование нескольких линий и графиков

Во многих ситуациях требуется использовать несколько линий графика, например, при сравнении двух наборов значений. Чтобы создать такие графики с помощью библиотеки MatPlotLib, следует вызвать функцию `plt.plot()`

несколько раз — по одному разу для каждой линии графика, как показано в следующем примере:

```
import matplotlib.pyplot as plt
%matplotlib inline

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
plt.plot(range(1,11), values)
plt.plot(range(1,11), values2)
plt.show()
```

Запустив этот пример, вы увидите две линии на графике (рис. 10.2). Даже если вы не видите этого в печатной книге, линейные графики имеют разные цвета, так что вы можете различить их.

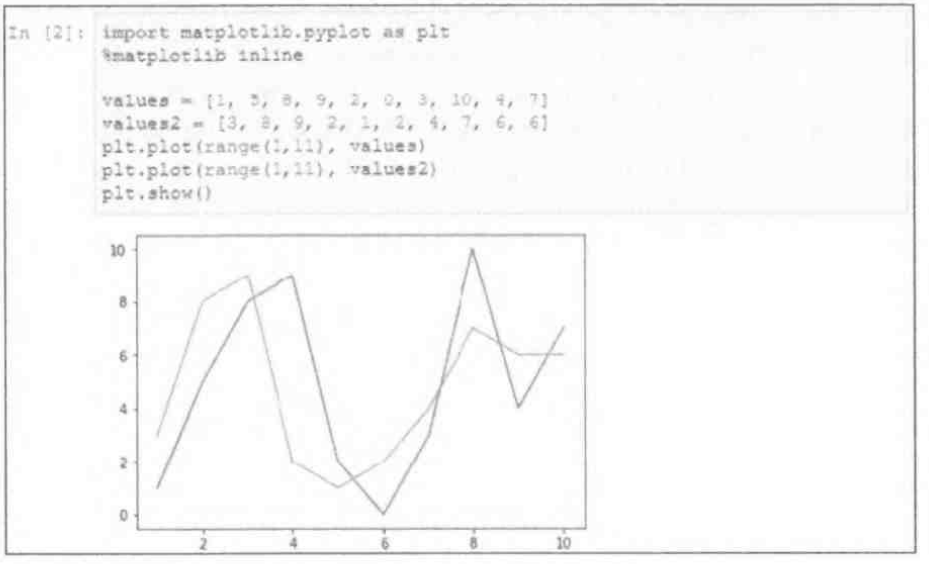


Рис. 10.2. Создание графика с несколькими линиями

Сохранение работы на диске

Jupyter Notebook позволяет легко включать графики в ячейки, которые вы создаете, так что можно создавать отчеты, легко понятные каждому. Если вам необходимо сохранить копию своей работы на диске для последующего использования или использовать ее в составе большого отчета, сохраните график программно с помощью функции `plt.savefig()`, как показано в следующем коде:

```
import matplotlib.pyplot as plt
%matplotlib auto

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
```

```
plt.plot(range(1,11), values)
plt.ioff()
plt.savefig('MySamplePlot.png', format='png')
```

В данном случае вы должны предоставить как минимум два ввода. Первый ввод — это имя файла. По желанию можете указать путь для сохранения файла. Второй ввод — это формат файла. В данном примере файл сохраняется в формате PNG (Portable Network Graphic), но есть и другие варианты: PDF (Portable Document Format), PS (Postscript), EPS (Encapsulated Postscript) и SVG (Scalable Vector Graphics).


ЗАПОМНИ!

Обратите внимание на наличие магической функции `%matplotlib auto`, в данном случае. Использование этого вызова удаляет вид `inline` графика. У вас есть параметры для других возможностей MatPlotLib, в зависимости от того, какую версию Python и MatPlotLib вы используете. Например, некоторые разработчики предпочитают виду `inline` вид `notebook`, поскольку он обеспечивает дополнительную функциональность. Но для использования `notebook` необходимо также перезапустить ядро, и вы не всегда сможете увидеть то, что ожидаете. Чтобы увидеть список возможностей, используйте магическую функцию `%matplotlib -l`. Кроме того, вызов метода `plt.ioff()` отключает взаимодействие с графиком.

Настройка осей, отметок, сеток

Довольно трудно понять, что на самом деле означают данные, если вы не предоставите их единицу измерения или хотя бы какое-нибудь средство для сравнения. Использование осей, отметок и сеток позволяет графически проиллюстрировать относительный размер элементов данных, чтобы зритель получил оценку сравнительного измерения. Вы не будете использовать эти функции в каждом графике и вы можете использовать функции по-разному в зависимости от потребностей зрителя, но важно знать, как использовать их для помощи в документировании ваших данных в графической среде.


ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

В следующих примерах используется магическая функция `%matplotlib notebook`, чтобы вы могли увидеть разницу между ней и магической функцией `%matplotlib inline`. Два встроенных вида используют другой графический механизм. Следовательно, вы должны выбрать пункт меню Kernel⇨Restart (Ядро⇨Перезапуск),

чтобы перезапустить ядро, прежде чем запускать какие-либо примеры, приведенные в следующих разделах.

Получение осей

Оси определяют плоскости x и y графика. Ось x проходит горизонтально, ось y — вертикально. Во многих случаях можно позволить MatPlotLib выполнить любое необходимое форматирование самостоятельно. Но иногда необходимо получить доступ к осям и отформатировать их вручную. Следующий код показывает, как получить доступ к осям графика:

```
import matplotlib.pyplot as plt
%matplotlib notebook

values = [0, 5, 8, 9, 2, 0, 3, 10, 4, 7]
ax = plt.axes()
plt.plot(range(1,11), values)
plt.show()
```

Причина, по которой оси размещают в переменной `ax`, а не манипулируют ими непосредственно, заключается в том, что это делает написание кода проще и эффективнее. В этом случае используйте стандартные оси, вызывая метод `plt.axes()`, а затем поместите дескриптор осей в `ax`. *Дескриптор* (handle) — это своего рода указатель на оси. Считайте это сковородой. Вы не будете поднимать саму сковороду, вы будете брать ее только за ручку (handle), когда ее нужно поднять.

Форматирование осей

Во многих случаях простого отображения осей будет недостаточно. Понадобится способ изменить отображения MatPlotLib. Например, вы не хотите, чтобы наибольшее значение *t* достигло вершины графика. В следующем примере показано лишь небольшое количество задач, которые можно решать после получения доступа к осям:

```
import matplotlib.pyplot as plt
%matplotlib notebook

values = [0, 5, 8, 9, 2, 0, 3, 10, 4, 7]
ax = plt.axes()
ax.set_xlim([0, 11])
ax.set_ylim([-1, 11])
ax.set_xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
ax.set_yticks([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
plt.plot(range(1,11), values)
plt.show()
```

В данном случае вызовы функций `set_xlim()` и `set_ylim()` изменяют пределы осей (длину каждой оси). Вызовы функций `set_xticks()` и `set_yticks()` изменяют отметки, используемые для отображения данных. С помощью этих вызовов можно изменить график во всех подробностях. Например, вы можете изменить метки отдельных отметок, если хотите. На рис. 10.3 показан вывод этого примера. Обратите внимание, как изменения влияют на отображение линейного графика.

```
In [3]: import matplotlib.pyplot as plt
        %matplotlib notebook

        values = [0, 5, 8, 9, 2, 0, 3, 10, 4, 7]
        ax = plt.axes()
        ax.set_xlim([0, 11])
        ax.set_ylim([-1, 11])
        ax.set_xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
        ax.set_yticks([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
        plt.plot(range(1,11), values)
        plt.show()
```

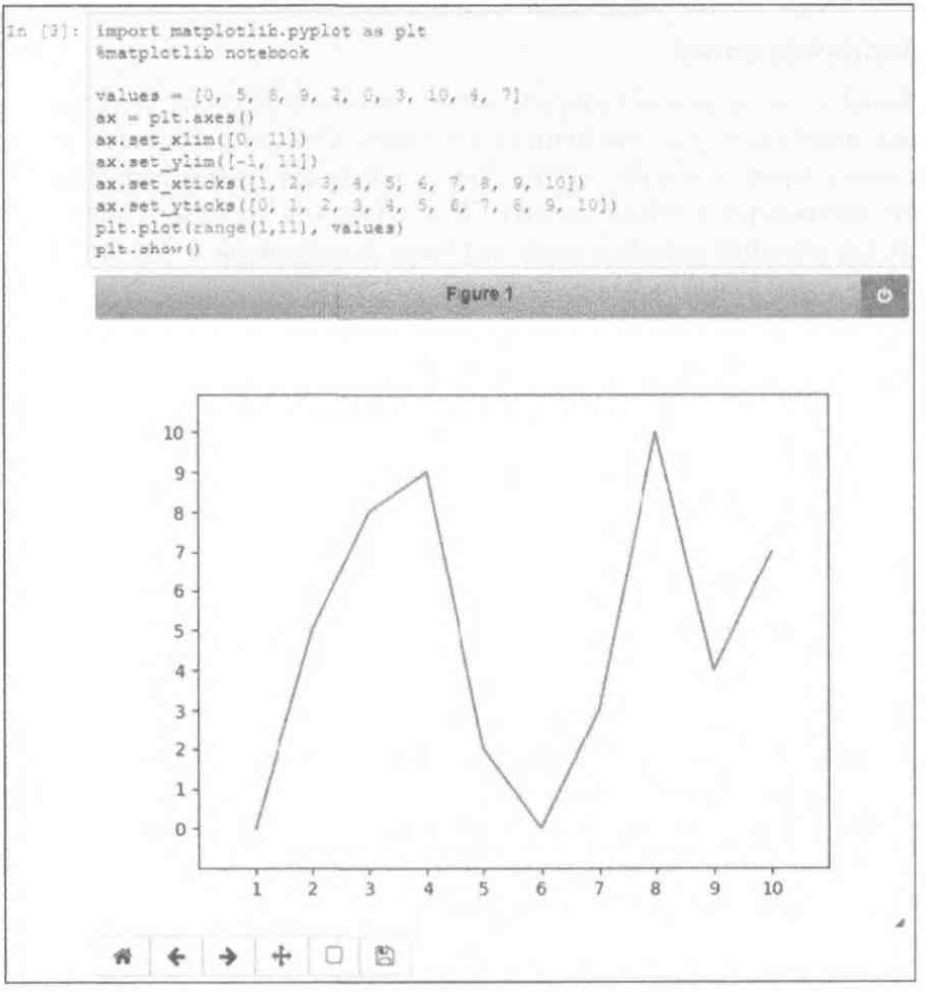


Рис. 10.3. Изменение отображения осей


ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Как можно заметить, просматривая различия между рис. 10.1–10.3, магическая функция `%matlplotlib notebook` дает совершенно иной вид. Элементы управления в нижней части представления позволяют перемещать и масштабировать экран, переходить между

созданными видами и загружать рисунок на диск. Кнопка справа от заголовка Figure 1 (Рисунок 1) на рис. 10.3 позволяет прекратить взаимодействие с графиком после завершения работы с ним. Любые изменения, внесенные вами в представление графика, после этого остаются, и любой, кто смотрит на ваш Notebook, увидит график так, как вы хотели. Возможность взаимодействия с графиком заканчивается, когда вы отображаете другой график.

Добавление сетки

Линии сетки позволяют увидеть точное значение каждого элемента графика. Вы можете быстрее определить координаты x и y, что облегчает сравнение отдельных точек. Конечно, сетки также добавляют шум и затрудняют просмотр фактического потока данных. Дело в том, что вы можете использовать сетки для создания определенных эффектов. Следующий код показывает, как добавить сетку в график из предыдущего раздела:

```
import matplotlib.pyplot as plt
%matplotlib notebook

values = [0, 5, 8, 9, 2, 0, 3, 10, 4, 7]
ax = plt.axes()
ax.set_xlim([0, 11])
ax.set_ylim([-1, 11])
ax.set_xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
ax.set_yticks([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
ax.grid()
plt.plot(range(1,11), values)
plt.show()
```

Все, что вам действительно нужно сделать, — это вызвать функцию `grid()`. Как и во многих других функциях библиотеки MatPlotLib, вы можете добавить параметры для создания сетки точно так, как вы хотите ее видеть. Например, вы можете решить, добавлять x строк сетки, y линий сетки или и то и другое. Вывод этого примера показан на рис. 10.4. В данном случае на рисунке показан блокнот с отключенным взаимодействием.

Определение внешнего вида линии

Простое рисование линий на странице мало что дает, если вы хотите помочь зрителю понять важность данных. В большинстве случаев нужно использовать разные стили линий, чтобы зритель мог отличить одну группу данных от

другой. Кроме того, чтобы подчеркнуть важность или ценность конкретной группы данных, нужно использовать цвет. Например, зеленый цвет часто означает безопасность, а красный — опасность. Следующие разделы помогут понять, как работать со стилем и цветом линий, чтобы донести идеи и концепции до зрителя без использования какого-либо текста.

```
In [4]: import matplotlib.pyplot as plt
        %matplotlib notebook

        values = [0, 5, 8, 9, 2, 0, 3, 10, 4, 7]
        ax = plt.axes()
        ax.set_xlim([0, 11])
        ax.set_ylim([-1, 11])
        ax.set_xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
        ax.set_yticks([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
        ax.grid()
        plt.plot(range(1,11), values)
        plt.show()
```

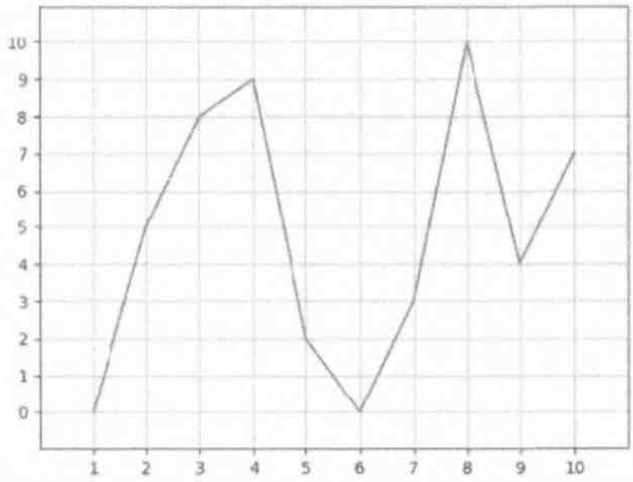


Рис. 10.4. Добавление сетки облегчает чтение значений

ДОСТУПНОСТЬ ГРАФИКИ

При создании презентации очень важно учесть, что страдающий дальтонизмом, не сможет сказать, что одна линия зеленая, а другая красная, а кто-то не в состоянии различить пунктирную и штрихпунктирную линии. Использование нескольких методов для различения каждой линии поможет всем увидеть ваши данные удобным для него способом.

Работа со стилями линий

Стили линий помогут различать графики, выводя линии различными способами. Использование уникального представления для каждой линии помогает различать каждую из них (даже если они отпечатаны в оттенках серого). Вы также можете выразить конкретный линейный график, используя для него другой стиль линии (и используя тот же стиль для других линий). Различные стили линий MatPlotLib приведены в табл. 10.1.

Таблица 10.1. Стили линий MatPlotLib

Символ	Стиль линии
'-'	Сплошная
'--'	Пунктирная
'-.'	Штрихпунктирная
':'	Точечная

Стиль линии является третьим аргументом вызова функции `plot()`. Вы просто указываете желаемую строку для типа линии, как показано в следующем примере:

```
import matplotlib.pyplot as plt
%matplotlib inline

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
plt.plot(range(1,11), values, '--')
plt.plot(range(1,11), values2, ':')
plt.show()
```

В данном случае первый линейный график использует стиль пунктирной линии, в то время как второй линейный график — точечных линий (рис. 10.5).

Использование цвета

Цвет — это еще один способ различения линейных графиков. Конечно, у этого метода есть определенные проблемы. Наиболее существенная из них возникает, когда кто-то делает черно-белую копию вашего цветного графика, скрывая различия цветов в оттенках серого. Другая проблема заключается в том, что кто-то, страдающий дальтонизмом, не сможет отличить одну линию от другой. Все это говорит о том, что цвет делает презентации более яркими

и привлекательными. В табл. 10.2 приведены цвета, поддерживаемые библиотекой MatPlotLib.

```
In [1]: import matplotlib.pyplot as plt
        %matplotlib inline

        values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
        values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
        plt.plot(range(1,11), values, '--')
        plt.plot(range(1,11), values2, ':')
        plt.show()
```

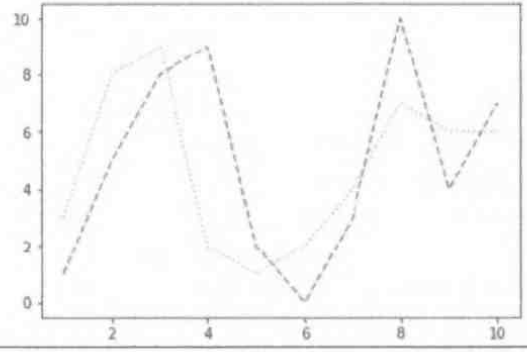


Рис. 10.5. Стили линий помогают различать графики

Таблица 10.2. Цвета MatPlotLib

Символ	Цвет
'b'	Blue (синий)
'g'	Green (зеленый)
'r'	Red (красный)
'c'	Cyan (голубой)
'm'	Magenta (пурпурный)
'y'	Yellow (желтый)
'k'	Black (черный)
'w'	White (белый)

Как и в случае стилей линий, цвет присутствует в строке как третий аргумент вызова функции `plot()`. В данном случае зритель видит две линии — одну красную, другую пурпурную. Фактическая презентация выглядит как на

рис. 10.2, но с заданными цветами, а не со стандартными, используемыми на этом снимке экрана. Если вы читаете печатную версию книги, на рис. 10.2 фактически используются оттенки серого.

```
import matplotlib.pyplot as plt
%matplotlib inline

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
plt.plot(range(1,11), values, 'r')
plt.plot(range(1,11), values2, 'm')
plt.show()
```

Добавление маркеров

Маркеры добавляют специальный символ к каждой точке данных на линейном графике. В отличие от стиля и цвета линий, маркеры, как правило, менее подвержены проблемам доступности и печати. Даже когда конкретный маркер неясен, люди обычно могут отличить один маркер от другого. В табл. 10.3 приведен список маркеров, предоставляемых библиотекой MatPlotLib.

Таблица 10.3. Маркеры MatPlotLib

Символ	Тип маркера
'.'	Точка
','	Пиксель
'o'	Круг
'v'	Треугольник 1 вниз
'^'	Треугольник 1 вверх
'<'	Треугольник 1 влево
'>'	Треугольник 1 вправо
'1'	Треугольник 2 вниз
'2'	Треугольник 2 вверх
'3'	Треугольник 2 влево
'4'	Треугольник 2 вправо
's'	Прямоугольник

Символ	Тип маркера
`'p'`	Пятиугольник
`'*'`	Звезда
`'h'`	Шестиугольник стиля 1
`'H'`	Шестиугольник стиля 2
`'+'`	Плюс
`'x'`	X
`'D'`	Ромб
`'d'`	Тонкий ромб
`'|'`	Вертикальная линия
`'_'`	Горизонтальная линия

Как и в случаях со стилем и цветом линии, маркеры добавляются в качестве третьего аргумента в вызов функции `plot()`. В следующем примере показаны эффекты объединения стиля линии с маркером для обеспечения уникального представления линейного графика:

```
import matplotlib.pyplot as plt
%matplotlib inline

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
plt.plot(range(1,11), values, 'o--')
plt.plot(range(1,11), values2, 'v:')
plt.show()
```

Обратите внимание, как сочетание стиля линии и маркера выделяет каждую линию на рис. 10.6. Даже при печати в черно-белом режиме вы можете легко отличить одну линию от другой, поэтому обычно вы будете комбинировать приемы презентации.

```
In [3]: import matplotlib.pyplot as plt
        %matplotlib inline

        values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
        values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
        plt.plot(range(1,11), values, 'o--')
        plt.plot(range(1,11), values2, 'v:')
        plt.show()
```

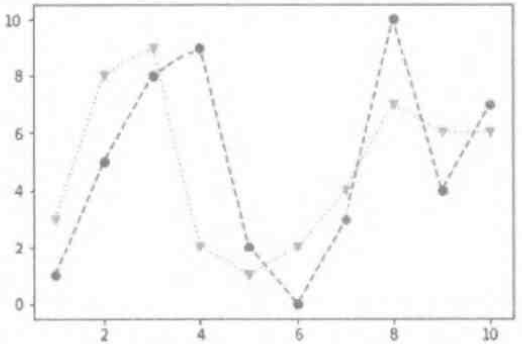


Рис. 10.6. Маркеры помогают подчеркнуть индивидуальные значения

Использование меток, аннотаций и легенд

Чтобы полностью документировать свой график, обычно приходится прибегать к меткам, аннотациям и легендам. Каждый из этих элементов имеет различное назначение, описанное ниже.

- **Метка**. Обеспечивает позитивную идентификацию конкретного элемента данных или группы. Облегчает понимание названия или вида показанных данных.
- **Аннотация**. Дополняет информацию, которую зритель может сразу увидеть в данных, с помощью заметок, источников или другой полезной информации. В отличие от метки, аннотация помогает расширить знания о данных, а не просто идентифицировать их.
- **Легенда**. Представляет список групп данных на графике и часто содержит подсказки (например, тип линии или цвет), чтобы упростить идентификацию группы данных. Например, все красные точки могут принадлежать группе A, тогда как все синие точки — группе B.

В следующих разделах описаны назначение и применение различных вспомогательных средств документации, поставляемых с MatPlotLib. Некоторые графики прекрасно работают без каких-либо вспомогательных документов, но в других случаях понадобится использовать все три элемента.

Добавление меток

Метки помогают понять значение каждой оси любого графика. Без меток отображаемые значения не имеют никакого смысла. Помимо аббревиатуры вы можете добавить единицы измерения, например дюймы или сантиметры, чтобы ваша аудитория знала, как интерпретировать показанные данные. В следующем примере показано, как добавить метки на график:

```
import matplotlib.pyplot as plt
%matplotlib inline

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
plt.xlabel('Entries')
plt.ylabel('Values')
plt.plot(range(1,11), values)
plt.show()
```

Вызов функции `xlabel()` документирует ось x графика, в то время как вызов функции `ylabel()` документирует ось y (рис. 10.7).

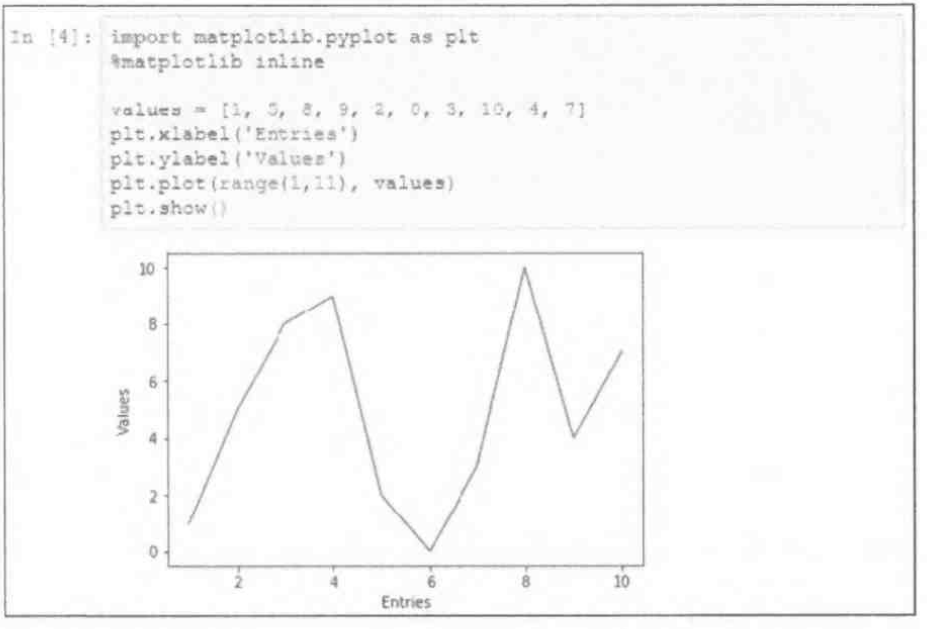


Рис. 10.7. Используйте метки для определения осей

Аннотирование диаграммы

Аннотацию используют для того, чтобы привлечь особое внимание к интересующим вас точкам на графике. Например, вы можете указать, что некая точка данных находится за пределами обычного диапазона, ожидаемого для

определенного набора данных. В следующем примере показано, как добавить аннотацию к графику:

```
import matplotlib.pyplot as plt
%matplotlib inline

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
plt.annotate(xy=[1,1], s='First Entry')
plt.plot(range(1,11), values)
plt.show()
```

Вызов функции `annotate()` обеспечивает необходимую маркировку. Необходимо указать местоположение аннотации с помощью параметра `xy`, а также указать текст для размещения в этом месте с помощью параметра `s`. Функция `annotate()` предоставляет также и другие параметры, которые можно использовать для создания специального форматирования или размещения на экране (рис. 10.8).

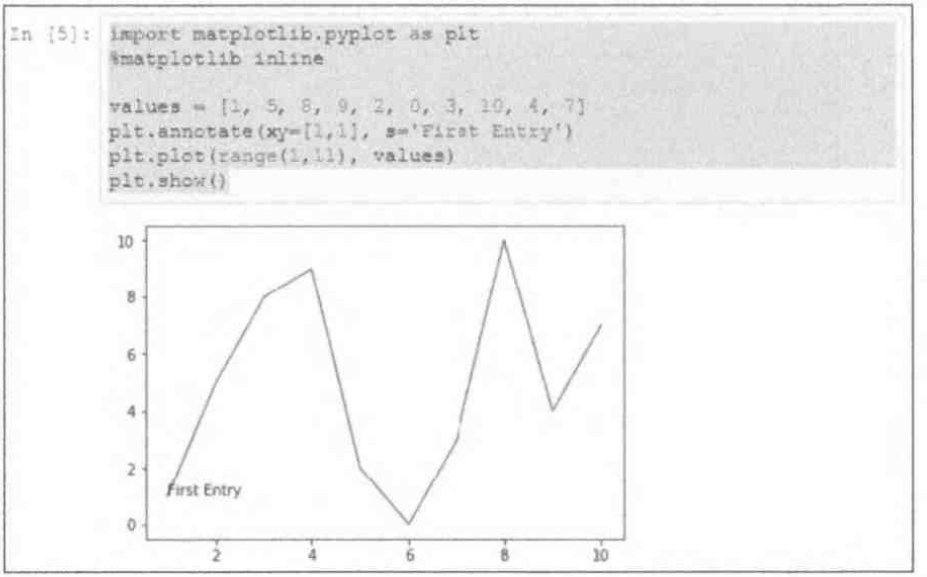


Рис. 10.8. Аннотация может идентифицировать интересные места

Создание легенды

Легенда документирует отдельные элементы графика. Каждая линия представлена в таблице, которая содержит ее метку, чтобы можно было различать их. Например, одна линия может представлять продажи в 2017 году, а другая — в 2018 году, поэтому нужно включить запись в легенду для каждой строки с метками 2017 и 2018. В следующем примере показано, как добавить легенду на график:

```
import matplotlib.pyplot as plt
%matplotlib inline

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
line1 = plt.plot(range(1,11), values)
line2 = plt.plot(range(1,11), values2)
plt.legend(['First', 'Second'], loc=4)
plt.show()
```

Функция `legend()` вызывается после создания графиков, а не до, как некоторые другие функции, описанные в этой главе. Вы должны предоставить дескриптор для каждого из участков. Обратите внимание, что `line1` установлен равным первому вызову `plot()`, а `line2` установлен равным второму вызову `plot()`.

СОВЕТ

Стандартное расположение легенды — верхний правый угол графика, который оказался неудобным для данного конкретного примера. Добавление параметра `loc` позволяет разместить легенду в другом месте. См. документацию функции `legend()` по адресу `https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.legend`, где можно найти дополнительные положения легенд. На рис. 10.9 показан результат этого примера.

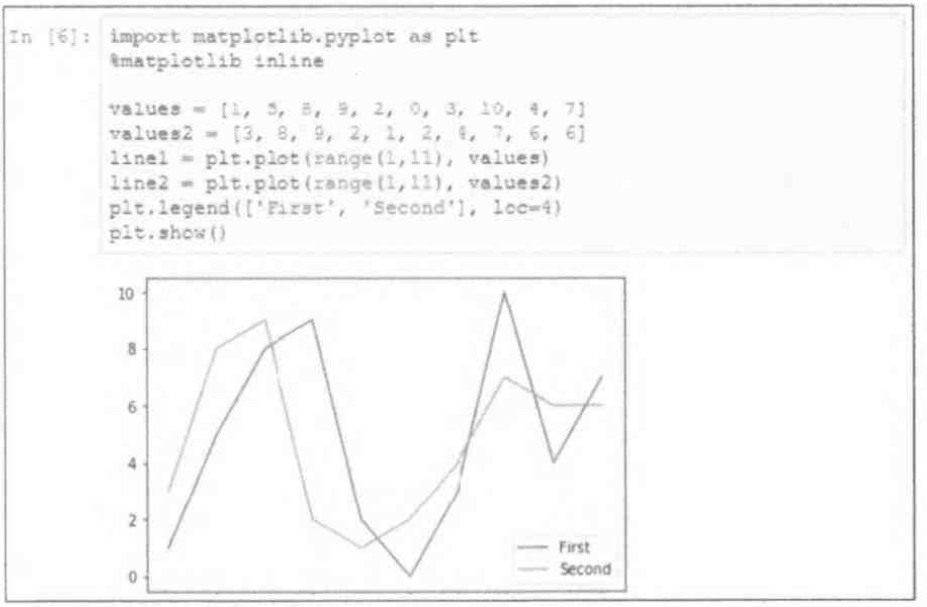


Рис. 10.9. Используйте легенду для идентификации отдельных линий

Глава 11

Визуализация данных

В ЭТОЙ ГЛАВЕ...

- Выбор правильного графика для работы
- Работа с расширенными диаграммами рассеяния
- Изучение временных и географических данных
- Создание графиков

В главе 10 мы рассмотрели механизм работы с библиотекой MatPlotLib, что является важным первым шагом к ее использованию. В данной главе вы научитесь использовать библиотеку MatPlotLib для выполнения полезной работы. Основная цель главы — помочь визуализировать ваши данные различными способами. Создание графического представления данных очень важно, если вы хотите помочь другим людям понять, что вы пытаетесь сказать. Несмотря на то что сами вы знаете, что значат цифры, другим людям, вероятно, дополнительно понадобится графика.

Глава начинается с рассмотрения некоторых основных типов графов, поддерживаемых библиотекой MatPlotLib. В этой главе вы не найдете полного перечня диаграмм и графиков, чтобы изучить их все в деталях, для этого может потребоваться целая книга. Здесь описаны лишь наиболее распространенные типы.

В этой главе вы начнете исследовать конкретные виды графиков, поскольку они связаны с наукой о данных. Конечно, ни одна книга по науке о данных не была бы полной без изучения *диаграмм рассеяния* (scatterplot), которые помогают увидеть шаблоны в, казалось бы, не связанных точках данных. Поскольку большая часть данных, с которыми вы работаете сегодня, имеет временную или географическую природу, данная глава посвящена этим двум типам. Вы

также можете работать как с направленными, так и с ненаправленными графами, которые хорошо подходят для анализа в социальных сетях.


СОВЕТ

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_11_Visualizing_the_Data.ipynb`.

Выбор правильного графика

Тип графика определяет, как люди будут просматривать связанные с ним данные, поэтому его выбор очень важен с самого начала. Например, если вы хотите показать, как различные элементы данных влияют на единое целое, используйте круговую диаграмму. Если хотите, чтобы люди формировали мнение, сравнивая элементы данных, используйте гистограмму. Таким образом, следует выбрать такой график, который естественным образом заставит людей прийти к тому выводу, для которого вы собирали данные из различных источников, тщательно обрабатывали и рисовали. (У вас также есть возможность использования линейных графиков; см. главу 10.) В следующих разделах описаны различные типы графиков и представлены основные примеры их использования.

Демонстрация части целого на круговой диаграмме

Круговые диаграммы сосредотачиваются на показе частей целого, которое составляет 100 процентов. Вопрос в том, сколько процентов из них занимает каждое значение. В следующем примере показано, как создать круговую диаграмму с множеством специальных признаков:

```
import matplotlib.pyplot as plt
%matplotlib inline

values = [5, 8, 9, 10, 4, 7]
colors = ['b', 'g', 'r', 'c', 'm', 'y']
labels = ['A', 'B', 'C', 'D', 'E', 'F']
explode = (0, 0.2, 0, 0, 0, 0)

plt.pie(values, colors=colors, labels=labels,
        explode=explode, autopct='%1.1f%%',
        counterclock=False, shadow=True)
plt.title('Values')

plt.show()
```

Важной частью круговой диаграммы являются значения. Вы можете создать простую круговую диаграмму, используя в качестве входных данных только значения.

Параметр `colors` позволяет выбирать собственные цвета для каждого сектора круга, а параметр `labels` идентифицирует каждый клин. Во многих случаях требуется выделить один клин среди других, и для этого добавляют параметр `explode` со списком выносимых значений. Значение 0 удерживает клин на месте, а любое другое значение перемещает его от центра круга.

Каждый клин может демонстрировать различные виды информации. В данном примере показан процент заполнения каждого клина параметром `autopct`. Для форматирования процентов следует предоставить строку формата.


СОВЕТ

Некоторые параметры влияют на способ построения круговой диаграммы. Параметр `counterclock` определяет направление клиньев (по часовой стрелке или против). Параметр `shadow` определяет тень под кругом (для трехмерного эффекта). По адресу `https://matplotlib.org/api/pyplot_api.html` вы можете найти и другие параметры.

В большинстве случаев вы захотите также дополнить круговую диаграмму заголовком, чтобы другие знали, что она представляет. Для этого используется функция `title()` (рис. 11.1).

Сравнение на гистограмме

Гистограммы облегчают сравнение значений. Широкие столбцы и отдельные измерения подчеркивают различия между значениями, а не переход одного значения в другое, как это было бы на линейном графике. К счастью, у вас есть все виды методов для выделения определенных значений и выполнения других трюков. В следующем примере показаны только некоторые вещи, которые вы можете сделать с вертикальной гистограммой:

```
import matplotlib.pyplot as plt
%matplotlib inline

values = [5, 8, 9, 10, 4, 7]
widths = [0.7, 0.8, 0.7, 0.7, 0.7, 0.7]
colors = ['b', 'r', 'b', 'b', 'b', 'b']
plt.bar(range(0, 6), values, width=widths,
    color=colors, align='center')

plt.show()
```

```
In [1]: import matplotlib.pyplot as plt
        %matplotlib inline

        values = [5, 8, 9, 10, 4, 7]
        colors = ['b', 'g', 'r', 'c', 'm', 'y']
        labels = ['A', 'B', 'C', 'D', 'E', 'F']
        explode = (0, 0.2, 0, 0, 0, 0)

        plt.pie(values, colors=colors, labels=labels,
                explode=explode, autopct='%1.1f%%',
                counterclock=False, shadow=True)
        plt.title('Values')

        plt.show()
```

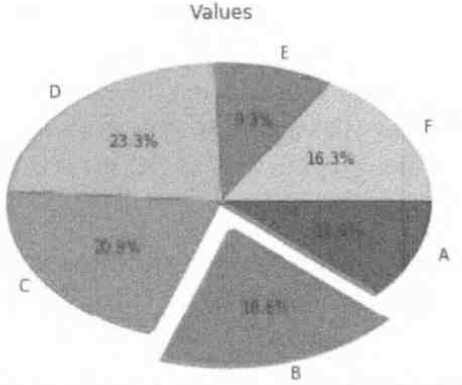


Рис. 11.1. Круговые диаграммы показывают процент от целого

Чтобы создать даже простую гистограмму, необходимо указать серию координат x и высоту столбцов. Для создания координат x в примере используется функция `range()`, а массив `values` содержит высоты.

Конечно, вам может потребоваться нечто большее, чем простая гистограмма, и библиотека MatPlotLib предоставляет несколько способов для этого. В данном примере для управления шириной каждого столбца используется параметр `width`, подчеркивая второй столбец, делая его немного больше. Большая ширина будет видна даже на черно-белой распечатке. Здесь также используется параметр `color`, позволяющий изменить цвет целевой панели на красный (остальные — синие).

Как и другие типы диаграмм, гистограмма предоставляет некоторые специальные средства, которые можно использовать для выделения презентации. В этом примере параметр `align` используется для центрирования данных по координате x (стандартная позиция — слева). Вы также можете использовать другие параметры, такие как `hatch` (штриховка), для улучшения визуального представления гистограммы. На рис. 11.2 показан вывод этого примера.

```
In [2]: import matplotlib.pyplot as plt
        %matplotlib inline

        values = [5, 8, 9, 10, 4, 7]
        widths = [0.7, 0.8, 0.7, 0.7, 0.7, 0.7]
        colors = ['b', 'r', 'b', 'b', 'b', 'b']
        plt.bar(range(0, 6), values, width=widths,
                color=colors, align='center')

        plt.show()
```

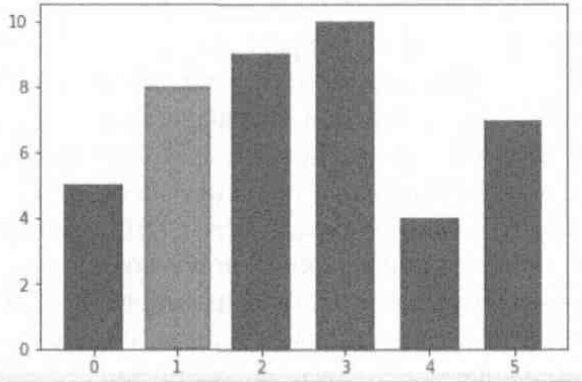


Рис. 11.2. Гистограммы облегчают сравнение

Эта глава поможет вам начать использовать библиотеку MatPlotLib для создания различных типов диаграмм и графиков. Конечно, чем больше примеров, тем лучше, поэтому вы также можете найти более передовые примеры на сайте MatPlotLib по адресу `https://matplotlib.org/1.2.1/examples/index.html`. Некоторые из примеров, демонстрирующие методы анимации, являются достаточно сложными, но на практике вы вполне можете использовать любой из них для улучшения ваших собственных диаграмм и графиков.

Отображение распределений с использованием гистограмм

Гистограммы классифицируют данные, разбивая их на *столбцы* (bin), каждый из которых содержит подмножество всего диапазона данных. Гистограмма отображает количество элементов в каждом столбце, чтобы было лучше видно распределение и последовательность данных от столбца к столбцу. В большинстве случаев вы видите кривую некого типа, такую как колоколообразная кривая. В следующем примере показано, как создать гистограмму со случайными данными:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

x = 20 * np.random.randn(10000)

plt.hist(x, 25, range=(-50, 50), histtype='stepfilled',
         align='mid', color='g', label='Test Data')
plt.legend()
plt.title('Step Filled Histogram')
plt.show()
```

В данном случае входные значения представляют собой последовательность случайных чисел. Распределение этих чисел должно показать кривую колоколообразного типа. Для построения графика необходимо, как минимум, предоставить серию значений, в данном случае x. Второй аргумент содержит количество столбцов, используемых при создании интервалов данных. Стандартное значение — 10. Использование параметра `range` помогает сфокусировать гистограмму на соответствующих данных и исключить любые выбросы.

Вы можете создавать гистограммы нескольких типов. При стандартных настройках создается столбчатая гистограмма. Вы также можете создать столбчатую гистограмму с накоплением, ступенчатый график или ступенчатый график с заполнением (тип, показанный в примере). Кроме того, можно контролировать ориентацию вывода; стандартно используется вертикальная ориентация.

Как и в большинстве других диаграмм и графиков, описанных в этой главе, вы можете добавить к выводу специальные возможности. Например, параметр `align` определяет выравнивание каждой полосы вдоль базовой линии. Используйте параметр `color` для управления цветом полос. Параметр `label` фактически не появляется, если вы не создаете легенду (как показано в этом примере). Рис. 11.3 демонстрирует типичный вывод этого примера.


ЗАПОМНИ!

Случайные данные варьируются от вызова к вызову. Каждый раз, когда вы запускаете пример, вы видите немного разные результаты, поскольку процесс генерации случайный.

Обозначение групп с использованием диаграмм размаха

Диаграммы размаха (boxplot), или *ящики с усами*, позволяют изобразить группы чисел через их *квартили* (quartile) (три точки, разделяющие группу на четыре равные части). Ящик может также иметь линии, называемые *усами* (whiskers), указывающими данные за пределами верхнего и нижнего квартилей. Интервал, показанный на диаграмме размаха, позволяет указать перекос

и разброс данных. В следующем примере показано, как создать диаграмму размаха со случайными данными:

```
In [3]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        x = 20 * np.random.randn(10000)

        plt.hist(x, 25, range=(-50, 50), histtype='stepfilled',
                 align='mid', color='g', label='Test Data')
        plt.legend()
        plt.title('Step Filled Histogram')
        plt.show()
```

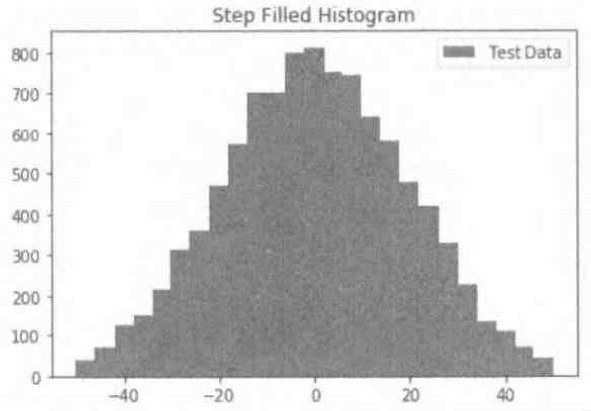


Рис. 11.3. Гистограммы позволяют увидеть распределение чисел

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

spread = 100 * np.random.rand(100)
center = np.ones(50) * 50
flier_high = 100 * np.random.rand(10) + 100
flier_low = -100 * np.random.rand(10)
data = np.concatenate((spread, center,
                       flier_high, flier_low))

plt.boxplot(data, sym='gx', widths=.75, notch=True)
plt.show()
```

Чтобы получить пригодный для использования набор данных, нужно объединить несколько различных методов генерации чисел, как показано в начале примера. Ниже описаны эти методы.

- spread. Содержит набор случайных чисел от 0 до 100.
- center. Обеспечивает 50 значений непосредственно в центре диапазона 50.
- flier_high. Имитирует выбросы от 100 до 200.
- flier_low. Имитирует выбросы от 0 до –100.

Код объединяет все эти значения в один набор данных с помощью функции concatenate(). Будучи созданными случайно с конкретными характеристиками (такими, как большое количество точек в середине), вывод будут демонстрировать конкретные характеристики и нормально работать для примера.

В качестве входных данных вызов функции boxplot() требует только data. Все остальные параметры имеют стандартные настройки. В данном случае код устанавливает представление выбросов как зеленые X, используя параметр sym. Для изменения размера блока используется параметр widths (в этом случае он установлен очень большим, чтобы ящик было легче увидеть). Наконец, вы можете создать ящик или ящик с надрезом, используя параметр notch (стандартное значение False). Рис. 11.4 демонстрирует типичный вывод этого примера.

```
In [4]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        spread = 100 * np.random.rand(100)
        center = np.ones(50) * 50
        flier_high = 100 * np.random.rand(10) + 100
        flier_low = -100 * np.random.rand(10)
        data = np.concatenate((spread, center,
                               flier_high, flier_low))

        plt.boxplot(data, sym='gx', widths=.75, notch=True)
        plt.show()
```

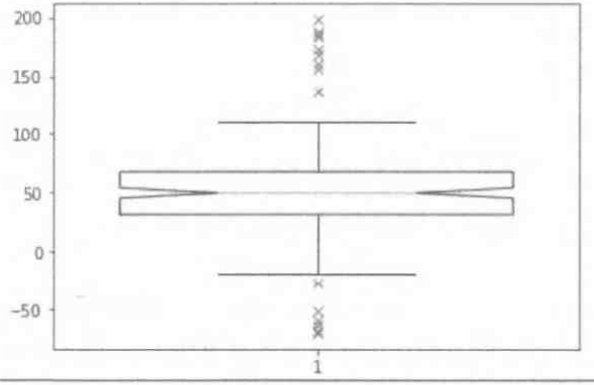


Рис. 11.4. Использование диаграммы размаха для представления групп чисел

Ящик показывает три точки данных в виде прямоугольника, а красная линия в середине — это медиана. Две черные горизонтальные линии, соединенные с ящиком при помощи усов, показывают верхний и нижний пределы (для четырех квартилей). Выбросы появляются выше и ниже верхней и нижней предельных линий в виде зеленых крестиков.

Просмотр шаблонов данных с использованием диаграмм рассеяния

Диаграммы рассеяния показывают кластеры данных, а не тренды (как линейные графики) или дискретные значения (как гистограммы). Цель диаграммы рассеяния — помочь увидеть шаблоны данных. В следующем примере показано, как создать диаграмму рассеяния с использованием случайных данных:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

x1 = 5 * np.random.rand(40)
x2 = 5 * np.random.rand(40) + 25
x3 = 25 * np.random.rand(20)
x = np.concatenate((x1, x2, x3))

y1 = 5 * np.random.rand(40)
y2 = 5 * np.random.rand(40) + 25
y3 = 25 * np.random.rand(20)
y = np.concatenate((y1, y2, y3))

plt.scatter(x, y, s=[100], marker='^', c='m')
plt.show()
```

Пример начинается с генерации случайных координат x и y. Для каждой координаты x у вас должна быть соответствующая координата y. Диаграмму рассеяния можно создать, используя только координаты x и y.

Диаграмму рассеяния можно создать несколькими способами. В данном случае параметр `s` определяет размер каждой точки данных. Параметр `marker` определяет форму точки данных. Используйте параметр `c`, чтобы задать цвета для всех точек данных, или определите отдельный цвет для отдельных точек данных. На рис. 11.5 показан вывод этого примера.

Создание расширенных диаграмм рассеяния

Диаграммы рассеяния особенно важны для науки о данных, поскольку они способны демонстрировать те шаблоны данных, которые не очевидны при

просмотре другими способами. Вы можете просматривать группы данных с относительной легкостью и помочь зрителю понять, когда данные принадлежат определенной группе. Вы также можете показывать перекрытия между группами и даже демонстрировать, когда определенные данные находятся за пределами ожидаемого диапазона. Отображение всех столь различных видов отношений в данных — это передовой метод, который необходимо знать, чтобы наилучшим образом использовать библиотеку MatPlotLib. В следующих разделах показано, как применять эти передовые методы к диаграмме рассеяния, созданной ранее в этой главе.

```
In [5]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        x1 = 5 * np.random.rand(40)
        x2 = 5 * np.random.rand(40) + 25
        x3 = 25 * np.random.rand(20)
        x = np.concatenate((x1, x2, x3))

        y1 = 5 * np.random.rand(40)
        y2 = 5 * np.random.rand(40) + 25
        y3 = 25 * np.random.rand(20)
        y = np.concatenate((y1, y2, y3))

        plt.scatter(x, y, s=[100], marker='^', c='m')
        plt.show()
```

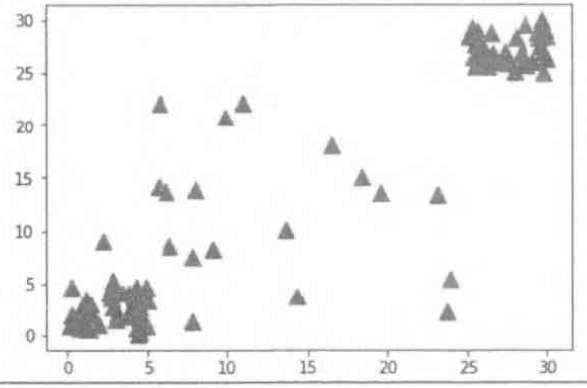


Рис. 11.5. Использование диаграммы рассеяния для демонстрации группы точек данных и связанных с ними шаблонов

Отображение групп

Пример работает по существу так же, как пример диаграммы рассеяния в предыдущем разделе, за исключением того, что здесь используется массив для

цветов. К сожалению, в черно-белой печатной книге различия между оттенками серого на рис. 11.6 будет трудно увидеть. Тем не менее первая группа — синего цвета, вторая — зеленого. Все выбросы отображаются красным цветом.

```
In [6]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        x1 = 5 * np.random.rand(50)
        x2 = 5 * np.random.rand(50) + 25
        x3 = 30 * np.random.rand(25)
        x = np.concatenate((x1, x2, x3))

        y1 = 5 * np.random.rand(50)
        y2 = 5 * np.random.rand(50) + 25
        y3 = 30 * np.random.rand(25)
        y = np.concatenate((y1, y2, y3))

        color_array = ['b'] * 50 + ['g'] * 50 + ['r'] * 25

        plt.scatter(x, y, s=[50], marker='D', c=color_array)
        plt.show()
```

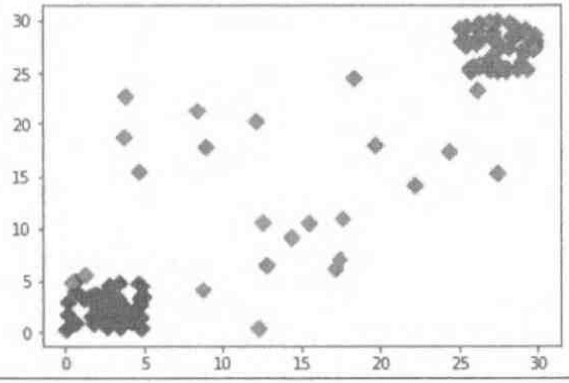


Рис. 11.6. Цветные массивы помогут лучше выделить группы рассеяния

Отображение корреляций

В некоторых случаях вам нужно будет знать общее направление, в котором движутся данные при просмотре диаграммы рассеяния. Даже если вы создадите четкое описание групп, фактическое направление, в котором эти данные движутся в целом, может быть неясным. В этом случае добавьте в вывод линию тренда. Ниже приведен пример добавления линии тренда к диаграмме рассеяния, которая включает в себя группы, но она не так ясна, как диаграмма рассеяния, показанная ранее на рис. 11.6.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pylab as plb
%matplotlib inline

x1 = 15 * np.random.rand(50)
x2 = 15 * np.random.rand(50) + 15
x3 = 30 * np.random.rand(25)
x = np.concatenate((x1, x2, x3))

y1 = 15 * np.random.rand(50)
y2 = 15 * np.random.rand(50) + 15
y3 = 30 * np.random.rand(25)
y = np.concatenate((y1, y2, y3))

color_array = ['b'] * 50 + ['g'] * 50 + ['r'] * 25
plt.scatter(x, y, s=[90], marker='*', c=color_array)
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
plb.plot(x, p(x), 'm-')

plt.show()
```

Код создания диаграммы рассеяния, по сути, такой же, как в примере из предыдущего раздела, но график уже не определяет группы так четко. Добавление линии тренда означает вызов с данными функции `polyfit()` библиотеки NumPy, которая возвращает вектор коэффициентов, `p`, минимизирующий ошибку наименьших квадратов. (Регрессия по методу наименьших квадратов — это метод для нахождения линии, которая суммирует отношения между двумя переменными, `x` и `y`, в данном случае, по крайней мере, в пределах области объясняемой переменной `x`. Третий параметр функции `polyfit()` выражает степень аппроксимации с помощью полинома.) Выходной вектор функции `polyfit()` используется в качестве входных данных для функции `poly1d()`, который вычисляет фактические точки данных для оси `y`. Вызов функции `plot()` создает линию тренда на диаграмме рассеяния. Типичный результат этого примера приведен на рис. 11.7.

Построение временны*х* рядов

Ничто не является действительно статичным. Когда вы просматриваете большинство данных, вы видите некий момент времени — снимок того, какими были данные в один конкретный момент. Конечно, такие представления общеприняты и полезны. Но иногда нужно просматривать данные по мере их

изменения во времени. Только просматривая данные по мере их изменения, можно понять основные силы, которые их формируют. В следующих разделах описано, как работать с данными на временной основе.

```
In [7]: import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.pylab as plb
        %matplotlib inline

        x1 = 15 * np.random.rand(50)
        x2 = 15 * np.random.rand(50) + 15
        x3 = 30 * np.random.rand(30)
        x = np.concatenate((x1, x2, x3))

        y1 = 15 * np.random.rand(50)
        y2 = 15 * np.random.rand(50) + 15
        y3 = 30 * np.random.rand(30)
        y = np.concatenate((y1, y2, y3))

        color_array = ['b'] * 50 + ['g'] * 50 + ['r'] * 25
        plt.scatter(x, y, s=[90], marker='*', c=color_array)
        z = np.polyfit(x, y, 1)
        p = np.poly1d(z)
        plb.plot(x, p(x), 'm-')

        plt.show()
```

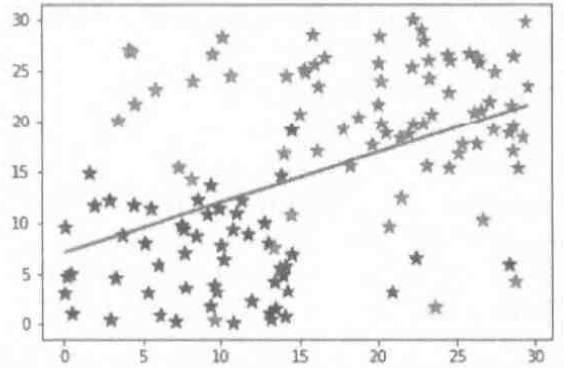


Рис. 11.7. Линия тренда на диаграмме рассеяния может показать общее направление данных

Представление времени по осям

Нередко нужно представлять изменение данных в течение времени. Данные могут поступать во многих формах, но обычно есть некоторый тип отметок времени (одна единица времени), за которым следует одна или несколько функций, описывающих происходящее во время этого конкретного отрезка

времени. В следующем примере показан простой набор дней и продаж в эти дни для определенного элемента в целочисленных значениях:

```
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt
%matplotlib inline

start_date = dt.datetime(2018, 7, 30)
end_date = dt.datetime(2018, 8, 5)
daterange = pd.date_range(start_date, end_date)
sales = (np.random.rand(len(daterange)) * 50).astype(int)
df = pd.DataFrame(sales, index=daterange,
                  columns=['Sales'])

df.loc['Jul 30 2018':'Aug 05 2018'].plot()
plt.ylim(0, 50)
plt.xlabel('Sales Date')
plt.ylabel('Sale Value')
plt.title('Plotting Time')
plt.show()
```

Пример начинается с создания объекта `DataFrame` для хранения информации. Источником информации может быть что угодно, но пример генерирует ее случайным образом. Обратите внимание, что в примере создается `date_range` для хранения начальной и конечной временных рамок даты для упрощения обработки с использованием цикла `for`.

Важной частью этого примера является создание отдельных строк. Каждая строка имеет фактическое значение времени, чтобы вы не теряли информацию. Однако обратите внимание, что индекс (свойство `row_s.name`) является строкой. Эта строка должна присутствовать в той форме, в которой вы хотите, чтобы даты отображались при представлении на графике.

Использование `loc[]` позволяет выбрать диапазон дат из общего числа доступных записей. Обратите внимание, что в этом примере для вывода используются только некоторые созданные данные. Затем добавляется некоторая уточняющая информация о сюжете и отображается на экране. В этом случае в вызове функции `plot()` следует указать значения `x` и `y`, иначе вы получите ошибку. На рис. 11.8 показан типичный вывод для случайно созданных данных.

Отображение трендов с течением времени

Как и в случае любого другого представления данных, иногда без посторонней помощи действительно трудно увидеть, в каком направлении движутся данные. Следующий пример начинается с графика из предыдущего раздела, и к нему добавляется линия тренда:

```
In [8]: import pandas as pd
        import matplotlib.pyplot as plt
        import datetime as dt

        start_date = dt.datetime(2018, 7, 29)
        end_date = dt.datetime(2018, 8, 7)
        daterange = pd.date_range(start_date, end_date)
        sales = (np.random.rand(len(daterange)) * 50).astype(int)
        df = pd.DataFrame(sales, index=daterange,
                          columns=['Sales'])

        df.loc['Jul 30 2018':'Aug 05 2018'].plot()
        plt.ylim(0, 50)
        plt.xlabel('Sales Date')
        plt.ylabel('Sale Value')
        plt.title('Plotting Time')
        plt.show()
```

Рис. 11.8. Использование линейного графика для демонстрации изменения данных с течением времени

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt
%matplotlib inline

start_date = dt.datetime(2018, 7, 29)
end_date = dt.datetime(2018, 8, 7)
daterange = pd.date_range(start_date, end_date)
sales = (np.random.rand(len(daterange)) * 50).astype(int)
df = pd.DataFrame(sales, index=daterange,
                  columns=['Sales'])

lr_coef = np.polyfit(range(0, len(df)), df['Sales'], 1)
lr_func = np.poly1d(lr_coef)
trend = lr_func(range(0, len(df)))
df['trend'] = trend
```

```
df.loc['Jul 30 2018':'Aug 05 2018'].plot()

plt.xlabel('Sales Date')
plt.ylabel('Sale Value')
plt.title('Plotting Time')
plt.legend(['Sales', 'Trend'])
plt.show()
```


ЗАПОМНИ!

Выше, в разделе "Отображение корреляций", показано, как большинство людей добавляют линию тренда на свой график. Фактически этот подход часто используется в Интернете. Вы также заметите, что в некоторых ситуациях у многих возникают проблемы с использованием этого подхода. В данном примере используется немного другой подход: линия тренда добавляется непосредственно в объект `DataFrame`. Если вы выведете `df` после вызова `df['trend'] = trend`, то увидите данные линии тренда, похожие на значения, показанные ниже.

```
            Sales      trend
2018-07-29      6  18.890909
2018-07-30     13  20.715152
2018-07-31     38  22.539394
2018-08-01     22  24.363636
2018-08-02     40  26.187879
2018-08-03     39  28.012121
2018-08-04     36  29.836364
2018-08-05     21  31.660606
2018-08-06      7  33.484848
2018-08-07     49  35.309091
```

Использование этого подхода упрощает построение данных. Вы вызываете функцию `plot()` только один раз и не полагаетесь на `pylab` библиотеки MatPlotLib, как показано в примере раздела "Отображение корреляций". Полученный код проще и менее склонен к проблемам, которые вы видите в Интернете.

Когда вы отображаете начальные данные, вызов функции `plot()` автоматически создает легенду. Библиотека MatPlotLib не добавляет линию тренда автоматически, поэтому вы должны также создать новую легенду для графика. На рис. 11.9 показан типичный вывод этого примера с использованием случайно созданных данных.

```
In [9]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import datetime as dt

        start_date = dt.datetime(2018, 7, 29)
        end_date = dt.datetime(2018, 8, 7)
        daterange = pd.date_range(start_date, end_date)
        sales = (np.random.rand(len(daterange)) * 50).astype(int)
        df = pd.DataFrame(sales, index=daterange,
                          columns=['Sales'])

        lr_coef = np.polyfit(range(0, len(df)), df['Sales'], 1)
        lr_func = np.poly1d(lr_coef)
        trend = lr_func(range(0, len(df)))
        df['trend'] = trend
        df.loc['Jul 30 2018':'Aug 05 2018'].plot()

        plt.xlabel('Sales Date')
        plt.ylabel('Sale Value')
        plt.title('Plotting Time')
        plt.legend(['Sales', 'Trend'])
        plt.show()
```

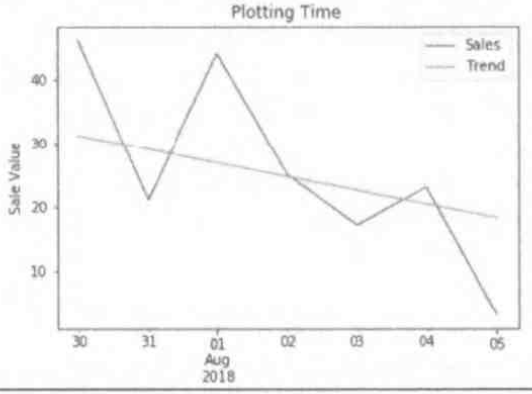


Рис. 11.9. Добавьте линию тренда, чтобы показать среднее направление изменения данных на графике

Отображение географических данных

Может быть важным знание того, откуда поступают данные или как они относятся к конкретному месту. Например, если вы хотите знать, где возникла нехватка продовольствия, и спланировать, как с ней справиться, вам необходимо сопоставить имеющиеся у вас данные с географическим местоположением. То же самое относится и к прогнозированию будущих продаж. Вы можете обнаружить, что вам нужно использовать существующие данные и определить, где разместить новые магазины. В противном случае вы могли бы разместить

магазин в том месте, где продажи окажутся невысоки, и вы скорее потеряете силы и деньги, а не заработаете. В следующих разделах описано, как работать с простой картой при взаимодействии с географическими данными.


ВНИМАНИЕ!

Вы должны закрыть среду Notebook перед тем, как вносить в нее какие-либо изменения, в противном случае conda пожалуется, что некоторые файлы еще используются. Чтобы закрыть среду Notebook, закройте и остановите ядро для всех открытых вами файлов Notebook, а затем нажмите комбинацию клавиш <Ctrl + C> в окне терминала Notebook. Подождите несколько секунд, прежде чем пытаться что-либо делать, чтобы дать файлам время для правильного закрытия.

Использование среды Notebook

Некоторые из устанавливаемых вами пакетов также имеют тенденцию изменять среду вашего Notebook, устанавливая другие пакеты, которые могут плохо работать с вашей базовой конфигурацией. Следовательно, появятся проблемы с кодом, который функционировал ранее. Обычно эти проблемы состоят в основном из предупреждающих сообщений, таких как предупреждения об устаревании, как обсуждается далее, в разделе "Решение проблем устаревания библиотек". Однако в некоторых случаях измененные пакеты также могут менять вывод кода. Возможно, более новый пакет использует обновленный алгоритм или по-другому взаимодействует с кодом. Если у вас есть такой пакет, как Basemap, который вносит изменения в общую базовую конфигурацию, и вы хотите сохранить текущую конфигурацию, вам необходимо настроить для него среду. Среда сохраняет базовую конфигурацию без изменений, но также позволяет новому пакету создавать среду, необходимую для правильного выполнения. Ниже описано, как создать среду Basemap, используемую в этой главе.

1. **Откройте Anaconda Prompt.**

   Обратите внимание, что в приглашении указывается местоположение вашей папки в вашей системе, но ей предшествует `(base)`. Индикатор `(base)` говорит о том, что вы находитесь в своей базовой среде — той, которую хотите сохранить.

2. **Введите `create -n Basemap python=3 anaconda=5.2.0` и нажмите клавишу <Enter>.**

   Это действие создает новую среду Basemap, которая будет использовать Python 3.6 и Anaconda 5.2.0. Вы получаете точно такую же среду, которую использовали до сих пор.

3. **Введите `source activate Basemap`, если используете OS X или Linux, или `activate Basemap`, если используете Windows, и нажмите клавишу <Enter>.**

   Вы перешли в среду Basemap. Обратите внимание, что приглашение `(base)` сменилось на `(Basemap)`.

4. **Для установки своей копии Basemap следуйте инструкциям раздела "Получение набора инструментов Basemap".**

5. **Введите `Jupyter Notebook` и нажмите клавишу <Enter>.**

   Вы видите запуск Notebook, но теперь он использует среду Basemap, а не Baseline. Эта копия Notebook работает точно так же, как и любая другая, которую вы использовали. Единственное отличие — это среда, в которой он работает.


ЗАПОМНИ!

Та же техника применима для любого специального пакета, который вы хотите установить. Вы должны зарезервировать ее для пакетов, которые не собираетесь использовать каждый день. Например, в этой книге Basemap используется только для одного примера, поэтому для нее целесообразно создать среду.

После того как закончите использовать среду Basemap, введите в командной строке **`deactivate`** и нажмите клавишу <Enter>. Вы увидите быстрое изменение обратно на `(base)`.

Получение набора инструментов Basemap

Прежде чем вы сможете работать с сопоставлением данных, вам потребуется библиотека, которая поддерживает необходимые функции. Доступно несколько таких пакетов, но проще всего устанавливать и работать с таковыми из набора Basemap Toolkit. Вы можете получить этот инструментарий по адресу `https://matplotlib.org/basemap/users/intro.html`. (Чтобы избежать ошибок доступа к файлам, убедитесь, что закрыли Notebook и остановили сервер, прежде чем продолжить действия, описанные в этом разделе.) Однако самый простой способ — использовать инструмент conda из Anaconda Prompt для ввода следующих команд:

```
conda install -c conda-forge basemap=1.1.0
conda install -c conda-forge basemap-data-hires
conda install -c conda-forge proj4=5.2.0
```

По тому же адресу содержится дополнительная информация о наборе инструментов. В отличие от некоторых других пакетов, данный пакет содержит инструкции для пользователей Mac, Windows и Linux. Кроме того, вы можете получить специальный установщик для Windows. Обязательно посмотрите

также видео о его использовании по адресу `http://nbviewer.ipython.org/github/mqlaql/geospatialdata/blob/master/Geospatial-Data-with-Python.ipynb`.

Вам понадобится следующий код, чтобы использовать набор инструментов после его установки:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
%matplotlib inline
```

Решение проблем устаревания библиотек

Одним из основных преимуществ работы с Python является огромное количество пакетов, которые он поддерживает. К сожалению, не каждый пакет получает обновления достаточно быстро, чтобы избежать использования устаревших функций в других пакетах. *Устаревшая функция* (deprecated feature) — это функция, которая все еще существует в целевом пакете, но разработчики этого пакета планируют удалить ее в следующем обновлении. Следовательно, при запуске вашего кода вы получаете предупреждение об устаревшем пакете. Даже несмотря на то, что предупреждение об устаревании не мешает вашему коду работать, оно, как правило, вызывает подозрения к вашему приложению. В конце концов, никто не хочет видеть то, что кажется сообщением об ошибке. Тот факт, что Notebook стандартно отображает эти сообщения светло-красным цветом, не имеет значения.

ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

К сожалению, ваша копия набора инструментов Basemap может выдать предупреждение об устаревшей функции, поэтому в данном разделе рассказывается, как решить эту проблему. Вы можете узнать больше о потенциальных проблемах по адресу `https://github.com/matplotlib/basemap/issues/382`. Эти сообщения выглядят примерно так:

```
C:\Users\Luca\Anaconda3\lib\site-packages\mpl_toolkits
\basemap\__init__.py:1708: MatplotlibDeprecationWarning:
The axesPatch function was deprecated in version 2.1.
Use Axes.patch instead.
    limb = ax.axesPatch

C:\Users\Luca\Anaconda3\lib\site-packages\mpl_toolkits
\basemap\__init__.py:1711: MatplotlibDeprecationWarning:
The axesPatch function was deprecated in version 2.1. Use
Axes.patch instead.
    if limb is not ax.axesPatch:
```

Этот текст выглядит действительно ужасающе, но такие сообщения указывают на две проблемы. В первую очередь, проблема в библиотеке MatPlotLib,

и она вращается вокруг вызова axesPatch. В сообщениях также говорится, что этот конкретный вызов устарел, начиная с версии 2.1. Используйте следующий код для проверки вашей версии MatPlotLib:

```
import matplotlib
print(matplotlib.__version__)
```

Если вы установили Anaconda, следуя инструкциям в главе 3, то увидите, что у вас есть как минимум MatPlotLib 2.2.2. Следовательно, одним из способов решения этой проблемы является понижение версии вашей копии библиотеки MatPlotLib с помощью следующей команды в приглашении Anaconda:

```
conda install -c conda-forge matplotlib=2.0.2
```

Проблема с этим подходом заключается в том, что он может также вызвать проблемы для любого кода, который использует более новые функции, находящиеся в MatPlotLib 2.2.2. Это не оптимально, но если вы часто используете Basemap в своем приложении, то это может быть практическим решением.

СОВЕТ

Лучшее решение — просто признать, что проблема существует, документируя ее как часть вашего кода. Документирование проблемы и ее конкретной причины облегчает поиск проблемы позже, после обновления пакета. Для этого вы добавляете две следующие строки кода:

```
import warnings
warnings.filterwarnings("ignore")
```

ЗАПОМНИ!

Вызов функции `filterwarnings()` выполняет указанное действие, которым в данном случае является `"ignore"`. Чтобы отменить эффекты фильтрации предупреждений, вызовите функцию `resetwarnings()`. Обратите внимание, что атрибут `module` совпадает с источником проблем в предупреждающих сообщениях. Вы также можете определить более широкий фильтр с помощью атрибута `category`. Данный конкретный вызов затрагивает только один модуль.

Использование Basemap для вывода географических данных

Теперь, когда у вас установлен пакет Basemap, вы можете с ним что-то сделать. В следующем примере показано, как вывести карту и разместить на ней указатели на определенные места:

```
austin = (-97.75, 30.25)
hawaii = (-157.8, 21.3)
washington = (-77.01, 38.90)
chicago = (-87.68, 41.83)
```

```
losangeles = (-118.25, 34.05)

m = Basemap(projection='merc',llcrnrlat=10,urcrnrlat=50,
            llcrnrlon=-160,urcrnrlon=-60)

m.drawcoastlines()
m.fillcontinents(color='lightgray',lake_color='lightblue')
m.drawparallels(np.arange(-90.,91.,30.))
m.drawmeridians(np.arange(-180.,181.,60.))
m.drawmapboundary(fill_color='aqua')

m.drawcountries()

x, y = m(*zip(*[hawaii, austin, washington,
                chicago, losangeles]))
m.plot(x, y, marker='o', markersize=6,
       markerfacecolor='red', linewidth=0)

plt.title("Mercator Projection")
plt.show()
```

Пример начинается с определения долготы и широты различных городов. Затем создается простая карта, а параметр `projection` определяет ее базовый внешний вид. Следующие четыре параметра, `llcrnrlat`, `urcrnrlat`, `llcrnrlon` и `urcrnrlon`, определяют стороны карты. Вы можете определить и другие параметры, но эти обычно создают используемую карту.

Следующий набор вызовов определяет подробности карты. Например, `drawcoastlines()` определяет, будут ли выделены береговые линии, чтобы их было легче увидеть. Чтобы облегчить разметку суши по воде, необходимо вызвать функцию `fillcontinents()` с цветами по вашему выбору. При работе с определенными местоположениями, как в примере, вы будете вызывать функцию `drawcountries()`, чтобы убедиться, что границы стран отображаются на карте. На данный момент у вас есть карта, которая готова для заполнения данными.

В данном примере координаты x и y создаются с использованием ранее сохраненных значений долготы и широты. Затем эти местоположения отображаются на карте контрастным цветом, чтобы их легче было увидеть. Последний шаг — отобразить карту, как показано на рис. 11.10.

Визуализация графов

Граф (graph) представляет собой изображение данных, демонстрирующее связи между точками данных с использованием линий. Цель в том, чтобы показать, что некоторые точки данных относятся к другим точкам данных, но не

ко всем точкам данных, отображаемым в графе. Возьмем, например, карту системы метро. Каждая из станций соединяется с другими станциями, но ни одна станция не соединяется со всеми станциями в системе метро. Графы являются популярной темой в науке о данных из-за их использования в анализе социальных сетей. Выполняя анализ в социальных сетях, вы изображаете и анализируете сети отношений, таких как друзья или деловые связи, из социальных сетей, таких как Facebook, Google+, Twitter или LinkedIn.

```
In [13]: austin = (-97.75, 30.25)
         hawaii = (-157.8, 21.3)
         washington = (-77.01, 38.90)
         chicago = (-87.68, 41.83)
         losangeles = (-118.25, 34.05)


         m = Basemap(projection='merc',llcrnrlat=10,urcrnrlat=50,
                     llcrnrlon=-160,urcrnrlon=-60)

         m.drawcoastlines()
         m.fillcontinents(color='lightgray',lake_color='lightblue')
         m.drawparallels(np.arange(-90.,91.,30.))
         m.drawmeridians(np.arange(-180.,181.,60.))
         m.drawmapboundary(fill_color='aqua')

         m.drawcountries()

         x, y = m(*zip(*[hawaii, austin, washington,
                         chicago, losangeles]))
         m.plot(x, y, marker='o', markersize=6,
                markerfacecolor='red', linewidth=0)

         plt.title("Mercator Projection")
         plt.show()
```

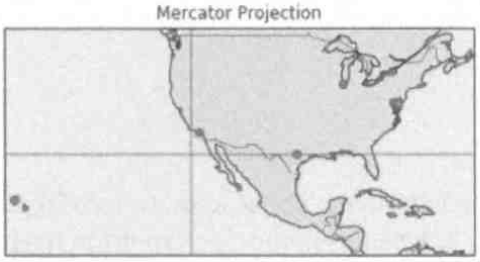


Рис. 11.10. Карты способны иллюстрировать данные так, как не могут другие графические объекты


ЗАПОМНИ!

Есть два основных типа графов: один является *ненаправленным* (undirected), когда граф просто показывает линии между элементами данных, другой — *направленным* (directed), когда к линии добавляются стрелки, показывающие, что данные перемещаются в определенном направлении. Рассмотрим, например, изображение системы

водоснабжения. В большинстве случаев вода будет течь только в одном направлении, поэтому вы можете использовать направленный граф, чтобы изобразить не только связи между источниками и потребителями воды, но и показать направление течения воды с помощью стрелок. В следующих разделах описаны два типа графов и показано, как их создавать.

Разработка ненаправленных графов

Как уже упоминалось, ненаправленный граф просто демонстрирует связи между узлами. Выходные данные не обеспечивают направления от одного узла к другому. Например, при установлении соединения между веб-страницами направление не подразумевается. В следующем примере показано, как создать ненаправленный граф:

```
import networkx as nx
import matplotlib.pyplot as plt
%matplotlib inline

G = nx.Graph()
H = nx.Graph()
G.add_node(1)
G.add_nodes_from([2, 3])
G.add_nodes_from(range(4, 7))
H.add_node(7)
G.add_nodes_from(H)

G.add_edge(1, 2)
G.add_edge(1, 1)
G.add_edges_from([(2,3), (3,6), (4,6), (5,6)])
H.add_edges_from([(4,7), (5,7), (6,7)])
G.add_edges_from(H.edges())

nx.draw_networkx(G)
plt.show()
```

В отличие от стандартного примера, приведенного в разделе "Использование основ NetworkX" главы 8, в данном примере строится граф с использованием ряда различных методов. Он начинается с импорта пакета Networkx, который мы использовали в главе 8. Чтобы создать новый ненаправленный граф, код вызывает конструктор `Graph()`, способный получать несколько входных аргументов для использования в качестве атрибутов. Тем не менее вы можете построить отличный граф и без использования атрибутов, что и показано в данном примере.

Самый простой способ добавить узел — вызвать функцию `add_node()` с номером узла. Вы также можете добавить список, словарь или `range()`

(диапазон) узлов, используя функцию `add_nodes_from()`. Фактически, при желании, можно импортировать узлы из других графов.


ЗАПОМНИ!

Несмотря на то что используемые в примере узлы основаны на числах, для своих узлов вы не обязаны использовать именно числа. Узел может использовать одну букву, строку или даже дату. Узлы имеют некоторые ограничения. Например, вы не сможете создать узел, используя логическое значение.

Поскольку узлы не имеют никакой связи с самого начала, вы должны определить связи (ребра) между ними. Чтобы добавить одно ребро, вызовите функцию `add_edge()` с номерами узлов, которые хотите добавить. Как и с узлами, вы можете использовать функцию `add_edges_from()`, чтобы создать более одного ребра, используя в качестве входных данных список, словарь или другой граф. На рис. 11.11 показан вывод этого примера (ваш вывод может немного отличаться, но соединения должны быть те же).

```
In [12]: import networkx as nx
         import matplotlib.pyplot as plt
         %matplotlib inline

         G = nx.Graph()
         H = nx.Graph()
         G.add_node(1)
         G.add_nodes_from([2, 3])
         G.add_nodes_from(range(4, 7))
         H.add_node(7)
         G.add_nodes_from(H)

         G.add_edge(1, 2)
         G.add_edge(1, 1)
         G.add_edges_from([(2,3), (3,6), (4,6), (5,6)])
         H.add_edges_from([(4,7), (5,7), (6,7)])
         G.add_edges_from(H.edges())

         nx.draw_networkx(G)
         plt.show()
```

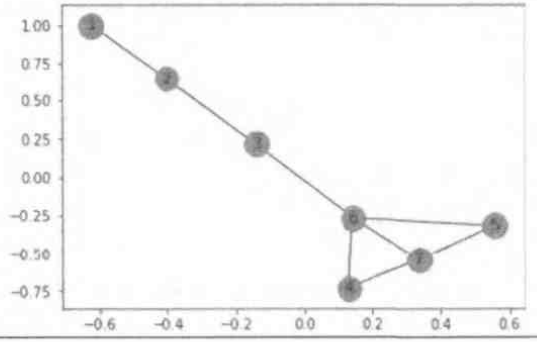


Рис. 11.11. Ненаправленные графы соединяют узлы вместе, образуя шаблоны

Разработка направленных графов

Направленные графы используют, когда нужно показать направление, скажем, от начальной точки до конечной. Когда вы получаете карту, на которой показано, как добраться из одной конкретной точки в другую, начальный и конечный узлы помечаются как таковые, а линии между этими узлами (и всеми промежуточными) показывают направление.

Ваши графики не должны быть скучными. Вы можете украсить их всевозможными способами, чтобы зритель получал дополнительную информацию по-разному. Например, можете создать собственные метки, использовать специальные цвета для определенных узлов или полагаться на цвет, чтобы помочь людям увидеть значения, лежащие в основе ваших графиков. Вы также можете изменить вес линии ребра и использовать другие методы, чтобы пометить конкретный путь между узлами в качестве лучшего. В следующем примере показаны многие (но не все) способы, позволяющие сделать направленный граф более интересным:

```
import networkx as nx
import matplotlib.pyplot as plt
%matplotlib inline

G = nx.DiGraph()

G.add_node(1)
G.add_by nodes_from([2, 3])
G.add_nodes_from(range(4, 6))
G.add_path([6, 7, 8])

G.add_edge(1, 2)
G.add_edges_from([(1,4), (4,5), (2,3), (3,6), (5,6)])

colors = ['r', 'g', 'g', 'g', 'g', 'm', 'm', 'r']
labels = {1:'Start', 2:'2', 3:'3', 4:'4',
                     5:'5', 6:'6', 7:'7', 8:'End'}
sizes = [800, 300, 300, 300, 300, 600, 300, 800]

nx.draw_networkx(G, node_color=colors, node_shape='D',
                 with_labels=True, labels=labels,
                 node_size=sizes)
plt.show()
```

Пример начинается с создания направленного графа с помощью конструктора `DiGraph()`. Следует отметить, что пакет NetworkX также поддерживает типы графов `MultiGraph()` и `MultiDiGraph()`. Список всех типов графов приведен по адресу `https://networkx.lanl.gov/reference/classes.html`.

Добавление узлов очень похоже на работу с ненаправленным графом. Вы можете добавлять отдельные узлы, используя функцию `add_node()`, и несколько узлов, используя функцию `add_nodes_from()`. Вызов функции `add_path()` позволяет создавать узлы и ребра одновременно. Порядок узлов в вызове важен. Переход от одного узла к другому осуществляется слева направо по списку, предоставленному для вызова.

В этом примере к выводу добавляются специальные цвета узлов, метки, фигура (используется только одна) и размеры. Для этого вы по-прежнему вызываете функцию `draw_networkx()`. Однако добавление показанных параметров меняет внешний вид графа. Обратите внимание, что вы должны установить `with_labels` в `True`, чтобы увидеть метки, предоставленные параметром `labels` (рис. 11.12).

```
In [13]: import networkx as nx
         import matplotlib.pyplot as plt
         %matplotlib inline

         G = nx.DiGraph()

         G.add_node(1)
         G.add_nodes_from([2, 3])
         G.add_nodes_from(range(4, 6))
         G.add_path([6, 7, 8])

         G.add_edge(1, 2)
         G.add_edges_from([(1,4), (4,5), (2,3), (3,6), (5,6)])

         colors = ['r', 'g', 'g', 'g', 'g', 'm', 'm', 'r']
         labels = {1:'Start', 2:'2', 3:'3', 4:'4',
                   5:'5', 6:'6', 7:'7', 8:'End'}
         sizes = [800, 300, 300, 300, 300, 600, 300, 800]

         nx.draw_networkx(G, node_color=colors, node_shape='D',
                          with_labels=True, labels=labels,
                          node_size=sizes)
         plt.show()
```

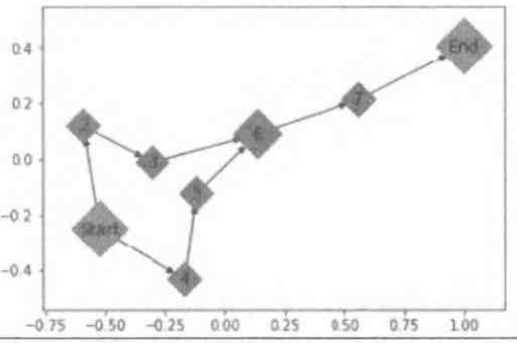


Рис. 11.12. Используйте направленные графы, чтобы показать направление между узлами

Добавление ребер во многом аналогично работе с ненаправленным графом. Вы можете использовать функцию `add_edge()` для добавления одного ребра или `add_edges_from()` для добавления нескольких ребер одновременно. Однако порядок номеров узлов важен. Переход идет от левого узла к правому в каждой паре.

4 Манипулирование данными

В ЭТОЙ ЧАСТИ...

- Установка и применение различных пакетов науки о данных
- Анализ данных
- Сокращение измерений в наборе данных
- Кластеризация данных в наборах
- Повышение надежности за счет обнаружения выбросов

Глава 12

Расширение возможностей Python

В ЭТОЙ ГЛАВЕ...

- Как пакет Scikit-learn работает с классами
- Использование разреженных матриц и трюк хеширования
- Проверка производительности и потребления памяти
- Экономия времени с многоядерными алгоритмами

В предыдущих главах мы рассмотрели все основные методы загрузки и обработки данных, предлагаемые языком Python. Теперь пришло время использовать более сложные инструменты для обработки данных (или манипулирования) и машинного обучения. Завершающим этапом большинства проектов в области науки о данных является создание инструмента данных, способного автоматически обобщать, прогнозировать и давать рекомендации непосредственно на основании ваших данных.

Прежде чем сделать этот последний шаг, вам все еще нужно обработать данные, применяя более радикальные преобразования. Это часть *обработки данных* (data wrangling) или *манипулирования данными* (data munging), где за сложными преобразованиями следуют визуальные и статистические исследования, а затем — дальнейшие преобразования. В следующих разделах вы узнаете, как обрабатывать огромные потоки текста, исследовать основные характеристики набора данных, оптимизировать скорость ваших экспериментов, сжимать данные и создавать новые искусственные признаки, создавать новые группы и классификации, а также обнаруживать неожиданные или исключительные случаи, способные погубить ваш проект.

Начиная с этого момента, вы будете больше использовать пакет Scikit-learn (а значит, вам нужно больше знать о нем — полная документация находится по адресу `https://scikit-learn.org/stable/documentation.html`). Пакет Scikit-learn предлагает единое хранилище, содержащее практически все инструменты, необходимые для того, чтобы стать аналитиком данных и чтобы ваш проект по науке данных был успешным. В этой главе вы познакомитесь с важными характеристиками пакета Scikit-learn, структурированными в модулях, классах и функциях, а также с некоторыми передовыми средствами экономии времени и повышения производительности при работе в Python с большими неструктурированными данными и длительными вычислительными операциями.


СОВЕТ

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле исходного кода `P4DS4D2_12_Stretching_Pythons_Capabilities.ipynb`.

Пакет Scikit-learn

Иногда наилучший способ узнать, как что-то использовать, — это потратить время, играя с ним. Чем сложнее инструмент, тем важнее становится игра. Учитывая сложные математические задачи, которые вы выполняете с помощью пакета Scikit-learn, игра становится особенно важной. В следующих разделах используется идея игры с пакетом Scikit-learn, чтобы помочь вам раскрыть важные концепции его использования в области обработки данных.

Понятие классов в Scikit-learn

Понимание того, как работают классы, является важной предпосылкой для правильного использования пакета Scikit-learn. Scikit-learn — это пакет для машинного обучения и экспериментов с данными, пользующийся популярностью у большинства аналитиков данных. Он содержит широкий спектр хорошо зарекомендовавших себя алгоритмов обучения, функций ошибок и процедур тестирования.

В своей основе пакет Scikit-learn имеет несколько базовых классов, на которых построены все алгоритмы. Помимо класса `BaseEstimator`, от которого происходят все остальные классы, существуют четыре типа классов, охватывающих все основные функции машинного обучения.

- Классификация (classifying).
- Регрессия (regressing).
- Группировка по кластерам (grouping by clusters)
- Преобразование данных (transforming data).

Несмотря на то что каждый базовый класс имеет определенные методы и атрибуты, основные функциональные возможности для обработки данных и машинного обучения гарантируются одним или несколькими сериями методов и атрибутов, называемых *интерфейсами* (interface). Интерфейсы предоставляют единый *интерфейс прикладных программ* (Application Programming Interface — API) для обеспечения сходства методов и атрибутов между всеми различными алгоритмами, имеющимися в пакете. Существуют четыре объектно-ориентированных интерфейса Scikit-learn:

- `estimator` (оценщик). Для *подбора* (fitting) параметров и изучения их по данным, согласно алгоритму.
- `predictor` (предсказатель). Для создания прогнозов по подобранным параметрам.
- `transformer` (трансформатор). Для преобразования данных и реализации установленных параметров.
- `model` (модель). Для выявления качества подбора или других показателей.

Пакет группирует алгоритмы, построенные на базовых классах и одном или нескольких объектных интерфейсах, в модули. Каждый модуль отображает специализацию в конкретном типе решения для машинного обучения. Например, модуль `linear_model` предназначен для линейного моделирования, а модуль `metrics` — для оценки результатов и потерь.

Чтобы найти конкретный алгоритм в пакете Scikit-learn, сначала вы должны найти модуль, содержащий тот же тип алгоритма, который вас интересует, а затем выбрать его из списка содержимого модуля. Алгоритм, как правило, представляет собой сам класс, методы и атрибуты которого уже известны, поскольку они являются общими для других алгоритмов в пакете Scikit-learn.

СОВЕТ

Привыкание к подходу классов Scikit-learn может занять некоторое время. Тем не менее API одинаковы для всех доступных в пакете инструментов, поэтому, изучая один класс, вы обязательно узнаете обо всех других классах. Лучший подход — выучить один класс полностью, а затем применять то, что вы знаете, к другим классам.

Определение приложений для науки о данных

Выяснение способов использования науки о данных для получения конструктивных результатов имеет важное значение. Например, вы можете применить интерфейс `estimator` для следующего.

- **Задача классификации**. Выявление принадлежности нового наблюдения к определенной группе.
- **Задача регрессии**. Выявление ценности нового наблюдения.

Рассмотрим метод `fit(X, y)`, где `X` — двумерный массив предикторов (набор наблюдений для изучения), а `y` — целевой результат (другой массив, одномерный).

После применения метода `fit`, информация в массиве `X` связывается с таковой в массиве `y`, так что, зная некую новую информацию с такими же характеристиками как в `X`, можно правильно угадать `y`. В процессе некоторые параметры оцениваются методом `fit` внутренне. Использование метода `fit` позволяет различать изучаемые параметры и гиперпараметры, которые вместо этого устанавливаются вами при создании экземпляра обучаемого.

Установка включает присвоение класса Scikit-learn переменной Python. В дополнение к гиперпараметрам вы также можете изменить другие рабочие параметры, такие как требование нормализации или установка случайного начального числа, чтобы воспроизводить одинаковые результаты для каждого вызова при одинаковых входных данных.

Ниже приведен пример с линейной регрессией — очень простой и распространенный алгоритм машинного обучения. Вы загружаете некоторые данные, чтобы использовать этот пример из примеров, которые предоставляет пакет Scikit-learn. Например, набор данных Boston содержит переменные предиктора, которые пример кода может сопоставить с ценами на дома, что помогает построить предиктор, способный рассчитать стоимость дома с учетом его характеристик.

```
from sklearn.datasets import load_boston
boston = load_boston()
X, y = boston.data,boston.target
print("X:%s y:%s" % (X.shape, y.shape))
```

Возвращенные размеры для переменных `X` и `y`:

```
X:(506, 13) y:(506,)
```

Вывод указывает, что оба массива имеют одинаковое количество строк и `X` содержит 13 объектов. Метод `shape` выполняет анализ массива и сообщает его размеры.

СОВЕТ

Количество строк в X должно равняться таковому в y. Вы также гарантируете, что массивы X и y совпадают, поскольку обучение на данных происходит, когда алгоритм сопоставляет строки в X с соответствующим элементом в y. Если значения обоих массивов случайны, обучение невозможно.

ЗАПОМНИ!

Характеристики X, выраженные в столбцах X, называются *переменными* (variable) (более статистический термин), или *признаками* (feature) (термин более связан с машинным обучением).

После импорта класса `LinearRegression` вы можете создать экземпляр переменной по имени `hypothesis` и установить параметр, указывающий алгоритм для стандартизации (т.е. установить в нуль среднее значение и единичное стандартное отклонение для всех переменных — это статистическая операция для выравнивания уровня всех переменных), прежде чем оценивать параметры для изучения.

```
from sklearn.linear_model import LinearRegression
hypothesis = LinearRegression(normalize=True)
hypothesis.fit(X, y)
print(hypothesis.coef)_
```

После этого выводятся коэффициенты гипотезы линейной регрессии:

```
[-1.07170557e-01  4.63952195e-02  2.08602395e-02
  2.68856140e+00 -1.77957587e+01  3.80475246e+00
  7.51061703e-04 -1.47575880e+00  3.05655038e-01
 -1.23293463e-02 -9.53463555e-01  9.39251272e-03
 -5.25466633e-01]
```

После подборки `hypothesis` содержит изученные параметры, и вы можете визуализировать их, используя метод `coef_`, который типичен для всех линейных моделей (где вывод модели представляет собой сумму переменных, взвешенных по коэффициентам). Вы также можете назвать эти действия подбором обучения (например, "обучение алгоритма машинного обучения").

ЗАПОМНИ!

Гипотеза (hypothesis) — это способ описания алгоритма обучения на данных. Гипотеза определяет возможное представление y для данного X, которое вы проверяете на достоверность. Таким образом, это гипотеза как научного, так и машинного языка.

Помимо класса `estimator`, важны также объекты классов `predictor` и `model`. Класс `predictor`, прогнозирующий вероятность определенного результата, получает результат новых наблюдений, используя методы `predict` и `predict_proba`, как в этом сценарии:

```
import numpy as np
new_observation = np.array([1, 0, 1, 0, 0.5, 7, 59,
                            6, 3, 200, 20, 350, 4],
                           dtype=float).reshape(1, -1)
print(hypothesis.predict(new_observation))
```

Таким образом, единичное наблюдение преобразуется в прогноз:

```
[25.8972784]
```

Убедитесь, что новые наблюдения имеют то же количество признаков и тот же их порядок, что и при обучении `X`; в противном случае прогноз будет неверным.

Класс `model` предоставляет информацию о качестве *подбора* (fit), используя метод `score`:

```
hypothesis.score(X, y)
```

Качество выражается как число с плавающей запятой:

```
0.7406077428649427
```

В этом случае `score` возвращает коэффициент детерминации R2 прогнозирования. R2 — это показатель в диапазоне от 0 до 1, сравнивающий наш предиктор с простым средним. Более высокие значения показывают, что предиктор работает хорошо. Различные алгоритмы обучения могут использовать разные функции оценки. Обратитесь за помощью к сетевой документации по каждому алгоритму или к консоли Python:

```
help(LinearRegression)
```

Класс `transform` применяет преобразования, полученные на этапе подбора, к другим массивам данных. В `LinearRegression` нет метода преобразования, но большинство алгоритмов предварительной обработки его имеют. Например, `MinMaxScaler` из модуля `preprocessing` пакета Scikit-learn может преобразовывать значения в определенный диапазон минимальных и максимальных значений, изучая формулу преобразования из примера массива:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
scaler.fit(X)
print(scaler.transform(new_observation))
```

Запуск этого кода возвращает преобразованные значения для наблюдений:

```
[ 0.01116872  0.          0.01979472  0.
  0.23662551  0.65893849  0.57775489  0.44288845
  0.08695652  0.02480916  0.78723404  0.88173887
  0.06263797]
```

В этом случае код применяет значения `min` и `max`, полученные из `X`, к переменной `new_observation` и возвращает преобразование.

Трюк хеширования

Пакет Scikit-learn предоставляет бо́льшую часть структур данных и функций, необходимых для завершения вашего проекта по науке о данных. Вы даже можете найти классы для самых сложных задач.

Например, при работе с текстом одним из наиболее полезных решений, предоставляемых пакетом Scikit-learn, является трюк с хешированием. Вы узнаете, как работать с текстом, используя модель набора слов (см. раздел "Использование модели наборов слов" главы 8) и преобразование TF-IDF (Term Frequency–Inverse Document Frequency — частота термина–инверсная частота в документе). Все эти мощные преобразования могут работать правильно, только если весь текст известен и доступен в памяти вашего компьютера.

Более серьезная задача в области науки о данных — это анализ текстовых потоков, создаваемых в сети, например, в социальных сетях или больших сетевых хранилищах текста. Этот сценарий представляет собой довольно сложную задачу при попытке превратить текст в матрицу данных, пригодную для анализа. При работе с такими задачами знание трюка хеширования может дать немало преимуществ, включая.

- оперативную обработку больших матриц данных на основе текста;
- исправление неожиданных значений или переменных в текстовых данных;
- создание масштабируемых алгоритмов для больших коллекций документов.

Использование хеш-функций

Хеш-функции могут преобразовывать любой ввод в вывод, характеристики которого предсказуемы. Обычно они возвращают значение, в котором выходные данные связаны с определенным интервалом, крайности которого варьируются от отрицательных до положительных чисел или охватывают только положительные числа. Вы можете считать, что они обеспечивают соблюдение стандарта для ваших данных, — независимо от того, какие значения вы предоставляете, они всегда возвращают определенный продукт данных.

Наиболее полезная характеристика хеш-функции заключается в том, что при определенном вводе они всегда возвращают одно и то же выходное числовое значение. Следовательно, их называют детерминированными функциями.

Например, введите такое слово как "собака", и функция хеширования всегда будет возвращать одно и то же число.

В определенном смысле хеш-функции похожи на секретный код, превращающий все в числа. Однако, в отличие от секретных кодов, вы не можете преобразовать хешированный код в его первоначальное значение. Кроме того, в некоторых редких случаях разные слова генерируют один и тот же результат (также называемый хеш-конфликтом (hash collision)).

Демонстрация трюка хеширования

Существует много хеш-функций, наиболее популярными из которых являются MD5 (обычно используются для проверки целостности файлов, поскольку можно хешировать целые файлы) и SHA (используется в криптографии). В Python есть встроенная хеш-функция `hash`, которую можно использовать для сравнения объектов данных перед их сохранением в словарях. Например, проверьте, как Python хеширует свое имя:

```
print(hash('Python'))
```

Команда возвращает большое целое число:

```
-1126740211494229687
```

Сеанс Python на вашем компьютере может возвращать значение, отличное от значения, указанного в предыдущей строке. Не беспокойтесь — встроенные хеш-функции не всегда одинаковы для разных компьютеров. Если вам нужен согласованный вывод, используйте вместо этого хеш-функции пакета Scikit-learn, поскольку их вывод одинаков для всех машин.

Хеш-функция Scikit-learn также может возвращать индекс в определенном положительном диапазоне. Вы можете получить что-то похожее, используя встроенную функцию `hash`, стандартное деление и остаток:

```
print(abs(hash('Python')) % 1000)
```

На этот раз полученный хеш представляет собой меньшее целое число:

```
687
```

Когда вы запрашиваете остаток от абсолютного числа результата хеш-функции, вы получаете число, которое никогда не превышает значение, использованное для деления. Чтобы увидеть, как работает этот метод, представьте, что вы хотите преобразовать текстовую строку из Интернета в числовой вектор (вектор признаков) и использовать его для запуска проекта машинного обучения. Хорошая стратегия решения этой задачи науки о данных — использовать

унитарное кодирование (one-hot encoding), создающее набор слов. Ниже описаны этапы унитарного кодирования строки ("Python for data science") в вектор.

1. Присвойте каждому слову произвольное число, например Python=0 for=1 data=2 science=3.
2. Инициализируйте вектор, подсчитав количество уникальных слов, которым вы присвоили код на первом этапе.
3. Используйте коды, присвоенные на первом этапе, в качестве индексов для заполнения вектора значениями, присваивая 1 в случае совпадения со словом, существующим во фразе.

Результирующий вектор признаков выражается в виде последовательности `[1,1,1,1]` и состоит ровно из четырех элементов. Вы запускаете процесс машинного обучения, указав программе ожидать последовательности из четырех текстовых объектов, когда неожиданно появляется новая фраза, и вы должны также преобразовать следующий текст в числовой вектор: "Python for machine learning". Теперь для работы у вас есть два новых слова — "machine learning". Следующие этапы помогут вам создать новые векторы.

1. Присвойте новые коды: machine=4 learning=5. Это и есть *кодирование* (encoding).
2. Увеличьте предыдущий вектор, чтобы включить новые слова: `[1,1,1,1,0,0]`.
3. Вычислите вектор для новой строки: `[1,1,0,0,1,1]`.

Унитарное кодирование является весьма оптимальным, поскольку оно создает эффективные и упорядоченные векторы признаков.

```
from sklearn.feature_extraction.text import *
oh_enconder = CountVectorizer()
oh_enconded = oh_enconder.fit_transform([
'Python for data science','Python for machine learning'])

print(oh_enconder.vocabulary_)
```

Команда возвращает словарь, содержащий слова и их кодировки:

```
{'python': 4, 'for': 1, 'data': 0, 'science': 5,
'machine': 3, 'learning': 2}
```

К сожалению, унитарное кодирование отказывает и становится трудным для обработки, если в проекте высокая изменчивость входных данных. Это обычная ситуация в проектах по науке о данных, работающих с текстом или другими символическими признаками, когда поток из Интернета или другой сетевой среды может внезапно создать или добавить исходные данные. Использование

хеш-функций — более разумный способ справиться с непредсказуемостью входных данных.

1. Определите диапазон для вывода хеш-функции. Все векторы признаков будут использовать этот диапазон. В примере используется диапазон значений от 0 до 24.
2. Используя хеш-функцию, вычислите индекс для каждого слова в строке.
3. Присвойте единичное значение позициям вектора в соответствии с индексами слов.

В Python вы можете определить простой трюк хеширования, создав функцию и проверив результаты, используя две тестовые строки:

```
string_1 = 'Python for data science'
string_2 = 'Python for machine learning'

def hashing_trick(input_string, vector_size=20):
    feature_vector = [0] * vector_size
    for word in input_string.split(' '):
        index = abs(hash(word)) % vector_size
        feature_vector[index] = 1
    return feature_vector
```

Теперь проверьте обе строки:

```
print(hashing_trick(
    input_string='Python for data science',
    vector_size=20))
```

Вот первая строка, закодированная как вектор:

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

Как и ранее, ваши результаты могут не совпадать с данными в книге, поскольку хеши могут не совпадать на разных машинах. Теперь код выводит вторую закодированную строку:

```
print(hashing_trick(
    input_string='Python for machine learning',
    vector_size=20))
```

Вот результат для второй строки:

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

При просмотре векторов признаков вы должны заметить, что

- не знаете, где находится каждое слово. Когда важно иметь возможность обратить процесс присвоения индексов словам, вы должны хранить отношения между словами и их хешированными

значениями отдельно (например, можете использовать словарь, в котором ключи — это хешированные значения, а значения — это слова);

- для небольших значений параметра функции `vector_size` (например, `vector_size = 10`) многие слова перекрываются в одних и тех же позициях списка, представляющего вектор признаков. Чтобы свести перекрытие к минимуму, вы должны создать границы хеш-функций, превышающие количество элементов, которые планируется индексировать позже.

Векторы признаков в этом примере состоят в основном из нулевых записей, что представляет собой растрату памяти по сравнению с более эффективным использованием унитарной кодировки. Одним из способов решения этой проблемы является использование разреженных матриц, описанное в следующем разделе.

Детерминированный отбор

Разреженные матрицы (sparse matrix) являются ответом при работе с данными, имеющими несколько значений, т.е. когда большинство значений матрицы равно нулю. Разреженные матрицы хранят только координаты ячеек и их значения вместо информации всех ячеек в матрице. Когда приложение запрашивает данные из пустой ячейки, разреженная матрица возвращает нулевое значение после поиска координат и обнаружения их отсутствия. Пример вектора приведен ниже.

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

Следующий код превращает его в разреженную матрицу:

```
from scipy.sparse import csc_matrix
print csc_matrix([1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                  0, 0, 0, 1, 0, 1, 0])
```

Вот представление, обеспечиваемое `csc_matrix`:

```
(0, 0)    1
(0, 5)    1
(0, 16)   1
(0, 18)   1
```

Обратите внимание, что данные представлены в координатах (выраженных в кортеже индекса строки и столбца) и значениях ячейки.

Пакет SciPy предлагает большое разнообразие разреженных матричных структур, каждая из которых хранит данные и работает по-своему. (Некоторые хороши для разделения (slicing), другие — для вычислений.) Обычно

csc_matrix (сжатая матрица, основанная на строках) является хорошим выбором, поскольку большинство алгоритмов Scikit-learn принимают ее как ввод, и это оптимально для матричных операций.

Как аналитику данных вам не нужно заботиться о программировании собственной версии трюка с хешированием, если только вам не нужна особая реализация идеи. Библиотека Scikit-learn предлагает класс HashingVectorizer, который быстро преобразует любую коллекцию текста в разреженную матрицу данных, используя метод хеширования. Вот пример сценария, который копирует предыдущий пример:

```
import sklearn.feature_extraction.text as txt
htrick = txt.HashingVectorizer(n_features=20,
                               binary=True, norm=None)
hashed_text = htrick.transform(['Python for data science',
                                'Python for machine learning'])
hashed_text
```

Python сообщает о размере разреженной матрицы и количестве хранящихся в ней элементов:

```
<2x20 sparse matrix of type '<class 'numpy.float64'>'
with 8 stored elements in Compressed Sparse Row format>
```

Как только поступает новый текст, CountVectorizer преобразует его на основе предыдущей схемы кодирования, где новые слова не присутствовали. Следовательно, результатом является пустой вектор нулей. Вы можете проверить это, преобразовав разреженную матрицу в нормальную, плотную, с помощью todense:

```
oh_enconder.transform(['New text has arrived']).todense()
```

Как и ожидалось, выведенная матрица пуста:

```
matrix([[0, 0, 0, 0, 0, 0]], dtype=int64)
```

Сравните вывод CountVectorizer и HashingVectorizer, который всегда обеспечивает место для новых слов в матрице данных:

```
htrick.transform(['New text has arrived']).todense()
```

Матрица, заполненная классом HashingVectorizer, представляет новые слова:

```
matrix([[1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
         0., 0., 0., 0., 0., 0., 1.]])
```

В худшем случае слово попадает в уже занятую позицию, в результате чего два разных слова рассматриваются алгоритмом как одно и то же (что не приведет к заметному снижению производительности алгоритма).

СОВЕТ

`HashingVectorizer` — это идеальная функция для использования, когда данные не помещаются в память и их признаки не исправлены. В других случаях рассмотрите возможность использования более интуитивно понятного класса `CountVectorizer`.

Учет сроков и производительности

Поскольку книга переходит к все более сложным темам, таким как классы машинного обучения пакета Scikit-learn и разреженные матрицы пакета SciPy, вы можете задаться вопросом: как вся эта обработка может повлиять на скорость приложения? Повышенные требования к обработке влияют как на время работы, так и на доступную память.

Обеспечение лучшего использования машинных ресурсов — это действительно искусство, искусство оптимизации, и для его освоения требуется время. Тем не менее вы можете сразу же стать опытным специалистом в этом, выполнив некоторые точные измерения скорости и поняв, какие у вас проблемы на самом деле. Профилирование времени, которое требуется операциям, измерение того, сколько памяти занимает добавление дополнительных данных, или выполнение их преобразования поможет обнаружить узкие места в вашем коде и начать искать альтернативные решения.

Как описано в главе 5, Jupyter — это идеальная среда для экспериментов, настройки и улучшения кода. Работа с блоками кода, запись результатов и вывода, а также написание дополнительных заметок и комментариев помогут вашим решениям в области обработки данных принимать контролируемые и воспроизводимые формы.

Сравнительный анализ с использованием timeit

Работая с примером трюка хеширования в разделе "Трюк хеширования" ранее в этой главе, вы сравнивали две альтернативы для кодирования текстовой информации в матрицу данных, которая может удовлетворить различным потребностям.

- `CountVectorizer`. Оптимально кодирует текст в матрицу данных, но не учитывает последующие нововведения в тексте.
- `HashingVectorizer`. Обеспечивает гибкость в ситуациях, когда существует вероятность того, что приложение получит новые данные, но менее оптимально, чем методы, основанные на хеш-функциях.

Хотя их преимущества достаточно очевидны с точки зрения того, как они обрабатывают данные, вы можете задаться вопросом: каково влияние

использования одного или другого на вашу обработку данных с точки зрения скорости и возможности использования памяти.

Что касается скорости, то Jupyter предлагает простое, готовое решение, строковую магию `%timeit` и магическую функцию ячейки `%%timeit`.

- `%timeit`. Рассчитывает лучшее время выполнения для инструкции.
- `%%timeit`. Вычисляет наилучшую временную производительность для всех инструкций в ячейке, кроме той, которая размещена в той же строке ячейки, что и магическая функция ячейки (которая, следовательно, может быть инструкцией инициализации).

Обе магические команды сообщают о лучшей производительности в `r` испытаниях, повторенных для `n` циклов. Когда вы добавляете параметры `-r` и `-n`, Notebook автоматически выбирает номер, чтобы предоставить быстрый ответ.

Вот пример определения времени, необходимого для присвоения списка 10**6 порядковых значений с использованием *генератора списков* (list comprehension):

```
%timeit l = [k for k in range(10**6)]
```

Сообщаемое время:

```
109 ms ± 11.8 ms per loop
(mean ± std. dev. of 7 runs, 10 loops each)
```

Результат для генератора списков может быть проверен при инкременте производительности выборки и повторений теста:

```
%timeit -n 20 -r 5 l = [k for k in range(10**6)]

After a while, the timing is reported:109 ms ± 5.43 ms per loop
(mean ± std. dev. of 5 runs, 20 loops each)
```

Для сравнения можете проверить время, необходимое для присвоения значений в цикле `for`. Поскольку для цикла `for` требуется целая ячейка, в примере используется вызов магической функции ячейки `%%timeit`. Обратите внимание, что первая строка, которая присваивает переменной значение 10**6, не учитывается в производительности.

```
%%timeit
l = list()
for k in range(10**6):
    l.append(k)
```

Итоговое время:

```
198 ms ± 6.62 ms per loop
(mean ± std. dev. of 7 runs, 10 loops each)
```

Результаты показывают, что генератор списков примерно на 50 процентов быстрее, чем при использовании цикла `for`. Затем вы можете повторить тест, используя различные стратегии кодирования текста:

```
import sklearn.feature_extraction.text as txt
htrick = txt.HashingVectorizer(n_features=20,
                               binary=True,
                               norm=None)
oh_enconder = txt.CountVectorizer()
texts = ['Python for data science',
         'Python for machine learning']
```

Выполнив начальную загрузку классов и создав их экземпляры, протестируйте два решения:

```
%timeit oh_enconded = oh_enconder.fit_transform(texts)
```

Ниже приведено время для кодировщика слов на основе `CountVectorizer`.

```
1.15 ms ± 22.5 µs per loop
(mean ± std. dev. of 7 runs, 1000 loops each)
```

Теперь запустите тест на `HashingVectorizer`:

```
%timeit hashing = htrick.transform(texts)
```

Будут получены следующие, гораздо лучшие временны*е* характеристики: время в микросекундах (µs) меньше, чем в миллисекундах (ms):

```
186 µs ± 13 µs per loop
(mean ± std. dev. of 7 runs, 10000 loops each)
```

Трюк хеширования работает быстрее, чем один унитарный кодировщик, и различие можно объяснить, отметив, что последний является оптимизированным алгоритмом, отслеживающим кодирование слов, чего не делает трюк хеширования.

Jupyter — лучшая среда для измерения скорости кода решения для науки о данных. Если вы хотите отслеживать производительность в командной строке или в сценарии, запущенном из IDE, импортируйте класс `timeit` и используйте функцию `timeit` для отслеживания производительности команды, предоставив входной параметр в виде строки.

Если вашей команде нужны переменные, классы или функции, которые недоступны в базовом языке Python (например, классы Scikit-learn), предоставьте их в качестве второго входного параметра. Сформулируйте строку, в которой Python импортирует все необходимые объекты из основной среды, как показано в следующем примере:

```
import timeit
cumulative_time = timeit.timeit(
```

```
    "hashing = htrick.transform(texts)",
    "from __main__ import htrick, texts",
    number=10000)
print(cumulative_time / 10000.0)
```

ИСПОЛЬЗОВАНИЕ ПРЕДПОЧТИТЕЛЬНОЙ ПРОГРАММЫ УСТАНОВКИ И CONDA

Python предоставляет огромное количество пакетов, которые вы можете установить. Многие из этих пакетов поставляются в виде отдельных загружаемых модулей. Некоторые из них имеют исполняемый файл, подходящий для такой платформы, как Windows, а значит, вы можете легко установить пакет. Однако многие другие пакеты полагаются на `pip`, *предпочитаемую программу установки* (preferred installer program), к которой можно обращаться прямо из командной строки.

Чтобы использовать `pip`, откройте Anaconda Prompt. Если вам нужно установить пакет с нуля, например NumPy, введите **`pip install numpy`**, и программа загрузит и установит пакет, а также все связанные с ним пакеты, с которыми он должен работать. Вы даже можете установить конкретную версию, введя, например, **`pip install -U numpy==1.14.5`**, или просто обновить пакет до последней версии, если она уже установлена: **`pip install -U numpy`**.

Если вы установили Anaconda, вместо `pip` можете использовать `conda`, что еще более эффективно, поскольку все остальные пакеты устанавливаются в правильной версии для только что установленного пакета Python (а значит, можно устанавливать, обновлять и даже понижать версию пакетов, существующих в вашей системе). Использовать `conda` для установки нового пакета также можно из командной строки Anaconda, введя команду **`conda install numpy`**. Программное обеспечение проанализирует вашу систему, сообщит об изменениях, а затем спросит, следует ли продолжить. Введите **`y`**, если хотите продолжить установку. Используйте `conda` для обновления существующих пакетов (введите **`conda update numpy`**) или всей системы (введите **`conda update --all`**).

В качестве окружения эта книга использует Jupyter. Установка и обновление при использовании Jupyter немного сложнее. Джейк ВандерПлас из Вашингтонского университета написал очень информативное сообщение об этой проблеме (`https://jakevdp.github.io/blog/2017/12/05/installing-python-packages-from-jupyter/`). В статье предлагается несколько способов установки и обновления пакетов при использовании интерфейса Jupyter. В начале, до тех пор, пока вы не приобретете уверенность и опыт, лучшим вариантом будет сначала установка и обновление системы, а также последующий запуск Jupyter, что намного облегчит и упростит установку.

Работа с профилировщиком памяти

Как вы видели при тестировании кода приложения на характеристики производительности (скорости), вы можете получить аналогичную информацию об использовании памяти. Отслеживание потребления памяти может рассказать о возможных проблемах, связанных с обработкой или передачей данных в алгоритмы обучения. Пакет `memory_profiler` реализует необходимые функции, но не предоставляется как стандартный пакет Python и требует отдельной установки. Используйте следующую команду, чтобы установить пакет непосредственно из ячейки вашего блокнота Jupyter, как объясняется в сообщении Джейка ВандерПласа, упомянутом в предыдущем разделе:

```
import sys
!{sys.executable} -m pip install memory_profiler
```

Используйте следующую команду для каждого сеанса Jupyter Notebook, который хотите отслеживать:

```
%load_ext memory_profiler
```

После выполнения этих задач вы можете легко отслеживать, сколько памяти занимает команда:

```
hashing = htrick.transform(texts)
%memit dense_hashing = hashing.toarray()
```

Пиковый объем памяти и приращение сообщают об использовании памяти:

```
peak memory: 90.42 MiB, increment: 0.09 MiB
```

Получить полный обзор потребления памяти можно, сохранив ячейку блокнота на диске, а затем профилировав ее с помощью магической строки `%mprun` для импортируемой извне функции. (Магическая строка работает только с внешними сценариями Python.) Профилирование создает подробный отчет, команда за командой, как показано в следующем примере:

```
%%writefile example_code.py
def comparison_test(text):
    import sklearn.feature_extraction.text as txt
    htrick = txt.HashingVectorizer(n_features=20,
                                   binary=True,
                                   norm=None)
    oh_enconder = txt.CountVectorizer()
    oh_enconded = oh_enconder.fit_transform(text)
    hashing = htrick.transform(text)
    return oh_enconded, hashing

from example_code import comparison_test
text = ['Python for data science',
```

```
        'Python for machine learning']
%mprun -f comparison_test comparison_test(text)
```

Вы получите вывод, похожий на этот (стандартно вывод осуществляется в отдельном окне в нижней части экрана Notebook):

```
Line # Mem usage Increment Line Contents
================================================
     1 94.8 MiB  94.8 MiB  def  comparison_test(text):
     2 94.8 MiB   0.0 MiB       import...
     3 94.8 MiB   0.0 MiB       htrick = ...
     4 94.8 MiB   0.0 MiB                ...
     5 94.8 MiB   0.2 MiB                ...
     6 94.8 MiB   0.0 MiB       oh_encoder = ...
     7 94.8 MiB   0.0 MiB       oh_encoded = ...
     8 94.8 MiB   0.0 MiB       hashing = ...
     9 94.8 MiB   0.0 MiB       return ...
```

Результирующий отчет детализирует использование памяти каждой строкой в функцией, указывая основные приращения.

СОКРАЩЕНИЕ ИСПОЛЬЗОВАНИЯ ПАМЯТИ И УСКОРЕНИЕ ВЫЧИСЛЕНИЙ

При работе с данными вы используете `arrays` из NumPy или `DataFrames` из pandas. Но даже если они выглядят как разные структуры данных, одна сфокусирована на хранение данных в виде матрицы, а другая — на обработке сложных наборов данных, хранящихся по-разному, в `DataFrames` используются массивы NumPy. Понимание того, как работают `arrays` и используются pandas, позволяет сократить использование памяти и ускорить вычисления.

Массивы NumPy — это инструмент для обработки данных с использованием непрерывных блоков памяти для хранения значений. Поскольку данные располагаются в одной и той же области компьютерной памяти, Python может быстрее извлекать и легче разделять их. Это тот же принцип, что и фрагментация диска: если данные разбросаны по диску, они занимают больше места и требуют больше времени на обработку.

В зависимости от ваших потребностей, вы можете упорядочить данные массива по строкам (стандартный выбор для языков программирования NumPy и C/C++) или по столбцам. Компьютерная память хранит ячейки одну за другой в строке. Следовательно, вы можете записывать массив строка за строкой, что позволяет быстрее выполнять обработку по строкам или столбцам. Хотя все эти детали и скрыты от глаз, они имеют решающее значение, поскольку делают работу с массивами NumPy быстрой и эффективной для науки о данных (которая использует числовые матрицы и часто вычисляет информацию по строкам). Вот почему все алгоритмы Scikit-learn ожидают массивы NumPy в ка-

честве входных данных и массивы NumPy имеют фиксированный тип данных (они могут быть только того же типа, что и последовательность данных, и не могут изменяться).

`DataFrames` из pandas — это просто упорядоченные коллекции массивов NumPy. Ваши переменные в `DataFrame`, в зависимости от типа, сжимаются в массиве `array`. Например, все ваши целочисленные переменные находятся вместе в `IntBlock`, все ваши данные с плавающей точкой — во `FloatBlock`, а остальное — в `ObjectBlock`. Это значит, что когда вы хотите работать с одной переменной, вы фактически работаете со всеми переменными. Следовательно, если у вас есть операция, которую нужно применить, лучше применить ее ко всем переменным одного типа одновременно. Кроме того, это также означает, что работа со строковыми переменными невероятно дорога с точки зрения памяти и вычислений. Даже если вы храните в переменной что-то простое, например короткую серию названий цветов, это потребует использования полной строки (не менее 50 байт), и обработка будет довольно громоздкой с использованием механизма NumPy. Как предложено в главе 7, вы можете преобразовать строковые данные в категориальные переменные; тем самым за кулисами строки превращаются в числа. Таким образом, вы значительно уменьшите использование памяти и увеличите скорость работы.

Параллельная работа на нескольких ядрах

Большинство современных компьютеров являются многоядерными (два или более процессора в одном корпусе), некоторые с несколькими физическими процессорами. Одним из наиболее важных ограничений Python является то, что стандартно он использует только одно ядро. (Он был создан в то время, когда одноядерные системы были нормой.) Проекты науки о данных требуют довольно большого количества вычислений. В частности, часть научного аспекта науки о данных опирается на повторные тесты и эксперименты с различными матрицами данных. Не забывайте, что работа с огромными объемами данных означает, что большинство трудоемких преобразований повторяют наблюдение за наблюдением (например, идентичные и несвязанные операции над различными частями матрицы).

Использование большего количества процессорных ядер ускоряет вычисления почти в два раза. Например, четыре ядра будут работать в лучшем случае в четыре раза быстрее. Вы не получите полного четырехкратного увеличения, поскольку при запуске параллельного процесса возникают накладные расходы: необходимо настроить новые экземпляры Python с правильной информацией в памяти и запустить их; следовательно, улучшение будет менее чем потенциально достижимо, но все же значительным. Поэтому знание того,

как использовать более одного процессора, является сложным, но невероятно полезным навыком для увеличения количества выполненных анализов и для ускорения операций как при настройке, так и при использовании продуктов данных.

Многопроцессорная обработка осуществляется за счет репликации одного и того же кода, а также содержимого памяти в различных новых экземплярах Python, вычисления результата для каждого из них и возврата объединенных результатов в основную исходную консоль. Если ваш исходный экземпляр уже занимает *бо*льшую часть доступной оперативной памяти, невозможно будет создать новые экземпляры, и компьютеру может не хватить памяти.

Реализация многоядерного параллелизма

Для реализации многоядерного параллелизма на языке Python вы интегрируете пакет Scikit-learn с пакетом joblib для длительных операций, таких как репликация моделей при проверке результатов, или для поиска лучших гиперпараметров. В частности, Scikit-learn допускает многопроцессорную работу в следующих случаях.

- **При перекрестной проверке**. Проверка результатов гипотезы машинного обучения с использованием различных учебных и тестовых данных.
- **При сеточном поиске**. Систематическое изменение гиперпараметров гипотезы машинного обучения и проверка полученных результатов.
- **В многозначном прогнозе**. Многократный запуск алгоритма для нескольких целей, когда одновременно можно прогнозировать много разных результатов.
- **В методах ансамблевого машинного обучения**. Моделирование большого массива классификаторов, каждый из которых независим от другого, например, при использовании моделирования на основе `RandomForest`.

Вам не нужно делать ничего особенного, чтобы воспользоваться преимуществами параллельных вычислений, — активизируйте параллелизм, установив для параметра `n_jobs` количество ядер, превышающее 1, или установив значение в –1, что означает, что вы хотите использовать все доступные экземпляры процессора.

ВНИМАНИЕ!

Если вы не запускаете код из консоли или из Jupyter Notebook, крайне важно отделить его от любого импорта пакета или назначения глобальной переменной в сценарии с помощью команды `if __name__=='__main__':` в начале любого кода, который реализует многоядерный параллелизм. Оператор `if` проверяет, выполняется ли программа напрямую или вызывается уже запущенной консолью Python, избегая путаницы или ошибок параллельного процесса (например, рекурсивного вызова параллелизма).

Демонстрация многопроцессорности

Когда вы запускаете демонстрацию многопроцессорной обработки, рекомендуется использовать Notebook, что реально сэкономит ваше время в проектах по науке о данных. Использование Jupyter обеспечивает преимущество использования магической команды `%timeit` для определения времени выполнения. Вы начинаете с загрузки многоклассового набора данных, сложного алгоритма машинного обучения (*классификатор опорных векторов* (Support Vector Classifier — SVC) и процедуры перекрестной проверки для оценки надежности итоговых оценок по всем процедурам. Подробности обо всех этих инструментах вы найдете далее в книге. Самая важная вещь, которую нужно знать, — это то, что процедуры становятся достаточно большими, потому что SVC производит 10 моделей, которые повторяются 10 раз, используя перекрестную проверку в общей сложности 100 моделей.

```
from sklearn.datasets import load_digits
digits = load_digits()
X, y = digits.data,digits.target
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

%timeit single_core = cross_val_score(SVC(), X, y, \
                                      cv=20, n_jobs=1)
```

В результате вы получаете записанное среднее время работы для одного ядра:

```
18.2 s ± 265 ms per loop
(mean ± std. dev. of 7 runs, 1 loop each)
```

После этого теста активизируйте многоядерный параллелизм и синхронизируйте результаты, используя следующие команды:

```
%timeit multi_core = cross_val_score(SVC(), X, y, \
                                     cv=20, n_jobs=-1)
```

Работа на нескольких ядрах обеспечивает лучшее среднее время:

```
10.8 s ± 137 ms per loop
(mean ± std. dev. of 7 runs, 1 loop each)
```

Пример на компьютере демонстрирует положительное преимущество применения многоядерной обработки, несмотря на использование небольшого набора данных, с которым Python проводит большую часть времени, запуская консоли и выполняя часть кода в каждой. Эти издержки, составляющие несколько секунд, все еще значительны, учитывая, что общее время выполнения увеличивается на несколько секунд. Только представьте, что произойдет, если вы работаете с большими наборами данных — время выполнения может быть легко сокращено в два или три раза.

Хотя код отлично работает с Jupyter, его можно записать в сценарии и указать Python запустить его в консоли или использовать IDE, что может вызвать ошибки из-за внутренних операций многоядерной задачи. Решение, как было упомянуто ранее, заключается в том, чтобы поместить весь код в оператор `if`, который проверяет, была ли программа запущена напрямую и не вызывалась ли позже. Пример сценария приведен ниже.

```
from sklearn.datasets import load_digits
from sklearn.svm import SVC
from sklearn.cross_validation import cross_val_score
if __name__ == '__main__':
       digits = load_digits()
       X, y = digits.data,digits.target
       multi_core = cross_val_score(SVC(), X, y,
                                    cv=20, n_jobs=-1)
```

Глава 13

Разведочный анализ данных

В ЭТОЙ ГЛАВЕ...

- Философия разведочного анализа данных (EDA)
- Числовые и категориальные распределения
- Оценка корреляции и ассоциации
- Проверка средних различий в группах
- Визуализация распределений, отношений и групп

Наука о данных опирается на сложные алгоритмы построения прогнозов и определения важных сигналов в данных, и каждый алгоритм имеет свои сильные и слабые стороны. Короче говоря, вы выбираете ряд алгоритмов, запускаете их для данных, максимально оптимизируете их параметры и решаете, какой из них лучше всего поможет создать продукт данных или получить представление о вашей задаче.

Это выглядит немного автоматическим, частично благодаря мощному аналитическому программному обеспечению и языкам сценариев, таким как Python. Алгоритмы обучения сложны, и их сложные процедуры, естественно, кажутся вам автоматическими и немного непрозрачными. Но, даже если некоторые из этих инструментов кажутся черными или даже волшебными ящиками, помните об этой простой аббревиатуре: GIGO (Garbage In/Garbage Out — мусор на входе, мусор на выходе). На протяжении долгого времени это была известная пословица в статистике (и информатике). Независимо от того,

какими мощными алгоритмами машинного обучения вы пользуетесь, вы не получите хороших результатов, если в ваших данных что-то не так.

Разведочный анализ данных (Exploratory Data Analysis — EDA) — это общий подход к исследованию наборов данных с помощью простой сводной статистики и графических визуализаций для более глубокого понимания данных. EDA помогает вам стать более эффективным в последующем анализе данных и моделировании. В этой главе описаны все необходимые базовые данные, которые помогут вам решить, как действовать, используя наиболее подходящие методы преобразования данных и выработки решения.

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_13_Exploring_Data_Analysis.ipynb`.

Подход EDA

Разведочный анализ данных (EDA) был разработан в лаборатории Bell Labs Джоном Тьюки (John Tukey), математиком и статистиком, который хотел задавать больше вопросов и осуществлять больше действий над данными на основе самих данных (разведочный мотив) в отличие от доминирующего подтверждающего подхода того времени. Подтверждающий подход основан на использовании теории или процедуры — данные предназначены только для тестирования и применения. EDA появился в конце 1970-х, задолго до появления большого потока данных. Тьюки уже видел, что некоторые виды действий, такие как тестирование и моделирование, можно было легко выполнить автоматически. В одной из своих знаменитых работ Тьюки сказал:

"*Единственный шанс людей сделать что-то ЛУЧШЕ, чем компьютеры, — это попытаться сделать МЕНЬШЕ, чем они*".

Это утверждение объясняет, почему роль человека как аналитика данных равно как и его инструменты не ограничиваются автоматическими алгоритмами обучения, но включают также ручные и творческие исследовательскими задачи. Компьютеры непобедимы в оптимизации, но люди сильны в творчестве, выбирая неожиданные пути и пробуя маловероятные, но в итоге очень эффективные решения.

Если вы ознакомились с примерами из предыдущих глав, то уже поработали над небольшим количеством данных, но использование EDA немного отличается, поскольку он проверяет не только основные предположения о работоспособности данных, что фактически включает в себя *первоначальный анализ*

данных (Initial Data Analysis — IDA). На настоящий момент в книге было показано, как

- завершить наблюдения или отметить пропущенные случаи с помощью соответствующих признаков;
- преобразовать текстовые или категориальные переменные;
- создавать новые признаки, основываясь на знании предметной области данных;
- получать числовой набор данных, где строки — это наблюдения, а столбцы — переменные.

EDA идет дальше, чем IDA. Им двигают другие отношения: выход за рамки основных предположений. С EDA вы сможете

- описать ваши данные;
- внимательно изучить распределение данных;
- понять отношения между переменными;
- обратить внимание на необычные или неожиданные ситуации;
- поместить данные в группы;
- обратить внимание на неожиданные шаблоны в группах;
- обратить внимание на различия в группах.

ЗАПОМНИ!

Далее вы много узнаете о том, как EDA может помочь узнать о распределении переменных в вашем наборе данных. *Распределение переменных* (variable distribution) — это список значений, которые вы найдете в этой переменной по сравнению с их *частотой* (frequency), т.е. как часто они встречаются. Способность определять распределение переменных многое говорит о том, как переменная может вести себя при подаче в алгоритм машинного обучения, и, таким образом, поможет предпринять соответствующие шаги для обеспечения ее эффективной работы в вашем проекте.

Определение описательной статистики для числовых данных

Первые действия, которые вы можете предпринять с данными, — это выработать некоторые искусственные меры, чтобы выяснить, что в них происходит. Вы приобретете знания о мерах, таких как максимальные и минимальные значения, и сможете определить, с каких интервалов лучше всего начинать.

Во время разведки используйте простой, но полезный набор данных Fisher's Iris, который использовался в предыдущих главах. Загрузите его из пакета Scikit-learn, используя следующий код:

```
from sklearn.datasets import load_iris
iris = load_iris()
```

Загрузив набор данных Iris в переменную пользовательского класса Scikit-learn, извлеките из него `nparray` от NumPy и `DataFrame` от pandas:

```
import pandas as pd
import numpy as np

print('Your pandas version is: %s' % pd.__version__)
print('Your NumPy version is %s' % np.__version__)
from sklearn.datasets import load_iris
iris = load_iris()
iris_nparray = iris.data

iris_dataframe = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_dataframe['group'] = pd.Series([iris.target_names[k] for k in iris.
target],
    dtype="category")
```

```
Your pandas version is: 0.23.3
Your NumPy version is 1.14.5
```


ЗАПОМНИ!

NumPy, Scikit-learn и особенно pandas являются пакетами, находящимися в постоянной разработке, поэтому перед началом работы с EDA рекомендуется проверить номера их версий. Использование более старой или более новой версии может привести к тому, что ваш вывод будет отличаться от приведенного в книге, или к отказам некоторых команд. Для этого издания книги используйте версию 0.23.3 для pandas и версию 1.14.5 для NumPy.


СОВЕТ

В этой главе представлен ряд команд pandas и NumPy, которые помогут изучить структуру данных. Несмотря на то что применение отдельных разведочных команд дает больше свободы в анализе, приятно знать, что вы можете получить бо́льшую часть этой статистики, используя метод `describe`, применяемый к вашему объекту pandas `DataFrame`: например, `print iris_dataframe.describe()`, когда вы спешите с вашим проектом науки о данных.

Измерение центральной тенденции

Среднее значение и медиана являются первыми мерами для расчета числовых переменных при запуске EDA. Они могут дать оценку, когда переменные центрированы и каким-то образом симметричны.

Используя `pandas`, вы можете быстро вычислять как медианы, так и средние значения. Ниже приведена команда для получения среднего значения из Iris `DataFrame`.

```
print(iris_dataframe.mean(numeric_only=True))
```

Результирующий вывод для среднего статистического значения таков:

```
sepal length (cm) 5.843333
sepal width (cm) 3.054000
petal length (cm) 3.758667
petal width (cm) 1.198667
```

Аналогично, вот команда, которая выведет медиану:

```
print(iris_dataframe.median(numeric_only=True))

You then obtain the median estimates for all the variables:sepal length
(cm) 5.80
sepal width (cm) 3.00
petal length (cm) 4.35
petal width (cm) 1.30
```

Медиана — это центральное положение в ряду значений. При создании переменной этот показатель менее подвержен влиянию аномальных случаев или асимметричного распределения значений вокруг среднего значения. Здесь следует обратить внимание на то, что средние значения не центрированы (ни одна переменная не имеет нулевого среднего) и медиана длины лепестка сильно отличается от средней, что требует дальнейшего исследования.

При проверке центральных тенденций вы должны:

- проверить, что средние значения равны нулю;
- проверить, отличаются ли они друг от друга;
- обратить внимание, отличается ли медиана от среднего.

Измерение дисперсии и диапазона

Далее вы должны проверить дисперсию, используя ее квадратный корень и стандартное отклонение. Стандартное отклонение столь же информативно, как и дисперсия, но сравнение со средним легче, поскольку оно выражается в тех же единицах измерения. Дисперсия является хорошим показателем того, является ли среднее значение подходящим индикатором распределения

переменной, поскольку оно говорит о том, как значения переменной распределяются вокруг среднего значения. Чем выше дисперсия, тем дальше от среднего значения вы можете ожидать появления некоторых значений.

```
print(iris_dataframe.std())
```

Вывод для каждой переменной приведен ниже.

```
sepal length (cm) 0.828066
sepal width (cm) 0.433594
petal length (cm) 1.764420
petal width (cm) 0.763161
```

Кроме того, проверьте диапазон, который представляет собой разницу между максимальным и минимальным значениями для каждой количественной переменной, и он достаточно информативен в отношении различий между переменными.

```
print(iris_dataframe.max(numeric_only=True)
    - iris_dataframe.min(numeric_only=True))
```

Здесь вы можете найти вывод предыдущей команды:

```
sepal length (cm)   3.6
sepal width (cm)    2.4
petal length (cm)   5.9
petal width (cm)    2.4
```

Обратите внимание на стандартное отклонение и диапазон по отношению к среднему значению и медиане. Стандартное отклонение или диапазон, который слишком велик по отношению к показателям центральности (среднее значение и медиана), может указывать на возможную проблему с крайне необычными значениями, влияющими на расчет, или неожиданным распределением значений вокруг среднего значения.

Работа с процентилями

Поскольку медиана — это значение в центральной позиции распределения значений, вам может потребоваться рассмотреть другие заметные позиции. Помимо минимума и максимума, позиция в 25% ваших значений (нижний квартиль) и позиция в 75% (верхний квартиль) полезны для определения распределения данных и являются основой иллюстративного графика, *диаграммы размаха* (boxplot), которая является одной из тем, обсуждаемых в этой главе.

```
print(iris_dataframe.quantile([0,.25,.50,.75,1]))
```

На рис. 13.1 показан вывод в виде матрицы — сравнение, в котором квартили используются для строк, а различные переменные набора данных —

в качестве столбцов. Итак, квартиль с 25% для длины чашелистика (sepal length) (см) равен 5,1, а это значит, что 25% значений набора данных для этого показателя меньше 5,1.

```
In [8]: print(iris_dataframe.quantile([0,.25,.50,.75,1]))

      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0.00                4.3               2.0               1.00               0.1
0.25                5.1               2.8               1.60               0.3
0.50                5.8               3.0               4.35               1.3
0.75                6.4               3.3               5.10               1.8
1.00                7.9               4.4               6.90               2.5
```

Рис. 13.1. Использование квартилей в ходе сравнения данных

ЗАПОМНИ!

Разница между верхним и нижним процентилем составляет *интерквартильный диапазон* (Interquartile Range — IQR), который является мерой шкалы переменных, представляющих наибольший интерес. Вам не нужно рассчитывать его, но вы найдете его в диаграмме размаха, поскольку он помогает определить вероятные пределы распределения. То, что лежит между нижним квартилем и минимумом, а также верхним квартилем и максимумом, является исключительно редкими значениями, которые могут отрицательно повлиять на результаты анализа. Такие редкие случаи являются выбросами и рассматриваются в главе 16.

Определение мер нормальности

Последними показательными мерами того, как структурированы числовые переменные, используемые в этих примерах, являются асимметрия и эксцесс.

- *Асимметрия* (skewness) определяет смещение данных относительно среднего значения. Если перекос отрицательный, то левый хвост слишком длинный и масса наблюдений находится на правой стороне распределения. Если он положительный — все наоборот.
- *Эксцесс* (kurtosis) показывает, имеет ли распределение данных пик и хвосты правильной формы. Если эксцесс выше нуля, распределение имеет выраженный пик. Если оно ниже нуля — распределение будет довольно плоским.

Хотя чтение чисел поможет определить форму данных, принятие к сведению таких мер представляет собой формальный тест для выбора переменных, которые могут нуждаться в некоторой корректировке или преобразовании, чтобы стать более похожими на распределение Гаусса. Позднее вы будете визуализировать данные, так что это первый шаг весьма длительного процесса.

ЗАПОМНИ!

Нормальное, или Гауссово, распределение является наиболее полезным распределением в статистике благодаря его распространенности и особым математическим свойствам. По сути, это основа многих статистических тестов и моделей, некоторые из которых, такие как линейная регрессия, широко используются в науке о данных. В гауссовом распределении среднее значение и медиана имеют одинаковые значения, значения симметрично распределены вокруг среднего значения (оно имеет форму колокола), а его стандартное отклонение указывает расстояние от среднего значения, где кривая распределения изменяется от вогнутой к выпуклый (это называется точкой перегиба). Все эти характеристики делают распределение Гаусса особенным, и их можно использовать для статистических вычислений.

СОВЕТ

Вам будет редко встречаться Гауссово распределение в ваших данных. Даже если распределение по Гауссу важно для его статистических свойств, в действительности вам придется работать с совершенно другими распределениями, поэтому необходим EDA и такие меры, как асимметрия и эксцесс.

В качестве примера на рис. 13.1 показано, что функция длины лепестка (petal length) представляет различия между средним и медианным значением (см. выше раздел "Измерение дисперсии и диапазона"). В этом разделе мы проверим тот же пример на асимметрию и эксцесс, чтобы определить, требует ли переменная вмешательства.

Выполняя тесты на асимметрию и эксцесс, вы определяете, меньше ли p-значение или равно 0,05. Если это так, вы должны отказаться от нормальности (ваша переменная распределена как распределение Гаусса), что означает возможность получить лучшие результаты, если вы попытаетесь преобразовать переменную в нормальную. Следующий код показывает, как выполнить необходимый тест:

```
from scipy.stats import skew, skewtest
variable = iris_dataframe['petal length (cm)']
s = skew(variable)
zscore, pvalue = skewtest(variable)
print('Skewness %0.3f z-score %0.3f p-value %0.3f'
      % (s, zscore, pvalue))
```

Ниже приведены оценки асимметрии, которые вы получите.

```
Skewness -0.272 z-score -1.398 p-value 0.162
```

Вы можете выполнить еще один тест на эксцесс, как показано в следующем коде:

```
from scipy.stats import kurtosis, kurtosistest
variable = iris_dataframe['petal length (cm)']
k = kurtosis(variable)
zscore, pvalue = kurtosistest(variable)
print('Kurtosis %0.3f z-score %0.3f p-value %0.3f'
      % (k, zscore, pvalue))
```

Показатели эксцесса, которые вы получите:

```
Kurtosis -1.395 z-score -14.811 p-value 0.000
```

Результаты теста говорят о том, что данные слегка смещены влево, но этого не настолько, чтобы сделать данные неприменимыми для использования. Реальная проблема заключается в том, что кривая слишком плоская, чтобы иметь форму колокола, поэтому следует изучить этот вопрос подробнее.

СОВЕТ

Рекомендуется автоматически проверять все переменные на асимметрию и эксцесс. Затем следует приступить к визуальной проверке переменных, значения которых являются самыми высокими. Ненормальность распределения также может скрывать различные проблемы, такие как выбросы в группах, которые можно заметить только с помощью графической визуализации.

Подсчет для категориальных данных

Набор данных Iris состоит из четырех метрических переменных и качественного целевого результата. Подобно тому, как вы используете средние значения и дисперсию в качестве описательных мер для метрических переменных, так и частоты строго зависят от качественных переменных.

Поскольку набор данных состоит из метрических измерений (ширина и длина в сантиметрах), вы должны сделать его качественным, разделив на ячейки в соответствии с определенными интервалами. Пакет pandas содержит две полезные функции, `cut` и `qcut`, способные преобразовать метрическую переменную в качественную:

- **`cut`** ожидает серию крайних значений, используемых для нарезания измерений, или целочисленное количество групп, используемых для нарезания переменных на группы равной ширины;
- **`qcut`** ожидает серию процентилей, используемых для нарезания переменных.

Вы можете получить новый категориальный объект DataFrame, используя следующую команду, которая осуществляет группирование (см. раздел "Понятие группирования и дискретизации" главы 9) для каждой переменной:

```
pcts = [0, .25, .5, .75, 1]
iris_binned = pd.concat(
    [pd.qcut(iris_dataframe.iloc[:,0], pcts, precision=1),
    pd.qcut(iris_dataframe.iloc[:,1], pcts, precision=1),
    pd.qcut(iris_dataframe.iloc[:,2], pcts, precision=1),
    pd.qcut(iris_dataframe.iloc[:,3], pcts, precision=1)],
    join='outer', axis = 1)
```


СОВЕТ

Этот пример основан на группировании. Тем не менее это также может помочь выяснить, когда переменная находится выше или ниже единственного граничного значения, обычно среднего значения или медианы. В этом случае вы устанавливаете pd.qcut на 0,5 процентиля или pd.cut на среднее значение переменной.


ЗАПОМНИ!

Группирование преобразует числовые переменные в категориальные. Это преобразование может улучшить понимание данных и следующей фазы машинного обучения, уменьшив шум (выбросы) или нелинейность преобразованной переменной.

Понятие частот

Чтобы получить частоту для каждой категориальной переменной набора данных, как для прогнозирующей переменной, так и для результата, используйте следующий код:

```
print(iris_dataframe['group'].value_counts())
```

Результирующие частоты показывают, что каждая группа имеет одинаковый размер:

```
virginica  50
versicolor 50
setosa     50
```

```
You can try also computing frequencies for the binned petal length that you
obtained from the previous paragraph:print(iris_binned['petal length
(cm)'].value_counts())
```

В данном случае группирование создает разные группы:

```
(0.9, 1.6]    44
(4.4, 5.1]    41
(5.1, 6.9]    34
(1.6, 4.4]    31
```

Этот пример предоставляет также некоторую базовую информацию о частоте, такую как количество уникальных значений в каждой переменной и мода частот (строки `top` и `freq` в выводе).

```
print(iris_binned.describe())
```

На рис. 13.2 приведено описание группирования.

```
In [14]: print(iris_binned.describe())

       sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)
count                150               150                150               150
unique                 4                 4                  4                 4
top           (4.2, 5.1]        (1.9, 2.8]         (0.9, 1.6]        (0.0, 0.3]
freq                  41                47                 44                41
```

Рис. 13.2. Описательная статистика для группирования

Частоты могут свидетельствовать о множестве интересных характеристик качественных признаков.

- Мода распределения частот, которая является наиболее популярной категорией.
- Другие наиболее популярные категории, особенно когда они сравнимы с модой (бимодальное распределение) или между ними существует большая разница.
- Распределение частот по категориям, если оно быстро уменьшается или распределяется поровну.
- Редкие категории.

Создание таблиц сопряженности

Сопоставляя различные категориальные распределения частот, вы можете отобразить взаимосвязь между качественными переменными. Функция `pandas.crosstab` может сопоставлять переменные или группы переменных, помогая найти возможные структуры данных или отношения.

В следующем примере мы проверим, как переменная результатов связана с длиной лепестка, и понаблюдаем за тем, как определенные результаты и классы, связанные с лепестками, никогда не появляются вместе. На рис. 13.3 показаны различные типы ирисов вдоль левой стороны вывода, за которым следует вывод, связанный с длиной лепестка.

```
print(pd.crosstab(iris_dataframe['group'],
                  iris_binned['petal length (cm)']))
```

```
In [15]: print(pd.crosstab(iris_dataframe['group'],
                           iris_binned['petal length (cm)']))

petal length (cm)  (0.9, 1.6]  (1.6, 4.4]  (4.4, 5.1]  (5.1, 6.9]
group
setosa                      44           6           0           0
versicolor                   0          25          25           0
virginica                    0           0          16          34
```

Рис. 13.3. Таблица сопряженности на основе групп и группировок

Создание прикладной визуализации для EDA

До сих пор в этой главе переменные рассматривались каждая по отдельности. Если вы следовали примерам, то создали *одномерное* (univariate) (т.е. внимание уделялось только отдельным вариациям данных) описание данных. Данные богаты информацией, поскольку они предлагают перспективу, которая выходит за рамки одной переменной, представляя больше переменных с их взаимными вариациями. Способ использования большего количества данных заключается в том, чтобы создать *двумерное* (bivariate) (с учетом того, как связаны друг с другом пары переменных) исследование. Это также основа для сложного анализа данных, основанного на *многомерном* (multivariate) (одновременно учитывающем все существующие отношения между переменными) подходе.

Если одномерный подход проверяет ограниченное количество описательной статистики, то сопоставление различных переменных или групп переменных увеличивает количество возможностей. Такое исследование перегружает аналитика различными тестами и двумерным анализом. Использование визуализации — это быстрый способ ограничить тестирование и анализ только интересующими следами и подсказками. Визуализации, использующие несколько информативных графических изображений, могут с большей легкостью передавать различные статистические характеристики переменных и их взаимные отношения.

Исследование диаграмм размаха

Диаграммы размаха (или ящики с усами) представляют распределения и их экстремальные диапазоны и показывают, находятся ли некоторые наблюдения слишком далеко от ядра данных — проблемная ситуация для некоторых алгоритмов обучения. В следующем коде показано, как создать простую диаграмму размаха с использованием набора данных Iris:

```
boxplots = iris_dataframe.boxplot(fontsize=9)
```

На рис. 13.4 приведена структура распределения каждой переменной в ее ядре, представленной процентилями 25° и 75° (стороны прямоугольника) и

медианой (в центре прямоугольника). Линии, так называемые усы, представляют 1,5-кратное IQR от боковых сторон ящика (или расстояние до наиболее экстремального значения, если в пределах 1,5-кратного IQR). На диаграмме размаха каждое наблюдение обозначается знаком за пределами усов (что считается необычным значением).

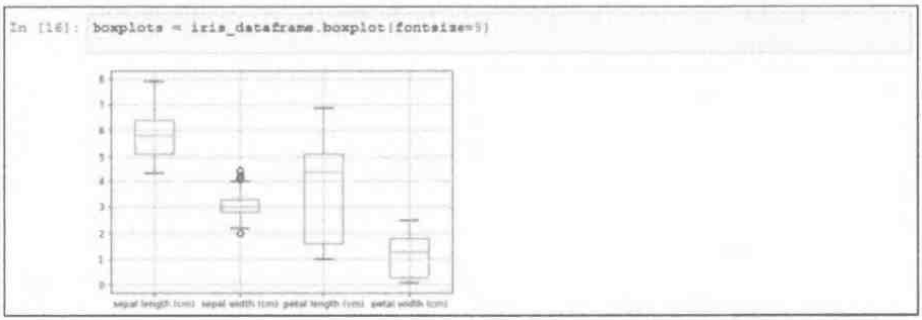


Рис. 13.4. Диаграмма размаха, упорядоченная по переменным

Диаграммы размаха также чрезвычайно полезны для визуальной проверки групп различий. Обратите внимание, что на рис. 13.5 показано, как на диаграмме размаха можно указать, что три группы, `setosa`, `versicolor` и `virginica`, имеют разную длину лепестков, причем значения только частично перекрывают края двух последних из них.

```
import matplotlib.pyplot as plt
boxplots = iris_dataframe.boxplot(
                              column='petal length (cm)',
                              by='group', fontsize=10)
plt.suptitle("")
plt.show()
```

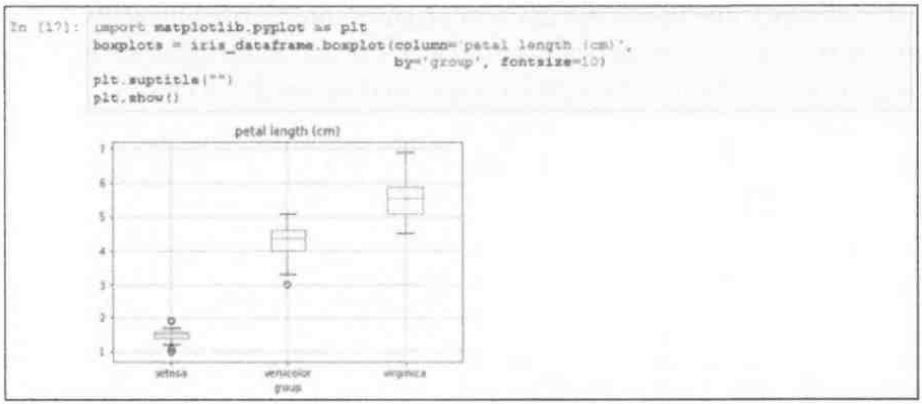


Рис. 13.5. Диаграмма размаха для лепестков, упорядоченная по группам

Поиск t-критериев после диаграмм размаха

После того как вы обнаружили возможное различие в группах относительно переменной, t-критерий (t-критерий используется в ситуациях, когда выборочная совокупность имеет точное нормальное распределение) или односторонний *дисперсионный анализ* (Analysis Of Variance — ANOVA) может предоставить статистическую проверку значимости различий между средними значениями групп.

```
from scipy.stats import ttest_ind

group0 = iris_dataframe['group'] == 'setosa'
group1 = iris_dataframe['group'] == 'versicolor'
group2 = iris_dataframe['group'] == 'virginica'
variable = iris_dataframe['petal length (cm)']

print('var1 %0.3f var2 %03f' % (variable[group1].var(),
                                variable[group2].var()))
```

```
var1 0.221 var2 0.304588
```

T-критерий сравнивает две группы одновременно, и для этого необходимо определить, имеют ли группы одинаковую дисперсию или нет. Поэтому необходимо заранее рассчитать дисперсию, например, так:

```
variable = iris_dataframe['sepal width (cm)']
t, pvalue = ttest_ind(variable[group1], variable[group2],
                      axis=0, equal_var=False)
print('t statistic %0.3f p-value %0.3f' % (t, pvalue))
```

Результирующая t-статистика и ее p-значения:

```
t statistic -3.206 p-value 0.002
```

Вы интерпретируете `pvalue` как вероятность того, что вычисленная разница `t` в статистике будет просто случайностью. Обычно, когда он ниже 0,05, можно подтвердить, что среднее значение групп значительно отличается.

Вы можете одновременно проверить более двух групп, используя односторонний критерий ANOVA. В этом случае `pvalue` имеет интерпретацию, аналогичную t-критерию:

```
from scipy.stats import f_oneway
variable = iris_dataframe['sepal width (cm)']
f, pvalue = f_oneway(variable[group0],
                     variable[group1],
                     variable[group2])
print('One-way ANOVA F-value %0.3f p-value %0.3f'
      % (f,pvalue))
```

Результат критерия ANOVA будет таким:

```
One-way ANOVA F-value 47.364 p-value 0.000
```

Наблюдение параллельных координат

Параллельные координаты (parallel coordinate) помогут определить, какие группы в выходной переменной можно легко отделить от других. Это действительно многомерный график, поскольку он представляет все данные одновременно. В следующем примере показано, как использовать параллельные координаты:

```
from pandas.plotting import parallel_coordinates
iris_dataframe['group'] = iris.target
iris_dataframe['labels'] = [iris.target_names[k]
                    for k in iris_dataframe['group']]
pll = parallel_coordinates(iris_dataframe, 'labels')
```

Как показано на рис. 13.6, на оси абсцисс отложены все количественные переменные. На ординате вы найдете все наблюдения, тщательно представленные в виде параллельных линий, каждая из которых имеет свой цвет, представляя принадлежность к другой группе.

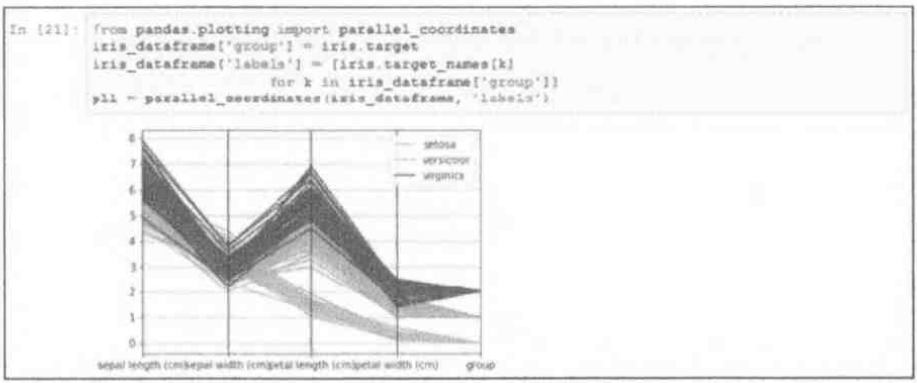


Рис. 13.6. Параллельные координаты определяют, легко ли разделить группы

Если параллельные линии каждой группы объединяются на визуализации вместе вдоль отдельной части графика вдали от других групп, то группа легко отделима. Визуализация предоставляет также средства для утверждения способности определенных признаков разделять группы.

Графики распределения

Обычно обзор полного распределения значений вы представляете на диаграмме размаха либо в виде кривой или гистограммы. Вывод на рис. 13.7

представляет все распределения в наборе данных. Сразу видны различные масштабы и формы переменных, например, тот факт, что у лепестков есть два пика.

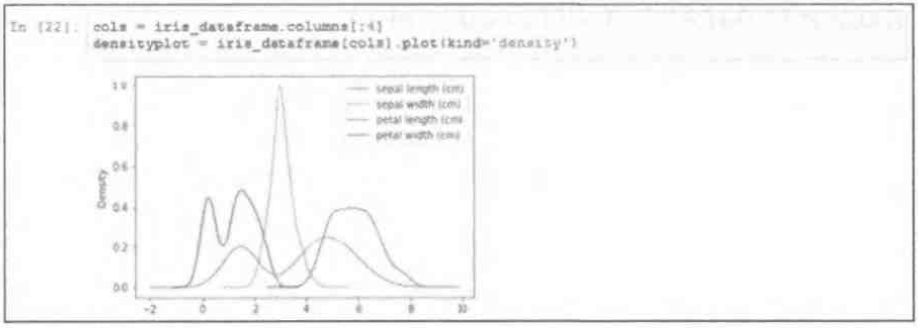


Рис. 13.7. Распределения и плотности признаков

Гистограммы представляют другой, более подробный вид распределений:

```
variable = iris_dataframe['petal length (cm)']
single_distribution = variable.plot(kind='hist')
```

На рис. 13.8 представлена гистограмма длины лепестка. Она обнаруживает пробел в распределении, который может стать многообещающим открытием, если его получится связать с определенной группой цветов ириса.

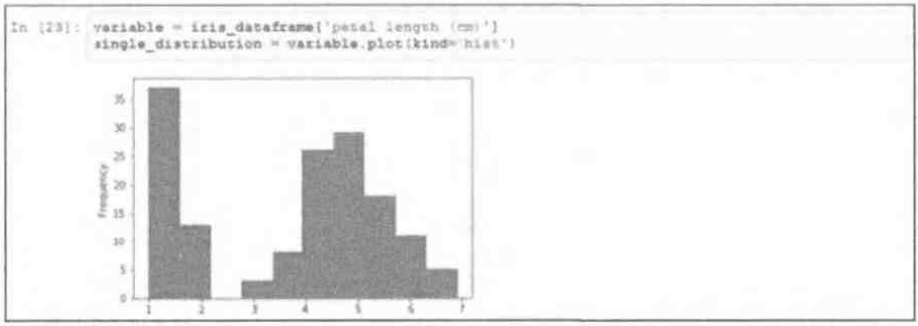


Рис. 13.8. Гистограммы могут лучше детализировать распределения

Построение диаграмм рассеяния

На диаграммах рассеяния две сравниваемые переменные дают координаты для построения наблюдений в виде точек на плоскости. Результатом обычно является облако точек. Когда облако вытянуто и напоминает линию, можно сделать вывод, что переменные коррелируют. Следующий пример демонстрирует этот принцип:

```
palette = {0: 'red', 1: 'yellow', 2:'blue'}
colors = [palette[c] for c in iris_dataframe['group']]
simple_scatterplot = iris_dataframe.plot(
                kind='scatter', x='petal length (cm)',
                y='petal width (cm)', c=colors)
```

Эта простая диаграмма рассеяния, представленная на рис. 13.9, сравнивает длину и ширину лепестков. Диаграмма рассеяния выделяет разные группы, используя разные цвета. Удлиненная форма, описанная точками, указывает на сильную корреляцию между двумя наблюдаемыми переменными, а разделение облака на группы предполагает возможность разделения групп.

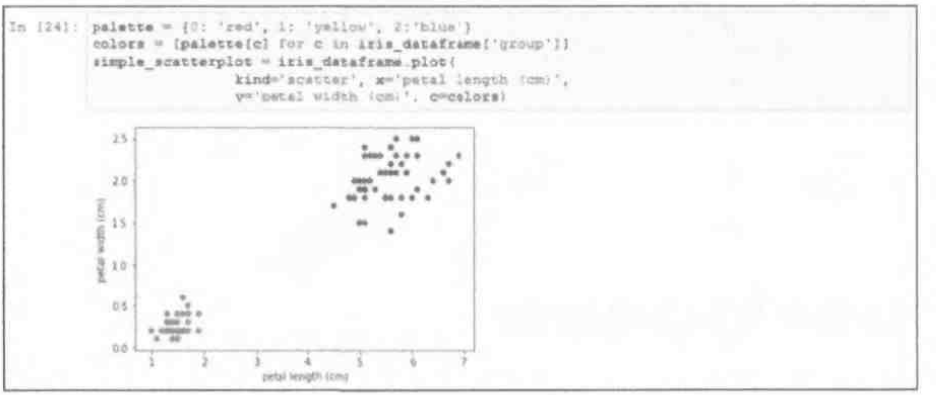


Рис. 13.9. Диаграмма рассеяния показывает, как связаны две переменные

Поскольку количество переменных не слишком велико, вы также можете автоматически создать все диаграммы рассеяния из комбинации переменных. Это представление является матрицей диаграмм рассеяния. В следующем примере показано, как ее создать:

```
from pandas.plotting import scatter_matrix
palette = {0: "red", 1: "yellow", 2: "blue"}
colors = [palette[c] for c in iris_dataframe['group']]
matrix_of_scatterplots = scatter_matrix(
    iris_dataframe, figsize=(6, 6),
    color=colors, diagonal='kde')
```

На рис. 13.10 представлена итоговая визуализация для набора данных Iris. Диагональ, представляющая оценку плотности, можно заменить гистограммой с использованием параметра `diagonal='hist'`.

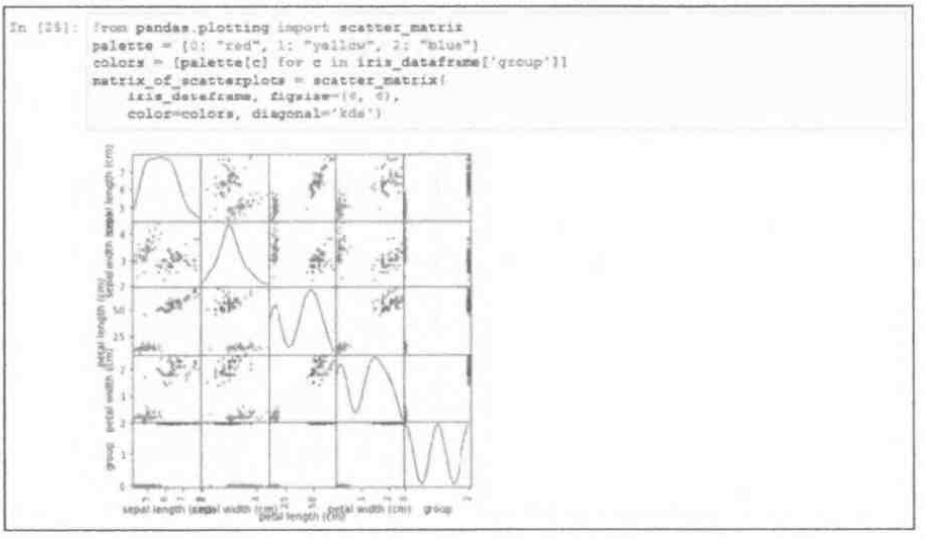


Рис. 13.10. Матрица диаграмм рассеяния отображает больше информации за один раз

Понятие корреляции

Подобно тому, как взаимосвязь переменных можно представить графически, она также измерима статистической оценкой. При работе с числовыми переменными оценкой является корреляция, и наиболее известной является *корреляция Пирсона* (Pearson's correlation) — основа для сложных моделей линейной оценки. При работе с категориальными переменными оценкой является связь, а наиболее часто используемым инструментом для измерения связи между объектами является *критерий хи-квадрат* (chi-square statistic).

Использование ковариации и корреляции

Ковариация (covariance) является первой мерой отношения двух переменных. Она определяет, имеют ли обе переменные совпадающее поведение относительно их среднего значения. Если отдельные значения двух переменных обычно выше или ниже их соответствующих средних значений, то эти две переменные имеют положительную связь. Это означает, что они склонны к согласованности, и вы можете выяснить поведение одной, глядя на другую. В таком случае их ковариация будет положительным числом, и чем выше это число, тем выше согласованность.

Если, напротив, одна переменная обычно выше, а другая ниже их соответствующих средних значений, то эти две переменные связаны отрицательно.

Хотя они не согласованы, это интересная ситуация для прогнозирования, поскольку, наблюдая за состоянием одной из них, вы можете выяснить вероятное состояние другой (хотя они и противоположны). В этом случае их ковариация будет отрицательным числом.

Третье состояние заключается в том, что две переменные не согласуются систематически или не согласуются совсем. В этом случае ковариация будет стремиться к нулю, что является признаком того, что переменные имеют мало общего или их поведение независимо.

В идеале, если у вас есть числовая целевая переменная, вы хотите, чтобы она имела высокую положительную или отрицательную ковариацию с прогнозирующими переменными. Наличие высокой положительной или отрицательной ковариации среди прогнозирующих переменных является признаком избыточности информации. *Избыточность информации* (information redundancy) сигнализирует о том, что переменные указывают на одни и те же данные, т.е. переменные говорят нам об одном и том же в несколько разных отношениях.

С помощью библиотеки `pandas` довольно просто вычислить ковариационную матрицу. Вы можете немедленно применить ее к объекту `DataFrame` набора данных Iris, как показано ниже.

```
iris_dataframe.cov()
```

Матрица на рис. 13.11 демонстрирует переменные, присутствующие как в строках, так и в столбцах. Наблюдая за различными комбинациями строк и столбцов, вы можете определить значение ковариации между выбранными переменными. Наблюдая за этими результатами, вы сразу поймете, что существует небольшая взаимосвязь между длиной и шириной чашелистика, что означает, что они имеют различные информативные значения. Тем не менее между шириной и длиной лепестка могут существовать особые отношения, но в примере не сказано, что это за отношение, поскольку меру трудно интерпретировать.

```
In [26]: iris_dataframe.cov()
```

Out[26]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	group
sepal length (cm)	0.685694	-0.039268	1.273682	0.516904	0.530872
sepal width (cm)	-0.039268	0.188004	-0.321713	-0.117981	-0.148993
petal length (cm)	1.273682	-0.321713	3.113179	1.296387	1.371812
petal width (cm)	0.516904	-0.117981	1.296387	0.582414	0.597987
group	0.530872	-0.148993	1.371812	0.597987	0.671141

Рис. 13.11. Ковариационная матрица набора данных Iris

Масштаб переменных, которые вы наблюдаете, влияет на ковариацию, поэтому вы должны использовать другую, но стандартную меру. Решение

заключается в том, чтобы использовать корреляцию, которая является оценкой ковариации после стандартизации переменных. Вот пример получения корреляции с использованием простого метода pandas:

```
iris_dataframe.corr()
```

Проверить полученную матрицу корреляции можно на рис. 13.12.

```
In [27]: iris_dataframe.corr()
```

Out[27]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	group
sepal length (cm)	1.000000	-0.109369	0.871754	0.817954	0.782561
sepal width (cm)	-0.109369	1.000000	-0.420516	-0.356544	-0.419446
petal length (cm)	0.871754	-0.420516	1.000000	0.962757	0.949043
petal width (cm)	0.817954	-0.356544	0.962757	1.000000	0.956464
group	0.782561	-0.419446	0.949043	0.956464	1.000000

Рис. 13.12. Корреляционная матрица набора данных Iris

Теперь это еще более интересно, поскольку значения корреляции находятся между –1 и +1, поэтому соотношение между шириной и длиной лепестка является положительным, а при 0,96 — почти максимально возможным.

Можно также вычислить ковариационные и корреляционные матрицы с помощью команд NumPy:

```
covariance_matrix = np.cov(iris_nparray, rowvar=0)
correlation_matrix = np.corrcoef(iris_nparray, rowvar=0)
```

ЗАПОМНИ!

В статистике такого рода корреляция является *корреляцией Пирсона* (Pearson correlation), а ее коэффициент — *коэффициентом Пирсона* (Pearson's r).

СОВЕТ

Еще один приятный трюк — выровнять корреляцию. Возводя ее в квадрат, вы теряете знак отношений. Новое число сообщает процент информации, совместно используемой двумя переменными. В этом примере корреляция 0,96 подразумевает, что 96% информации являются общими. Получить квадрат корреляционной матрицы можно с помощью команды `iris_dataframe.corr()**2`.

ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Важно помнить, что ковариация и корреляция основаны на среднем значении, поэтому они, как правило, представляют отношения, которые можно выразить с помощью линейных формул. Переменные в реальных наборах данных обычно не имеют хороших линейных формул. Напротив, они очень нелинейные, с изгибами и поворотами. Чтобы получить линейные отношения между переменными в любом случае, воспользуйтесь математическими преобразованиями.

Запомните хорошее правило: используйте корреляции только для того, чтобы установить отношения между переменными, а не исключать их.

Использование непараметрической корреляции

Корреляции могут хорошо работать, когда переменные являются числовыми, а их отношения строго линейны. Иногда функция может быть порядковой (числовая переменная, но с порядками), или вы можете подозревать некоторую нелинейность из-за ненормального распределения в данных. Возможное решение состоит в проверке сомнительной корреляции непараметрической корреляцией, такой как *ранговая корреляция Спирмена* (Spearman rank-order correlation) (а значит, у нее меньше требований с точки зрения распределения рассматриваемых переменных). Корреляция Спирмена преобразует числовые значения в ранги, а затем коррелирует ранги, сводя таким образом к минимуму влияние любых нелинейных отношений между двумя переменными. Результирующая корреляция, обычно обозначаемая как *rho*, должна интерпретироваться так же, как корреляция Пирсона.

В качестве примера проверим взаимосвязь между длиной и шириной чашелистика, у которой корреляция Пирсона была довольно слабой:

```
from scipy.stats import spearmanr
from scipy.stats.stats import pearsonr
a = iris_dataframe['sepal length (cm)']
b = iris_dataframe['sepal width (cm)']
rho_coef, rho_p = spearmanr(a, b)
r_coef, r_p = pearsonr(a, b)
print('Pearson r %0.3f | Spearman rho %0.3f'
    % (r_coef, rho_coef))
Here is the resulting comparison:Pearson r -0.109 | Spearman rho -0.159
```

В данном случае код подтверждает слабую связь между двумя переменными с использованием непараметрического критерия.

Учет критерия хи-квадрат для таблиц

Вы можете применить другой непараметрический критерий для отношений в работе со сводными таблицами. Этот критерий применим как к категориальным, так и к числовым данным (после того, как они были разбиты на группы). Статистика хи-квадрат сообщает, когда распределение в таблице двух переменных статистически сопоставимо с таблицей, в которой две переменные предположены как не связанные одна с другой (так называемая гипотеза независимости). Вот пример того, как используется эта техника:

```
from scipy.stats import chi2_contingency
table = pd.crosstab(iris_dataframe['group'],
                    iris_binned['petal length (cm)'])
chi2, p, dof, expected = chi2_contingency(table.values)
print('Chi-square %0.2f p-value %0.3f' % (chi2, p))
```

Результирующая статистика хи-квадрат такова:

```
Chi-square 212.43 p-value 0.000
```

Как было показано ранее, p-значение — это вероятность того, что разница хи-квадрат будет случайной. Высокое значение хи-квадрат и зн*а*чимое p-значение указывают на то, что переменную длины лепестка можно эффективно использовать для различения групп ирисов.


ЗАПОМНИ!

Чем больше значение критерия хи-квадрат, тем больше вероятность того, что две переменные связаны, но значение критерия хи-квадрат зависит от количества ячеек в таблице. Не используйте меру хи-квадрат для сравнения различных критериев хи-квадрат, если только не знаете, что сравниваемые таблицы имеют одинаковую форму.


СОВЕТ

Критерий хи-квадрат особенно интересен для оценки взаимосвязей между числовыми переменными в двоичном виде, даже при наличии сильной нелинейности, которая может обмануть корреляцию Пирсона. В отличии от мер корреляции, он может информировать о возможной связи, но он не предоставит четких деталей своего направления или абсолютной величины.

Изменение распределения данных

В качестве побочного продукта исследования данных на этапе EDA можно сделать следующее.

- Создать новый признак из комбинации разных, но связанных переменных.
- Найти скрытые группы или странные значения, скрывающиеся в данных.
- Опробовать некоторые полезные модификации распределений данных с помощью группировки (или других дискретизаций, таких как бинарные переменные).

При выполнении EDA необходимо учитывать важность преобразования данных при подготовке к этапу обучения, что также означает использование определенных математических формул. Большинство алгоритмов машинного

обучения работают лучше всего при максимальной корреляции Пирсона между переменными, которые вы должны предсказать, и переменной, которую вы используете для их предсказания. В следующих разделах представлен обзор наиболее распространенных процедур, используемых во время EDA для улучшения взаимосвязи между переменными. Преобразование данных, которое вы выбираете, зависит от фактического распределения данных, поэтому это не то, что вы решаете заранее, скорее вы обнаружите это с помощью EDA и многократного тестирования. Кроме того, в этих разделах подчеркивается необходимость согласования процесса преобразования с математической формулой, которую вы используете.

Использование разных статистических распределений

В своей практике науки о данных вы встретитесь с широким спектром различных распределений, одни из которых связаны с теорией вероятностной, а другие — нет. Для некоторых распределений предполагается, что они должны вести себя как нормальные, а для других — нет, и это может быть проблемой, в зависимости от того, какие алгоритмы используются для процесса обучения. Как правило, если модель представляет собой линейную регрессию или часть семейства линейных моделей, поскольку она сводятся к суммированию коэффициентов, следует учитывать как стандартизацию переменных, так и преобразование распределения.


ЗАПОМНИ!

Помимо линейных моделей, многие другие алгоритмы машинного обучения безразличны к распределению переменных, которые вы используете. Однако преобразование переменных в наборе данных, чтобы сделать их распределение более похожим на гауссово, может привести к положительным эффектам.

Создание стандартизации z-оценки

В процессе EDA вы, возможно, поняли, что ваши переменные имеют разные масштабы и неоднородны по своему распределению. Как следствие, вам необходимо преобразовать переменные таким образом, чтобы они были легко сопоставимы при анализе:

```
from sklearn.preprocessing import scale
variable = iris_dataframe['sepal width (cm)']
stand_sepal_width = scale(variable)
```


ЗАПОМНИ!

Некоторые алгоритмы будут работать неожиданным образом, если вы не масштабируете переменные с помощью стандартизации. Как правило, обращайте внимание на любые линейные модели,

кластерный анализ и любой алгоритм, который претендует на то, чтобы основываться на статистических показателях.

Преобразование других известных распределений

Когда вы проверяете на корреляцию переменные с высокой асимметрией и эксцессом, результаты могут вас разочаровать. Как вы узнали ранее в этой главе, использование непараметрической меры корреляции, такой как коэффициент Спирмена, может рассказать о двух переменных больше, чем коэффициент Пирсона. В этом случае следует преобразовать обретенное знание в новый признак:

```
from scipy.stats.stats import pearsonr
tranformations = {'x': lambda x: x,
                  '1/x': lambda x: 1/x,
                  'x**2': lambda x: x**2,
                  'x**3': lambda x: x**3,
                  'log(x)': lambda x: np.log(x)}
a = iris_dataframe['sepal length (cm)']
b = iris_dataframe['sepal width (cm)']
for transformation in tranformations:
    b_transformed = tranformations[transformation](b)
    pearsonr_coef, pearsonr_p = pearsonr(a, b_transformed)
    print('Transformation: %s \t Pearson\'s r: %0.3f'
        % (transformation, pearsonr_coef))
```

```
Transformation:  x        Pearson's r:  -0.109
Transformation:  x**2     Pearson's r:  -0.122
Transformation:  x**3     Pearson's r:  -0.131
Transformation:  log(x)   Pearson's r:  -0.093
Transformation:  1/x      Pearson's r:   0.073
```

При изучении различных возможных преобразований использование цикла `for` может сказать вам, что степенное преобразование увеличит корреляцию между двумя переменными, увеличивая таким образом производительность алгоритма линейного машинного обучения. Вы также можете выполнить другие дополнительные преобразования, такие как квадратный корень `np.sqrt(x)`, экспонента `np.exp(x)` и различные комбинации всех преобразований, такие как `np.log(1/x)`.

СОВЕТ

Логарифмическое преобразование переменной иногда сопряжено с трудностями, поскольку оно не будет работать для отрицательных или нулевых значений. Поэтому нужно изменить масштаб значений переменной так, чтобы минимальное значение было равно 1. Этого можно добиться с помощью некоторых функций из пакета NumPy: `np.log(x + np.abs(np.min(x)) + 1)`.

Глава 14

Уменьшение размерности

В ЭТОЙ ГЛАВЕ...

- Магия разложения по сингулярным значениям
- Различие между факторами и компонентами
- Автоматическое получение и сопоставление изображений и текста
- Создание системы рекомендации фильмов

Большие данные (big data) определяются как наборы данных, настолько огромные, что их становится трудно обрабатывать с использованием традиционных методов. Манипулирование большими данными отличают статистические задачи, основанные на небольших выборках, кроме задач науки о данных. Обычно для небольших задач используют традиционные статистические методы и методы обработки данных — для больших.

Данные можно считать большими, поскольку они состоят из множества примеров, и это первый вид больших данных, который приходит в голову спонтанно. Анализ базы данных для миллионов клиентов и одновременное взаимодействие с ними — это действительно сложная задача, но это не единственно возможная перспектива больших данных. Другой вид больших данных — это размерность данных, а именно количество аспектов, отслеживаемых приложением. Данные с высокой размерностью могут предлагать множество признаков (переменных), зачастую сотни и даже тысячи. И это может превратиться в настоящую проблему. Даже если вы наблюдаете только несколько случаев

в течение короткого времени, слишком большое количество признаков может сделать анализ невозможным.

Сложность работы с таким большим количеством измерений приводит к необходимости использования различных методов обработки данных для фильтрации информации — сохранения тех данных, которые, кажется, лучше решают задачу. Фильтр уменьшает размерность, удаляя избыточную информацию в наборах данных большого размера. В этой главе основное внимание уделяется сокращению измерений данных, когда в них слишком много повторений одной и той же информации. Вы можете считать это сокращение своего рода сжатием информации, которое похоже на сжатие файлов на жестком диске для экономии места.


СОВЕТ

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_14_Reducing_Dimensionality.ipynb`.

Понятие SVD

Ядром магии сокращения данных являются такие операции линейной алгебры, как *разложение по сингулярным значениям* (Singular Value Decomposition — SVD). SVD — это математический метод, который получает на входе данные в виде одной матрицы и возвращает три результирующие матрицы, которые, будучи перемножены, возвращают исходную матрицу. (Краткое введение в SVD см. по адресу `https://machinelearningmastery.com/singular-value-decompositionfor-machine-learning/`.) Формула SVD:

$$M = U * s * Vh$$

Ниже кратко объясняются буквы, используемые в уравнении.

- **U.** Содержит всю информацию о строках (наблюдениях).
- **Vh.** Содержит всю информацию о столбцах (признаках).
- **s.** Записывает процесс SVD (тип журнальной записи).

Создание трех матриц из одной кажется контрпродуктивным, когда целью является уменьшение измерений данных. При использовании SVD вы, похоже, создаете больше данных, а не сокращаете их. Тем не менее магия SVD скрыта в процессе, поскольку, строя эти новые матрицы, он отделяет информацию о строках от столбцов исходной матрицы и в результате сжимает всю ценную информацию в первые столбцы новых матриц.

Полученная матрица s показывает, как произошло сжатие. Сумма всех значений в s говорит вам, сколько информации сохранено ранее в исходной матрице, а каждое значение в r сообщает, сколько данных накоплено в каждом соответствующем столбце U и Vh.

Чтобы понять, как все это работает, нужно рассмотреть отдельные значения. Например, если сумма s равна 100, а первое значение s равно 99, это значит, что 99% информации теперь хранится в первом столбце U и Vh. Таким образом, вы можете с радостью отбросить все оставшиеся столбцы после первого, не теряя никакой информации, важной для процесса обнаружения знаний в контексте науки о данных.

В поисках уменьшения размерности

Пришло время посмотреть, как Python может помочь уменьшить сложность данных. В следующем примере демонстрируется метод сокращения больших данных. Вы можете использовать эту технику и во многих других интересных случаях.

```
import numpy as np
A = np.array([[1, 3, 4], [2, 3, 5], [1, 2, 3], [5, 4, 6]])
print(A)
```

Код выводит матрицу A, которая появляется в следующих примерах:

```
[[1 3 4]
 [2 3 5]
 [1 2 3]
 [5 4 6]]
```

Матрица A содержит данные, которые вы хотите уменьшить. Она состоит из четырех наблюдений, содержащих по три признака в каждом. Используя модуль linalg из NumPy, вы можете получить доступ к функции svd, которая точно разделяет исходную матрицу на три переменные: U, s и Vh.

```
U, s, Vh = np.linalg.svd(A, full_matrices=False)
print(np.shape(U), np.shape(s), np.shape(Vh))
print(s)
```

Вывод перечисляют формы U, s и Vh соответственно и выводят содержимое переменной s:

```
(4, 3) (3,) (3, 3)
[12.26362747 2.11085464 0.38436189]
```

Матрица U, представляющая строки, имеет четыре значения строки. Матрица Vh является квадратной, и три ее строки представляют исходные столбцы. Матрица s является диагональной. Диагональная матрица содержит

нули в каждом элементе, кроме диагонали. Длина ее диагонали точно равна длине трех исходных столбцов. Внутри s вы обнаружите, что большинство значений находятся в первых элементах, что указывает на то, что в первом столбце содержится больше всего информации (около 83%), во втором есть некоторые значения (около 14%), а в третьем — остаток значений. Чтобы получить эти проценты, вы складываете три значения и получаете 14,758844, а затем используете это число для разделения отдельных столбцов. Например, 12,26362747/14,758844 составляет 0,8309341483655495, или около 83%.

Чтобы проверить, выполняет ли SVD свои обещания, просмотрите вывод примера. В примере восстанавливается исходная матрица с использованием функции NumPy dot для умножения U, s (диагональ) и Vh. Функция dot осуществляет матричное умножение, представляющее собой процедуру умножения, немного отличающуюся от арифметической. Вот пример полной реконструкции матрицы:

```
print(np.dot(np.dot(U, np.diag(s)), Vh))
```

Код выводит восстановленную исходную матрицу A:

```
[[ 1. 3. 4.]
 [ 2. 3. 5.]
 [ 1. 2. 3.]
 [ 5. 4. 6.]]
```

Реконструкция идеальна, но, очевидно, вам нужно сохранить в результирующей матрице U то же количество компонентов, что и в переменных, как в исходном наборе данных. Никакого уменьшения размерности на самом деле не произошло, вы просто реструктурировали данные таким образом, чтобы новые переменные не коррелировали (и это полезно для алгоритмов кластеризации, как вы узнаете в главе 15).

При работе с SVD вы обычно заботитесь о результирующей матрице U, представляющей строки, поскольку она заменяет ваш начальный набор данных.

Теперь пришло время немного поиграть с результатами и добиться реального снижения. Например, посмотрим, что произойдет, если исключить из матрицы U третий столбец, менее важный из всех. В следующем примере показано, что происходит при удалении последнего столбца из всех трех матриц:

```
print np.round(np.dot(np.dot(U[:,:2], np.diag(s[:2])),
                      Vh[:2,:]),1) # k=2 реконструкция
```

Код выводит реконструкцию исходной матрицы с использованием первых двух компонентов:

```
[[ 1. 2.8 4.1]
 [ 2. 3.2 4.8]
 [ 1. 2.  3. ]
 [ 5. 3.9 6. ]]
```

Вывод почти идеален. Это значит, что вы можете удалить последний компонент и использовать U как идеальную замену оригинальному набору данных. Здесь есть несколько знаков после запятой. Чтобы продолжить этот пример, следующий код удаляет второй и третий столбцы из матрицы U:

```
print np.round(np.dot(np.dot(U[:,:1], np.diag(s[:1])),
               Vh[:1,:]),1) # k=1 реконструкция
```

Ниже приведена реконструкция исходной матрицы с использованием одного компонента.

```
[[ 2.1 2.5 3.7]
 [ 2.6 3.1 4.6]
 [ 1.6 1.8 2.8]
 [ 3.7 4.3 6.5]]
```

Теперь ошибок больше. В некоторых элементах матрицы пропущено несколько десятичных знаков. Однако бо́льшая часть числовой информации не повреждена, и вы можете смело использовать матрицу U вместо исходных данных. Просто представьте себе возможность применения такого метода к матрице большего размера, содержащей сотни столбцов, которые сначала можно преобразовать в матрицу U, а затем безопасно отбросить большинство столбцов.

Один из серьезных вопросов, который следует учитывать, — это определение количества столбцов. Создание накопленной суммы диагональной матрицы s (функция cumum NumPy идеально подходит для этой задачи) полезно для отслеживания того, как выражается информация и во сколько столбцов. Как правило, следует рассмотреть решения, обеспечивающие от 70 до 99% исходной информации; но это не строгое правило — оно зависит от того, насколько важно для вас восстановить исходный набор данных.

Использование SVD для измерения невидимого

Свойство SVD заключается в том, чтобы сжимать исходные данные на таком уровне и таким интеллектуальным способом, что в определенных ситуациях техника может действительно создавать новые значимые и полезные признаки, а не только сжатые переменные. Следовательно, вы могли бы использовать три столбца матрицы U в предыдущем примере в качестве новых признаков.

Если данные содержат подсказки и намеки о скрытой причине или мотиве, SVD может собрать их вместе и предложить правильные ответы и идеи. Это особенно верно, когда данные состоят из интересных фрагментов информации, подобных приведенным ниже.

- **Текст в документах возможно содержит идеи и значимые категории**. Точно так же, как вы можете составить свое мнение о рассматриваемых темах, читая блоги и группы новостей, SVD поможет сделать содержательную классификацию групп документов или конкретных тем, о которых написано в каждой из них.
- **Отзывы о конкретных фильмах или книгах возможно содержит ваши личные предпочтения и для других категорий продуктов**. Поэтому, если вы скажете, что вам понравилась оригинальная коллекция сериалов *Star Trek* ("Звездный путь") на рейтинговом сайте, будет легко определить, что вам нравится в отношении других фильмов, потребительских товаров или даже типов личности.

Примером метода, основанного на SVD, является *латентное семантическое индексирование* (Latent Semantic Indexing — LSI), которое успешно используется для связывания документов и слов на основе того, что слова, хотя и различаются, как правило, имеют одинаковое значение в аналогичных контекстах. Этот тип анализа предлагает не только синонимичные слова, но и более высокие концепции группирования. Например, анализ LSI по некоторым спортивным новостям может объединять бейсбольные команды высшей лиги исключительно на основе совпадения названий команд в аналогичных статьях, без каких-либо предварительных знаний о том, что такое бейсбольная команда или высшая лига.

Выполнение факторного анализа и PCA

SVD работает непосредственно с числовыми значениями в данных, но данные также можно выражать как отношения между переменными. У каждого признака есть определенная вариация. Вы можете рассчитать изменчивость как меру дисперсии вокруг среднего. Чем больше дисперсия, тем больше информации содержится в переменной. Кроме того, если вы поместите переменную в набор, то сможете сравнить дисперсию двух переменных, чтобы определить, коррелируют ли они, а это является показателем того, насколько сильно сходные значения они имеют.

Проверяя все возможные корреляции переменной с другими в наборе, вы можете обнаружить, что может быть два типа дисперсии.

- **Уникальная дисперсия**. Некая вариация уникальна для рассматриваемой переменной. Это не может быть связано с тем, что происходит с любой другой переменной.
- **Общая дисперсия**. Некая вариация является общей для одной или нескольких других переменных, создавая избыточность данных. Избыточность подразумевает, что вы можете найти одну и ту же информацию с немного разными значениями в различных признаках и во многих наблюдениях.

Конечно, следующий шаг — определить причину общей дисперсии. Попытка определить, как бороться с уникальными и общими вариациями, привела к созданию факторного анализа и *анализа основных компонентов* (Principal Component Analysis — PCA).

Психометрическая модель

Задолго до того, как были придуманы многие алгоритмы машинного обучения, появилась *психометрия* (psychometric) — дисциплина в психологии, которая занимается психологическим измерением, пытаясь найти статистическое решение для эффективного измерения параметров личности. Наша личность, как и другие аспекты, не поддается непосредственному измерению. Например, невозможно точно измерить, насколько человек интровертен или интеллектуален. Анкеты и психологические тесты только намекают на эти значения.

Психологи знали о SVD и пытались применить его к задачам. Общие вариации привлекли их внимание: они думали, что если некоторые переменные почти одинаковы, то они должны иметь одну и ту же базовую причину. Для решения этой задачи психологи создали *факторный анализ* (factor analysis) и, вместо того, чтобы применять SVD непосредственно к данным, применили его к недавно созданной матрице, отслеживающей общую дисперсию, в надежде на сжатие всей информации и восстановление новых полезных признаков, *факторов* (factor).

В поисках скрытых факторов

Для демонстрации использования факторного анализа начнем с набора данных Iris:

```
from sklearn.datasets import load_iris
from sklearn.decomposition import FactorAnalysis
iris = load_iris()
X = iris.data
Y = iris.target
cols = [s[:12].strip() for s in iris.feature_names]
factor = FactorAnalysis(n_components=4).fit(X)
```

После загрузки данных и сохранения всех прогностических функций класс `FactorAnalysis` инициализируется запросом на поиск четырех факторов. Затем данные подбираются. Вы можете изучить результаты, наблюдая за атрибутом `components_`, который возвращает массив, содержащий показатели взаимосвязи между вновь созданными факторами, размещенными в строках, и исходными объектами, размещенными в столбцах:

```
import pandas as pd
print(pd.DataFrame(factor.components_, columns=cols))
```

В выходных данных вы обнаружите, как факторы, создаваемые кодом, указанным в строках, связаны с исходными переменными, изображенными в столбцах. Вы можете интерпретировать числа, как если бы они были корреляциями:

```
   sepal length   sepal width   petal length   petal width
0      0.707227     -0.153147       1.653151      0.701569
1      0.114676      0.159763      -0.045604     -0.014052
2     -0.000000      0.000000       0.000000      0.000000
3     -0.000000      0.000000       0.000000     -0.000000
```

Положительное число на пересечении каждого фактора и признака указывает, что между ними существует положительная пропорция; отрицательное число указывает на то, что они расходятся и что одно противоречит другому. Например, при тестировании набора данных Iris результирующие факторы должны составлять максимум 2, а не 4, поскольку только два фактора имеют существенную связь с исходными признаками. Вы можете использовать эти два фактора в качестве новых переменных в вашем проекте, поскольку они отражают невидимую, но важную особенность, на которую только намекали ранее доступные данные.


СОВЕТ

Вам придется протестировать разные значения `n_components`, поскольку вы не можете знать, сколько факторов существует в данных. Если алгоритму требуется большее количества факторов, чем существует, он будет генерировать факторы с низкими или нулевыми значениями в массиве `components_`.

Использование компонентов, а не факторов

Если SVD может быть успешно применен к общей дисперсии, вы можете задаться вопросом: почему нельзя применить его ко всем дисперсиям? Используя слегка измененную исходную матрицу, все отношения в данных могут быть уменьшены и сжаты аналогично тому, как это делает SVD. Результаты этого процесса, которые очень похожи на SVD, называют *анализом основных*

компонентов (Principal Component Analysis — PCA). Вновь созданные признаки называются *компонентами* (component). В отличие от факторов, компоненты не описываются как основная причина структуры данных, а представляют собой реструктурированные данные, поэтому их можно рассматривать как большое интеллектуальное суммирование выбранных переменных.

Для приложений, работающих с данными, PCA и SVD очень похожи. Тем не менее PCA не зависит от масштаба исходных признаков (поскольку он работает на показателях корреляции, которые располагаются между –1 и +1), а PCA фокусируется на восстановлении взаимосвязи между переменными, предлагая, таким образом, различные результаты из SVD.

Уменьшение размерности

Процедура получения PCA очень похожа на факторный анализ. Различие в том, что вы не указываете количество компонентов для извлечения. Сколько компонентов оставить, вы решаете позже, после проверки атрибута `explained_variance_ratio_`, который обеспечивает количественную оценку (в процентах) информативного значения каждого извлеченного компонента. В следующем примере показано, как решить эту задачу:

```
from sklearn.decomposition import PCA
import pandas as pd
pca = PCA().fit(X)
print('Explained variance by each component: %s'
      % pca.explained_variance_ratio_)
print(pd.DataFrame(pca.components_,
                   columns=iris.feature_names))
```

В выходных данных вы можете наблюдать, как исходная дисперсия набора данных распределяется по компонентам (например, здесь на первый компонент, изначально имеющейся в наборе данных, приходится 92,5% дисперсии) и результирующей матрице компонентов PCA, где каждый компонент (отображается в строках) относится к каждой исходной переменной (размещается в столбцах):

```
Explained variance by each component:
    [0.92461621 0.05301557 0.01718514 0.00518309]
   sepal length    sepal width   petal length   petal width
0      0.361590      -0.082269       0.856572      0.358844
1      0.656540       0.729712      -0.175767     -0.074706
2     -0.580997       0.596418       0.072524      0.549061
3      0.317255      -0.324094      -0.479719      0.751121
```

В этом разложении набора данных Iris векторный массив, представленный `explained_variance_ratio_`, указывает, что бо́льшая часть информации

сосредоточена в первом компоненте (92,5%). Вы видели такой же результат после факторного анализа. Таким образом, можно сократить весь набор данных до двух компонентов, что снизит уровень шума и избыточной информации по сравнению с исходным набором данных.

Сжатие информации с использованием t-SNE

Поскольку SVD и PCA уменьшают сложность данных, вы можете использовать их для визуализации. Но диаграммы рассеяния PCA зачастую не помогают в визуализации, поскольку вам нужно куда больше графиков, чтобы увидеть, как связаны между собой примеры. Поэтому для визуализации отношений в сложных наборах данных из сотен переменных с использованием простых двумерных диаграмм рассеяния ученые создали алгоритмы *нелинейного уменьшения размерности* (nonlinear dimensionality reduction) (или *обучения многообразием* (manifold learning)), такие как t-SNE.

Алгоритм t-SNE начинает со случайного проецирования данных в указанное количество измерений (обычно два для двумерного представления) в виде точек. Затем в серии итераций алгоритм пытается объединить точки, которые ссылаются на аналогичные примеры в наборе данных (сходство рассчитывается с использованием вероятности), и выделить точки, которые слишком отличаются одна от другой. После нескольких итераций похожие точки должны располагаться в кластерах, отделенных от других точек. Такое расположение помогает представлять данные в виде графика, чтобы получить представление о данных и их значении.

В этом примере используется набор данных рукописных цифр из Scikit-learn. Набор данных содержит изображения рукописных чисел в оттенках серого, представленных в виде матрицы значений 8-х-8 в диапазоне от нуля до единицы. (Это оттенки, где нуль — абсолютно черный, а единица — белый.)

```
from sklearn.datasets import load_digits
digits = load_digits()
X = digits.data
ground_truth = digits.target
```

После загрузки набора данных запустите алгоритм t-SNE, чтобы сжать данные:

```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2,
            init='pca',
            random_state=0,
            perplexity=50,
            early_exaggeration=25,
```

```
            n_iter=300)
Tx = tsne.fit_transform(X)
```

В этом примере устанавливаются начальные параметры `perplexity`, `early_exaggeration` и `n_iter`, которые влияют на качество конечного представления. Вы можете опробовать разные значения этих параметров и получить несколько разных решений. Когда набор данных будет уменьшен, нанесите его на график и поместите метку исходного номера в ту область графика, где находится большинство подобных примеров:

```
import numpy as np
import matplotlib.pyplot as plt
plt.xticks([], [])
plt.yticks([], [])
for target in np.unique(ground_truth):
    selection = ground_truth==target
    X1, X2 = Tx[selection, 0], Tx[selection, 1]
    c1, c2 = np.median(X1), np.median(X2)
    plt.plot(X1, X2, 'o', ms=5)
    plt.text(c1, c2, target, fontsize=18)
```

На рис. 14.1 приведен результирующий график, на котором показано, как некоторые рукописные числа, такие как нуль, шесть или четыре, легко отличить от других, тогда как такие числа как три и девять (или пять и восемь), легче было бы интерпретировать неправильно.

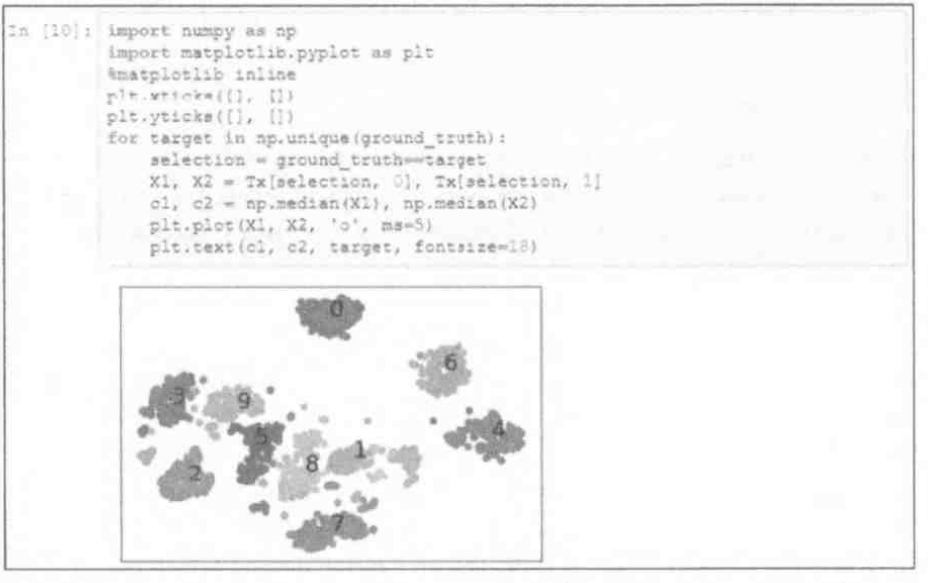


Рис. 14.1. Результирующая проекция набора данных рукописных цифр по алгоритму t-SNE

Понимание некоторых приложений

Довольно сложно понять алгоритмы, составляющие семейство методов разложения данных, полученных из SVD, из-за сложности их математического аппарата и многочисленных вариантов (таких, как Factor, PCA и SVD). Несколько примеров помогут понять, как лучше всего использовать эти мощные инструменты для обработки данных. В следующих разделах описаны алгоритмы, которые вы, вероятно, видели в действии, когда

- выполняли поиск изображений в поисковой системе или публиковали изображения в социальной сети;
- автоматически маркировали сообщения блогов или вопросов на веб-сайтах вопросов и ответов;
- получали рекомендаций по покупке на сайтах электронной коммерции.

Распознавание лиц с помощью PCA

В следующем примере показано, как использовать изображения лиц для объяснения того, как социальные сети маркируют изображения соответствующей меткой или именем:

```
from sklearn.datasets import fetch_olivetti_faces
dataset = fetch_olivetti_faces(shuffle=True, random_state=101)
train_faces = dataset.data[:350,:]
test_faces = dataset.data[350:,:]
train_answers = dataset.target[:350]
test_answers = dataset.target[350:]
```

Пример начинается с импорта набора данных лиц Olivetti — набора изображений, доступных из Scikit-learn. Для этого эксперимента код делит набор маркированных изображений на обучающий и тестовый. Вам нужно сделать вид, что вы знаете метки обучающего набора, но ничего не знаете о тестовом. В результате вы хотите связать изображения из тестового набора с наиболее похожим изображением из обучающего набора.

Набор данных Olivetti состоит из 400 фотографий 40 человек (таким образом, для каждого человека есть 10 фотографий). Несмотря на то что фотографии представляют одного и того же человека, каждая фотография сделана в разное время дня, с разным освещением, выражением лица или деталями (например, в очках и без них). Размеры изображения составляют 64×64 пикселя, поэтому при развертывании пикселей в элементы создается набор данных, состоящий из 400 случаев и 4096 переменных. Вы можете получить дополнительную информацию о наборе данных, используя вызов `print(dataset.`

DESCR), как показано в загружаемом исходном коде. Дополнительную информацию о наборе данных см. на веб-страницах AT&T Laboratories Cambridge: https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html. Следующий фрагмент кода преобразует и уменьшает изображения с использованием алгоритма PCA от Scikit-learn:

```
from sklearn.decomposition import RandomizedPCA
n_components = 25
Rpca = PCA(svd_solver='randomized',
           n_components=n_components,
           whiten=True)
Rpca.fit(train_faces)
print('Explained variance by %i components: %0.3f'
 % (n_components, np.sum(Rpca.explained_variance_ratio_)))
compressed_train_faces = Rpca.transform(train_faces)
compressed_test_faces = Rpca.transform(test_faces)
```

При запуске код выводит долю дисперсии для первых 25 компонентов результирующего PCA, объясняя дисперсию по 25 компонентам: 0,794.

Класс RandomizedPCA — это приблизительная версия PCA, которая работает лучше, когда набор данных большой (много строк и переменных). Разложение создает 25 новых переменных (параметр n_components) и отбеливание (whiten=True), удаляя таким образом некоторый постоянный шум (создаваемый текстовой и фото гранулярностью) из изображений. Результирующее разложение использует 25 компонентов, что составляет около 80% информации, хранящейся в 4096 признаках.

```
import matplotlib.pyplot as plt
%matplotlib inline

photo = 17
print('The represented person is subject %i'
      % test_answers[photo])
plt.subplot(1, 2, 1)
plt.axis('off')
plt.title('Unknown photo '+str(photo)+' in test set')
plt.imshow(test_faces[photo].reshape(64,64),
           cmap=plt.cm.gray, interpolation='nearest')
plt.show()
```

На рис. 14.2 представлен субъект № 34, чья фотография № 17 была выбрана в качестве тестового набора.

После разложения тестового набора пример приложения получает данные, относящиеся только к фотографии 17, и вычитает их из разложения тренировочного набора. Теперь тренировочный набор состоит из различий по сравнению с примером фото. Код возводит их в квадрат (для удаления отрицательных

```
In [14]: import matplotlib.pyplot as plt
         %matplotlib inline

         photo = 17
         print('We are looking for face id=%i'
               % test_answers[photo])
         plt.subplot(1, 2, 1)
         plt.axis('off')
         plt.title('Unknown face '+str(photo)+' in test set')
         plt.imshow(test_faces[photo].reshape(64,64),
                    cmap=plt.cm.gray, interpolation='nearest')

         We are looking for face id=34

Out[14]: <matplotlib.image.AxesImage at 0xb94f9b0>
```

Рис. 14.2. Пример приложения пытается найти похожие фотографии

значений) и суммирует по строкам, что приводит к серии суммированных ошибок. Наиболее похожими являются фотографии с наименьшими квадратами ошибок, отличия которых наименьшие.

```
mask = compressed_test_faces[photo,]
squared_errors = np.sum((compressed_train_faces
                         - mask)**2, axis=1)
minimum_error_face = np.argmin(squared_errors)
most_resembling = list(np.where(squared_errors < 20)[0])
print('Best resembling subject in training set: %i'
      % train_answers[minimum_error_face])
```

Предыдущий код возвращает номер кода наиболее похожего человека в наборе данных, который фактически соответствует коду субъекта, выбранного из набора тестов:

```
Best resembling subject in training set: 34
```

Вы проверяете работу, проделанную кодом, отображая фотографию 17 из тестового набора рядом с тремя верхними изображениями из обучающего набора, которые лучше всего напоминают его (рис. 14.3):

```
import matplotlib.pyplot as plt
plt.subplot(2, 2, 1)
plt.axis('off')
plt.title('Unknown face '+str(photo)+' in test set')
plt.imshow(test_faces[photo].reshape(64, 64),
```

```
            cmap=plt.cm.gray,
            interpolation='nearest')
for k,m in enumerate(most_resembling[:3]):
    plt.subplot(2, 2, 2+k)
    plt.title('Match in train set no. '+str(m))
    plt.axis('off')
    plt.imshow(train_faces[m].reshape(64, 64),
        cmap=plt.cm.gray,
        interpolation='nearest')
plt.show()
```

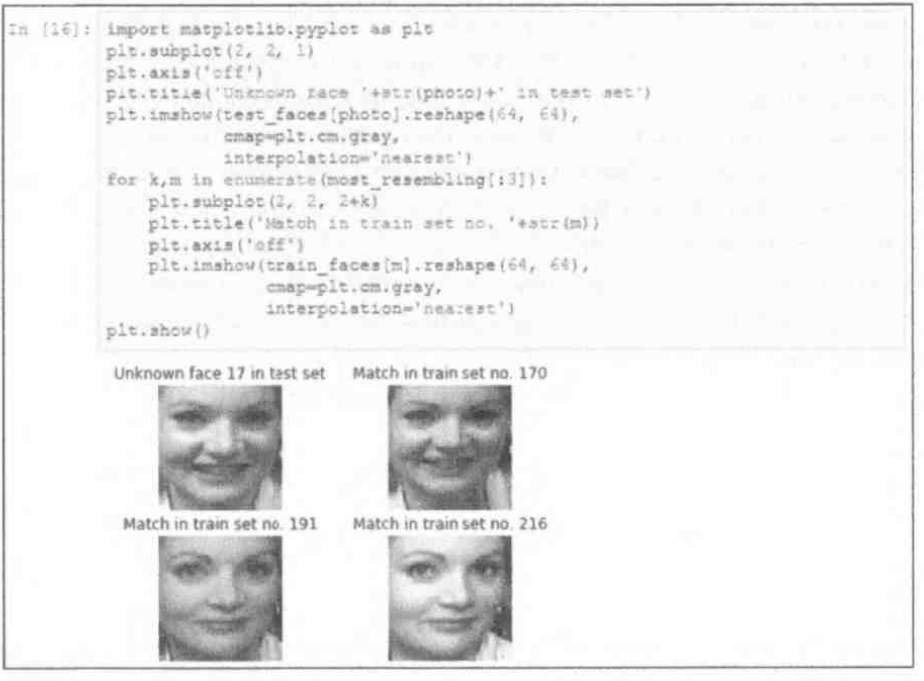


Рис. 14.3. Вывод демонстрирует результаты, которые напоминают тестовое изображение

Даже несмотря на то, что наиболее похожая фотография из учебных данных — это просто измененная в масштабе версия той, которая представлена в тестовом наборе, на двух других фотографиях показано другое изображение того же человека, что и на тестовой фотографии 17. В этом примере использования PCA, начиная с образцового изображения, точно находятся другие фотографии того же человека из набора изображений.

Извлечение тем с использованием NMF

Текстовые данные — еще одна область применения семейства алгоритмов сокращения данных. Первоначальная идея заключается в том, что если люди

говорят или пишут о некой теме, они, как правило, используют слова из ограниченного набора, относящегося к этой теме. Следовательно, если у вас есть коллекция текстов и вы не знаете, к каким темам они относятся, вы можете искать группы слов, которые имеют тенденцию связываться друг с другом, поэтому группа слов, образованная методами уменьшением размерности, дает представление о темах, к которым они могли бы относиться.

Это идеальное применение семейства SVD, поскольку за счет уменьшения количества столбцов признаков (в документе слова являются элементами) они будут объединяться в измерениях, и вы сможете найти темы, проверив слова, набирающие баллы. SVD и PCA предоставляют признаки, которые положительно и отрицательно связаны с вновь созданными измерениями. Таким образом, результирующая тема может быть выражена наличием слова (высокая положительная ценность) или его отсутствием (высокая отрицательная ценность), что делает интерпретацию сложной и нелогичной для человека. К счастью, Scikit-learn включает в себя класс разложения для *неотрицательной матричной факторизации* (Non-Negative Matrix Factorization — NMF), которая позволяет исходному признаку только положительно соотноситься с получаемыми измерениями.

Этот пример начинается с загрузки набора данных 20newsgroups, выбора только сообщений, касающихся объектов для продажи, и автоматического удаления верхних и нижних колонтитулов, а также кавычек:

```
from sklearn.datasets import fetch_20newsgroups
dataset = fetch_20newsgroups(shuffle=True,
                  categories = ['misc.forsale'],
                  remove=('headers', 'footers', 'quotes'),
                  random_state=101)
print('Posts: %i' % len(dataset.data))
```

Код загружает набор данных и выводит количество сообщений, которые он содержит:

```
Posts: 585
```

Класс `TfidVectorizer` импортируется и настраивается для удаления стоп-слов (общих слов, таких как *the* или *and*) и сохранения только отличительных слов, создавая матрицу, столбцы которой указывают на разные слова.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF

vectorizer = TfidfVectorizer(max_df=0.95, min_df=2,
                             stop_words='english')
tfidf = vectorizer.fit_transform(dataset.data)
n_topics = 5
nmf = NMF(n_components=n_topics,
          random_state=101).fit(tfidf)
```

ЗАПОМНИ!

Член `tfidf` (*частота термина–инверсная частота в документе* — term frequency–inverse document frequency — TF-IDF) представляет собой простой расчет, основанный на частоте слова в документе. Это взвешенная редкость слова в доступных документах. Взвешивание слов — это эффективный способ исключения слов, которые не могут помочь классифицировать или идентифицировать документ при обработке текста. Например, можно исключить общие части речи или другие общие слова.

Как и в случае других алгоритмов из модуля `sklearn.decomposition`, параметр `n_components` указывает количество требуемых компонентов. Если вы хотите искать больше тем, используйте большее число. По мере увеличения необходимого количества тем метод `reconstruction_err_` сообщает о более низкой частоте появления ошибок. Вам решать, когда остановиться, учитывая компромисс между большим количеством времени, потраченным на вычисления, и большим количеством тем.

Последняя часть сценария выводит пять итоговых тем. Читая выведенные слова, вы можете выбрать значение извлеченных тем благодаря характеристикам продукта (например, слова *drive* (диск), *hard* (жесткий диск), *card* (плата) и *floppy* (дискета) относятся к компьютерам) или к конкретному продукту (например, *comics* (комиксы), *car* (автомобиль), *stereo* (стерео), *games* (игры)).

```
feature_names = vectorizer.get_feature_names()
n_top_words = 15
for topic_idx, topic in enumerate(nmf.components_):
    print('Topic #%d:' % (topic_idx+1),)
    topics = topic.argsort()[:-n_top_words - 1:-1]
    print(' '.join([feature_names[i] for i in topics]))
```

Темы отображаются в порядке, сопровождаемом наиболее характерными ключевыми словами. Вы можете исследовать полученную модель, изучая атрибут `components_` из обученной модели NMF. Он состоит из NumPy `ndarray`, содержащего положительные значения для связанных с темой слов. Используя метод `argsort`, вы можете получить индексы главных ассоциаций, столь высокие значения которых указывают на то, что они являются наиболее представительными словами. Этот код извлекает индексы лучших репрезентативных слов для темы 1:

```
print(nmf.components_[0,:].argsort()[:-n_top_words-1:-1])
```

Вывод представляет собой список индексов, каждый из которых соответствует слову:

```
[2463 740 2200 2987 2332 853 3727 3481 2251 2017 556 842
 2829 2826 2803]
```

Декодирование индексов слов создает читаемые строки, вызывая их из массива, полученного из метода `get_feature_names`, примененного к установленному ранее `TfidfVectorizer`. В следующем фрагменте показано, как извлечь слово, относящееся к индексу 2463, верхнему репрезентативному слову темы 1:

```
word_index = 2463
print(vectorizer.get_feature_names()[word_index])
```

Слово, относящееся к индексу 2463 — это "offer" (предложение):

```
offer
```

Рекомендация фильмов

Другим интересным применением для сокращения данных являются системы, которые вырабатывают рекомендации для вещей, о которых вы, возможно, хотели бы узнать побольше или которые хотели бы купить. Скорее всего, вы увидите рекомендации на большинстве веб-сайтов электронной коммерции после входа в систему и посещения нескольких страниц товаров. По мере просмотра вы оцениваете товары или кладете их в свою электронную корзину. Основываясь на этих действиях и действиях других клиентов, вы видите другие возможности покупки (это метод *совместной фильтрации* (collaborative filtering)).

Вы можете реализовать совместные рекомендации, основываясь на простых средствах или расчете частот на основе приобретенных предметов других клиентов или на основе рейтингов с использованием SVD. Такой подход помогает надежно вырабатывать рекомендации даже в случае товаров, которые продавец редко продает или которые являются совершенно новыми для пользователей. В этом примере используется хорошо известная база данных, созданная веб-сайтом MovieLens и собранная из оценок пользователями фильма, который им понравился или не понравился. Поскольку это внешний набор данных, сначала загрузите его по адресу `http://files.grouplens.org/datasets/movielens/ml-1m.zip`. После загрузки базы данных извлеките ее в рабочий каталог Python. Вы обнаружите свой рабочий каталог с помощью таких команд:

```
import os
print(os.getcwd())
```

Обратите внимание на отображаемый каталог и распакуйте базу данных `ml-1m`. Затем выполните следующий код:

```
import pandas as pd
from scipy.sparse import csr_matrix
users = pd.read_table('ml-1m/users.dat', sep='::',
        header=None, names=['user_id', 'gender',
```

```
        'age', 'occupation', 'zip'], engine='python')
ratings = pd.read_table('ml-1m/ratings.dat', sep='::',
          header=None, names=['user_id', 'movie_id',
          'rating', 'timestamp'], engine='python')
movies = pd.read_table('ml-1m/movies.dat', sep='::',
         header=None, names=['movie_id', 'title',
         'genres'], engine='python')
MovieLens = pd.merge(pd.merge(ratings, users), movies)
```

Используя pandas, код загружает различные таблицы данных, а затем объединяет их на основе признаков с одинаковым именем (переменные user_id и movie_id).

```
ratings_mtx_df = MovieLens.pivot_table(values='rating',
        index='user_id', columns='title', fill_value=0)
movie_index = ratings_mtx_df.columns
```

Пакет pandas также поможет создать в строках сводную информацию о пользователях и о названиях фильмов в столбцах. Индекс фильма будет отслеживать, какой фильм представляет каждый столбец.

```
from sklearn.decomposition import TruncatedSVD
recom = TruncatedSVD(n_components=15, random_state=101)
R = recom.fit_transform(ratings_mtx_df.values.T)
```

Класс TruncatedSVD сокращает объем данных до десяти компонентов. Этот класс предлагает более масштабируемый алгоритм, чем linalg.svd из SciPy, используемый в предыдущих примерах. Класс TruncatedSVD вычисляет результирующие матрицы именно той формы, которую вы определили с помощью параметра n_components (полные результирующие матрицы не рассчитываются), что приводит к более быстрому выводу и меньшему использованию памяти.

Вычисляя матрицу Vh, вы можете уменьшить оценки разных, но схожих пользователей (оценки каждого пользователя выражены строкой) в сжатые измерения, которые воссоздают общие вкусы и предпочтения. Обратите также внимание на то, что поскольку вас интересует матрица Vh (сокращение столбцов/фильмов), но алгоритм предоставляет вам только матрицу U (разложение на основе строк), вам необходимо ввести транспонирование данных (в ходе транспонирования столбцы становятся строками, и вы получаете вывод TruncatedSVD, который является матрицей Vh). Теперь вы ищете конкретный фильм:

```
movie = 'Star Wars: Episode V \
- The Empire Strikes Back (1980)'
movie_idx = list(movie_index).index(movie)
print("movie index: %i" %movie_idx)
print(R[movie_idx])
```

Вывод указывает на индекс эпизода *Star Wars* ("Звездные войны") и его координаты SVD:

```
movie index: 3154
[184.72254552 -17.77612872 47.33450866 51.4664494
 47.92058216 17.65033116 14.3574635 -12.82219207
 17.51347857 5.46888807 7.5430805 -0.57117869
 -30.74032355 2.4088565 -22.50368497]
```

Используя метку фильма, вы можете узнать, в каком столбце находится фильм (в данном случае индекс столбца 3154), и вывести значения 10 компонентов. Эта последовательность обеспечивает профиль фильма. Теперь вы пытаетесь получить все фильмы с баллами, аналогичными целевому фильму и с сильной корреляцией с ним. Хорошая стратегия подразумевает вычисление матрицы корреляции всех фильмов, получение фрагмента, относящегося к вашему фильму, и выяснения, какие из них наиболее связаны (характеризуются высокой положительной корреляцией — скажем, по крайней мере 0,98) с названием фильмов, используя индексацию:

```
import numpy as np
correlation_matrix = np.corrcoef(R)
P = correlation_matrix[movie_idx]
print(list(movie_index[(P > 0.985) & (P < 1.0)]))
```

Код возвращает названия фильмов, наиболее похожие на ваш фильм (они предназначены для предложений, основанных на предпочтениях к данному фильму):

```
['Raiders of the Lost Ark (1981)',
 'Star Wars: Episode IV - A New Hope (1977)',
 'Star Wars: Episode V - The Empire Strikes Back (1980)',
 'Star Wars: Episode VI - Return of the Jedi (1983)',
 'Terminator, The (1984)']
```

Поклонникам "Звездных войн" хотелось бы иметь несколько названий, таких как "Звездные войны". *Эпизод IV* и *VI* (конечно). Кроме того, фанатам может понравиться *Raiders of the Lost Ark* ("Индиана Джонс: В поисках утраченного ковчега") из-за актера Харрисона Форда, сыгравшего главного героя во всех этих фильмах.

SVD всегда найдет лучший способ связать строку или столбец в данных, обнаружив сложные взаимодействия или отношения, которых вы раньше не представляли. Вам не нужно ничего придумывать заранее; это подход полностью основаны на данных.

Глава 15

Кластеризация

В ЭТОЙ ГЛАВЕ...

- Возможности кластеризации без учителя
- Как кластеризация методом k-средних работает с маленькими и большими данными
- Применение DBScan в качестве альтернативы

Одной из основных способностей, которой люди обладали с первобытных времен, является разделение известного мира на отдельные классы, в которых отдельные объекты имеют общие черты, которые классификатор считает важными. Начиная с примитивных обитателей пещер, классифицирующих природный мир, в котором они жили, выделяя растения и животных, полезных или опасных для их выживания, мы переходим к современному времени, когда отделы маркетинга классифицируют потребителей на целевые сегменты, а затем применяют надлежащие маркетинговые планы.

Классификация имеет решающее значение для нашего процесса создания новых знаний, поскольку, собирая похожие объекты, мы можем:

- упоминать все предметы в классе одинаковыми названиями;
- суммировать релевантные признаки по типу класса;
- связывать определенные действия или вспоминать определенные знания автоматически.

Сегодня работа с большими потоками данных требует той же способности классификации, но в другом масштабе. Чтобы обнаружить неизвестные группы сигналов, присутствующих в данных, нам нужны специализированные алгоритмы, способные научиться относить примеры к определенным классам (*подход с учителем* (supervised approach)) и обнаруживать новые интересные

классы, о которых мы ранее не знали (*обучение без учителя* (unsupervised learning)).

Несмотря на то что ваша основная задача в качестве аналитика данных будет заключаться в применении на практике ваших навыков прогнозирования, вам также необходимо будет найти в ваших данных информацию о возможной новой полезной информации. Например, вам часто нужно будет находить новые признаки, чтобы усилить прогнозирующую силу ваших моделей, найти простой способ проводить сложные сравнения в данных и находить сообщества в социальных сетях.

Управляемый данными подход (data-driven approach) классификации, *кластеризация* (clustering), окажет большую помощь в достижении успеха вашим проектом данных, когда вам необходимо обеспечить новое понимание с нуля и не хватает маркированных данных или вы хотите создать для них новые метки.

Технологии кластеризации — это набор методов *классификации без учителя* (unsupervised classification), способных создавать значимые классы в ходе непосредственной обработки данных, без каких-либо предварительных знаний или гипотез о группах, которые в них могут присутствовать. Если все *контролируемые алгоритмы с учителем* (supervised algorithm) нуждаются в маркированных примерах (метках классов), *алгоритмы без учителя* (unsupervised algorithm) способны сами определить, какими могут быть наиболее подходящие метки.

СОВЕТ

Кластеризация поможет обобщить огромное количество данных. Это эффективный метод представления данных нетехнической аудитории и обеспечения контролируемого алгоритма группами переменных, что дает им концентрированную и значимую информацию.

Есть несколько видов кластеризации. Чтобы различить их, используйте рекомендации, приведенные ниже.

- Присвоение каждого примера уникальной группе (разбиение) или нескольким (*нечеткая кластеризация* (fuzzy clustering)).
- Определение эвристики, т.е. эмпирического правила, используемого для выяснения, является ли пример частью группы.
- Определение того, как они количественно определяют разницу между наблюдениями, т.е. так называемую *меру расстояния* (distance measure).

В большинстве случаев вы будете использовать методы *разделяющей кластеризации* (partition-clustering) (точка данных может быть частью только одной группы, поэтому группы не пересекаются; членство в них раздельно), а среди методов разделения — метод *k*-средних (k-means). Кроме того, в этой

главе упоминаются другие полезные методы, основанные на агломерационных методах и плотности данных.

Агломерационные методы (agglomerative method) устанавливают данные в кластеры на основе метрики расстояния. Подходы *плотности данных* (data density) основаны на той идее, что группы очень плотные и непрерывные, поэтому, если вы заметите уменьшение плотности при исследовании части группы точек, это может означать, что вы достигли одной из ее границ.


СОВЕТ

Поскольку вы обычно не знаете, что ищете, разные методы могут предоставить вам разные решения и точки зрения на данные. Секрет успешной кластеризации в том, чтобы опробовать как можно больше способов, сравнить результаты и попытаться найти причину, по которой вы можете рассматривать одни наблюдения как группу по отношению к другим.


СОВЕТ

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_15_Clustering.ipynb`.

Кластеризация методом k-средних

Метод k-средних (k-means) — это итеративный алгоритм, ставший очень популярным в машинном обучении благодаря своей простоте, скорости и масштабируемости для большого количества точек данных. Алгоритм k-средних основывается на идее, что существует определенное количество групп данных, называемых *кластерами* (cluster). Каждая группа данных распределена вокруг центральной точки, которая имеет некоторые ключевые характеристики.

Фактически вы можете рассматривать центральную точку скопления, *центроид* (centroid), как солнце. Точки данных распределяются вокруг центроида, как планеты. Поскольку звездные системы разделены космической пустотой, ожидается, что скопления также четко отделяются друг от друга, поэтому как группы точек они внутренне однородны и отличаются друг от друга.


ЗАПОМНИ!

Алгоритм k-средних предполагает поиск кластеров в данных. Поэтому он найдет их даже тогда, когда их не будет. Чтобы определить, является ли группа настоящим золотым самородком, необходима проверка внутри групп.

С учетом того, что вам достаточно указать количество ожидаемых групп (вы можете использовать предположение или опробовать несколько возможных

решений), алгоритм k-средних будет искать их, используя эвристику для установления положения центральных точек.

Центроиды скоплений должны быть очевидны по различию их характеристик и по положениям относительно друг друга. Даже если вы начнете со случайного угадывания, где они могут быть, в конце концов, после нескольких коррекций, вы всегда найдете их, используя множество тяготеющих к ним точек данных.

Понятие алгоритмов на основе центроидов

Процедура нахождения центроидов проста.

1. **Угадайте количество кластеров k.**

   K центроидов выбираются из точек данных случайным образом или так, чтобы они помещались в данные в очень отдаленных друг от друга положениях. Все остальные точки присваиваются ближайшему центроиду на основе евклидова расстояния.

2. **Сформируйте начальные кластеры.**

3. **Пересчитывайте кластеры до тех пор, пока не заметите, что ваше решение больше не меняется.**

   Вы пересчитываете центроиды как среднее значение всех точек, присутствующих в группе. Все точки данных переназначаются группам в зависимости от расстояния до новых центроидов.

Итеративный процесс назначения случаев наиболее вероятному центроиду и последующего усреднения назначенных точек для поиска нового центроида будет медленно смещать положение центроида в те области, к которым тяготеет большинство точек данных. В результате вы получите истинное положение центроида.

Процедура имеет только два слабых места, которые нужно учитывать. Сначала вы выбираете начальные центроиды случайным образом, а значит вы можете начать с плохой начальной точки. В результате итеративный процесс остановится на каком-то маловероятном решении — например, расположении центроида между двумя группами. Чтобы убедиться, что ваше решение является наиболее вероятным, опробуйте алгоритм несколько раз и отследите результаты. Чем больше попыток, тем больше шансов подтвердить правильное решение. Реализация метода k-средних пакета Scikit-learn для языка Python сделает это за вас, так что вам просто нужно решить, сколько раз вы собираетесь пробовать. (Противоречие в том, что большее количество итераций дает лучшие результаты, но каждая итерация требует драгоценного времени.) Второе слабое место связано с *евклидовым расстоянием* (Euclidean distance),

которое использует метод k-средних; оно является расстоянием между двумя точками на плоскости (концепция, которую вы, вероятно, изучали в школе). В приложении k-средних каждая точка данных является вектором объектов, поэтому при сравнении расстояния двух точек сделайте следующее.

1. Создайте список, содержащий различия элементов в двух векторах.
2. Возведите в квадрат все элементы разностного вектора.
3. Рассчитайте квадратный корень из суммируемых элементов.

Вы можете опробовать простой пример на языке Python. Представьте, что есть две точки, A и B, и у них есть три числовых элемента. Если A и B являются данными двух человек, их отличительные признаки могут быть измерены по росту (см), весу (кг) и возрасту (годы), как показано в следующем коде:

```
import numpy as np
A = np.array([165, 55, 70])
B = np.array([185, 60, 30])
```

В следующем примере показано, как рассчитать различия между тремя элементами, возвести в квадрат все полученные элементы и определить квадратный корень из суммированных квадратов значений:

```
D = (A - B)
D = D**2
D = np.sqrt(np.sum(D))
print(D)
```

```
45.0
```

В конце концов, евклидово расстояние — это действительно большая сумма. Когда составляющие вектор разности переменные значительно отличаются друг от друга по масштабу (в этом примере высота могла бы быть выражена в метрах), в результате получается расстояние, в котором преобладают элементы с наибольшим масштабом. Очень важно изменить масштаб переменных так, чтобы перед применением алгоритма k-средних они использовали сходный масштаб. Для решения этой задачи можно использовать фиксированный диапазон или статистическую нормализацию с нулевым средним и единицей дисперсии.

Другая возможная проблема связана с корреляцией между переменными, что приводит к избыточности информации. Если две переменные имеют сильную корреляцию, это означает, что часть их информационного содержания повторяется. Репликация подразумевает подсчет одной и той же информации более одного раза в сумме, используемой для вычисления расстояния. Если вы не знаете о проблеме корреляции, некоторые переменные будут доминировать в расчете показателя расстояния — ситуация, которая может привести к тому,

что вы не найдете нужные кластеры. Решение состоит в том, чтобы удалить корреляцию благодаря алгоритму уменьшения размерности, такому как *анализ основных компонентов* (Principal Components Analysis — PCA). Вы должны помнить, что следует оценить масштаб и корреляцию, прежде чем применять метод k-средних и другие методы кластеризации, использующие евклидову меру расстояния.

Пример с данными изображения

Пример с данными изображения демонстрирует, как применить инструменты и получить представление о кластерах. Идеальный пример — кластеризация набора рукописных цифр, предоставляемого пакетом Scikit-learn. Рукописные числа естественно отличаются друг от друга — они обладают изменчивостью в том смысле, что есть несколько способов написания определенных цифр. Конечно, у всех нас разный подчерк, поэтому вполне естественно, что цифры, написанные каждым человеком немного различаются. Следующий код показывает, как импортировать данные изображения:

```
from sklearn.datasets import load_digits
digits = load_digits()
X = digits.data
ground_truth = digits.target
```

Пример начинается с импорта набора данных цифр из пакета Scikit-learn и присвоения данных переменной. Затем он сохраняет метки в другой переменной для последующей проверки. Следующим шагом является обработка данных с использованием PCA.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
pca = PCA(n_components=30)
Cx = pca.fit_transform(scale(X))
print('Explained variance %0.3f'
      % sum(pca.explained_variance_ratio_))
```

```
Explained variance 0.893
```

Применяя PCA к масштабированным данным, код решает проблемы масштаба и корреляции. Несмотря на то что PCA может воссоздать то же количество переменных, что и в исходных данных, пример кода отбрасывает некоторые с помощью параметра `n_components`. Решение использовать 30 компонентов по сравнению с исходными 64 переменными позволяет сохранить бо́льшую часть исходной информации (около 90% исходной вариации в данных) и упростить набор данных, удалив корреляцию, сократив избыточные переменные и их шум.

СОВЕТ

В оставшейся части главы используется набор данных `Cx`. Если вам нужно запустить отдельные примеры кода из этой главы, сначала потребуется запустить код, представленный ранее в этом разделе.

В этом примере преобразованные PCA данные отображаются в переменной `Cx`. После импорта класса `KMeans` код определяет его основные параметры:

- `n_clusters` — количество k центроидов, которые нужно найти;
- `n_init` — количество попыток метода k-средних с разными начальными центроидами. Код должен опробовать процедуру достаточное количество раз, например 10, как показано здесь:

```
from sklearn.cluster import KMeans
clustering = KMeans(n_clusters=10,
                    n_init=10, random_state=1)
clustering.fit(Cx)
```

После создания параметров класс кластеризации готов к использованию. Вы можете применить метод `fit()` к набору данных `Cx`, который создает масштабированный и сокращенный набор данных.

Поиск оптимального решения

Как упоминалось в предыдущем разделе, в примере кластеризуется десять разных чисел. Пришло время проверить сначала решение с $k = 10$. Следующий код сравнивает предыдущий результат кластеризации с *истинной правдой* (ground truth) — истинными метками, чтобы определить, есть ли какое-либо соответствие:

```
import numpy as np
import pandas as pd
ms = np.column_stack((ground_truth,clustering.labels_))
df = pd.DataFrame(ms,
                  columns = ['Ground truth','Clusters'])
pd.crosstab(df['Ground truth'], df['Clusters'],
            margins=True)
```

Преобразование решения, заданного переменной `labels`, внутренней для класса `clustering`, в `pandas DataFrame`, позволяет применить сводную таблицу и сравнивать исходные метки с метками, полученными из кластеризации (рис. 15.1). Поскольку строки представляют истинную правду, вы можете искать числа, большинство из наблюдений которых разделены между различными кластерами. Эти наблюдения являются примерами, созданными вручную, которые сложнее понять с помощью метода k-средних.

```
In [5]: import numpy as np
        import pandas as pd
        ms = np.column_stack((ground_truth,clustering.labels_))
        df = pd.DataFrame(ms,
                          columns = ['Ground truth','Clusters'])
        pd.crosstab(df['Ground truth'], df['Clusters'],
                    margins=True)
```

Out[5]:

Clusters	0	1	2	3	4	5	6	7	8	9	All
Ground truth											
0	177	0	0	1	0	0	0	0	0	0	178
1	0	27	0	0	0	1	0	96	58	0	182
2	1	41	6	0	1	0	0	24	4	0	177
3	0	1	160	0	7	8	0	7	0	0	183
4	0	0	0	157	4	2	0	2	7	9	181
5	0	0	39	2	0	137	2	2	0	0	182
6	1	0	0	0	0	0	174	5	1	0	181
7	0	0	0	0	157	1	0	1	3	17	179
8	1	1	46	0	2	7	2	103	12	0	174
9	0	0	144	0	8	4	0	2	19	3	180
All	180	70	395	160	179	160	178	242	104	29	1797

Рис. 15.1. Сводная таблица кластеров истинной правды и k-средних

Обратите внимание, что цифры, такие как шесть и нуль, сосредоточены в одном главном кластере, тогда как другие, такие как три, девять и множество примеров пяти и восьми, как правило, собираются в одну группу, кластер 1. Кластер 9 состоит из одного числа четыре (пример, который настолько отличается от всех остальных, так как у него есть свой кластер). Из этого открытия вы можете сделать вывод, что некоторые рукописные цифры легко распознать, а другие нет.

СОВЕТ

Сводная таблица была особенно полезна в этом примере, поскольку вы можете сравнить результаты кластеризации с истинной правдой. Тем не менее во многих приложениях кластеризации у вас не будет никакой истинной правды для сравнения. В таких случаях особенно полезно представлять значения переменных с помощью найденных вами центроидов кластера. Для решения этой задачи можете использовать описательную статистику, среднее значение или медиану (см. главу 13) к каждому кластеру и сравнивать их различные описательные характеристики.

Еще одно наблюдение, которое вы можете сделать, — это то, что, хотя в данном примере всего десять цифр, существует куда больше типов рукописных

форм каждого, следовательно, необходимо найти больше кластеров. Конечно, проблема в том, чтобы определить, сколько кластеров вам нужно.

Для измерения жизнеспособности кластера используйте инерцию. *Инерция* (inertia) — это сумма всех различий между каждым членом кластера и его центроидом. Если примеры в группе похожи на центроид, разница невелика, как и инерция. Инерция как индивидуальная мера мало информативна. Более того, сравнивая инерцию разных кластеров в целом, вы замечаете, что чем больше у вас групп, тем меньше инерция. Вы хотите сравнить инерцию кластерного решения с предыдущим кластерным решением. Это сравнение дает вам *скорость изменения* (rate of change) — более понятную меру. Чтобы получить скорость изменения инерции в Python, нужно создать цикл. Опробуйте прогрессивные кластерные решения внутри цикла, записывая их значения. Вот сценарий для примера рукописной цифры:

```
import numpy as np
inertia = list()
for k in range(1,21):
    clustering = KMeans(n_clusters=k,
                        n_init=10, random_state=1)
    clustering.fit(Cx)
    inertia.append(clustering.inertia_)
delta_inertia = np.diff(inertia) * -1
```

Используйте переменную `inertia` в классе кластеризации после подбора кластеризации. Переменная `inertia` представляет собой список, содержащий скорость изменения инерции между текущим решением и предыдущим. Ниже приведен код, выводящий линейный график скорости изменения, показанный на рис. 15.2.

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.figure()
x_range = [k for k in range(2, 21)]
plt.xticks(x_range)
plt.plot(x_range, delta_inertia, 'ko-')
plt.xlabel('Number of clusters')
plt.ylabel('Rate of change of inertia')
plt.show()
```

При изучении скорости изменения инерции ищите скачки в самой скорости. Если скорость увеличивается, это значит, что добавление кластеров больше, чем в предыдущем решении, приносит гораздо больше пользы, чем ожидалось. Если она вместо этого снизится, вы, скорее всего, форсируете кластер больше, чем необходимо. Все кластерные решения перед скачком могут быть хорошими кандидатами в соответствии с принципом просты (скачок говорит о

сложности в нашем анализе, но правильные решения обычно самые простые). В этом примере довольно много скачков при $k = 7, 9, 11, 14, 17$, но $k = 17$ представляется наиболее многообещающим пиком из-за гораздо более высокой скорости изменения относительно нисходящего тренда.

```
In [7]: import matplotlib.pyplot as plt
        %matplotlib inline
        plt.figure()
        x_range = [k for k in range(2, 21)]
        plt.xticks(x_range)
        plt.plot(x_range, delta_inertia, 'ko-')
        plt.xlabel('Number of clusters')
        plt.ylabel('Rate of change of inertia')
        plt.show()
```

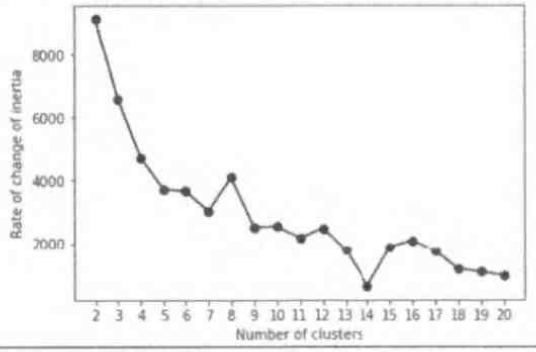


Рис. 15.2. Скорость изменения инерции для решения при $k = 20$


ЗАПОМНИ!

Скорость изменения инерции даст всего несколько советов, где могут быть хорошие кластерные решения. Вам решать, какой из них выбрать, если вам необходимо получить дополнительную информацию о данных. Если кластеризация вместо этого является лишь шагом в сложном проекте науки о данных, вам не нужно тратить много усилий на поиск оптимального количества кластеров. Просто передайте решение с достаточным количеством кластеров следующему алгоритму машинного обучения и позвольте ему выбрать лучшее.

Кластеризация больших данных

Метод k-средних — это способ уменьшить сложность данных, суммируя множество примеров в наборе данных. Для решения этой задачи следует загрузить данные в память компьютера, но это не всегда возможно, особенно если вы работаете с большими данными. Пакет Scikit-learn предлагает альтернативный способ применения метода k-средних; `MiniBatchKMeans` — это вариант, способный постепенно кластеризовать отдельные фрагменты данных.

Фактически процедура группового обучения обычно обрабатывает данные по частям. Есть только два различия между стандартной функцией k-средних и `MiniBatchKMeans`.

- Вы не сможете автоматически протестировать разные исходные центроиды, если не попытаетесь снова выполнить анализ.
- Анализ начнется, когда есть группа созданная, по крайней мере, из минимального количества случаев. С помощью параметра `batch_size` это значение обычно устанавливается равным 100 (но чем больше случаев, тем лучше результат).

Простая демонстрация предыдущего рукописного набора данных показывает, насколько эффективно и легко использовать класс кластеризации `MiniBatchKMeans`. Вначале в примере проверяется алгоритм k-средних на всех доступных данных и записывается инерция решения:

```
k = 10
clustering = KMeans(n_clusters=k,
                    n_init=10, random_state=1)
clustering.fit(Cx)
kmeans_inertia = clustering.inertia_
print("K-means inertia: %0.1f" % kmeans_inertia)
```

Обратите внимание, что результирующая инерция составляет 58253,3. Далее в этом примере проверяются те же данные и количество кластеров в ходе подбора кластера `MiniBatchKMeans` небольшими отдельными группами из 100 примеров:

```
from sklearn.cluster import MiniBatchKMeans
batch_clustering = MiniBatchKMeans(n_clusters=k,
                                   random_state=1)
batch = 100
for row in range(0, len(Cx), batch):
    if row+batch < len(Cx):
        feed = Cx[row:row+batch,:]
    else:
        feed = Cx[row:,:]
    batch_clustering.partial_fit(feed)
batch_inertia = batch_clustering.score(Cx) * -1

print("MiniBatchKmeans inertia: %0.1f" % batch_inertia)
```

Этот сценарий просматривает индексы заранее отмасштабированного и упрощенного PCA набора данных (`Cx`), создавая группы по 100 наблюдений в каждом. Используя метод `partial_fit`, он подбирает кластеризацию методом k-средних в каждой группе, используя центроиды, найденные предыдущим

вызовом. Алгоритм останавливается, когда у него заканчиваются данные. Используя метод оценки для всех доступных данных, он сообщает о своей инерции для решения с 10 кластерами. Теперь заявленная инерция составляет 64633,6. Обратите внимание, что `MiniBatchKmeans` приводит к более высокой инерции, чем стандартный алгоритм. Хотя разница минимальна, подходящее решение является негативным, поэтому нужно зарезервировать этот подход для тех случаев, когда вы действительно не можете работать с наборами данных в памяти.

Иерархическая кластеризация

Если алгоритм k-средних связан с центроидами, иерархическая (или агломерационная) кластеризация пытается связать каждую точку данных с помощью меры расстояния с ближайшим соседом, создав кластер. Итеративное выполнение алгоритма с различными методами связи позволяет алгоритму собрать все доступные точки и быстро снижает количество кластеров, пока в конце все точки не объединятся в одну группу.

Результаты, если они будут визуализированы, будут очень похожи на биологические классификации живых существ, которые вы, возможно, изучали в школе или видели на плакатах в музее естественной истории — перевернутое дерево, все ветви которого сходятся в ствол. Такое образное дерево является *дендрограммой* (dendrogram), и вы увидите, что оно используется в медицинских и биологических исследованиях. Реализация агломерационной кластеризации с помощью Scikit-learn не позволяет отобразить дендрограмму из ваших данных, поскольку такая техника визуализации хорошо работает только в нескольких случаях, в то время как вы можете рассчитывать на множество примеров.

По сравнению с методом k-средних агломерационные алгоритмы более громоздки и плохо масштабируются для больших наборов данных. Агломерационные алгоритмы больше подходят для статистических исследований (их легко найти в естественных науках, археологии, а иногда в психологии и экономике). Эти алгоритмы обладают преимуществом создания полного спектра вложенных кластерных решений, поэтому вам нужно выбрать подходящий для ваших целей.

Чтобы эффективно использовать агломерационную кластеризацию, вы должны знать о различных методах связи (эвристика для кластеризации) и метриках расстояния. Есть три метода связи.

- **Метод Уорда** (Ward's). Стремится найти сферические скопления, очень сплоченные внутри и чрезвычайно дифференцированные от других групп. Еще одно преимущество: метод способен находить кластеры одинакового размера. Работает только с евклидовым расстоянием.
- **Метод полной связи** (complete). Связывает кластеры, используя свои самые отдаленные наблюдения, т.е. их самые разнящиеся точки данных. Следовательно, кластеры, созданные с использованием этого метода, как правило, состоят из очень похожих наблюдений, что делает полученные группы достаточно компактными.
- **Центроидный метод** (average). Связывает кластеры, используя их центроиды и игнорируя их границы. Метод создает большие группы, чем метод полной связи. Кроме того, кластеры могут быть разных размеров и форм, в отличие от решений Уорда. Следовательно, этот подход находит успешное применение в области биологических наук, легко улавливая естественное разнообразие.

Есть также три метрики расстояния.

- **Евклидова** (euclidean или l2). Применяется методом k-средних.
- **Манхэттен** (manhattan или l1). Аналогична евклидовой, но расстояние рассчитывается при суммировании абсолютной величины разницы между размерами. На карте, если евклидово расстояние является кратчайшим путем между двумя точками, манхэттенское расстояние подразумевает движение сначала прямо вдоль одной оси, а затем вдоль другой оси, как автомобиль в городе, проезжая по городским кварталам к пункту назначения (известно также как *расстояние городских кварталов* (city block distance)).
- **Косинусная** (cosine). Хороший выбор, когда слишком много переменных, и вы беспокоитесь о том, что некоторая переменная может быть несущественной (или просто шумом). Косинусное расстояние уменьшает шум, принимая во внимание форму переменных, больше, чем их значения. Это имеет тенденцию связывать наблюдения, которые имеют одинаковые максимальные и минимальные переменные, независимо от их фактического значения.

Использование иерархического кластерного решения

Если набор данных не содержит слишком много наблюдений, стоит попробовать агломерационную кластеризацию со всеми комбинациями связи и расстояния, а затем тщательно сравнить результаты. В кластеризации правильные ответы известны редко, и агломерационная кластеризация может предоставить вам еще одно полезное потенциальное решение. Например, вы можете

воссоздать предыдущий анализ с помощью метода k-средних и рукописных цифр, используя метод полной связи и евклидово расстояние следующим образом (вывод показан на рис. 15.3):

```
from sklearn.cluster import AgglomerativeClustering
Hclustering = AgglomerativeClustering(n_clusters=10,
                                      affinity='euclidean',
                                      linkage='ward')
Hclustering.fit(Cx)
ms = np.column_stack((ground_truth,Hclustering.labels_))
df = pd.DataFrame(ms,
                  columns = ['Ground truth','Clusters'])
pd.crosstab(df['Ground truth'],
            df['Clusters'], margins=True)
```

```
In [10]: from sklearn.cluster import AgglomerativeClustering

         Hclustering = AgglomerativeClustering(n_clusters=10,
                                               affinity='euclidean',
                                               linkage='ward')
         Hclustering.fit(Cx)

         ms = np.column_stack((ground_truth,Hclustering.labels_))
         df = pd.DataFrame(ms,
                           columns = ['Ground truth','Clusters'])
         pd.crosstab(df['Ground truth'],
                     df['Clusters'], margins=True)
```

Out[10]:

Ground truth \ Clusters	0	1	2	3	4	5	6	7	8	9	All
0	0	0	0	0	0	178	0	0	0	0	178
1	1	27	154	0	0	0	0	0	0	0	182
2	0	166	10	0	1	0	1	0	0	0	177
3	0	4	13	0	0	0	166	0	0	0	183
4	1	0	4	0	1	0	0	14	0	161	181
5	168	1	0	1	0	0	11	0	0	1	182
6	0	0	1	180	0	0	0	0	0	0	181
7	1	0	1	0	0	0	0	8	169	0	179
8	2	4	167	0	0	0	1	0	0	0	174
9	3	0	29	1	0	0	143	4	0	0	180
All	176	201	379	182	2	178	322	26	169	162	1797

Рис. 15.3. Сводная таблица истиной правды и агломерационных кластеров Уорда

В этом случае результаты немного лучше, чем у метода k-средних, хотя, возможно, вы заметили, что выполнение анализа с использованием этого подхода, безусловно, занимает больше времени, чем использование метода k-средних.

При работе с большим количеством наблюдений вычисления для иерархического кластерного решения могут занять несколько часов, что делает его менее осуществимым. Для того чтобы обойти проблему времени, используйте двухфазную кластеризацию, которая быстрее и предоставляет иерархическое решение, даже если вы работаете с большими наборами данных.

Использование двухфазного кластерного решения

Чтобы реализовать решение двухфазной кластеризации, обработайте исходные наблюдения с использованием метода k-средних при большом количестве кластеров. Хорошее практическое правило — взять квадратный корень из количества наблюдений и использовать это число. Более того, для правильной работы всегда нужно, чтобы количество кластеров не выходило из диапазона 100–200 для второго этапа, основанного на иерархической кластеризации. В следующем примере используется 50 кластеров:

```
from sklearn.cluster import KMeans
clustering = KMeans(n_clusters=50,
                    n_init=10,
                    random_state=1)
clustering.fit(Cx)
```

На этом этапе сложнее всего отследить, какому кластеру какой случай был присвоен методом k-средних. Для этой цели мы используем словарь.

```
Kx = clustering.cluster_centers_
Kx_mapping = {case:cluster for case,
    cluster in enumerate(clustering.labels_)}
```

Новый набор данных, `Kx`, состоит из кластерных центроидов, которые обнаружил алгоритм k-средних. Вы можете рассматривать каждый кластер как хорошую представленную сводку исходных данных. Если вы кластеризуете сводку сейчас, она будет почти такой же, как кластеризация исходных данных.

```
from sklearn.cluster import AgglomerativeClustering
Hclustering = AgglomerativeClustering(n_clusters=10,
                                      affinity='cosine',
                                      linkage='complete')
Hclustering.fit(Kx)
```

Теперь вы сопоставляете результаты с центроидами, которые использовали изначально, чтобы можно было легче определить, состоит ли иерархический кластер из определенных центроидов k-средних. Результат состоит из наблюдений, составляющих кластеры k-средних, содержащих эти центроиды:

```
H_mapping = {case:cluster for case,
    cluster in enumerate(Hclustering.labels_)}
```

```
final_mapping = {case:H_mapping[Kx_mapping[case]]
    for case in Kx_mapping}
```

Теперь вы можете оценить решение, которое получили, используя ту же матрицу неточностей, что и ранее, как для k-средних, так и для иерархической кластеризации (результаты приведены на рис. 15.4):

```
ms = np.column_stack((ground_truth,
    [final_mapping[n] for n in range(max(final_mapping)+1)]))
df = pd.DataFrame(ms,
                  columns = ['Ground truth','Clusters'])
pd.crosstab(df['Ground truth'],
            df['Clusters'], margins=True)
```

```
In [15]: ms = np.column_stack((ground_truth,
          [final_mapping[n] for n in range(max(final_mapping)+1)]))
         df = pd.DataFrame(ms,
                           columns = ['Ground truth','Clusters'])
         pd.crosstab(df['Ground truth'],
                     df['Clusters'], margins=True)
```

Out[15]:

Clusters	0	1	2	3	4	5	6	7	8	9	All
Ground truth											
0	0	0	0	0	0	0	178	0	0	0	178
1	1	59	27	0	0	95	0	0	0	0	182
2	0	11	160	0	1	4	0	0	0	1	177
3	2	1	3	0	167	4	0	0	0	6	183
4	1	1	0	0	0	3	4	15	154	3	181
5	169	0	0	0	11	2	0	0	0	0	182
6	1	0	0	177	0	1	2	0	0	0	181
7	0	0	3	0	0	66	0	26	0	84	179
8	0	23	3	0	51	92	0	0	0	5	174
9	3	18	0	1	140	6	0	2	0	10	180
All	177	113	196	178	370	273	184	43	154	109	1797

Рис. 15.4. Сводная таблица истиной правды и двухступенчатой кластеризации

Решение, которое вы получаете, аналогично предыдущим решениям. Результат доказывает, что этот подход является подходящим методом для обработки больших наборов данных или даже наборов больших данных, сокращения их до меньших представлений и последующей работы с менее масштабируемой кластеризацией, но с более разнообразными и точными методами. Двухфазный подход обладает также еще одним преимуществом, поскольку он хорошо работает с зашумленными данными и данными с выбросами, — начальная фаза k-средних хорошо фильтрует такие проблемы и переводит их в отдельные кластерные решения.

Обнаружение новых групп с DBScan

Как метод k-средних, так и агломеративная кластеризация, особенно если вы используете критерии связи Уорда, будут давать сплоченные группы, похожие на пузырьки, одинаково распределенные по всем направлениям. Иногда реальность может давать сложные и тревожные результаты — группы могут иметь странные формы, далекие от канонического пузыря. Модуль наборов данных Scikit-learn (см. обзор на `http://scikitlearn.org/stable/modules/clustering.html`) предлагает широкий спектр форм, которые не могут быть успешно использованы с помощью метода k-средних или агломерационных кластеризации: большие круги, содержащие более мелкие, чередующиеся маленькие круги и спиральные наборы данных Swiss (названные в честь бисквитного рулета из-за того, как расположены точки данных).

DBScan — это еще один алгоритм кластеризации, основанный на осмысленной интуиции (smart intuition) и способный решать даже самые сложные проблемы. Алгоритм DBScan опирается на идею, что кластеры плотные, поэтому для начала исследования пространства данных в каждом направлении и маркировки границ кластеров при уменьшении плотности должно быть достаточно. Области пространства данных с недостаточной плотностью точек считаются пустыми, и во всех точках присутствуют помехи или *выбросы* (outlier), т.е. точки характеризуются необычными или странными значениями.

Алгоритм DBScan является более сложным и требует больше времени для выполнения, чем метод k-средних (но он быстрее, чем агломерационная кластеризация). Он автоматически угадывает количество кластеров и указывает на странные данные, которые плохо вписываются в любой класс. Это отличает DBScan от предыдущих алгоритмов, которые пытаются объединить в класс каждое наблюдение.

Для репликации рукописной кластеризации цифр требуется всего несколько строк кода Python:

```
from sklearn.cluster import DBSCAN
DB = DBSCAN(eps=3.7, min_samples=15)
DB.fit(Cx)
```

Используя DBScan, вам не нужно устанавливать K ожидаемых кластеров; алгоритм найдет их сам. По-видимому, отсутствие числа K, кажется, упрощает использование DBScan; в действительности алгоритм требует, чтобы для правильной работы вы исправили два основных параметра, `eps` и `min_sample`.

» `eps`. Максимальное расстояние между двумя наблюдениями, которое позволяет им быть частью одной и той же окрестности.

» `min_sample`. Минимальное количество наблюдений в окрестности, которые превращают их в центральную точку.

Алгоритм работает, обходя данные и создавая кластеры, а затем связывает наблюдения, расположенные в окрестностях. *Окрестность* (neighborhood) — это небольшой кластер точек данных, все в пределах расстояния `eps`. Если количество точек в окрестности меньше, чем число `min_sample`, то DBScan не образует окрестность.

Независимо от формы кластера, DBScan связывает все окрестности вместе, если они находятся достаточно близко (согласно значению расстояния `eps`). Когда больше нет окрестностей, DBScan пытается объединить в группы даже отдельные точки данных, если они находятся на расстоянии `eps`. Точки данных, которые не связаны с какой-либо группой, рассматриваются как шум (точки слишком специфичные, чтобы быть частью группы).


СОВЕТ

Опробуйте несколько значений `eps` и `min_sample`. Результирующие кластеры также могут резко измениться относительно значений, установленных в этих двух параметрах. Начните с небольшого значения `min_samples`. Использование меньшего значения позволяет объединить многие окрестности. Стандартное значение 5 вполне подходит. Затем опробуйте разные числа для `eps`, начиная с 0,1 и выше. Не разочаровывайтесь, если изначально не можете получить подходящий результат, — продолжайте пробовать разные комбинации.

Несколько исследований данных могут позволить взглянуть на результаты с правильной точки зрения. Сначала подсчитаем кластеры:

```
from collections import Counter
print('No. clusters: %i' % len(np.unique(DB.labels_)))
print(Counter(DB.labels_))

ms = np.column_stack((ground_truth, DB.labels_))
df = pd.DataFrame(ms,
                  columns = ['Ground truth', 'Clusters'])
pd.crosstab(df['Ground truth'],
            df['Clusters'], margins=True)
```

Почти половина наблюдений относится к кластеру, помеченному как –1, что представляет шум (шум определяется как примеры, которые слишком необычны для группировки). Учитывая количество измерений (30 некоррелированных переменных из анализа PCA) в данных и их высокую изменчивость (они представляют собой рукописные образцы), многие случаи не попадают ни в одну из групп. На рис. 15.5 приведен вывод этого примера.

```
No. clusters: 12
Counter({-1: 836, 6: 182, 0: 172, 2: 159, 1: 156, 4: 119,
         5: 77, 3: 28, 10: 21, 7: 18, 8: 16, 9: 13})
```

```
In [17]: from collections import Counter
         print('No. clusters: %i' % len(np.unique(DB.labels_)))
         print(Counter(DB.labels_))

         ms = np.column_stack((ground_truth, DB.labels_))
         df = pd.DataFrame(ms,
                           columns = ['Ground truth', 'Clusters'])

         pd.crosstab(df['Ground truth'],
                     df['Clusters'], margins=True)

         No. clusters: 12
         Counter({-1: 829, 6: 183, 0: 172, 1: 159, 2: 158, 4: 119, 5: 80, 3:
         27, 10: 21, 8: 19, 7: 17, 9: 13})

Out[17]:
```

Clusters	-1	0	1	2	3	4	5	6	7	8	9	10	All
Ground truth													
0	6	172	0	0	0	0	0	0	0	0	0	0	178
1	68	0	88	0	26	0	0	0	0	0	0	0	182
2	143	0	2	0	0	0	0	0	0	19	13	0	177
3	75	0	4	0	0	0	0	104	0	0	0	0	183
4	84	0	0	0	0	0	80	0	17	0	0	0	181
5	157	0	0	1	0	0	0	3	0	0	0	21	182
6	23	0	1	157	0	0	0	0	0	0	0	0	181
7	60	0	0	0	0	119	0	0	0	0	0	0	179
8	110	0	63	0	1	0	0	0	0	0	0	0	174
9	103	0	1	0	0	0	0	76	0	0	0	0	180
All	829	172	159	158	27	119	80	183	17	19	13	21	1797

Рис. 15.5. Сводная таблица истиной правды и DBScan

ЗАПОМНИ!

Сила DBScan заключается в предоставлении надежных, согласованных кластеров. DBScan, как и метод k-средних и агломерационная кластеризация, не обязаны искать решение с определенным количеством кластеров, особенно если такого решения не существует.

Глава 16

Поиск выбросов в данных

В ЭТОЙ ГЛАВЕ...

- Что является выбросом
- Различие между экстремальными значениями и новыми
- Использование простой статистики для поиска выбросов
- Поиск наиболее сложных выбросов с помощью передовых методов

Ошибки происходят, когда их меньше всего ждешь, и это также верно в отношении ваших данных. Кроме того, ошибки в данных трудно обнаружить, особенно если набор данных содержит много переменных разных типов и масштабов. Ошибки в данных могут принимать различные формы. Например, значения могут систематически отсутствовать для определенных переменных, ошибочные числа могут появляться здесь и там, а данные могут включать выбросы. Красный флаг должен быть поднят, когда:

- отсутствие значений в определенных группах случаев или переменных означают, что ошибку вызывает какая-то конкретная причина;
- ошибочные значения зависят от того, как приложение создало или обработало данные. Например, вам нужно знать, получило ли приложение данные от измерительного прибора. На надежность приборов могут повлиять внешние условия или ошибка человека;
- случай, по-видимому, действителен, но весьма отличается от обычных значений, которые характеризуют эту переменную. Когда вы не можете объяснить причину разницы, вы можете наблюдать выброс.

Среди описанных ошибок самая сложная проблема, которую нужно решить, — это когда в наборе данных есть выбросы, поскольку у вас не всегда есть уникальное определение выбросов или нет явной причины их наличия в данных. В результате многое остается только на ваше усмотрение.

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_16_Detecting_Outliers.ipynb`.

Обнаружение выбросов

Как общее определение, *выбросы* (outlier) — это данные, которые значительно отличаются (далеки) от других данных в выборке. Причина, по которой они находятся далеко, в том, что одно или несколько значений являются слишком высокими или слишком низкими по сравнению с большинством других. Они также могут отображать почти уникальную комбинацию значений. Например, если вы анализируете записи студентов, зачисленных в университет, ваше внимание могут привлечь студенты, которые слишком молоды или слишком стары. Студенты, изучающие несколько необычных предметов, также потребуют тщательного изучения.

Выбросы искажают распределения данных и влияют на все ваши основные статистические данные о тенденциях. Средние значения выдвигаются вверх или вниз, влияя на все другие описательные меры. Выброс всегда будет увеличивать дисперсию и изменять корреляции, поэтому вы можете получить неверные предположения о данных и отношениях между переменными.

Этот простой пример может отображать влияние (в небольшом масштабе) одного выброса на более чем тысячу регулярных наблюдений:

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
%matplotlib inline

import numpy as np
from scipy.stats.stats import pearsonr
np.random.seed(101)
normal = np.random.normal(loc=0.0, scale= 1.0, size=1000)
print('Mean: %0.3f Median: %0.3f Variance: %0.3f' %
                                  (np.mean(normal),
                                   np.median(normal),
                                   np.var(normal)))
```

Генератор случайных чисел NumPy создает в примере переменную `normal`, которая содержит 1000 наблюдений, полученных из стандартного нормального распределения. Базовая описательная статистика (среднее, медиана, дисперсия) не показывает ничего неожиданного. Вот итоговое среднее, медиана и дисперсия:

```
Mean: 0.026 Median: 0.032 Variance: 1.109
```

Теперь мы изменим одно значение, вставив выброс:

```
outlying = normal.copy()
outlying[0] = 50.0
print('Mean: %0.3f Median: %0.3f Variance: %0.3f' %
                                  (np.mean(outlying),
                                   np.median(outlying),
                                   np.var(outlying)))

print('Pearson''s correlation: %0.3f p-value: %0.3f' %
                             pearsonr(normal,outlying))
```

Вы можете взять эту новую переменную, `outlying`, и поместить в нее выброс (при индексе 0 у вас есть положительное значение 50,0). Теперь будет получена немного другая описательная статистика:

```
Mean: 0.074 Median: 0.032 Variance: 3.597
Pearsons correlation coefficient: 0.619 p-value: 0.000
```

Теперь статистика показывает, что среднее значение стало в три раза выше, чем раньше, как и дисперсия. Изменение влияет не только на медиану, которая зависит от положения (она сообщает значение, занимающее среднее положение, когда все наблюдения расположены по порядку).

Что еще более важно, корреляция исходной переменной и переменной с выбросом довольно далека от +1,0 (значение корреляции переменной по отношению к самой себе), что указывает на то, что мера линейных отношений между двумя переменными была серьезно повреждена.

Что еще может пойти не так

Выбросы не просто смещают ключевые показатели исследовательской статистики, они также меняют структуру отношений между переменными в данных. Выбросы могут повлиять на алгоритмы машинного обучения двумя способами.

- Алгоритмы, основанные на коэффициентах, могут получать неправильные коэффициенты, чтобы минимизировать их неспособность понять отдаленные случаи. Ярким примером являются линейные модели (суммы коэффициентов), но они не единственные. Выбросы

могут также влиять на алгоритмы на основе *обучения дерева решений* (tree-based learner), таких как алгоритмы AdaBoost или Grastent Boosting Machines.

» Поскольку алгоритмы обучаются на выборках данных, выбросы могут побудить алгоритм к искажению веса вероятности чрезвычайно низких или высоких значений при определенной конфигурации переменных.

Обе ситуации ограничивают способность алгоритма обучения хорошо обобщать новые данные. Другими словами, они приводят к искажению процесса обучения на этом наборе данных.


ЗАПОМНИ!

Существует несколько способов устранения выбросов — некоторые из них требуют изменения существующих данных, а другие — выбора подходящей функции ошибок для алгоритма машинного обучения. (Некоторые алгоритмы позволяют выбрать другую функцию ошибок в качестве параметра при настройке процедуры обучения.) Большинство алгоритмов машинного обучения могут применять различные функции ошибок. Функция ошибок важна, поскольку она помогает алгоритму учиться, понимая ошибки и применяя корректировки в процессе обучения, но некоторые функции ошибок чрезвычайно чувствительны к выбросам, в то время как другие довольно устойчивы к ним. Например, мера квадратичной ошибки имеет тенденцию подчеркивать выбросы, поскольку ошибки, получаемые из примеров с большими значениями, возводятся в квадрат, становясь тем самым еще более заметными.

Понятие аномалий и новых данных

Поскольку выбросы возникают как ошибки или крайне редкие случаи, их обнаружение никогда не бывает легкой задачей; но это важно для получения эффективных результатов от вашего проекта по науке о данных. В определенных областях обнаружение аномалий само по себе является целью науки о данных: обнаружение мошенничества в страховании и банковском деле, обнаружение неисправностей на производстве, мониторинговые системы в здравоохранении и других критически важных областях, а также обнаружение событий в системах безопасности и раннего предупреждения.

Важное различие заключается в поиске выбросов в существующих данных или в проверке любых новых данных, содержащих аномалии относительно существующих случаев. Возможно, вы потратили много времени на очистку данных или разработали приложение для машинного обучения на основе

доступных данных, поэтому было бы важно выяснить, похожи ли новые данные на старые, и будут ли продолжать работать алгоритмы классификации или прогнозирования.

В таких случаях аналитики данных говорят об обнаружении *новизны* (novelty), поскольку они должны знать, насколько хорошо новые данные напоминают старые. Быть исключительно новым считается аномалией: новизна может скрывать значительное событие или помешать правильной работе алгоритма, поскольку машинное обучение в значительной степени основано на обучении, исходя из прошлых примеров, и оно может не обобщаться до совершенно новых случаев. При работе с новыми данными вам следует переучить алгоритм.

Опыт учит, что мир редко бывает стабилен. Иногда новинки появляются вполне естественно, поскольку мир так изменчив. Следовательно, данные изменяются с течением времени неожиданным образом, как в целевых, так и в прогнозирующих переменных. Это явление называется *дрейфом понятий* (concept drift). Термин *понятие* (concept) относится к вашей цели и *дрейфует* к исходным данным, используемым для выполнения прогноза, который движется медленно и не поддается контролю, как лодка, дрейфующая из-за сильных приливов. Рассматривая модель науки о данных, вы будете различать разные дрейфы концепций и ситуации новизны.

- **Физическая** (physical). Системы распознавания лиц или голоса или даже климатические модели никогда не меняются. Не ожидайте новинок, но проверьте те выбросы, которые возникают в результате проблем с данными, таких как ошибочные измерения.
- **Политическая и экономическая** (political and economic). Эти модели иногда меняются, особенно в долгосрочной перспективе. Вы должны следить за долгосрочными эффектами, которые начинаются медленно, а затем распространяются и консолидируются, делая модели неэффективными.
- **Социальное поведение** (social behavior). Социальные сети и язык, которым вы пользуетесь каждый день, со временем меняются. Ожидайте появления новинок и предпринимайте меры предосторожности; в противном случае ваша модель внезапно испортится и станет непригодной для использования.
- **Данные поисковых систем, банковские операции и схемы мошенничества в электронной торговле** (search engine data, banking, and e-commerce fraud schemes). Эти модели меняются довольно часто. Вам необходимо проявлять особую осторожность при проверке появившихся новинок, предлагая обучить новую модель для поддержания точности.

» **Угрозы кибербезопасности и рекламные тренды** (cyber security threats and advertising trends). Эти модели постоянно меняются. Обнаружение новинок является нормой, и повторное использование одних и тех же моделей в течение длительного времени является опасным.

Изучение простого одномерного метода

Хорошим началом при поиске выбросов, независимо от количества имеющихся в данных переменных, является рассмотрение каждой переменной по отдельности, используя как графическую, так и статистическую проверку. Это однофакторный подход, позволяющий определить выбросы по неподходящему значению переменной. Пакет pandas позволяет легко обнаружить выбросы благодаря

» простому методу describe, информирующему о среднем значении, дисперсии, квартилях и экстремумах числовых значений для каждой переменной;

» системе автоматической визуализации диаграмм размаха.

Использование обоих методов позволяет легко узнать, есть ли выбросы и где их искать. Набор данных diabetes из модуля наборов данных Scikit-learn является хорошим примером для начала.

```
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
X,y = diabetes.data, diabetes.target
```

После этих команд все данные содержатся в переменной X типа ndarray NumPy. Затем код преобразует его в объект pandas DataFrame и запрашивает некоторую описательную статистику (вывод приведен на рис. 16.1):

```
import pandas as pd
pd.options.display.float_format = '{:.2f}'.format
df = pd.DataFrame(X)
df.describe()
```

Вы можете определить проблемные переменные, посмотрев на экстремумы распределения (максимальные значения переменной). Например, вы должны рассмотреть, находятся ли минимальное и максимальное значения соответственно далеко от 25-го и 75-го процентиля. Как показано в выводе, многие переменные имеют подозрительно большие максимальные значения. Анализ диаграммы размаха прояснит ситуацию. Следующая команда создает диаграмму размаха всех переменных, показанных на рис. 16.2:

```
In [5]: import pandas as pd
        pd.options.display.float_format = '{:.2f}'.format
        df = pd.DataFrame(X)
        df.describe()
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9
count	442.00	442.00	442.00	442.00	442.00	442.00	442.00	442.00	442.00	442.00
mean	-0.00	0.00	-0.00	0.00	-0.00	0.00	-0.00	0.00	-0.00	-0.00
std	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
min	-0.11	-0.04	-0.09	-0.11	-0.13	-0.12	-0.10	-0.06	-0.13	-0.14
25%	-0.04	-0.04	-0.03	-0.04	-0.03	-0.03	-0.04	-0.04	-0.03	-0.03
50%	0.01	-0.04	-0.01	-0.01	-0.00	-0.00	-0.01	-0.00	-0.00	-0.00
75%	0.04	0.05	0.03	0.04	0.03	0.03	0.03	0.03	0.03	0.03
max	0.11	0.05	0.17	0.13	0.15	0.20	0.18	0.19	0.13	0.14

Рис. 16.1. Описательная статистика для `DataFrame`

```
fig, axes = plt.subplots(nrows=1, ncols=1,
                         figsize=(10, 5))
df.boxplot(ax=axes);
```

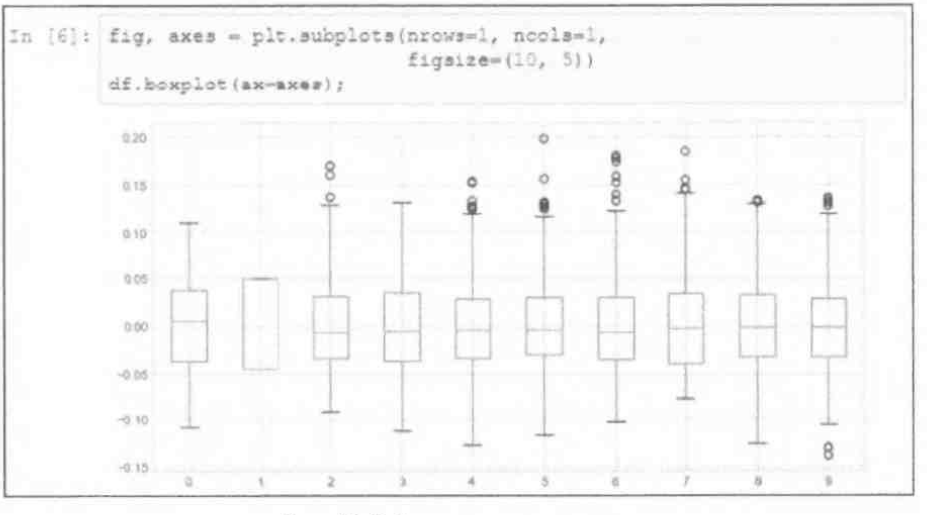


Рис. 16.2. Диаграмма размаха

Диаграммы размаха, созданные `pandas DataFrame`, будут иметь усы, установленные на "плюс" или "минус" 1,5 IQR (межквартильный диапазон или расстояние между нижним и верхним квартилем) по отношению к верхней и нижней стороне ящика (верхний и нижний квартили). Этот стиль диаграммы размаха Тьюки (по имени статистика Джона Тьюки (John Tukey), который

создал и продвигал его среди статистиков вместе с другими методами описания данных), позволяет визуализировать случаи за пределами усов. (Все точки за пределами этих усов считаются выбросами.)

Опора на гауссово распределение

Еще одной эффективной проверкой на выбросы в данных является использование нормального распределения. Даже если данные не распределены нормально, их стандартизация позволит вам предположить определенные вероятности обнаружения аномальных значений. Например, 99,7% значений, находящихся в стандартизированном нормальном распределении, должны находиться в диапазоне +3 и –3 стандартных отклонений от среднего значения, как показано в следующем коде:

```
from sklearn.preprocessing import StandardScaler
Xs = StandardScaler().fit_transform(X)
# метод .any(1) позволит избежать дублирования
df[(np.abs(Xs)>3).any(1)
```

На рис. 16.3 приведены результаты, описывающие строки в наборе данных, и демонстрирующие некоторые возможные выбросы значений.

```
In [7]: from sklearn.preprocessing import StandardScaler
        Xs = StandardScaler().fit_transform(X)
        # .any(1) method will avoid duplicating
        df[(np.abs(Xs)>3).any(1)]
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9
58	0.04	-0.04	-0.06	0.04	0.01	-0.06	0.18	-0.08	-0.00	-0.05
123	0.01	0.05	0.03	-0.00	0.15	0.20	-0.06	0.19	0.02	0.07
216	0.01	0.05	0.04	0.05	0.05	0.07	-0.07	0.15	0.05	0.05
230	-0.04	0.05	0.07	-0.06	0.15	0.16	0.00	0.07	0.05	0.07
256	-0.05	-0.04	0.16	-0.05	-0.03	-0.02	-0.05	0.03	0.03	0.01
260	0.04	-0.04	-0.01	-0.06	0.01	-0.03	0.15	-0.08	-0.08	-0.02
261	0.05	-0.04	-0.04	0.10	0.04	-0.03	0.18	-0.08	-0.01	0.02
269	0.01	-0.04	-0.03	-0.03	0.04	-0.01	0.16	-0.08	-0.01	-0.04
322	0.02	0.05	0.06	0.06	0.02	-0.04	-0.09	0.16	0.13	0.08
336	-0.02	-0.04	0.09	-0.04	0.09	0.09	-0.06	0.15	0.08	0.05
367	-0.01	0.05	0.17	0.01	0.03	0.03	-0.02	0.03	0.03	0.03
441	-0.05	-0.04	-0.07	-0.08	0.08	0.03	0.17	-0.04	-0.00	0.00

Рис. 16.3. Сообщение о возможных выбросах

Модуль Scikit-learn предоставляет простой способ стандартизации данных и записи всех преобразований для последующего использования в различных наборах данных. Это означает, что все данные, независимо от того, используются ли они для машинного обучения или для проверки производительности, стандартизируются одинаково.


СОВЕТ

Правило 68-95-99.7 гласит, что в стандартизированном нормальном распределении 68% значений находятся в пределах одного стандартного отклонения, 95% — в пределах двух стандартных отклонений, а 99,7% — в пределах трех. При работе с искаженными данными правило 68-95-99.7 может не выполняться, и в таком случае может потребоваться более консервативная оценка, такая как неравенство Чебышева. *Неравенство Чебышева* опирается на формулу, гласящую, что для `k` стандартных отклонений вокруг среднего значения, количество случаев в процентах не должно превышать `1/k^2` вокруг среднего значения[1]. Следовательно, при семи стандартных отклонениях от среднего значения вероятность нахождения корректного значения составляет не более 2%, независимо от того, каково распределение (2% — это низкая вероятность, поэтому ваш случай можно считать почти наверняка выбросом).


СОВЕТ

Неравенство Чебышева является консервативным. Высокая вероятность быть выбросом соответствует семи или более стандартным отклонениям от среднего значения. Используйте это, когда велика цена посчитать ценность выбросом, когда это не так. Для всех остальных применений достаточно правила 68-95-99.7.

Предположения и проверка

Найдя некоторые возможные одномерные выбросы, вы должны решить, как с ними бороться. Если вы полностью не доверяете отдаленным случаям, полагая, что это ошибки или заблуждения, удалите их. (В Python можно просто отменить их выбор, используя *прихотливую индексацию* (fancy indexing).)


СОВЕТ

Изменение значений в данных или решение об исключении определенных значений — это решение, которое необходимо принять после того, как вы поймете, почему в данных есть некоторые выбросы. Вы можете исключить необычные значения или случаи, для которых

[1] Проще говоря, случайная величина в основном принимает значения, близкие к своему среднему. См. лучше `https://ru.wikipedia.org/wiki/Неравенство_Чебышёва`. — *Примеч. ред.*

предполагаете, что произошла некоторая ошибка в измерении, в записи или предшествующей обработке данных. Если же вы поймете, что отдаленный случай является вполне законным, хотя и редким, лучшим выходом будет снизить его вес (если алгоритмы обучения используют вес для наблюдений) или увеличить размер выборки данных.

В нашем случае, решив сохранить и стандартизировать данные, мы могли бы просто ограничить значения выбросов, используя простой множитель стандартного отклонения:

```
Xs_capped = Xs.copy()
o_idx = np.where(np.abs(Xs)>3)
Xs_capped[o_idx] = np.sign(Xs[o_idx]) * 3
```

В предлагаемом коде функция `sign` из NumPy восстанавливает знак отдаленного наблюдения (+1 или –1), который затем умножается на значение 3 и присваивается соответствующей точке данных, восстановленной с помощью логической индексации стандартизированного массива.

Этот подход имеет ограничение. Будучи стандартным отклонением, используемым как для высоких, так и для низких значений, оно подразумевает симметрию в распределении данных, чего в реальных данных зачастую нет. В качестве альтернативы можете использовать более сложный подход — *винзоризацию* (winsorizing). При его использовании, считающиеся выбросами значения обрезаются до значения определенных процентилей, которые действуют как пределы значений (обычно 5-й процентиль для нижней границы, 95-й для верхней):

```
from scipy.stats.mstats import winsorize
Xs_winsorized = winsorize(Xs, limits=(0.05, 0.95))
```

Таким образом, вы создаете другое значение барьера для бо́льших и меньших значений с учетом любой асимметрии в распределении данных. Независимо от того, что вы решите использовать для ограничения (стандартное отклонение или винзоризацию), ваши данные теперь готовы для дальнейшей обработки и анализа.

Наконец, альтернативное решение — позволить Scikit-learn автоматически преобразовать данные и обрезать выбросы с помощью преобразователя `RobustScaler`, основанного на IQR (как в диаграмме размаха, обсуждавшейся ранее в этой главе).

Выработка многомерного подхода

Работа с отдельными переменными позволяет обнаружить большое количество выбросов в наблюдениях. Но выбросы не обязательно являются значениями, слишком далекими от нормы. Иногда выбросы состоят из необычных комбинаций значений в нескольких переменных. Это редкие, но влиятельные комбинации, способные обмануть алгоритмы машинного обучения.

В таких случаях точная проверка каждой переменной не сможет исключить аномальные случаи из набора данных. Только несколько выбранных методов, учитывающих больше переменных за раз, смогут выявить проблемы в данных.

Следующие методики позволяют подойти к проблеме с разных точек зрения.

- Уменьшение размерности.
- Кластеризация плотности.
- Нелинейное моделирование распределения.

Использование этих методов и сравнение их результатов позволяет обратить внимание на повторяющиеся сигналы в конкретных случаях — иногда уже найденных в одномерном исследовании, а иногда еще нет.

Использование анализа основных компонентов

Анализ основных компонентов может полностью реструктурировать данные, устраняя избыточность и упорядочивая вновь полученные компоненты в соответствии с количеством исходных отклонений, которые они выражают. Этот тип анализа предлагает синтетическое и полное представление о распределении данных, делая многовариантные выбросы особенно очевидными.

Первые два компонента, будучи наиболее информативными с точки зрения дисперсии, могут отображать общее распределение данных, если они визуализируются. Вывод обеспечивает хороший намек на возможные очевидные выбросы.

Последние два компонента, являющиеся наиболее остаточными, они отображают всю информацию, которая иначе не могла бы быть подобрана методом PCA. Они также могут дать представление о возможных, но менее очевидных выбросах.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from pandas.plotting import scatter_matrix
pca = PCA()
Xc = pca.fit_transform(scale(X))
```

```
first_2 = sum(pca.explained_variance_ratio_[:2]*100)
last_2 = sum(pca.explained_variance_ratio_[-2:]*100)

print('variance by the components 1&2: %0.1f%%' % first_2)
print('variance by the last components: %0.1f%%' % last_2)

df = pd.DataFrame(Xc, columns=['comp_' + str(j)
                  for j in range(10)])
fig, axes = plt.subplots(nrows=1, ncols=2,
                         figsize=(15, 5))
first_two = df.plot.scatter(x='comp_0', y='comp_1',
                            s=50, grid=True, c='Azure',
                            edgecolors='DarkBlue',
                            ax=axes[0])
last_two = df.plot.scatter(x='comp_8', y='comp_9',
                           s=50, grid=True, c='Azure',
                           edgecolors='DarkBlue',
                           ax=axes[1])
plt.show()
```

На рис. 16.4 показаны две диаграммы рассеяния первого и последнего компонентов. В выводе также сообщается об отклонении, объясненном первыми двумя компонентами (половина информативного содержимого набора данных) PCA и последними двумя:

```
variance by the components 1&2: 55.2%
variance by the last components: 0.9%
```

Обратите особое внимание на точки данных вдоль осей (где ось x определяет независимую переменную, а ось y — зависимую). Вы можете увидеть возможный порог, используемый для отделения обычных данных от подозрительных.

Используя два последних компонента, вы можете найти несколько точек для исследования, используя порог –0,3 для десятого компонента и –1,0 для девятого. Все случаи ниже этих значений являются возможными выбросами (рис. 16.5).

```
outlying = (Xc[:,-1] > 0.3) | (Xc[:,-2] > 1.0)
df[outlying]
```

Использование кластерного анализа для определения выбросов

Выбросы являются изолированными точками в пространстве переменных, а DBScan является алгоритмом кластеризации, который связывает плотные части данных и отмечает разреженные. Таким образом, DBScan — это идеальный инструмент для автоматического исследования данных и проверки возможных выбросов.

```
first_2 = sum(pca.explained_variance_ratio_[:2]*100)
last_2 = sum(pca.explained_variance_ratio_[-2:]*100)

print('variance by the components 1&2: %0.1f%%' % first_2)
print('variance by the last components: %0.1f%%' % last_2)

df = pd.DataFrame(Xc, columns=['comp_' + str(j)
                               for j in range(10)])

fig, axes = plt.subplots(nrows=1, ncols=2,
                         figsize=(15, 5))
first_two = df.plot.scatter(x='comp_0', y='comp_1',
                            s=50, grid=True, c='Azure',
                            edgecolors='DarkBlue',
                            ax=axes[0])
last_two  = df.plot.scatter(x='comp_8', y='comp_9',
                            s=50, grid=True, c='Azure',
                            edgecolors='DarkBlue',
                            ax=axes[1])

plt.show()
```

```
variance by the components 1&2: 55.2%
variance by the last components: 0.9%
```

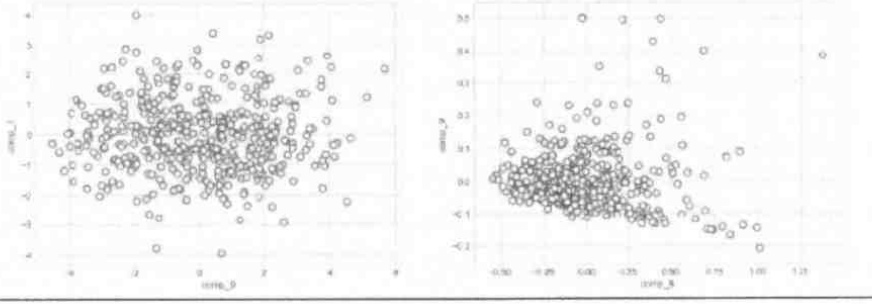


Рис. 16.4. Два первых и два последних компонента из PCA

```
In [12]: outlying = (Xc[:,-1] > 0.3) | (Xc[:,-2] > 1.0)
         df[outlying]
```

Out[12]:

	comp_0	comp_1	comp_2	comp_3	comp_4	comp_5	comp_6	comp_7	comp_8	comp_9
23	3.77	-1.76	1.09	0.72	-0.64	1.90	0.56	1.09	0.44	0.50
58	-2.65	2.23	2.79	-0.63	0.26	-0.13	1.44	0.67	1.01	-0.21
110	-2.04	-0.76	0.74	-1.93	-0.07	0.24	-1.75	-0.41	0.47	0.31
169	2.35	0.15	-0.13	1.19	-0.64	0.64	2.65	-0.31	0.22	0.50
254	3.82	-1.03	1.06	0.44	0.27	0.86	0.97	0.66	0.43	0.33
322	4.52	-2.24	-0.14	0.85	-0.47	0.73	1.28	0.34	1.39	0.38
323	3.87	0.69	0.26	0.59	0.07	0.76	1.79	0.36	0.89	0.40
353	0.98	1.61	-1.16	1.14	-0.36	1.46	2.53	0.90	-0.02	0.50
371	2.11	-0.28	0.64	-0.65	-0.36	-0.26	2.22	1.09	0.07	0.35
394	2.24	-1.13	0.51	1.54	-1.30	-0.12	2.28	-0.10	0.40	0.43

Рис. 16.5. Возможные случаи выбросов, обнаруженные PCA

Ниже приведен пример использования DBScan для обнаружения выбросов.

```
from sklearn.cluster import DBSCAN
DB = DBSCAN(eps=2.5, min_samples=25)
DB.fit(Xc)

from collections import Counter
print(Counter(DB.labels_))
df[DB.labels_==-1]
```

Однако для DBSCAN требуются два параметра, eps и min_samples, которые требуют многократных попыток найти правильные значения, что делает их использование немного сложным.

Как упоминалось в предыдущей главе, начинайте с низкого значения min_samples и пробуйте увеличивать значения eps с 0,1 и выше. После каждого испытания с измененными параметрами проверяйте ситуацию, подсчитав количество наблюдений в классе –1 внутри атрибута labels, и остановитесь, когда количество выбросов окажется разумным при визуальном осмотре.

СОВЕТ

На границе распределения плотных частей всегда будут точки, поэтому трудно предоставить пороговое значение для количества случаев, которые могут быть классифицированы в классе –1. Обычно выбросы не должны превышать 5% случаев, поэтому используйте это указание как общее эмпирическое правило.

Вывод предыдущего примера сообщат, сколько примеров попадает в группу –1, которую алгоритм считает не частью основного кластера, и предоставляет список случаев, которые являются его частью.

СОВЕТ

Алгоритм кластеризации k-средних менее автоматизирован, но вы также можете использовать его для обнаружения выбросов. Сначала запустите кластерный анализ с достаточным количеством кластеров. (Вы можете опробовать разные решения, если не уверены.) Затем отыщите кластеры с несколькими примерами (или, может быть, с одним); они, вероятно, являются выбросами, поскольку выглядят как небольшие отдельные кластеры, которые отделены от больших кластеров, содержащих большинство примеров.

Автоматическое обнаружение с помощью изоляционного леса

Случайные леса (random forest) и *сверхслучайные деревья* (extremely randomized tree) — это мощные методы машинного обучения, которые будут описаны в главе 20. Они работают, разделяя набор данных на меньшие наборы

на основании определенных значений переменных, чтобы упростить прогнозирование классификации или регрессии для каждого меньшего подмножества (принцип "разделяй и властвуй").

`IsolationForest` — это алгоритм, который использует тот факт, что выброс легче выделить из большинства случаев на основе различий между его значениями или комбинацией значений. Алгоритм отслеживает, сколько времени требуется, чтобы отделить случай от других и поместить его в свое собственное подмножество. Чем меньше усилий требуется на его отделение, тем больше вероятность того, что случай является выбросом. В качестве меры `IsolationForest` производит измерение расстояния (чем короче расстояние, тем больше вероятность того, что это выброс).


СОВЕТ

Пока ваши алгоритмы машинного обучения находятся в разработке, обученный `IsolationForest` может действовать как проверка работоспособности, поскольку многие алгоритмы машинного обучения не могут справиться с выбросами и новыми примерами.

Чтобы настроить `IsolationForest` на обнаружение выбросов, достаточно выбрать уровень, который определит процент случаев, считающихся выбросами, на основе измерения расстояния. Выберите такой процент, исходя из вашего опыта и ожиданий качества данных. Для создания работающего `IsolationForest` выполните следующий сценарий:

```
from sklearn.ensemble import IsolationForest
auto_detection = IsolationForest(max_samples=50,
                                 contamination=0.05,
                                 random_state=0)
auto_detection.fit(Xc)
evaluation = auto_detection.predict(Xc)
df[evaluation==-1]
```

Вывод содержит список случаев, подозреваемых как выбросы. Кроме того, алгоритм обучен распознавать нормальные примеры наборов данных. Когда вы предоставляете в набор данных новые наборы данных и оцениваете их с помощью обученного `IsolationForest`, вы можете сразу определить, что с вашими новыми данными что-то не так.


ЗАПОМНИ!

`IsolationForest` — вычислительно требовательный алгоритм. Выполнение анализа большого набора данных занимает много времени и памяти.

5

Обучение на данных

В ЭТОЙ ЧАСТИ...

- Четыре простых, но важных алгоритма
- Методы перекрестной проверки, отбора и оптимизации
- Линейные и нелинейные трюки для увеличения сложности
- Работа с ансамблями данных для получения лучшего результата

Глава 17

Четыре простых, но эффективных алгоритма

В ЭТОЙ ГЛАВЕ...

- Использование линейной и логистической регрессии
- Теорема Байеса и ее применение для наивной классификации
- Прогнозирование на основе случаев, сходных с KNN

В этой части вы начнете изучать все алгоритмы и инструменты, необходимые для обучения на основе данных (обучение модели с использованием данных) и способности прогнозировать числовую оценку (например, цены на жилье) или класс (например, вид ирисов) на новых примерах. В этой главе мы начнем с самых простых алгоритмов и перейдем к более сложным. Четыре алгоритма, описанных в этой главе, представляют собой хорошую отправную точку для любого аналитика данных.

СОВЕТ

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_17_Exploring_Four_Simple_and_Effective_Algorithms.ipynb`.

Угадай число: линейная регрессия

Регрессия имеет долгую историю в статистике: от построения простых, но эффективных линейных моделей экономических, психологических, социальных и политических данных до проверки гипотез о понимании различий в группах, моделирования более сложных задач с порядковыми значениями, двоичными и множественными классами, подсчетом данных и иерархическими отношениями. Это также распространенный инструмент науки о данных, швейцарский армейский нож машинного обучения, который можно использовать для любой задачи. Аналитики данных воспринимают линейную регрессию как простой, понятный, но эффективный алгоритм для оценок, а в версии логистической регрессии и для классификации.

В СВЕТЕ ПРОСТОГО И СЛОЖНОГО

Простое и *сложное* не являются абсолютными терминами в машинном обучении; их значение связано с конкретной задачей данных. Некоторые алгоритмы являются простым суммированием, в то время как другие требуют сложных вычислений и манипулирования с данными (язык Python работает как с простыми, так и со сложными алгоритмами). Данные имеют значение. Хорошей практикой является тестирование нескольких моделей, начиная с простых. Вы можете обнаружить, что простое решение работает лучше во многих случаях. Например, вы можете захотеть сохранить простоту и использовать линейную модель вместо более сложного подхода, а получите более точные результаты. По сути, это то, что подразумевает теорема об отсутствии бесплатных обедов: ни один подход не годится для всех проблем, и даже самое простое решение может содержать ключ к решению важной проблемы.

Теорема "бесплатных обедов не бывает" (no free lunch) Дэвида Вольперта (David Wolpert) и Уильяма Макриди (William Macready) утверждает, что два "любых алгоритма оптимизации эквивалентны, когда их производительности близки по всем возможным задачам". Если алгоритмы абстрактно эквивалентны, ни один не превосходит другой, если только обратное не доказано на конкретной, практической задаче. Более подробную информацию о теореме см. на `http://www.no-free-lunch.org/`; некоторая непосредственно относится к машинному обучению.

Определение семейства линейных моделей

Линейная регрессия (linear regression) — это статистическая модель, которая определяет взаимосвязь между целевой переменной и набором прогнозирующих признаков. Для этого применяется формула следующего типа:

```
y = bx + a.
```

Вы можете перевести эту формулу в нечто читабельное и полезное для многих задач. Например, если вы пытаетесь предсказать свои продажи на основании прошлых результатов и доступных данных о расходах на рекламу, предыдущая формула преобразуется в

```
продажи = b * (расходы на рекламу) + a
```

СОВЕТ

Если вспомнить алгебру и геометрию, изученные в средней школе, то формула `y = bx + a` — это линия на координатной плоскости, состоящей из оси `x` (абсцисса) и оси `y` (ордината). Большая часть математики в машинном обучении относится на самом деле к старшей школе, но Python хорошо справляется с ней.

Вы можете упростить понимание формулы, объяснив ее компоненты: `a` — это значение пересечения (значение `y`, когда `x` равно нулю); `b` — это коэффициент, который выражает наклон линии (соотношение между `x` и `y`). Если `b` положительно, `y` увеличивается и уменьшается по мере увеличения и уменьшения `x`, а когда `b` отрицательно, `y` ведет себя противоположным образом. Вы можете понимать `b` как единичное изменение по `y`, учитывая единичное изменение по `x`. Когда значение `b` близко к нулю, влияние `x` на `y` незначительно, но если значение `b` высокое, положительное или отрицательное, влияние изменений `x` на `y` будет значительным.

Следовательно, линейная регрессия может найти наилучшее значение `y = bx + a` и представить отношение между целевой переменной `y` и вашим прогнозирующим признаком `x`. Как `a` (альфа), так и `b` (коэффициент бета) оцениваются на основе данных, и их находят с использованием алгоритма линейной регрессии так, чтобы разность между всеми действительными целевыми значениями `y` и всеми значениями `y`, полученными из формулы линейной регрессии, была минимально возможной.

Вы можете выразить это отношение графически как сумму квадратов всех вертикальных расстояний между всеми точками данных и линией регрессией. Такая сумма всегда является минимально возможной, если вы правильно рассчитываете линию регрессии, используя такую оценку, как обычные наименьшие квадраты, которая получается из статистики или эквивалентного градиентного спуска методом машинного обучения. Различия между реальными значениями `y` и линией регрессии (прогнозируемыми значениями `y`) определяются как остатки (поскольку они являются тем, что осталось после регрессии, — ошибками).

Использование большего количества переменных

При использовании для прогнозирования y одной переменной применяйте простую линейную регрессию, но при работе со многими переменными используйте множественную линейную регрессию. Если у вас много переменных, их масштаб не важен для создания точных прогнозов линейной регрессии. Однако хорошей привычкой является стандартизация X, поскольку масштаб переменных весьма важен для некоторых вариантов регрессии (которые вы увидите позже), а для вашего понимания данных полезно сравнить коэффициенты в соответствии с их влиянием на y.

В следующем примере используется набор данных Boston из Scikit-learn. Он пытается прогнозировать цены на жилье в Бостоне, используя линейную регрессию. В примере также делается попытка определить, какие переменные больше влияют на результат, поэтому пример стандартизирует предикторы.

```
from sklearn.datasets import load_boston
from sklearn.preprocessing import scale
boston = load_boston()
X = scale(boston.data)
y = boston.target
```

Класс регрессии в пакете Scikit-learn является частью модуля `linear_model`. Предварительно отмасштабировав переменную X, вам не будут нужны другие подготовки или специальные параметры, решение о которых нужно было бы принять при использовании этого алгоритма.

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression(normalize=True)
regression.fit(X, y)
```

Теперь, когда алгоритм настроен, вы можете использовать метод `score`, чтобы сообщить о мере *R2*, которая представляет собой показатель в диапазоне от 0 до 1 и указывает, что использование конкретной регрессионной модели лучше для прогнозирования y, чем использование простого среднего значения. (В процессе *подбора* (fitting) создается линия или кривая, которая наилучшим образом соответствует точкам предоставленных данных. Вы подбираете линию или кривую к точкам данных для решения различных задач, таких как прогнозирование на основе тенденций или шаблонов, создаваемых данными.) Оценку R^2 можно также рассматривать как количество целевой информации, объясняемой моделью (такой же, как квадратичная корреляция), поэтому получение значения около 1 означает возможность объяснить бо́льшую часть переменной y с помощью модели:

```
print(regression.score(X, y))
```

Вот итоговая оценка:

```
0.740607742865
```

В этом случае оценка R^2 по ранее установленным данным составляет около 0,74, что является хорошим результатом для простой модели. Вы можете интерпретировать оценку R^2 как процент информации, имеющейся в целевой переменной, которая была объяснена моделью с использованием предикторов. Таким образом, оценка 0,74 означает, что модель соответствует большей части информации, которую вы хотели прогнозировать, и только 26% из них остаются необъясненными.

Расчет оценки R^2 на том же наборе данных, который использовался для обучения, считается разумным в статистике при использовании линейных моделей. В науке о данных и машинном обучении всегда имеет смысл проверять результаты на данных, которые не использовались для обучения. Более сложные алгоритмы способны запоминать данные лучше, чем извлечь из них зависимости, но иногда это утверждение может быть верным и для более простых моделей, таких как линейная регрессия.

Чтобы понять, что движет оценками в модели множественной регрессии, нужно взглянуть на атрибут `coefficients_`, который представляет собой массив, содержащий коэффициенты регрессии бета. *Коэффициенты* (coefficient) — это числа, полученные с помощью модели линейной регрессии для эффективного преобразования входных переменных формулы в целевой прогноз y. Выводя одновременно атрибут `boston.DESCR`, вы поймете, на какую переменную ссылаются коэффициенты. Функция `zip` обеспечивает итерацию обоих атрибутов, и вы можете вывести ее для отчетов.

```
print([a + ':' + str(round(b, 2)) for a, b in zip(
    boston.feature_names, regression.coef_,)])
```

Перечисленные переменные и их округленные коэффициенты (значения `b`, или наклон линии, описаны выше, в разделе "Определение семейства линейных моделей") таковы:

```
['CRIM:-0.92', 'ZN:1.08', 'INDUS:0.14', 'CHAS:0.68',
 'NOX:-2.06', 'RM:2.67', 'AGE:0.02', 'DIS:-3.1', 'RAD:2.66',
 'TAX:-2.08', 'PTRATIO:-2.06', 'B:0.86', 'LSTAT:-3.75']
```

`DIS` — это взвешенные расстояния для пяти центров занятости. Оно показывает главное абсолютное единичное изменение. Например, в сфере недвижимости ценность домов снижается, если они находятся слишком далеко от мест интересов людей (например, работы). В отличие от него, `AGE` и `INDUS`, в

обеих пропорциях, описывают возраст здания и демонстрируют, доступны ли в этом районе нетрадиционные виды деятельности; они не сильно влияют на результат, поскольку абсолютное значение их коэффициентов бета ниже, чем у DIS.

Ограничения и проблемы

Хотя линейная регрессия является простым, но эффективным инструментом оценки, у нее довольно много проблем. И в некоторых случаях они способны уменьшить выгоду от использования линейной регрессии, но в действительности это зависит от данных. Вы определяете, существуют ли какие-либо проблемы, используя метод и проверяя его эффективность. Если вы не обработаете данные (см. главу 19), то можете столкнуться с такими ограничениями.

- Линейная регрессия может моделировать только количественные данные. При моделировании категорий в качестве ответа необходимо преобразовать данные в логистическую регрессию.
- Если данные отсутствуют и вы не исправите это должным образом, модель перестает работать. Важно заполнить отсутствующие значения, или использовать нулевое значение для переменной, или создать дополнительную двоичную переменную, указывающую, что значение отсутствует.
- Кроме того, выбросы весьма разрушительны для линейной регрессии, поскольку она пытается минимизировать квадратичное значение остатков, а выбросы имеют большие остатки, заставляя алгоритм сосредоточиться на них больше, чем на массе обычных точек.
- Отношение между целью и каждой переменной предиктора основывается на одном коэффициенте — не существует автоматического способа представления сложных отношений, таких как параболический (существует уникальное значение x, максимизирующее y) или экспоненциальный рост. Единственный способ моделировать такие отношения — использовать математические преобразования x (и иногда y) или добавлять новые переменные. В главе 19 рассматривается как использование преобразований, так и добавление переменных.
- Наибольшим ограничением является то, что линейная регрессия суммирует члены, способные варьироваться независимо один от другого. Трудно понять, как представить влияние определенных переменных, которые влияют на результат очень по-разному в зависимости от их значения. Решение состоит в том, чтобы создать условия взаимодействия, т.е. умножить две или более переменных на новую переменную. Однако для этого необходимо, чтобы вы знали,

какие переменные нужно умножить, и создали новую переменную перед запуском линейной регрессии. Короче говоря, вы не можете легко представить сложные ситуации с вашими данными, просто упростив их.

Переход к логистической регрессии

Линейная регрессия хорошо подходит для оценки значений, но это не лучший инструмент для прогнозирования класса наблюдения. Несмотря на то что статистическая теория не советует этого, вы можете попытаться классифицировать бинарный класс, оценив один класс как 1, а другой как 0. В большинстве случаев результаты разочаровывают, поэтому статистическая теория не ошиблась!

Дело в том, что линейная регрессия работает на континууме числовых оценок. Для правильной классификации нужна более подходящая мера, например, вероятность владения классом. Благодаря следующей формуле можно преобразовать числовую оценку линейной регрессии в вероятность, более подходящую для описания соответствия наблюдения классу:

```
вероятность класса = exp(r) / (1+exp(r))
```

`r` — это результат регрессии (сумма переменных, взвешенных по коэффициентам), а `exp` — экспоненциальная функция. `exp(r)` соответствует числу Эйлера `e`, возведенному в степень `r`. Линейная регрессия, использующая такую формулу (также называемую функцией связи) для преобразования ее результатов в вероятности, является логистической регрессией.

Применение логистической регрессии

Логистическая регрессия (logistic regression) аналогична линейной, и единственное отличие заключается в данных y, которые должны содержать целочисленные значения, указывающие класс относительно наблюдения. Используя набор данных Iris из модуля `datasets` Scikit-learn, вы можете использовать значения 0, 1 и 2 для обозначения трех классов, соответствующих трем видам:

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data[:-1,:],
y = iris.target[:-1]
```

Чтобы упростить работу с примером, оставьте одно значение, чтобы позже можно было использовать его для проверки эффективности модели логистической регрессии:

```
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression()
logistic.fit(X, y)
single_row_pred = logistic.predict(
    iris.data[-1, :].reshape(1, -1))
single_row_pred_proba = logistic.predict_proba(
    iris.data[-1, :].reshape(1, -1))
print ('Predicted class %s, real class %s'
       % (single_row_pred, iris.target[-1]))
print ('Probabilities for each class from 0 to 2: %s'
       % single_row_pred_proba)
```

Предыдущий фрагмент кода выводит следующее:

```
Predicted class [2], real class 2
Probabilities for each class from 0 to 2:
    [[ 0.00168787 0.28720074 0.71111138]]
```

В отличие от линейной регрессии, логистическая регрессия не только выводит результирующий класс (в данном случае класс 2), но также оценивает вероятность того, что наблюдение является частью всех трех классов. Основываясь на наблюдении, используемом для прогнозирования, логистическая регрессия оценивает вероятность в 71% происхождения из класса 2 — высокая вероятность, но не точная оценка, поэтому оставляется запас неопределенности.

СОВЕТ

Использование вероятностей позволяет предсказать наиболее вероятный класс, но вы также можете упорядочить прогнозы в отношении принадлежности к этому классу. Это особенно полезно в медицинских целях: ранжирование прогноза с точки зрения вероятности относительно других помогает выявить, какие пациенты подвергаются наибольшему риску заболеть или уже имеют заболевание.

Учет нескольких классов

Логистическая регрессия автоматически решает задачу с несколькими классами (она началась с распознавания трех видов ирисов). Большинство алгоритмов, предоставляемых библиотекой Scikit-learn, прогнозируют вероятности или оценку для класса. Они способны автоматически решать задачи с несколькими классами, используя две стратегии.

- **"Один против всех"** (one versus rest). Алгоритм сравнивает каждый класс со всеми остальными классами, создавая модель для каждого класса. Если для прогноза есть десять классов, значит, есть десять моделей. Этот подход основан на классе `OneVsRestClassifier` из Scikit-learn.
- **"Один против одного"** (one versus one). Алгоритм сравнивает каждый класс с каждым отдельным оставшимся классом, создавая количество моделей, эквивалентное `n * (n - 1) / 2`, где `n` — количество классов. Если у вас есть десять классов, то у вас 45 моделей, `10 * (10 - 1) / 2`. Этот подход основан на классе `OneVsOneClassifier` из Scikit-learn.

В случае логистической регрессии стандартной многоклассовой стратегией является "один против всех". Пример, приведенный в этом разделе, показывает, как использовать обе стратегии с набором рукописных цифр, содержащим классы для чисел от 0 до 9. Следующий код загружает данные и помещает их в переменные:

```
from sklearn.datasets import load_digits
digits = load_digits()
train = range(0, 1700)
test = range(1700, len(digits.data))
X = digits.data[train]
y = digits.target[train]
tX = digits.data[test]
ty = digits.target[test]
```

На самом деле наблюдения являются сеткой из значений пикселей. Размеры сетки составляют 8×8 пикселей. Чтобы облегчить изучение данных с помощью алгоритмов машинного обучения, код объединяет их в список из 64 элементов. Пример резервирует часть доступных примеров для теста.

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.multiclass import OneVsOneClassifier
OVR = OneVsRestClassifier(LogisticRegression()).fit(X, y)
OVO = OneVsOneClassifier(LogisticRegression()).fit(X, y)
print('One vs rest accuracy: %.3f' % OVR.score(tX, ty))
print('One vs one accuracy: %.3f' % OVO.score(tX, ty))
```

Производительность двух многоклассовых стратегий составляет

```
One vs rest accuracy: 0.938
One vs one accuracy: 0.969
```

Два многоклассовых класса, `OneVsRestClassifier` и `OneVsOneClassifier`, работают за счет включения оценщика (в данном случае `LogisticRegression`). После включения они обычно работают так же, как и любой другой алгоритм

обучения в Scikit-learn. Интересно, что стратегия "один против одного" обеспечила высочайшую точность благодаря бо́льшому количеству конкурирующих моделей.

Просто, как наивный байесовский классификатор

Вы можете удивиться, почему алгоритм Байеса называют наивным. Наивная часть исходит из его формулировки; он делает некоторые крайние упрощения стандартных вычислений вероятности. Ссылка на Байеса в его названии относится к преподобному Байесу и его теореме о вероятности.

Преподобный Томас Байес (1701–1761), английский статистик и философ, сформулировал свою теорему в первой половине XVIII века. Теорема никогда не была опубликована при его жизни. Он глубоко революционизировал теорию вероятности, введя идею условной вероятности, т.е. вероятности, обусловленной доказательствами.

Конечно, это помогает начать с самого начала — с самой вероятности. *Вероятность* (probability) говорит о вероятности события и выражается в числовой форме. Вероятность события измеряется в диапазоне от 0 до 1 (от 0 до 100%), и ее эмпирически определяют подсчетом количества случаев, когда произошло конкретное событие относительно всех событий. Вы можете рассчитать это по данным!

Когда вы наблюдаете события (например, когда у объекта есть определенная характеристика) и хотите оценить вероятность, связанную с неким событием, вы подсчитываете количество раз, когда признак появляется в данных, и делите эту цифру на общее количество доступных наблюдений. Результатом является число в диапазоне от 0 до 1, которое выражает вероятность.

Когда вы оцениваете вероятность события, вы склонны полагать, что можете применить вероятность в каждой ситуации. Термином для этого убеждения является *априорность* (a priori), поскольку он представляет собой первую оценку вероятности в отношении события (которое приходит на ум первым). Например, если вы оцениваете вероятность того, что неизвестный человек является женщиной, вы можете после некоторого подсчета сказать, что это 50%, что и является априорной или первой вероятностью, которой вы будете придерживаться.

Априорная вероятность может измениться перед лицом доказательств, т.е. чего-то, что может радикально изменить ваши ожидания. Например, свидетельством того, является ли человек мужчиной или женщиной, может быть

наличие у него длинных или коротких волос. Вы можете оценить наличие длинных волос как событие с вероятностью 35% для населения в целом, но среди женщин — это 60%. Если этот процент выше среди женского населения, в отличие от общей вероятности (априори наличие длинных волос), это должна быть некоторая полезная информация, которую вы можете использовать.

Представьте, что вам нужно угадать, мужчина это или женщина, и есть доказательство, что у него длинные волосы. Это похоже на задачу прогнозирования, и, в конце концов, эта ситуация действительно похожа на прогнозирование категориальной переменной на основе данных: у нас есть целевая переменная с различными категориями, и нужно выявить вероятность каждой категории на основе свидетельств и данных. Байес представил полезную формулу:

$$P(A|B) = P(B|A)*P(A) / P(B)$$

Формула выглядит как статистический жаргон и немного нелогична, поэтому ее необходимо объяснить подробно. Чтение формулы в контексте предыдущего примера придает ей смысл и делает немного более понятной.

- P(A|B) — это вероятность быть женщиной (событие A) с длинными волосами (свидетельство B). Эта часть формулы определяет то, что вы хотите прогнозировать. Другими словами, она говорит, что нужно предсказать y, учитывая x, где y — результат (мужчина или женщина), а x — свидетельство (длинные или короткие волосы).
- P(B|A) — вероятность наличия длинных волос, когда человек — женщина. В данном случае вы уже знаете, что это 60%. В любой задаче с данными вы можете легко получить этот показатель, сопоставив признаки с целевым результатом.
- P(A) — вероятность быть женщиной, общий шанс 50% (априори).
- P(B) — вероятность иметь длинные волосы, которая составляет 3% (также априори).

СОВЕТ

Такие части формулы, как P(A|B), читаются следующим образом: вероятность A задана B. Символ | означает задано. Вероятность, выраженная таким образом, является условной, потому что это вероятность A, обусловленная свидетельством B. В этом примере подстановка чисел в формулу приводит к: 60% * 50% / 35% = 85,7%.

Поэтому, возвращаясь к предыдущему примеру, даже если вероятность быть женщиной составляет 50%, знание только таких данных, как длинные волосы, увеличивает их до 85,7%, что является куда более благоприятным шансом для предположения. Вы можете быть более уверенными в предположении,

что человек с длинными волосами — женщина, поскольку у вас есть чуть менее 15% шансов ошибиться.

Наивный Байес не такой уж и наивный

Наивный байесовский метод, использующий простое правило Байеса, применяет все имеющиеся доказательства для изменения предыдущей базовой вероятности ваших прогнозов. Поскольку данные содержат так много доказательств, т.е. имеют много признаков, они составляют бо́льшую сумму всех вероятностей, полученных из упрощенной наивной байесовской формулы.

Как обсуждалось выше, в разделе "Угадай число: линейная регрессия", суммирование переменных подразумевает, что модель воспринимает их как отдельные и уникальные фрагменты информации. Но в действительности это не так, поскольку приложения существуют в мире взаимосвязей, где каждая часть информации соединяется со многими другими частями. Использование одного фрагмента информации более одного раза означает, что этому конкретному фрагменту уделяется больше внимания.

Поскольку вы не знаете (или просто игнорируете) отношения между каждым доказательством, вы, вероятно, просто передаете их все наивному байесовскому классификатору. Простая и наивная передача всего, что вы знаете, формуле, действительно работает хорошо, и многие исследования сообщают о хорошей производительности, несмотря на то, что вы делаете наивное предположение. Для прогнозирования можно использовать все, хоть и не кажется, что это должно быть хорошо, учитывая сильную связь между переменными. Ниже приведены некоторые из наиболее популярных способов использования наивного байесовского классификатора.

- Создание детекторов спама (перехват всех раздражающих писем в вашем почтовом ящике).
- Анализ настроений (прогнозирование, содержит ли текст положительное или отрицательное отношение к теме, и определение настроения автора).
- Задачи обработки текста, такие как исправление орфографии или определение используемого для написания языка, либо классификация текста в более крупную категорию. Наивный байесовский классификатор здесь очень популярен, поскольку для его работы не требуется большого количества данных. Естественно, он может обрабатывать несколько классов. С некоторыми небольшими изменениями переменных (превратив их в классы) он также может

обрабатывать числовые переменные. Scikit-learn предоставляет три класса наивного байесовского классификатора в модуле `sklearn.naive_bayes`.

- `MultinomialNB`. Использует вероятности, полученные из наличия признаков. Когда признак присутствует, конечному результату присваивается определенная вероятность, которую текстовые данные указывают для прогноза.
- `BernoulliNB`. Обеспечивает полиномиальную функциональность наивного байесовского классификатора, но штрафует за отсутствие признаков. Когда признак присутствует, назначается другая вероятность, не такая, как в случае его отсутствия. Фактически он рассматривает все признаки как дихотомические переменные (распределением дихотомической переменной является распределение Бернулли). Вы также можете использовать его с текстовыми данными.
- `GaussianNB`. Определяет версию наивного байесовского классификатора, ожидающего нормального распределения всех признаков. Следовательно, этот класс является неоптимальным для текстовых данных, в которых слова редки (вместо этого используйте полиномиальное распределение или распределение Бернулли). Если ваши переменные имеют положительные и отрицательные значения, это лучший выбор.

Прогнозирование текстовых классификаций

Наивный байесовский классификатор особенно популярен при классификации документов. В текстовых задачах нередко задействованы миллионы признаков, по одному на каждое слово, написанных правильно или неправильно. Иногда текст ассоциируется с другими близлежащими словами в *n-граммах* (n-gram), т.е. последовательностями следующих друг за другом слов. Наивный байесовский алгоритм может быстро выучить текстовые признаки и обеспечить быстрые прогнозы на основе входных данных.

В этом разделе проверяется классификация текста с использованием биномиальных и полиномиальных наивных байесовских моделей, предлагаемых Scikit-learn. Примеры основаны на наборе данных `20newsgroups`, содержащем большое количество сообщений из 20 видов групп новостей. Набор данных разделен на обучающий набор для построения текстовых моделей и тестовый набор, состоящий из сообщений, которые временно следуют обучающему набору. Используйте набор тестов для проверки точности ваших прогнозов:

```
from sklearn.datasets import fetch_20newsgroups
newsgroups_train = fetch_20newsgroups(
```

```
                    subset='train', remove=('headers', 'footers',
                                            'quotes'))
newsgroups_test = fetch_20newsgroups(
        subset='test', remove=('headers', 'footers',
                               'quotes'))
```

После загрузки двух наборов в память импортируйте две наивных байесовских модели и создайте их экземпляры. На этом этапе вы устанавливаете значения альфа, которые полезны во избежание нулевой вероятности для редких объектов (нулевая вероятность исключит эти признаки из анализа). Обычно для альфа используется небольшое значение, как показано в следующем коде:

```
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
Bernoulli = BernoulliNB(alpha=0.01)
Multinomial = MultinomialNB(alpha=0.01)
```

В главе 12 вы используете прием хеширования для моделирования текстовых данных, не опасаясь появления новых слов при использовании модели после фазы обучения. Вы можете использовать два разных метода хеширования: один для подсчета слов (для полиномиального подхода) и один для записи того, появилось ли слово в двоичной переменной (биномиальный подход). Вы также можете удалить *стоп-слова* (stop word), т.е. общие слова, встречающиеся в английском языке, например *a*, *the*, *in* и т.д.

```
import sklearn.feature_extraction.text as txt
multinomial = txt.HashingVectorizer(stop_words='english',
                                    binary=False, norm=None)
binary = txt.HashingVectorizer(stop_words='english',
                               binary=True, norm=None)
```

На этом этапе вы можете обучить два классификатора и протестировать их на тестовом наборе, представляющем собой набор сообщений, которые временно появляются после обучающего набора. Мера теста — это точность, которая представляет собой процент правильных предположений, которые делает алгоритм.

```
import numpy as np
target = newsgroups_train.target
target_test = newsgroups_test.target
multi_X = np.abs(
    multinomial.transform(newsgroups_train.data))
multi_Xt = np.abs(
    multinomial.transform(newsgroups_test.data))
bin_X = binary.transform(newsgroups_train.data)
bin_Xt = binary.transform(newsgroups_test.data)
Multinomial.fit(multi_X, target)
Bernoulli.fit(bin_X, target)
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import accuracy_score
for name, model, data in [('BernoulliNB', Bernoulli, bin_Xt),
                          ('MultinomialNB', Multinomial, multi_Xt)]:
    accuracy = accuracy_score(y_true=target_test,
                              y_pred=model.predict(data))
    print ('Accuracy for %s: %.3f' % (name, accuracy))
```

Сообщаемая точность для двух наивных байесовских моделей такова:

```
Accuracy for BernoulliNB: 0.570
Accuracy for MultinomialNB: 0.651
```

Вы можете заметить, что обе модели не займут много времени на обучение и сообщение своих прогнозов для тестового набора. Учтите, что учебный набор состоит из более чем 11 000 сообщений, содержащих 300 000 слов, а тестовый набор содержит около 7 500 других сообщений.

```
print('number of posts in training: %i'
      % len(newsgroups_train.data))
D={word:True for post in newsgroups_train.data
    for word in post.split(' ')}
print('number of distinct words in training: %i'
      % len(D))
print('number of posts in test: %i'
      % len(newsgroups_test.data))
```

Запуск кода возвращает следующую полезную статистику о тексте:

```
number of posts in training: 11314
number of distinct words in training: 300972
number of posts in test: 7532
```

Ленивое обучение с ближайшими соседями

Метод *k-ближайших соседей* (K-Nearest Neighbors — KNN) — это не правила построения на основе данных, основанных на коэффициентах или вероятности. KNN работает на основе сходства. Когда нужно прогнозировать что-то вроде класса, лучше всего найти наблюдения, наиболее похожие на те, которые вы хотите классифицировать или оценить. Затем вы можете получить нужный ответ из аналогичных случаев.

Выявление того, как много наблюдений сходны, подразумевает не изучение чего-то, а измерение. Поскольку KNN ничего не изучает, он считается *ленивым* (lazy), и вы услышите, что его называют *ленивым учеником* (lazy learner) или *учеником на основе примеров* (instancebased learner). Идея в том, что подобные предпосылки обычно ведут к схожим результатам, и важно не забывать сорвать такой низко висящий фрукт, прежде чем пытаться взобраться на дерево!

Алгоритм быстр во время обучения, поскольку он должен запомнить только данные о наблюдениях. Во время прогнозов он на самом деле больше рассчитывает. Когда наблюдений слишком много, алгоритм может стать очень медленным и занимать много памяти. Лучше всего не использовать его с большими данными, или может потребоваться почти вечность, чтобы спрогнозировать что-либо! Более того, этот простой и эффективный алгоритм работает лучше, когда у вас есть отдельные группы данных без слишком большого количества переменных, поскольку алгоритм чувствителен также к *проклятию размерности* (dimensionality curse).

Проклятие размерности случается при увеличении количества переменных. Рассмотрим ситуацию, в которой вы измеряете расстояние между наблюдениями, а когда пространство становится все больше и больше, трудно найти реальных соседей — проблема KNN, которая иногда ошибочно принимает отдаленное наблюдение за ближнее. Представление этой идеи похоже на игру в шахматы на многомерной шахматной доске. При игре на классической двумерной доске большинство фигур находятся рядом, и вы можете легче находить возможности и угрозы для своих пешек, когда у вас 32 фигуры и 64 позиции. Однако, когда вы начинаете играть на трехмерной доске, например, как в некоторых научно-фантастических фильмах, ваши 32 фигуры могут потеряться в 512 возможных позициях. А теперь представьте, что играете с 12-мерной шахматной доской. Вы можете легко запутаться, что находится рядом, а что далеко, это и происходит с KNN.


СОВЕТ

Вы все еще можете сделать KNN интеллектуальным при обнаружении сходства между наблюдениями, удаляя избыточную информацию и упрощая размерность данных (используя методы сокращения данных, как объяснено в главе 14).

Прогнозирование после наблюдения соседей

Для примера, демонстрирующего использование KNN, можете снова начать с набора цифровых данных. KNN особенно полезен, как и наивный байесовский классификатор, когда нужно прогнозировать много классов, или построить слишком много моделей, или полагаться на очень сложную модель.

```
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
digits = load_digits()
train = range(0, 1700)
test = range(1700, len(digits.data))
pca = PCA(n_components = 25)
pca.fit(digits.data[train])
```

```
X = pca.transform(digits.data[train])
y = digits.target[train]
tX = pca.transform(digits.data[test])
ty = digits.target[test]
```

KNN — это алгоритм, очень чувствительный к выбросам. Кроме того, вам нужно изменить масштаб переменных и удалить некоторую избыточную информацию. В этом примере вы используете PCA. Изменение масштаба не требуется, поскольку данные представляют пиксели, а это значит, что они уже масштабированы.


СОВЕТ

Вы можете избежать проблемы с выбросами, сохраняя соседство небольшим, т.е. не заглядывая слишком далеко в поисках похожих примеров.


СОВЕТ

Знание типа данных может сэкономить вам много времени и избежать ошибок. Например, в этом случае вы знаете, что данные представляют значения пикселей. Выполнение EDA (см. главу 13) всегда является первым шагом и может дать полезную информацию, но получение дополнительной информации о том, как были получены данные и что они представляют, также является хорошей практикой и может быть столь же полезным. Чтобы увидеть эту задачу в действии, зарезервируйте случаи в `tX` и опробуйте несколько случаев, которые KNN не будет искать при поиске соседей.

```
from sklearn.neighbors import KNeighborsClassifier
kNN = KNeighborsClassifier(n_neighbors=5, p=2)
kNN.fit(X, y)
```

KNN использует меру расстояния, чтобы определить, какие наблюдения следует рассматривать в качестве возможных соседей для целевого случая. Вы можете легко изменить предопределенное расстояние, используя параметр `p`.

- Когда `p` равно 2, используйте евклидово расстояние (обсуждается в главе 15 как часть темы кластеризации).
- Когда `p` равно 1, используйте метрику манхэттенского расстояния, которая является абсолютным расстоянием между наблюдениями. В двумерном квадрате, когда вы идете от одного угла к противоположному, манхэттенское расстояние такое же, как и по периметру, а евклидово — как ход по диагонали. Хотя манхэттенское расстояние не является самым коротким маршрутом, оно более реалистично, чем евклидово, и менее чувствительно к шуму и высокой размерности.

Обычно евклидово расстояние — это правильная мера, но иногда она может дать худшие результаты, особенно если анализ включает много взаимосвязанных переменных. Следующий код показывает, что анализ с ним выглядит хорошо:

```
print('Accuracy: %.3f' % kNN.score(tX,ty) )
print('Prediction: %s Actual: %s'
      % (kNN.predict(tX[-15:,:]),ty[-15:]))
```

Код возвращает точность и пример прогноза, который можно сравнить с фактическими значениями, чтобы выявить различия:

```
Accuracy: 0.990
Prediction: [2 2 5 7 9 5 4 8 1 4 9 0 8 9 8]
    Actual: [2 2 5 7 9 5 4 8 8 4 9 0 8 9 8]
```

Осмысленный выбор параметра k

Критическим параметром, который вы должны определить в KNN, является k. По мере увеличения k KNN учитывает больше точек для своих прогнозов, и на решения меньше влияют шумные случаи, способные оказать чрезмерное влияние. Ваши решения основаны в среднем на большем количестве наблюдений, и они становятся более надежными. Когда значение k, которое вы используете, слишком велико, рассматривайте соседей, которые находятся слишком далеко, все меньше и меньше разделяя случай, который вы должны предсказать.

Это важный компромисс. Когда значение параметра k меньше, вы рассматриваете более однородный пул соседей, но можете легче совершить ошибку, приняв несколько подобных случаев как должное. Когда значение параметра k больше, вы рассматриваете больше случаев с более высоким риском наблюдения соседей, которые находятся слишком далеко или являются выбросами. Возвращаясь к предыдущему примеру с данными рукописных цифр, поэкспериментируйте с изменением значения параметра k, как показано в следующем коде:

```
for k in [1, 5, 10, 50, 100, 200]:
    kNN = KNeighborsClassifier(n_neighbors=k).fit(X, y)
    print('for k = %3i accuracy is %.3f'
          % (k, kNN.score(tX, ty))
```

После запуска этого кода вы получите представление о том, что происходит при изменении k, и определите значение k, которое наилучшим образом соответствует данным:

```
for k =   1 accuracy is 0.979
for k =   5 accuracy is 0.990
```

```
for k =  10 accuracy is 0.969
for k =  50 accuracy is 0.959
for k = 100 accuracy is 0.959
for k = 200 accuracy is 0.907
```

Экспериментируя, вы обнаружите, что установка `n_neighbors` (параметра, представляющего k) в значение 5 является оптимальным выбором, что приводит к максимальной точности. Использование только ближайшего соседа (`n_neighbors` = 1) неплохой выбор, но установка значения выше 5, напротив, приводит к уменьшению результатов в задаче классификации.


СОВЕТ

Как правило, если в наборе данных мало наблюдений, установите для k значение, близкое к квадрату количества доступных наблюдений. Но общего правила не существует, и использование разных значений k — это всегда хороший способ оптимизировать производительность KNN. Всегда начинайте с низких значений и работайте с более высокими.

Глава 18

Перекрестная проверка, отбор и оптимизация

В ЭТОЙ ГЛАВЕ...

- Чрезмерный и недостаточный подбор
- Выбор правильной метрики для проверки
- Перекрестная проверка результатов
- Выбор лучших признаков для машинного обучения
- Оптимизация гиперпараметров

Алгоритмы машинного обучения действительно могут учиться на данных. Например, четыре представленных в предыдущей главе алгоритма, хотя и довольно просты, способны эффективно оценить класс или значение после того, как им предоставлены примеры, связанные с результатами. Все это вопрос обучения по индукции, т.е. процесс извлечения общих правил из конкретных примеров. С самого детства люди обычно учатся, наблюдая примеры, выводя из них некие общие правила или идеи, а затем успешно применяют полученное правило в новых ситуациях по мере взросления. Например, если мы видим, что кто-то обжегся, прикоснувшись к огню, то понимаем, что огонь опасен и не нужно прикасаться к нему самим, чтобы узнать это.

Обучение на примере с использованием машинных алгоритмов имеет подводные камни. Ниже приведено несколько вопросов, которые могут возникнуть.

- Приведенных примеров недостаточно, чтобы судить о правиле, независимо от того, какой алгоритм машинного обучения вы используете.
- Приложение машинного обучения снабжено неправильными примерами, а следовательно, не может правильно рассуждать.
- Даже когда приложение видит достаточно правильных примеров, оно все равно не может определить правила, поскольку они слишком сложные. Сэр Исаак Ньютон, отец современной физики, рассказал историю о том, что в своей формулировке гравитации он был вдохновлен падением яблока с дерева. К сожалению, вывод универсального закона из серии наблюдений не является автоматическим следствием для большинства из нас, то же самое относится и к алгоритмам.

При изучении машинного обучения важно учитывать эти подводные камни. Количество данных, их качество и характеристики алгоритма обучения решают, может ли приложение машинного обучения хорошо обобщать новые случаи. Если с любым из пунктов что-то не так, могут возникнуть серьезные ограничения. Как аналитик данных, вы должны осознать и научиться избегать подобных ошибок в своих экспериментах по науке о данных.


СОВЕТ

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_18_Performing_Cross_Validation_Selection_and_Optimization.ipynb`.

Размышляя над проблемой подбора модели

Подбор модели (fitting a model) подразумевает, в первую очередь, изучение на основе данных представления правил, которые создали данные. С математической точки зрения подбор модели аналогичен выявлению неизвестной функции, с которой вы встречались в старших классах, например, `y=4x^2+2x`, в ходе простого наблюдая за результатами `y`. Следовательно, вы ожидаете, что алгоритмы машинного обучения создают математические формулы, определяя, как работает реальность, на основе предоставленных примеров.

Демонстрация того, являются ли такие формулировки реальными, выходит за рамки науки о данных. Важнее всего то, что они работают, создавая точные прогнозы. Например, даже если вы можете описать бо́льшую часть физического мира с помощью математических функций, вы зачастую не можете описать

таким образом социальную и экономическую динамику, но люди все равно пытаются угадать их.

Подводя итог, как аналитик данных вы всегда должны стремиться аппроксимировать реальные, неизвестные функции, лежащие в основе проблем, с которыми встретились, используя лучшую доступную информацию. Результат вашей работы оценивается на основе вашей способности прогнозировать конкретные результаты (целевой результат) с учетом определенных предпосылок (данных) благодаря полезному набору алгоритмов (алгоритмов машинного обучения).

Ранее в книге вы видели нечто похожее на реальную функцию или закон, когда была представлена линейная регрессия, имеющая свою собственную формулировку. Линейная формула `y = Bx + a`, которая математически представляет линию на плоскости, зачастую способна хорошо аппроксимировать обучающие данные, даже если они не представляют линию или что-то похожее на нее. Как и в случае линейной регрессии, все другие алгоритмы машинного обучения имеют внутреннюю формулу (а некоторые, например нейронные сети, даже требуют, чтобы их формулу определяли с нуля). Формула линейной регрессии является одной из самых простых, а формулы других алгоритмов обучения могут показаться довольно сложными. Вам не нужно точно знать, как они работают, необходимо иметь представление о том, насколько они сложны, представляют ли они линию или кривую и могут ли воспринимать выбросы или зашумленные данные. Планируя учиться на основе данных, вы должны учитывать следующие проблемные аспекты на основе формул, которые намереваетесь использовать.

1. Является ли алгоритм обучения наилучшим, который может аппроксимировать неизвестную функцию, которую вы представляете используемыми данными. Чтобы принять такое решение, рассмотрите эффективность формул алгоритма обучения на имеющихся данных и сравните их с другими, альтернативными формулами из других алгоритмов.
2. Является ли конкретная формула алгоритма обучения слишком простой по отношению к скрытой функции, чтобы сделать ее оценку (это называется *проблемой смещения* (bias problem)).
3. Является ли конкретная формула алгоритма обучения слишком сложной по отношению к скрытой функции, чтобы ее можно было выяснить (это приводит к *проблеме дисперсии* (variance problem)).

Не все алгоритмы подходят для каждой проблемы данных. Если у вас недостаточно данных или данные полны ошибочной информации, некоторым формулам может оказаться слишком сложно определить реальную функцию.

Понятие смещения и дисперсии

Если выбранный вами алгоритм обучения неспособен правильно учиться на основе данных и не дает хороших результатов, причина заключается в смещении или дисперсии его оценок.

- **Смещение** (bias). Учитывая простоту формул, ваш алгоритм имеет тенденцию переоценивать или недооценивать реальные правила, лежащие в основе данных, и систематически ошибочен в определенных ситуациях. Простые алгоритмы обладают высоким смещением; имея мало внутренних параметров, они обычно хорошо представляют только простые формулы.
- **Дисперсия** (variance). Учитывая сложность формул, ваш алгоритм имеет тенденцию извлекать слишком много информации из данных и обнаруживать несуществующие правила, что приводит к ошибочным прогнозам при работе с новыми данными. Вы можете считать дисперсию проблемой, связанной с запоминанием. Сложные алгоритмы могут запоминать особенности данных благодаря большому количеству внутренних параметров алгоритмов. Тем не менее запоминание не подразумевает какого-либо понимания правил.

Смещение и дисперсия зависят от сложности формул, лежащих в основе алгоритма обучения, а также от сложности формулы, которая, как предполагается, создает наблюдаемые вами данные. Тем не менее, когда вы рассматриваете конкретную проблему с использованием доступных правил данных, вам лучше иметь высокое смещение или дисперсию, если

- **У вас мало наблюдений**. Более простые алгоритмы работают лучше, независимо от того, какова неизвестная функция. Сложные алгоритмы, как правило, слишком многому учатся по данным, оценивая их неточно.
- **У вас много наблюдений**. Сложные алгоритмы всегда уменьшают дисперсию. Сокращение происходит потому, что даже сложные алгоритмы не могут многому научиться на данных, поэтому они изучают только правила, а не какой-либо беспорядочный шум.
- **У вас много переменных**. При условии, что у вас также есть много наблюдений, более простые алгоритмы стремятся найти способ аппроксимации даже сложных скрытых функций.

Определение стратегии выбора моделей

Когда вы сталкиваетесь с проблемой машинного обучения, вы обычно мало знаете об этой проблеме и не знаете, справится ли с ней конкретный алгоритм. Следовательно, вы на самом деле не знаете, вызвана ли проблема смещением

или дисперсией, хотя обычно можете использовать эмпирическое правило, что если алгоритм прост, он будет иметь высокое смещение, а если сложен — то высокую дисперсию. Даже работая с обычными, хорошо документированными приложениями обработки данных, вы заметите, что то, что работает в других ситуациях (как описано в научных и отраслевых статьях), зачастую не очень хорошо подходит для вашего приложения, поскольку данные отличаются.

Вы можете резюмировать эту ситуацию, используя знаменитую теорему "бесплатных обедов не бывает" математика Дэвида Вольперта: два любых алгоритма машинного обучения эквивалентны по производительности при тестировании по всем возможным задачам. Следовательно, невозможно сказать, что один алгоритм всегда лучше другого; один может быть лучше, чем другой, только когда используется для решения конкретных задач. Вы можете взглянуть на концепцию иначе: нет одного рецепта для всех проблем! Лучшая и единственная стратегия — опробовать все, что можно, и проверить результаты, используя контролируемый научный эксперимент. Использование этого подхода гарантирует, что кажущееся работающим — это то, что действительно работает, и самое главное, что будет продолжать работать с новыми данными. Несмотря на то что вы можете быть более уверенными в преимуществе одних обучаемых над другими, вы никогда не сможете сказать, какой алгоритм машинного обучения лучше, до того, как опробуете его и оцените его производительность по вашей задаче.

На данном этапе вы должны рассмотреть важный, но недооцененный аспект успеха обеспечения вашего проекта данных. Для поиска лучшей модели и наилучших результатов важно определить метрику оценки, которая отличает хорошую модель от плохой по деловой или научной проблеме, которую вы хотите решить. Фактически для некоторых проектов вам, возможно, придется избегать прогнозирования негативных случаев, когда они позитивные; в других может потребоваться определить только все позитивные; а в третьих достаточно будет упорядочить их так, чтобы позитивные предшествовали негативным, и вам не нужно проверять их все.

Выбирая алгоритм, вы также автоматически выбираете процесс оптимизации, управляемый метрикой оценки, которая сообщает о своей производительности алгоритму, чтобы он мог лучше настроить его параметры. Например, при использовании линейной регрессии метрика представляет собой среднеквадратическую ошибку, заданную вертикальным расстоянием наблюдений от линии регрессии. Следовательно, он автоматический, и вы можете легче выявить производительность алгоритма, обеспечиваемую такой стандартной метрикой оценки.

Помимо принятия стандартной метрики, некоторые алгоритмы позволяют выбирать предпочтительную функцию оценки. В других случаях, когда вы не

можете указать на лучшую функцию оценки, вы все равно можете влиять на существующую метрику оценки, соответствующим образом исправляя некоторые из ее гиперпараметров, тем самым косвенно оптимизируя алгоритм для другой метрики.

Прежде чем приступить к обучению на ваших данных и созданию прогнозов, всегда подумайте, что может быть лучшим показателем эффективности для вашего проекта. Scikit-learn предлагает доступ к широкому спектру мер как для классификации, так и для регрессии. Модуль `sklearn.metrics` позволяет вызывать процедуры оптимизации, используя простую строку или вызывая функцию ошибки из своих модулей. В табл. 18.1 приведены показатели, обычно используемые для задач регрессии.

Таблица 18.1. Меры оценки регрессии

Вызываемая строка	Функция
`mean_absolute_error`	`sklearn.metrics.mean_absolute_error`
`mean_squared_error`	`sklearn.metrics.mean_squared_error`
`r2`	`sklearn.metrics.r2_score`

Строка `r2` указывает статистическую меру линейной регрессии, R^2 (R в квадрате). Она выражает сравнение модели по прогнозирующей способности с простым средним значением. Приложения машинного обучения редко используют эту меру, поскольку она явно не сообщает об ошибках, допущенных моделью, хотя высокие значения R^2 подразумевают меньше ошибок. Более реальными метриками для регрессионных моделей являются среднеквадратические ошибки и средние абсолютные ошибки.

Квадратичные ошибки штрафуют экстремальные значения больше, тогда как абсолютная ошибка взвешивает все ошибки одинаково. Таким образом, на самом деле необходимо рассмотреть компромисс между максимально возможным снижением ошибки экстремальных наблюдений (квадратичная ошибка) или попыткой снизить ошибку для большинства наблюдений (абсолютная ошибка). Выбор, который вы делаете, зависит от конкретного приложения. Если экстремальные значения представляют критические ситуации для вашего приложения, лучше использовать квадрат ошибки. Однако, если ваша задача заключается в том, чтобы минимизировать общие и обычные наблюдения, как это часто бывает при прогнозировании задач с продажами, используйте в качестве эталона среднюю абсолютную ошибку. Выбор возможен даже для сложных задач классификации, как показано в табл. 18.2.

Таблица 18.2. Меры по оценке классификации

Вызываемая строка	Функция
`accuracy`	`sklearn.metrics.accuracy_score`
`precision`	`sklearn.metrics.precision_score`
`recall`	`sklearn.metrics.recall_score`
`f1`	`sklearn.metrics.f1_score`
`roc_auc`	`sklearn.metrics.roc_auc_score`

Точность (accuracy) — это простейшая мера ошибки в классификации, подсчитывающая (в процентах), сколько из прогнозов верны. Он учитывает, угадал ли алгоритм машинного обучения правильный класс. Эта мера работает как с бинарными, так и с многоклассовыми задачами. Несмотря на то что это простая мера, оптимизация точности может вызвать проблемы, если между классами существует дисбаланс. Например, это может быть проблемой, если класс является популярным или преобладающим, например, при обнаружении мошенничества, когда большинство транзакций фактически являются законными, по отношению к нескольким криминальным транзакциям. В таких ситуациях алгоритмы машинного обучения, оптимизированные для точности, склонны угадывать преобладающий класс и чаще всего ошибаются с редкими классами, что является нежелательным поведением для алгоритма, который, как вы ожидаете, правильно угадывает все классы, а не только несколько избранных.

Точность и отзыв (precision and recall), а также их совместная оптимизация по *оценке F1* (F1 score) могут устранить проблемы, не решаемые с помощью точности. Точность — это точность прогнозирования. При прогнозировании класса она отражает процент случаев, когда класс был прав. Например, вы можете использовать точность при диагностике рака у пациентов после оценки данных их обследования. Ваша точность в этом случае — процент пациентов, у которых действительно есть рак, среди тех, у кого рак диагностирован. Поэтому, если вы поставили диагноз десяти больным и девять из них действительно больны, ваша точность составляет 90%.

Вы сталкиваетесь с совершенно разными последствиями, когда рак не диагностирован у пациента, у которого он есть, или диагностирован у здорового пациента. Точность рассказывает только часть истории, поскольку есть пациенты с раком, которых вы диагностировали как здоровых, и это ужасная проблема. Мера отзывов рассказывает вторую часть истории. Она сообщает ваш

процент правильных догадок среди всего класса. Например, в предыдущем случае метрика отзыва — это процент пациентов, для которых вы правильно диагностировали заболевание раком. Если из 20 больных раком вы диагностировали только 9, ваш отзыв составит 45%.

При использовании вашей модели вы можете быть точны, но при этом иметь низкий уровень отзыва или получить высокий уровень отзыва, но потерять в процессе точность. К счастью, точность и отзыв можно максимизировать вместе, используя оценку F1, которая использует формулу: `F1 = 2 * (precision * recall) / (precision + recall)`. Использование оценки F1 гарантирует, что вы всегда получите наилучшую точность и отзыв.

Площадь под кривой рабочей характеристики приемника (Receiver Operating Characteristic Area Under Curve — ROC AUC) полезна, если вы хотите упорядочить классификации в соответствии с их вероятностью правильности. Следовательно, при оптимизации ROC AUC в предыдущем примере алгоритм обучения попытается сначала упорядочить (отсортировать) пациентов, начиная с тех, у кого наиболее вероятно заболевание раком, и заканчивая теми, кто менее всего подвержен риску. ROC AUC выше, если упорядоченность хороша, и низка, если она плоха. Если ваша модель имеет высокий ROC AUC, проверьте наиболее вероятно больных пациентов. Другой пример — проблема обнаружения мошенничества, когда вы хотите упорядочить клиентов в соответствии с риском мошенничества. Если ваша модель имеет хороший ROC AUC, тщательно проверьте только самых рискованных клиентов.

Различие между учебными и тестовыми наборами

Если вы научились выбирать метрику ошибок для классификации и регрессии, то следующим шагом в стратегии выбора лучшей модели является экспериментирование и оценка решений, а также их способность обобщения новых случаев. В качестве примера правильных процедур для экспериментов с алгоритмами машинного обучения начните с загрузки набора данных Boston (популярный пример набора данных, созданного в 1970-х годах), который состоит из цен на жилье в Бостоне, различных измерений характеристик домов и показателей жилого района, где находится каждый дом.

```
from sklearn.datasets import load_boston
boston = load_boston()
X, y = boston.data, boston.target
```

Обратите внимание, что набор данных содержит более 500 наблюдений и 13 объектов. Поскольку целью является мера цены, используйте линейную регрессию и оптимизируйте результат, используя среднеквадратическую ошибку. Вы должны гарантировать, что линейная регрессия является хорошей моделью

для набора данных Boston, и количественно оценить, насколько хорошо она используется согласно среднеквадратической ошибке (что позволяет сравнивать ее с альтернативными моделями).

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
regression = LinearRegression()
regression.fit(X,y)
print('Mean squared error: %.2f' % mean_squared_error(
       y_true=y, y_pred=regression.predict(X)))
```

Вот результирующая среднеквадратическая ошибка, созданная командами:

```
Mean squared error: 21.90
```

После того как модель была дополнена данными (обучающими данными, поскольку они предоставляет примеры для изучения), функция ошибок `mean_squared_error` сообщает об ошибке прогнозирования данных. Среднеквадратическая ошибка равна 21,90, что, по-видимому, является хорошей мерой, но рассчитывается непосредственно на обучающем наборе, поэтому вы не можете быть уверены, что он также может работать с новыми данными (алгоритмы машинного обучения хороши как для обучения, так и для запоминания примеров).

В идеале вам необходимо выполнить тест данных, которые алгоритм никогда не видел, чтобы исключить запоминание. Только так вы сможете определить, будет ли ваш алгоритм хорошо работать при поступлении новых данных. Для решения этой задачи вы ожидаете новые данные, делаете для них прогнозы, а затем сравниваете прогнозы с реальностью. Но решение этой задачи может занять много времени и может стать как рискованным, так и дорогостоящим, в зависимости от типа проблемы, которую вы хотите решить с помощью машинного обучения (например, некоторые приложения, такие как для обнаружения рака, могут быть невероятно рискованными для экспериментов, поскольку на карту поставлена жизнь).

К счастью, у вас есть другой способ получить тот же результат. Чтобы смоделировать наличие новых данных, вы можете разделить наблюдения на тестовые и обучающие примеры. В науке о данных довольно часто для теста выделяют 25–30% имеющихся данных, а для обучения прогностической модели используют оставшиеся 70–75%. Ниже приведен пример того, как вы можете разделить данные в Python.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
                    X, y, test_size=0.30, random_state=5)
print(X_train.shape, X_test.shape)
```

Код выводит результирующие формы обучающих и тестовых наборов, причем первый составляет 70% от начального размера набора данных, а второй — только 30%:

```
(354, 13) (152, 13)
```

В примере обучающие и тестовые переменные `X` и `Y` разделяются на отдельные переменные с помощью функции `train_test_split`. Параметр `test_size` указывает набор тестов, состоящий из 30% доступных наблюдений. Функция всегда выбирает тестовый образец случайным образом. Теперь вы можете использовать учебный набор для обучения:

```
regression.fit(X_train,y_train)
print('Train mean squared error: %.2f' % mean_squared_error(
      y_true=y_train, y_pred=regression.predict(X_train)))
```

Вывод показывают среднеквадратичную ошибку обучающего набора:

```
Train mean squared error: 19.07
```

В этот момент вы снова подбираете модель, и код сообщает о новой ошибке обучения 19.07, которая как-то отличается от предыдущей. Однако ошибка, на которую вы действительно должны обратить внимание, исходит из зарезервированного вами набора тестов:

```
print('Test mean squared error: %.2f' % mean_squared_error(
      y_true=y_test, y_pred=regression.predict(X_test)))
```

После выполнения оценки на тестовом наборе вы получите значение ошибки:

```
Test mean squared error: 30.70
```

Когда вы оцениваете ошибку на тестовом наборе, результаты показывают, что сообщаемое значение составляет 30,70. Действительно, какая разница! Оценка на обучающем наборе почему-то оказалась слишком оптимистичной. Хотя использование набора тестов реалистичней в оценке ошибок, оно делает ваш результат зависимым от небольшой части данных. Если вы измените эту небольшую часть, результат теста также изменится:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
                    X, y, test_size=0.30, random_state=6)
regression.fit(X_train,y_train)
print('Train mean squared error: %.2f' % mean_squared_error(
      y_true=y_train, y_pred=regression.predict(X_train)))
print('Test mean squared error: %.2f' % mean_squared_error(
      y_true=y_test, y_pred=regression.predict(X_test)))
```

Вы получаете новый отчет об ошибках в учебных и тестовых наборах:

```
Train mean squared error: 19.48
Test mean squared error: 28.33
```

В этом разделе рассматривалась общая проблема алгоритмов машинного обучения. Вы знаете, что каждый алгоритм имеет определенное смещение или дисперсию при прогнозировании результата, но проблема в том, что вы не можете точно оценить его влияние. Более того, если вам нужно сделать выбор алгоритма, вы не можете быть уверены, какое решение будет наиболее эффективным.

Использование обучающих данных всегда непригодно при оценке производительности алгоритма, поскольку обученный алгоритм на самом деле лучше прогнозирует обучающие данные. Это особенно верно, когда алгоритм имеет низкое смещение из-за своей сложности. В этом случае вы можете ожидать меньшую ошибку при прогнозировании обучающих данных, что означает, что вы получаете слишком оптимистичный результат, который не сравним по справедливости с другими алгоритмами (которые могут иметь другой профиль смещения/дисперсии), а также результаты, бесполезные для оценки этого примера. Используя тестовые данные, вы фактически сокращаете обучающие примеры (что может привести к снижению производительности алгоритма), но взамен получаете более надежную и сопоставимую оценку ошибок.

Перекрестная проверка

Если тестовые наборы дают нестабильные результаты из-за выборки, решение заключается в систематическом отборе определенного количества тестовых наборов и последующем усреднении результатов. Это статистический подход (наблюдение многих результатов и их усреднение) и основа перекрестной проверки. Рецепт прост.

1. Разделите данные на *блоки* (fold) (каждый блок представляет собой контейнер, содержащий равномерное распределение случаев), обычно 10, но варианты блоков 3, 5 и 20 также приемлемы.
2. Используйте один блок в качестве тестового набора, а остальные — в качестве обучающих.
3. Обучите и запишите результат тестового набора. Если у вас мало данных, лучше использовать большее количество блоков, поскольку количество данных и использование дополнительных блоков положительно влияет на качество обучения.
4. Выполните этапы 2-3 еще раз, используя каждый блок по очереди в качестве тестового набора.

5. Рассчитайте среднее значение и стандартное отклонение всех результатов тестов блоков. Среднее значение является надежной оценкой качества вашего предиктора. Стандартное отклонение укажет надежность предиктора (если оно слишком высокое, ошибка перекрестной проверки может быть неточной). Ожидайте, что предикторы с высокой дисперсией будут иметь высокое стандартное отклонение перекрестной проверки.

Хотя эта техника может показаться сложной, Scikit-learn вполне справляется с ней, используя функции из модуля `sklearn.model_selection`.

Перекрестная проверка k-блоков

Чтобы запустить перекрестную проверку, сначала нужно инициализировать итератор. `KFold` — это итератор, реализующий перекрестную проверку k-блоков. В модуле `sklearn.model_selection` доступны и другие итераторы, полученные в основном из статистической практики, но `KFold` является наиболее широко используемым в науке о данных.

Итератор `KFold` требует, чтобы вы указали значение `n_splits` (количество генерируемых блоков) и хотите ли перетасовывать данные (используя параметр `shuffle`). Как правило, чем выше ожидаемая дисперсия, тем больше увеличение числа разделений улучшает среднюю оценку. Рекомендуется перетасовывать данные, поскольку упорядоченные данные способны запутать процессы обучения при некоторых алгоритмах, если первые наблюдения отличаются от последних.

После установки `KFold` вызовите функцию `cross_val_score`, которая возвращает массив результатов, содержащий оценку (из функции оценки) для каждого блока перекрестной проверки. Вы должны предоставить функции `cross_val_score` исходные данные (`X` и `y`), свой оценщик (класс регрессии) и ранее созданный экземпляр итератора `KFold` (в качестве параметра `cv`). В течение нескольких секунд или минут, в зависимости от количества блоков и обработанных данных, функция возвращает результаты. Усредните эти результаты, чтобы получить среднюю оценку, и вычислите стандартное отклонение, чтобы проверить, насколько стабильно среднее значение:

```
from sklearn.model_selection import cross_val_score, KFold
import numpy as np
crossvalidation = KFold(n_splits=10, shuffle=True, random_state=1)
scores = cross_val_score(regression, X, y,
      scoring='neg_mean_squared_error', cv=crossvalidation, n_jobs=1)
print('Folds: %i, mean squared error: %.2f std: %.2f' %
       (len(scores),np.mean(np.abs(scores)),np.std(scores)))
```

Вот результат:

```
Folds: 10, mean squared error: 23.76 std: 12.13
```

СОВЕТ

Перекрестная проверка может работать параллельно, поскольку никакая оценка не зависит ни от какой другой оценки. Вы можете использовать преимущества нескольких ядер на вашем компьютере, установив параметр `n_jobs=-1`.

Выборка стратификации для сложных данных

Блоки перекрестной проверки определяются случайной выборкой. Иногда может потребоваться отследить, присутствует ли определенная характеристика в учебных и тестовых блоках и сколько их, чтобы избежать неправильных выборок. Например, у набора данных Boston есть двоичная переменная (признак, имеющий значение 1 или 0), указывающая, граничит ли дом с рекой Чарльз. Эта информация важна для понимания ценности дома и определения того, хотели бы люди потратить на него больше денег. Чтобы увидеть действие этой переменной, используйте следующий код:

```
%matplotlib inline
import pandas as pd
df = pd.DataFrame(X, columns=boston.feature_names)
df['target'] = y
df.boxplot('target', by='CHAS', return_type='axes');
```

Диаграмма размаха, представленная на рис. 18.1, показывает, что дома у реки, как правило, ценятся выше, чем другие. Конечно, по всему Бостону есть дорогие дома, но вы должны следить за тем, сколько домов у реки вы анализируете, поскольку ваша модель должна быть общей для всего Бостона, а не только для домов у реки.

В подобных ситуациях, когда характеристика является редкой или влиятельной, нельзя быть уверенным, присутствует ли она в образце, поскольку блоки создаются случайным образом. Наличие слишком большого или слишком малого количества определенных характеристик в каждом блоке означает, что алгоритм машинного обучения может выводить неправильные правила.

Класс `StratifiedKFold` предоставляет простой способ контроля риска создания искаженных выборок во время процедур перекрестной проверки. Он может контролировать выборку, так что определенные признаки или даже определенные результаты (когда целевые классы крайне разбалансированы) всегда будут присутствовать в ваших блоках в правильной пропорции. Вам просто нужно указать переменную, которой вы хотите управлять, используя параметр `y`, как показано в следующем коде:

```
In [8]: %matplotlib inline
        import pandas as pd
        df = pd.DataFrame(X, columns=boston.feature_names)
        df['target'] = y
        df.boxplot('target', by='CHAS', return_type='axes');
```

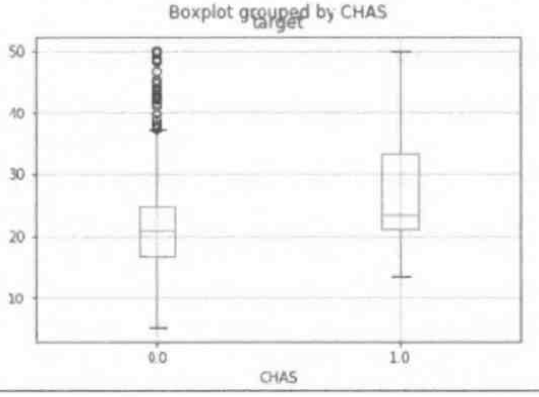


Рис. 18.1. Диаграмма размаха целевого результата, с сгруппированного по CHAS

```
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.metrics import mean_squared_error
strata = StratifiedShuffleSplit(n_splits=3,
                                test_size=0.35,
                                random_state=0)
scores = list()
for train_index, test_index in strata.split(X, X[:,3]):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    regression.fit(X_train, y_train)
    scores.append(mean_squared_error(y_true=y_test,
                                     y_pred=regression.predict(X_test)))
print('%i folds cv mean squared error: %.2f std: %.2f' %
     (len(scores),np.mean(np.abs(scores)),np.std(scores)))
```

Результатом трехкратной стратифицированной перекрестной проверки является

```
3 folds cv mean squared error: 24.30 std: 3.99
```

Хотя ошибка проверки аналогична, при контроле переменной CHAR стандартная ошибка оценок уменьшается, позволяя вам понять, что переменная влияла на предыдущие результаты перекрестной проверки.

Профессиональный выбор переменных

Выбор правильных переменных может улучшить процесс обучения за счет уменьшения количества шума (бесполезной информации), который способен повлиять на оценки обучаемого. Следовательно, выбор переменной способен эффективно уменьшить дисперсию прогнозов. Для того чтобы задействовать в обучении только полезные переменные и исключить лишние, используйте такие методы.

- **Одномерный подход**. Выбирайте переменные, наиболее связанные с целевым результатом.
- **Жадный или обратный подход**. Сохраняйте только те переменные, которые можете удалить из процесса обучения без ущерба для его производительности.

Выбор по одномерным критериям

Если вы решите выбрать переменную по уровню ее связи с ее целью, то класс `SelectPercentile` предоставит автоматическую процедуру для сохранения только определенного процента лучших связанных функций. Для ассоциации доступные такие метрики.

- **`f_regression`**. Используется только для числовых целевых переменных и основывается на производительности линейной регрессии.
- **`f_classif`**. Используется только для категориальных целевых переменных и основывается на статистическом тесте *дисперсионного анализа* (Analysis of Variance — ANOVA).
- **`chi2`**. Осуществляет *статистику хи-квадрат* (chi-square statistic) для категориальных целевых переменных. Она менее чувствительна к нелинейным отношениям между прогнозирующей переменной и ее целью.

СОВЕТ

При оценке кандидатов для задачи классификации метрики `f_classif` и `chi2` стремятся предоставить один и тот же набор первых переменных. По-прежнему рекомендуется тестировать выборки из обеих метрик ассоциации.

Помимо прямого выбора ассоциаций с верхним процентилем, `SelectPercentile` может также ранжировать лучшие переменные, чтобы упростить выбор того, на каком процентиле исключить функцию из процесса обучения. Класс `SelectKBest` аналогичен по своей функциональности, но он выбирает верхние переменные `k`, где `k` — число, а не процентиль.

```
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import f_regression
Selector_f = SelectPercentile(f_regression, percentile=25)
Selector_f.fit(X, y)
for n,s in zip(boston.feature_names,Selector_f.scores_):
    print('F-score: %3.2f\t for feature %s ' % (s,n))
```

После нескольких итераций код выводит следующие результаты:

```
F-score: 88.15 for     feature CRIM
F-score: 75.26 for     feature ZN
F-score: 153.95 for    feature INDUS
F-score: 15.97 for     feature CHAS
F-score: 112.59 for    feature NOX
F-score: 471.85 for    feature RM
F-score: 83.48 for     feature AGE
F-score: 33.58 for     feature DIS
F-score: 85.91 for     feature RAD
F-score: 141.76 for    feature TAX
F-score: 175.11 for    feature PTRATIO
F-score: 63.05 for     feature B
F-score: 601.62 for    feature LSTAT
```

Использование выходного уровня ассоциации (более высокие значения сигнализируют о большей ассоциации объекта с целевой переменной) помогает выбрать наиболее важные переменные для модели машинного обучения, но вы должны остерегаться следующих возможных проблем.

- Некоторые переменные с высокой связью также могут быть сильно коррелированными, отображая дублирующуюся информацию, которая действует как шум в процессе обучения.
- Некоторые переменные могут быть оштрафованы, особенно двоичные (переменные, обозначающие состояние или характеристику, используя значение 1, когда оно присутствует, и 0, когда его нет). Обратите, например, внимание на то, что выходные данные показывают двоичную переменную `CHAS` как наименее связанную с целевой переменной (но из предыдущих примеров вы знаете, что она влияет на фазу перекрестной проверки).


СОВЕТ

Одномерный процесс выбора может дать реальное преимущество, если у вас есть огромное количество переменных для выбора, а все другие методы становятся невозможными в вычислительном отношении. Лучшая процедура заключается в том, чтобы уменьшить значение `SelectPercentile` наполовину или более из доступных переменных, уменьшить количество переменных до управляемого

количества, а следовательно, разрешить использование более сложного и более точного метода, такого как жадный выбор.

Использование жадного поиска

При использовании одномерного выбора вы должны решить, сколько переменных оставить. *Жадный выбор* (greedy selection) автоматически уменьшает количество признаков, участвующих в модели обучения, на основе их вклада в производительность, измеряемую мерой ошибки. Подбирая данные, класс RFECV может предоставить вам информацию о количестве полезных признаков, указать их и автоматически преобразовать данные X за счет преобразования метода в набор сокращенных переменных, как показано в следующем примере:

```
from sklearn.feature_selection import RFECV
selector = RFECV(estimator=regression,
                 cv=10,
                 scoring='neg_mean_squared_error')
selector.fit(X, y)
print("Optimal number of features : %d"
      % selector.n_features_)
```

В примере выводится оптимальное количество признаков для задачи:

```
Optimal number of features: 6
```

Можно получить индекс для оптимального набора переменных, вызвав атрибут support_ из класса RFECV после его подбора:

```
print(boston.feature_names[selector.support_])
```

Команда выводит список, содержащий признаки:

```
['CHAS' 'NOX' 'RM' 'DIS' 'PTRATIO' 'LSTAT']
```

Обратите внимание, что CHAS теперь включен в число наиболее прогнозируемых признаков, что контрастирует с результатом однофакторного поиска в предыдущем разделе. Метод RFECV может определять, важна ли переменная, независимо от того, является ли она двоичной, категориальной или числовой, поскольку она непосредственно оценивает роль, которую играет признак в прогнозе.

ЗАПОМНИ

Метод RFECV, безусловно, более эффективен по сравнению с однофакторным подходом, поскольку он учитывает сильно коррелированные признаки и настроен для оптимизации оценки (которая обычно не является числовой или F-оценкой). Будучи жадным процессом, он требует вычислительных усилий и может лишь приблизиться к лучшему набору предикторов.

СОВЕТ

Поскольку RFECV получает лучший набор переменных из данных, выбор может быть переподобор, что происходит со всеми другими алгоритмами машинного обучения. Опробование RFECV на различных выборках обучающих данных может подтвердить выбор лучших переменных для использования.

Гиперпараметры

В качестве последнего примера рассмотрим процедуры поиска оптимальных гиперпараметров алгоритма машинного обучения для достижения наилучшей производительности прогнозирования. На самом деле *бо*льшая часть производительности вашего алгоритма уже была определена

- **Выбором алгоритма**. Не каждый алгоритм машинного обучения подходит для любого типа данных, и выбор правильного алгоритма для ваших данных может иметь решающее значение.
- **Подбор правильных переменных**. Производительность прогнозирования значительно увеличивается за счет создания признаков (вновь созданные переменные более предсказуемы, чем старые) и выбора признаков (устранение избыточности и шума). Точная настройка правильных гиперпараметров может обеспечить еще большую предсказуемость обобщения и улучшить результаты, особенно в случае сложных алгоритмов, которые не работают должным образом с использованием стандартных настроек.

ЗАПОМНИ!

Гиперпараметры (hyperparameter) — это параметры, которые вы должны определить самостоятельно, поскольку алгоритм не может автоматически извлечь их из данных. Как и в случае со всеми другими аспектами процесса обучения, которые связаны с решением аналитика данных, вы должны тщательно делать свой выбор после оценки перекрестно проверенных результатов.

Модуль `sklearn.grid_search` Scikit-learn специализируется на оптимизации гиперпараметров. Он содержит несколько утилит для автоматизации и упрощения процесса поиска их лучших значений. Код, приведенный в следующих разделах, иллюстрирует правильные процедуры, начиная с загрузки набора данных Iris в память:

```
import numpy as np
from sklearn.datasets import load_iris
iris = load_iris()
X, y = iris.data, iris.target
```

Пример готовится к выполнению своей задачи, загружая набор данных Iris и библиотеку NumPy. На данный момент пример способен оптимизировать алгоритм машинного обучения для прогнозирования видов ирисов.

Реализация сеточного поиска

Наилучший способ проверить лучшие гиперпараметры для алгоритма — это проверить их все, а затем выбрать лучшую комбинацию. Это значит, что в случае сложных настроек нескольких параметров вам придется запускать сотни, если не тысячи, немного иначе настроенных моделей. *Сеточный поиск* (grid search) — это метод систематического поиска, который объединяет все возможные комбинации гиперпараметров в отдельные наборы. Это трудоемкая техника. Тем не менее сеточный поиск обеспечивает один из лучших способов оптимизации приложения машинного обучения, которое может иметь много рабочих комбинаций, но только одну лучшую. Гиперпараметры, которые имеют много приемлемых решений (так называемые *локальные минимумы* (local minima)), могут создать впечатление, что вы нашли лучшее решение, тогда как вы действительно могли бы улучшить их производительность.


СОВЕТ

Сеточный поиск — это как забрасывание сети в море. Сначала лучше использовать большую сеть с широкой ячейкой. Большая сеть поможет понять, где в море есть стая рыбы. Узнав, где находится рыба, вы можете использовать меньшую сеть с более узкой ячейкой, чтобы поймать рыбу, которая находится в нужных местах. Точно так же, когда вы выполняете сеточный поиск, вы начинаете с поиска по нескольким разреженным значениям для проверки (широкие ячейки). После того как вы поймете, какие значения гиперпараметров нужно исследовать (косяки рыб), вы можете выполнить более тщательный поиск. Таким образом, вы также минимизируете риск переобучения в ходе перекрестной проверки слишком большого количества переменных, поскольку, как общий принцип в машинном обучении и научных экспериментах, чем больше попыток, тем больше шансов, что появится какой-то фальшиво хороший результат.


СОВЕТ

Сеточный поиск легко выполнить как параллельную задачу, поскольку результаты протестированной комбинации гиперпараметров не зависят от результатов других. Использование многоядерного компьютера на полную мощность требует, чтобы вы изменили значение атрибута `n_jobs` на `-1` при создании любого из классов сеточного поиска Scikit-learn.

У вас есть и другие варианты, кроме сеточного поиска. В качестве альтернативы сеточного поиска Scikit-learn реализует алгоритм случайного поиска. Существуют и другие методы оптимизации, основанные на байесовской оптимизации или нелинейных методах оптимизации, такие как метод Нелдера–Мида, которые не реализованы в пакетах данных, которые вы сейчас используете в Python.

В примере для демонстрации того, как эффективно реализовать сеточный поиск, используется один из ранее рассмотренных простых алгоритмов — классификатор k-ближайших соседей:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5,
       weights='uniform', metric= 'minkowski', p=2)
```

Классификатор k-ближайших соседей имеет довольно много гиперпараметров, которые вы можете установить для оптимальной производительности:

- количество соседних точек, учитываемых при оценке;
- способ взвешивания каждой из них;
- показатель, используемый для поиска соседей.

Используя диапазон возможных значений для всех параметров, вы можете легко понять, что собираетесь тестировать большое количество моделей — а именно 40 в данном случае:

```
grid = {'n_neighbors': range(1,11),
        'weights': ['uniform', 'distance'], 'p': [1,2]}
print ('Number of tested models: %i'
        % np.prod([len(grid[element]) for element in grid]))
score_metric = 'accuracy'
```

Код умножает количество всех тестируемых вами параметров и выводит результат:

```
Number of tested models: 40
```

Чтобы установить инструкции для поиска, вы должны создать словарь Python, ключами которого являются имена параметров, а значениями — списки значений, которые вы хотите проверить. Например, в примере записывается диапазон от 1 до 10 для гиперпараметра `n_neighbors` с использованием итератора `range(1,11)`, который создает последовательность чисел во время сеточного поиска.

Прежде чем начать, вы с помощью *"ванильной" модели* ("vanilla" model) со следующими стандартными параметрами выясните, каков показатель перекрестной проверки модели:

```
from sklearn.model_selection import cross_val_score
print('Baseline with default parameters: %.3f'
      % np.mean(cross_val_score(classifier, X, y,
                cv=10, scoring=score_metric, n_jobs=1)))
```

Вы принимаете к сведению результат, чтобы определить увеличение, обеспечиваемое оптимизацией параметров:

```
Baseline with default parameters: 0.967
```

Используя метрику точности (процент точных ответов), в этом примере сначала проверяется базовая линия, которая состоит из параметров стандартного алгоритма (также объясняется при создании экземпляра переменной `classifier` из ее класса). Трудно улучшить и без того высокую точность 0,967 (или 96,7%), но при поиске ответ будет найден с использованием десятикратной перекрестной проверки:

```
from sklearn.model_selection import GridSearchCV
search = GridSearchCV(estimator=classifier,
                      param_grid=grid,
                      scoring=score_metric,
                      n_jobs=1,
                      refit=True,
                      return_train_score=True,
                      cv=10)
search.fit(X,y)
```

После создания экземпляра с помощью алгоритма обучения, поискового словаря, метрики оценки и блоков перекрестной проверки класс `GridSearch` работает с методом `fit`. Дополнительно, после завершения сеточного поиска, она уточняет модель с наилучшей найденной комбинацией параметров (`refit=True`), позволяя ей немедленно начать прогнозирование с использованием самого класса `GridSearch`. Наконец, вы выводите лучшие параметры и результат лучшей комбинации:

```
print('Best parameters: %s' % search.best_params_)
print('CV Accuracy of best parameters: %.3f' %
      search.best_score_)
```

Выведенные значения таковы:

```
Best parameters: {'n_neighbors': 9, 'weights':
'uniform',
'p': 1}
CV Accuracy of best parameters: 0.973
```

Когда поиск будет завершен, проверьте результаты, используя атрибуты `best_params_` и `best_score:_`. Наилучшая найденная точность составила 0,973 — улучшение по сравнению с исходным уровнем. Вы также можете

проверить полную последовательность полученных баллов перекрестной проверки и их стандартное отклонение:

```
print(search.cv_results_)
```

Просматривая большое количество проверенных комбинаций, вы заметите, что более чем несколько из них получили оценку 0,973, когда комбинации имели девять или десять соседей. Чтобы лучше понять, как работает оптимизация в отношении количества соседей, используемых вашим алгоритмом, запустите класс Scikit-learn для визуализации. Метод `validation_curve` предоставляет вам подробную информацию о том, как ведут себя `train` и `validation` при использовании с разными гиперпараметрами `n_neighbors`.

```
from sklearn.model_selection import validation_curve
model = KNeighborsClassifier(weights='uniform',
                             metric= 'minkowski', p=1)
train, test = validation_curve(model, X, y,
                               param_name='n_neighbors',
                               param_range=range(1, 11),
                               cv=10, scoring='accuracy',
                               n_jobs=1)
```

Класс `validation_curve` предоставляет два массива, содержащих результаты, упорядоченные по значениям параметров в строках и блокам перекрестной проверки в столбцах:

```
import matplotlib.pyplot as plt
mean_train = np.mean(train,axis=1)
mean_test  = np.mean(test,axis=1)
plt.plot(range(1,11), mean_train,'ro--', label='Training')
plt.plot(range(1,11), mean_test,'bD-.', label='CV')
plt.grid()
plt.xlabel('Number of neighbors')
plt.ylabel('accuracy')
plt.legend(loc='upper right', numpoints= 1)
plt.show()
```

Проецирование строки означает создание графической визуализации, как показано на рис. 18.2, что помогает понять происходящее с процессом обучения.

Из визуализации вы можете получить две части информации.

- Пиковая точность перекрестной проверки с использованием девяти соседей выше, чем результат обучения. Результат обучения всегда должен быть лучше, чем любой показатель перекрестной проверки. Более высокая оценка указывает на то, что пример чрезмерно подобрал перекрестную проверку и удача сыграла свою роль в получении такого хорошего показателя перекрестной проверки.

» Второй пик точности перекрестной проверки у пяти соседей близок к самым низким результатам. Хорошо оцененные области обычно окружают оптимальные значения, поэтому данный пик немного подозрителен.

```
In [22]: import matplotlib.pyplot as plt
         mean_train  = np.mean(train,axis=1)
         mean_test   = np.mean(test,axis=1)
         plt.plot(range(1,11), mean_train,'ro--', label='Training')
         plt.plot(range(1,11), mean_test,'bD-.', label='CV')
         plt.grid()
         plt.xlabel('Number of neighbors')
         plt.ylabel('accuracy')
         plt.legend(loc='upper right', numpoints= 1)
         plt.show()
```

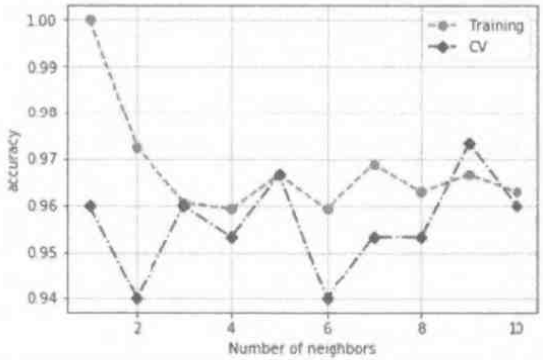


Рис. 18.2. Кривые проверки

Основываясь на визуализации, вы должны принять решение для девяти соседей (оно самое высокое и действительно окружено другими приемлемыми решениями). В качестве альтернативы, учитывая, что девять соседей являются решением по пределу поиска, можно вместо этого запустить новый сеточный поиск, расширив предел до большего количества соседей (более десяти), чтобы проверить, стабильна ли точность, уменьшается или даже увеличивается.

СОВЕТ

Запрос, проверка и снова запрос — это часть процесса науки о данных. Несмотря на то что язык Python и его пакеты предлагают множество автоматизированных процессов для изучения и обнаружения данных, вы должны задавать правильные вопросы и проверять, являются ли ответы наилучшими, используя статистические тесты и визуализации.

Попытка случайного поиска

Сеточный поиск, хоть и является исчерпывающим, действительно занимает много времени. Склонность к перекрестной проверке складывается из-за того, что в вашем наборе данных мало наблюдений и вы интенсивно ищете оптимизацию. Вместо этого интересным альтернативным вариантом является попытка *случайного поиска* (randomized search). В этом случае вы применяете сеточный поиск для проверки только некоторых комбинаций, выбранных случайным образом.

Несмотря на то что это может звучать как ставка на слепое везение, сеточный поиск на самом деле весьма полезен и неэффективен, а если вы выбираете достаточно случайных комбинаций, то есть высокая статистическая вероятность найти оптимальную комбинацию гиперпараметров, не рискуя чрезмерно близким подбором вообще. Например, в предыдущем примере код тестировал 40 различных моделей с использованием систематического поиска. Используя случайный поиск, вы можете сократить количество тестов на 75%, до 10 тестов, и достичь того же уровня оптимизации!

Использовать случайный поиск довольно просто. Импортируйте класс из модуля `grid_search` и введите те же параметры, что и в `GridSearchCV`, добавляя параметр `n_iter`, который указывает, сколько комбинаций нужно выбрать. Как правило, нужно выбрать четверть или треть от общего количества комбинаций гиперпараметров:

```
from sklearn.model_selection import RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=classifier,
                                   param_distributions=grid, n_iter=10,
    scoring=score_metric, n_jobs=1, refit=True, cv=10, )
random_search.fit(X, y)
```

Завершив поиск и используя ту же технику, что и ранее, вы можете исследовать результаты, выводя лучшие оценки и параметры:

```
print('Best parameters: %s' % random_search.best_params_)
print('CV Accuracy of best parameters: %.3f' %
       random_search.best_score_)
```

Результирующий поиск заканчивается следующими лучшими параметрами и оценкой:

```
Best parameters: {'n_neighbors': 9, 'weights': 'distance',
'p': 2}
Accuracy of best parameters: 0.973
```

Из результатов видно, что случайный поиск может дать результаты, аналогичные гораздо более дорогостоящему сеточному поиску.

Глава 19

Увеличение сложности с помощью линейных и нелинейных трюков

В ЭТОЙ ГЛАВЕ...

- Расширение функции с помощью полиномов
- Регуляризация модели
- Обучение на больших данных
- Использование метода опорных векторов и нейронной сети

В предыдущих главах вы ознакомились с некоторыми простейшими, но эффективными алгоритмами машинного обучения, такими как линейная и логистическая регрессия, наивный байесовский классификатор и K-ближайшие соседи (K-Nearest Neighbors — KNN). На этом этапе вы можете успешно завершить регрессионный или классификационный проект в области науки о данных. В данной главе рассматриваются еще более сложные и мощные методы машинного обучения, включая следующие: рассуждения о том, как улучшить данные; улучшение ваших оценок за счет регуляризации; обучение на больших данных при разделении их на управляемые фрагменты.

В настоящей главе речь пойдет также о *методе опорных векторов* (Support Vector Machine — SVM) — мощном семействе алгоритмов классификации и регрессии. В главе затрагиваются также нейронные сети. И SVM, и нейронные сети могут решать самые сложные задачи с данными в науке о данных. Впервые представленные несколько лет назад в качестве замены нейронных сетей,

ансамбли древовидных моделей вновь обогнали SVM (подробно об этом — в главе 20) как современный инструмент прогнозирования. Нейронные сети имеют долгую историю, но только за последние пять лет они улучшились до такой степени, что стали незаменимым инструментом для прогнозов, основанных на изображениях и тексте. Учитывая сложность как регрессии, так и классификации с использованием передовых методов, SVM в этой книге посвящено довольно много страниц, и лишь несколько страниц нейронным сетям. Тем не менее, углубленное понимание обеих стратегий определенно стоит времени и усилий.


СОВЕТ

Вам не нужно вводить исходный код этой главы вручную — используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_19_Increasing_Complexity.ipynb`. Вы также можете построить некоторые сложные графики, иллюстрирующие алгоритмы SVM, запустив код в исходном файле `P4DS4D2_19_Representing_SVM_boundaries.ipynb`.

Использование нелинейных преобразований

Линейные модели, такие как линейная и логистическая регрессия, на самом деле являются линейными комбинациями, которые суммируют признаки (взвешенные по изученным коэффициентам) и предоставляют простую, но эффективную модель. В большинстве ситуаций они предлагают хорошее приближение к сложной реальности, которую представляют. Даже при том, что они характеризуются высоким смещением, использование большого количества наблюдений может улучшить их коэффициенты и сделать более конкурентоспособными по сравнению со сложными алгоритмами.

Тем не менее они могут работать лучше при решении определенных задач, если вы предварительно проанализируете данные с использованием подхода *разведочного анализа данных* (Exploratory Data Analysis — EDA). После выполнения анализа вы можете преобразовать и улучшить имеющиеся признаки следующим образом.

- Линеаризация отношений между признаками и целевой переменной с использованием преобразований, которые увеличивают их корреляцию и делают облако их точек на диаграмме рассеяния более похожим на линию.
- Умножая переменные и заставляя их взаимодействовать, вы сможете лучше представить их совместное поведение.

» Расширение существующих переменных с использованием полиномиального расширения для более реалистичного представления отношений (таких, как кривые идеальной точки, когда в переменной есть пик, представляющий максимум, похожий на параболу).

Преобразования переменных

Пример — лучший способ объяснить, какие преобразования можно успешно применить к данным для улучшения линейной модели. Примеры в этом разделе, а также в разделах "Регуляризация линейных моделей" и "Как справиться с большими данными фрагмент за фрагментом" основаны на наборе данных Boston. Проблема заключается в регрессии, и данные изначально имеют десять переменных, объясняющих различные цены на жилье в Бостоне в 1970-х годах. Набор данных также имеет неявное упорядочение. К счастью, порядок не влияет на большинство алгоритмов, поскольку они изучают данные в целом. Когда алгоритм учится прогрессивно, упорядочение может мешать эффективному построению модели. Используя `seed` (чтобы исправить предопределенную последовательность случайных чисел) и `shuffle` из пакета `random` (чтобы перемешать индексы), вы можете переиндексировать набор данных:

```
From sklearn.datasets import load_boston
import random
from random import shuffle

boston = load_boston()
random.seed(0) # Создает воспроизводимые перетасовки
new_index = list(range(boston.data.shape[0]))
shuffle(new_index) # перетасовка индекса
X, y = boston.data[new_index], boston.target[new_index]
print(X.shape, y.shape, boston.feature_names)
```

В коде `random.seed(0)` создает воспроизводимую операцию перетасовки, а `shuffle(new_index)` — новый перетасованный индекс, используемый для переупорядочения данных. После этого код выводит формы `X` и `y`, а также список имен переменных набора данных:

```
(506, 13) (506,) ['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD'
 'TAX' 'PTRATIO' 'B' 'LSTAT']
```

СОВЕТ

Чтобы узнать больше о значении переменных, представленных в наборе данных Boston, выполните команду `print(boston.DESCR)`. Вы увидите вывод этой команды в загружаемом исходном коде.

Преобразование массива предикторов и целевой переменной в `pandas DataFrame` помогает поддерживать серию исследований и операций с

данными. Более того, хотя Scikit-learn требует в качестве ввода ndarray, он также будет принимать объекты DataFrame:

```
import pandas as pd
df = pd.DataFrame(X,columns=boston.feature_names)
df['target'] = y
```

Лучший способ определить возможные преобразования — это графическое исследование, а использование диаграммы рассеяния может многое рассказать о двух переменных. Вам необходимо сделать отношения между предикторами и целевым результатом как можно более линейными, поэтому попробуйте различные комбинации, например, следующие:

```
ax = df.plot(kind='scatter', x='LSTAT', y='target', c='b')
```

На рис. 19.1 представлена результирующая диаграмма рассеяния. Обратите внимание, что вы можете аппроксимировать облако точек, используя кривую, а не прямую линию. В частности, когда LSTAT составляет около 5, цель, по-видимому, располагается между значениями от 20 до 50. По мере увеличения LSTAT цель снижается до 10, снижая вариацию.

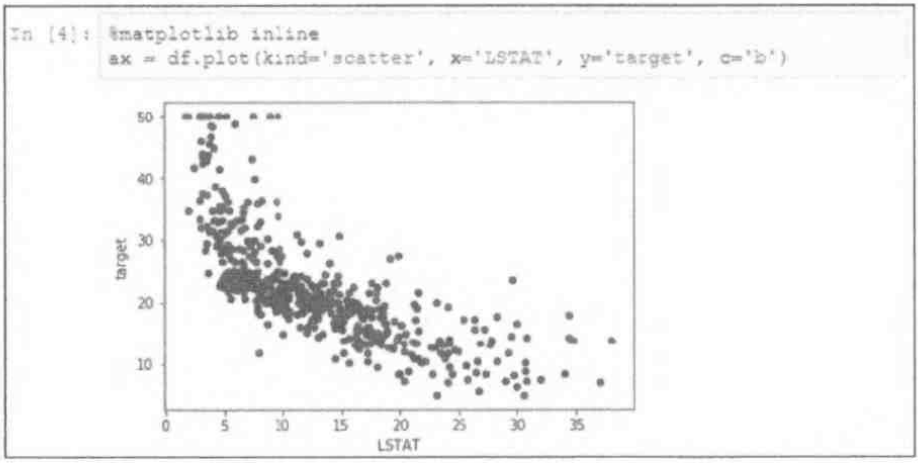


Рис. 19.1. Нелинейная связь между переменной LSTAT и целевыми ценами

В таких условиях может помочь логарифмическое преобразование. Но ваши значения должны находиться в диапазоне от нуля до единицы, например, в процентах, как показано в этом примере. В других случаях для переменной x могут быть полезны другие преобразования, включая x**2, x**3, 1/x, 1/x**2, 1/x**3 и sqrt(x). Главное опробовать их и проверить результат. Что касается тестирования, то можете использовать следующий сценарий в качестве примера:

```
import numpy as np
from sklearn.feature_selection import f_regression
single_variable = df['LSTAT'].values.reshape(-1, 1)
F, pval = f_regression(single_variable, y)
print('F score for the original feature %.1f' % F)
F, pval = f_regression(np.log(single_variable),y)
print('F score for the transformed feature %.1f' % F)
```

Код выводит F-оценку (`F score`), меру, позволяющую оценить прогнозируемость признака в задаче машинного обучения, как оригинального, так и преобразованного. Оценка за преобразованный признак является большим улучшением по сравнению с таковой но не преобразованный:

```
F score for the original feature 601.6
F score for the transformed feature 1000.2
```

F-оценка полезна для выбора переменных. Вы также можете использовать ее для оценки полезности преобразования, потому что и `f_regression`, и `f_classif` сами основаны на линейных моделях, а потому чувствительны к каждому эффективному преобразованию, используемому для того, чтобы сделать отношения переменных более линейными.

Создание взаимодействий между переменными

В линейной комбинации модель реагирует на изменения переменной независимо от изменений других переменных. В статистике такого рода модель является *моделью основных эффектов* (main effects model).

ЗАПОМНИ!

Наивный байесовский классификатор делает аналогичное предположение для вероятностей, а также хорошо работает со сложными текстовыми задачами.

Даже если машинное обучение работает с использованием аппроксимаций и набора независимых переменных, это может сделать ваши прогнозы хорошими в большинстве ситуаций, но иногда вы можете пропустить важную часть картины. Вы можете легко уловить эту проблему, представив изменение в вашей цели, связанное с объединенным изменением двух или более переменных двумя простыми и понятными способами.

- **Наличие знаний предметной области задачи**. Например, наличие шумного двигателя на автомобильном рынке является неприятностью для семейного автомобиля, но считается плюсом для спортивного (поклонники автомобилей хотят слышать, как звучит их сверхмощный и дорогой автомобиль). Зная предпочтения потребителя, вы можете смоделировать переменную уровня шума и переменную типа автомобиля, чтобы получить точные прогнозы,

используя аналитическую модель, прогнозирующую стоимость автомобиля на основе его характеристик.

- **Проверка комбинаций разных переменных**. Выполняя групповые тесты, вы можете увидеть влияние определенных переменных на целевую переменную. Поэтому, даже не зная о шумных двигателях и спортивных автомобилях, вы могли бы обнаружить другое среднее значение уровня предпочтений при анализе набора данных, разделенного по типу автомобилей и уровню шума.

В следующем примере показано, как проверять и обнаруживать взаимодействия в наборе данных Boston. Первая задача — загрузить несколько вспомогательных классов:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score, KFold
regression = LinearRegression(normalize=True)
crossvalidation = KFold(n_splits=10, shuffle=True,
                        random_state=1)
```

Код повторно инициализирует pandas DataFrame, используя только переменные предиктора. Цикл for сопоставляет различные предикторы и создает новую переменную, содержащую каждое взаимодействие. Математическая формулировка взаимодействия — это просто умножение.

```
df = pd.DataFrame(X,columns=boston.feature_names)
baseline = np.mean(cross_val_score(regression, df, y,
                   scoring='r2',
                   cv=crossvalidation))
interactions = list()
for var_A in boston.feature_names:
    for var_B in boston.feature_names:
        if var_A > var_B:
            df['interaction'] = df[var_A] * df[var_B]
            cv = cross_val_score(regression, df, y,
                                 scoring='r2',
                                 cv=crossvalidation)
            score = round(np.mean(cv), 3)
            if score > baseline:
               interactions.append((var_A, var_B, score))
print('Baseline R2: %.3f' % baseline)
print('Top 10 interactions: %s' % sorted(interactions,
                                         key=lambda x :x[2],
                                         reverse=True)[:10])
```

Код начинается с вывода базовой оценки R^2 для регрессии. Затем он сообщает о первых десяти взаимодействиях, добавление которых в моду увеличивает рейтинг:

```
Baseline R2: 0.716
Top 10 interactions: [('RM', 'LSTAT', 0.79), ('TAX', 'RM', 0.782), ('RM',
    'RAD', 0.778), ('RM', 'PTRATIO', 0.766), ('RM', 'INDUS', 0.76),
    ('RM', 'NOX', 0.747), ('RM', 'AGE', 0.742), ('RM', 'B', 0.738),
    ('RM', 'DIS', 0.736), ('ZN', 'RM', 0.73)]
```

Код проверяет добавление каждого конкретного взаимодействия к модели, используя перекрестную проверку по 10 блоков. (Более подробную информацию о работе с блоками см. в разделе "Перекрестная проверка" главы 18.) Код записывает изменение меры R^2 в стек (простой список), который приложение может упорядочить и исследовать позже.

Базовая оценка составляет 0,699, поэтому отмеченное улучшение стека взаимодействий до 0,782 выглядит весьма впечатляюще. Важно знать, как это улучшение стало возможным. Вовлечены две переменные — `RM` (среднее количество комнат) и `LSTAT` (процент населения с низким статусом). Расстановку этих двух переменных раскроет диаграмма:

```
colors = ['b' if v > np.mean(y) else 'r' for v in y]
scatter = df.plot(kind='scatter', x='RM', y='LSTAT',
                  c=colors)
```

Диаграмма рассеяния на рис. 19.2 поясняет улучшение. Для домов, расположенных в центре диаграммы, необходимо знать `LSTAT` и `RM`, чтобы правильно отделить дорогостоящие дома от дешевых; следовательно, взаимодействие в этом случае является обязательным.

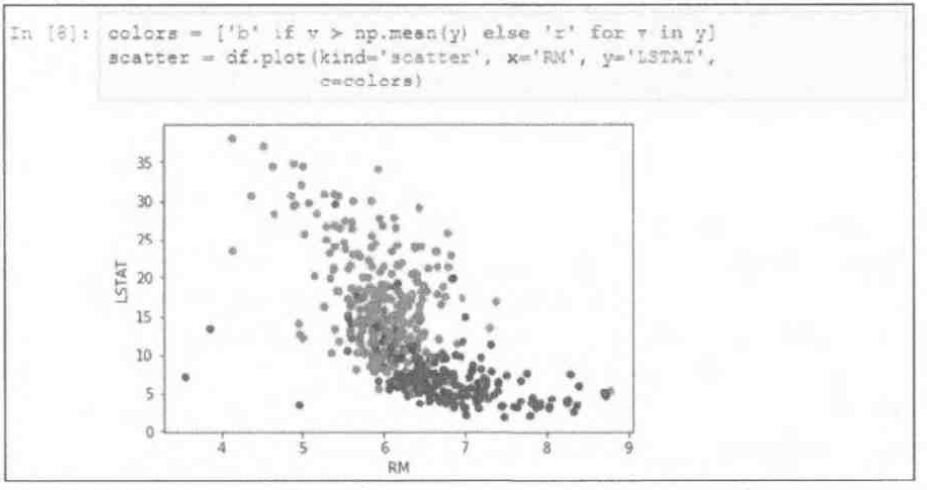


Рис. 19.2. Комбинированные переменные `LSTAT` *и* `RM` *помогают отделить высокие цены от низких*

Добавление взаимодействий и преобразованных переменных приводит к расширенной модели линейной регрессии — полиномиальной регрессии.

Аналитики данных полагаются на проверки и эксперименты для проверки подхода к решению проблемы, поэтому следующий код немного модифицирует предыдущий, чтобы переопределить набор предикторов, используя взаимодействия и квадратичные члены, полученные возведением переменных в квадрат:

```
polyX = pd.DataFrame(X,columns=boston.feature_names)
cv = cross_val_score(regression, polyX, y,
                     scoring='neg_mean_squared_error',
                     cv=crossvalidation)
baseline = np.mean(cv)
improvements = [baseline]
for var_A in boston.feature_names:
    polyX[var_A+'^2'] = polyX[var_A]**2
    cv = cross_val_score(regression, polyX, y,
                         scoring='neg_mean_squared_error',
                         cv=crossvalidation)
improvements.append(np.mean(cv))
for var_B in boston.feature_names:
    if var_A > var_B:
        poly_var = var_A + '*' + var_B
        polyX[poly_var] = polyX[var_A] * polyX[var_B]
        cv = cross_val_score(regression, polyX, y,
                             scoring='neg_mean_squared_error',
                             cv=crossvalidation)
        improvements.append(np.mean(cv))
```

```
import matplotlib.pyplot as plt
plt.figure()
plt.plot(range(0,92),np.abs(improvements),'-')
plt.xlabel('Added polynomial features')
plt.ylabel('Mean squared error')
plt.show()
```

Чтобы отслеживать улучшения по мере добавления в код новых комплексных членов, код помещает значения в список `improvements`. На рис. 19.3 показан график результатов, демонстрирующих, что некоторые дополнения хороши, поскольку квадратичная ошибка уменьшается, а другие дополнения плохи, поскольку они увеличивают ошибку.

Конечно, вместо того, чтобы безоговорочно добавлять все созданные переменные, вы можете выполнить текущий тест, прежде чем решите добавить квадратичный член или взаимодействие, проверяя перекрестной проверкой, действительно ли каждое добавление полезно для ваших прогностических целей. Этот пример является хорошей основой для проверки других способов управления текущей сложностью ваших наборов данных или сложностью преобразования и создания признаков в процессе исследования данных. Прежде

чем двигаться дальше, проверьте перекрестной проверкой как форму фактического набора данных, так и его среднеквадратичную ошибку:

```
In [10]: import matplotlib.pyplot as plt
         plt.figure()
         plt.plot(range(0,92),np.abs(improvements),'-')
         plt.xlabel('Added polynomial features')
         plt.ylabel('Mean squared error')
         plt.show()
```

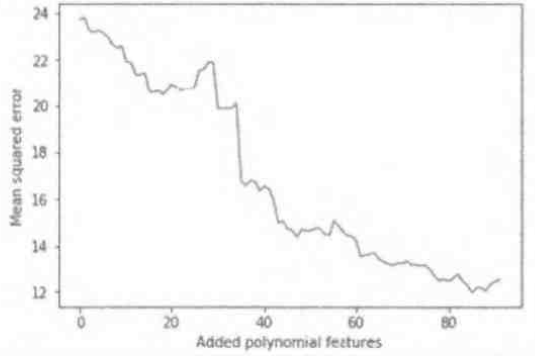


Рис. 19.3. Добавление полиномиальных функций увеличивает силу прогнозирования

```
print('New shape of X:', np.shape(polyX))
crossvalidation = KFold(n_splits=10, shuffle=True,
                        random_state=1)
cv = cross_val_score(regression, polyX, y,
                     scoring='neg_mean_squared_error',
                     cv=crossvalidation)
print('Mean squared error: %.3f' % abs(np.mean(cv)))
```

Несмотря на то что среднеквадратичная ошибка является хорошей, соотношение между 506 наблюдениями и 104 признаками не так уж и хорошо, поскольку количество наблюдений может быть недостаточным для правильной оценки коэффициентов.

```
New shape of X: (506, 104)
Mean squared error: 12.514
```

СОВЕТ

Эмпирическое правило: делите количество наблюдений на количество коэффициентов. В коде должно быть не менее 10–20 наблюдений для каждого коэффициента, который вы хотите оценить в линейных моделях. Однако опыт показывает, что лучше иметь как минимум 30 наблюдений.

Регуляризация линейных моделей

Линейные модели имеют высокое смещение, но по мере того, как вы добавляете больше признаков, больше взаимодействий и больше преобразований, они начинают приобретать способность адаптироваться к характеристикам данных и запоминать степень шума в данных, тем самым увеличивая дисперсию своих оценок. Компромисс с более высокой дисперсией для меньшего смещения не всегда лучший выбор, но, как упоминалось ранее, иногда это единственный способ увеличить прогнозирующую силу линейных алгоритмов.

Вы можете ввести регуляризацию L1 и L2 как способ контроля компромисса между смещением и дисперсией в пользу расширения возможности обобщения модели. Когда вы вводите одну из регуляризаций, аддитивная функция, которая зависит от сложности линейной модели, штрафует оптимизированную функцию стоимости. В линейной регрессии функция стоимости является квадратичной ошибкой прогнозов, а функция стоимости штрафуется с использованием суммирования коэффициентов переменных-предикторов.

Если модель сложная, но прогнозируемый выигрыш незначителен, штрафование вынуждает процедуру оптимизации удалять бесполезные переменные или уменьшать их влияние на оценку. Регуляризация также действует на сильно коррелированные признаки, ослабляя или исключая их вклад, стабилизируя тем самым результаты и уменьшая последующую дисперсию оценок.

- **Регрессия L1** (или Лассо (Lasso)). Уменьшает некоторые коэффициенты до нуля, прореживая коэффициенты. Осуществляет выбор переменной.
- **Регрессия L2** (или Ридж (Ridge)). Уменьшает коэффициенты наиболее проблемных признаков, делая их меньшими, но редко равными нулю. Все коэффициенты продолжают участвовать в оценке, но многие становятся слишком маленькими и неактуальными.


ЗАПОМНИ!

Для контроля степени регуляризации можно использовать гиперпараметр, либо сам коэффициент, зачастую называемый *альфа* (alpha). Когда альфа приближается к 1,0, вы получаете более сильную регуляризацию и большее уменьшение коэффициентов. В некоторых случаях коэффициенты сводятся к нулю. Не путайте альфа с `C`, параметром, используемым `LogisticRegression` и методами опорных векторов, поскольку `C` равно `1/alpha`, а значит, может быть больше 1. Меньшие значения `C` на самом деле соответствуют большей регуляризации, что прямо противоположно `alpha`.

СОВЕТ

Регуляризация работает, поскольку это сумма коэффициентов переменных-предикторов, а значит, важно, чтобы они имели один и тот же масштаб, иначе регуляризация может затруднить сходимость, и переменные с большими абсолютными значениями коэффициентов будут сильно влиять на нее, приводя к инвазивной регуляризации. Хорошей практикой является стандартизация значений предикторов или привязка их к общему минимуму-максимуму, такому как диапазон [–1, +1]. В следующих разделах демонстрируются различные методы использования регуляризации L1 и L2 для достижения различных эффектов.

Использование регрессии Ридж (L2)

В первом примере используется регуляризация типа L2, уменьшающая силу коэффициентов. Класс `Ridge` реализует регрессию L2 для линейной регрессии. Использовать его просто; он представляет только параметр альфа для исправления. `Ridge` имеет также другой параметр, `normalize`, который автоматически нормализует введенные предикторы до нулевого среднего значения и единичного блока.

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
ridge = Ridge(normalize=True)
search_grid = {'alpha':np.logspace(-5,2,8)}
search = GridSearchCV(estimator=ridge,
                      param_grid=search_grid,
                      scoring='neg_mean_squared_error',
                      refit=True, cv=10)
search.fit(polyX,y)
print('Best parameters: %s' % search.best_params_)
score = abs(search.best_score_)
print('CV MSE of best parameters: %.3f' % score)
```

Найдя наилучший параметр альфа, получаем лучшую модель.

```
Best parameters: {'alpha': 0.001}
CV MSE of best parameters: 11.630
```

СОВЕТ

Хорошее пространство поиска для значения альфа находится в диапазоне `np.logspace(-5,2,8)`. Конечно, если полученное оптимальное значение находится на одном из концов тестируемого диапазона, необходимо увеличить диапазон и провести повторное тестирование.

Переменные polyX и y, используемые для примеров в этом и последующих разделах, создаются как часть примера в разделе "Создание взаимодействий между переменными", ранее в этой главе. Если вы не проработали тот раздел, примеры в этом разделе не будут работать должным образом.

Использование регрессии Лассо (L1)

Во втором примере используется регуляризация L1, класс Lasso, основной характеристикой которого является уменьшение менее полезных эффекта до нуля. Это действие обеспечивает разреженность в коэффициентах, и лишь немногие из них имеют значения выше нуля. Класс использует те же параметры класса Ridge, которые были продемонстрированы в предыдущем разделе:

```
from sklearn.linear_model import Lasso
lasso = Lasso(normalize=True,tol=0.05, selection='random')
search_grid = {'alpha':np.logspace(-2,3,8)}
search = GridSearchCV(estimator=lasso,
                      param_grid=search_grid,
                      scoring='neg_mean_squared_error',
                      refit=True, cv=10)
search.fit(polyX,y)
print('Best parameters: %s' % search.best_params_)
score = abs(search.best_score_)
print('CV MSE of best parameters: %.3f' % score)
```

При настройке Lasso код использует менее чувствительный алгоритм (tol=0.05) и случайный подход для его оптимизации (selection='random'). Полученная среднеквадратичная ошибка выше, чем при использовании регуляризации L2:

```
Best parameters: {'alpha': 1e-05}
CV MSE of best parameters: 12.432
```

Использование регуляризации

Поскольку вы можете идентифицировать разреженные коэффициенты, возникающие в результате регрессии L1, в качестве процедуры выбора признака можно эффективно использовать класс Lasso, как и при выборе наиболее важных переменных. Настраивая параметр alpha, вы можете выбрать большее или меньшее количество переменных. В этом случае код устанавливает параметр alpha равным 0,01, в результате чего получается значительно упрощенное решение:

```
lasso = Lasso(normalize=True, alpha=0.01)
lasso.fit(polyX,y)
print(polyX.columns[np.abs(lasso.coef_)>0.0001].values)
```

Упрощенное решение состоит из нескольких взаимодействий:

```
['CRIM*CHAS' 'ZN*CRIM' 'ZN*CHAS' 'INDUS*DIS' 'CHAS*B'
 'NOX^2' 'NOX*DIS' 'RM^2' 'RM*CRIM' 'RM*NOX' 'RM*PTRATIO'
 'RM*B' 'RM*LSTAT' 'RAD*B' 'TAX*DIS' 'PTRATIO*NOX'
 'LSTAT^2']
```


СОВЕТ

Вы можете применить выбор переменных на основе L1 автоматически как к регрессии, так и к классификации, используя классы `RandomizedLasso` и `RandomizedLogisticRegression`. Оба класса создают серию рандомизированных L1 регуляризованных моделей. Код отслеживает полученные коэффициенты. В конце процесса приложение сохраняет все коэффициенты, которые класс не уменьшил до нуля, потому что они считаются важными. Вы можете обучать эти два класса, используя метод `fit`, но у них нет метода `predict`, только метод `transform`, который эффективно сокращает набор данных, как и большинство классов в модуле `sklearn.preprocessing`.

Объединение L1 и L2: ElasticNet

Регуляризация L2 уменьшает влияние коррелированных признаков, тогда как регуляризация L1 имеет тенденцию выбирать их. Хорошая стратегия заключается в том, чтобы объединить их, используя взвешенную сумму, применяя класс `ElasticNet`. Вы контролируете эффекты L1 и L2 с помощью одного и того же параметра `alpha`, но вы можете определить долю эффекта L1 с помощью параметра `l1_ratio`. Ясно, что если `l1_ratio` равно 0, то это будет регрессия Ридж; с другой стороны, если `l1_ratio` равно 1, то это регрессия Лассо.

```
from sklearn.linear_model import ElasticNet
elastic = ElasticNet(normalize=True, selection='random')
search_grid = {'alpha':np.logspace(-4,3,8),
               'l1_ratio': [0.10 ,0.25, 0.5, 0.75]}
search = GridSearchCV(estimator=elastic,
                      param_grid=search_grid,
                      scoring='neg_mean_squared_error',
                      refit=True, cv=10)
search.fit(polyX,y)
print('Best parameters: %s' % search.best_params_)
score = abs(search.best_score_)
print('CV MSE of best parameters: %.3f' % score)
```

Через некоторое время вы получите результат, который вполне сопоставим с L1:

```
Best parameters: {'alpha': 0.0001, 'l1_ratio': 0.75}
CV MSE of best parameters: 12.581
```

Как справиться с большими данными фрагмент за фрагментом

До сих пор мы рассматривали примеры небольших баз данных. Реальные данные, помимо беспорядка, могут быть также довольно большими — иногда такими большими, что они не помещаются в памяти, независимо от характеристик памяти машины.

Переменные `polyX` и `y`, используемые для примеров в следующих разделах, создаются как часть примера в разделе "Создание взаимодействий между переменными", ранее в этой главе. Если вы не проработали этот раздел, примеры в данном разделе не будут работать правильно.

Определение, когда данных слишком много

В проекте науки о данных данные могут считаться большими, когда возникает одна из следующих двух ситуаций:

- они не могут поместиться в доступной памяти компьютера;
- даже если в системе достаточно памяти для хранения данных, приложение не может обработать их с использованием алгоритмов машинного обучения за разумные сроки.

Реализация стохастического градиентного спуска

Если у вас слишком много данных, используйте в качестве линейного предиктора *стохастический регрессор градиентного спуска* (Stochastic Gradient Descent Regressor) (`SGDRegressor`), или *стохастический классификатор градиентного спуска* (Stochastic Gradient Descent Classifier) (`SGDClassifier`). Единственное отличие от других методов, описанных ранее в этой главе, заключается в том, что они оптимизируют свои коэффициенты, фактически используя только одно наблюдение за раз. Поэтому требуется больше итераций, прежде чем код достигнет результатов, сопоставимых с использованием регрессии Ридж или Лассо, но он требует гораздо меньше памяти и времени.

Это связано с тем, что оба предиктора полагаются на оптимизацию *стохастического градиентного спуска* (Stochastic Gradient Descent — SGD) — разновидность оптимизации, при которой корректировка параметров происходит после ввода каждого наблюдения, что приводит к более длинному и немного более ошибочному пути к минимизации функции ошибки. Конечно, оптимизация, основанная на отдельных наблюдениях, а не на огромных матрицах

данных, может оказать огромное положительное влияние на время обучения алгоритма и объем ресурсов памяти.

При использовании SGD, помимо различных функций стоимости, которые необходимо проверить на производительность, вы также можете попробовать использовать регуляризацию L1, L2 и ElasticNet, установив параметр `penalty` и соответствующие управляющие параметры `alpha` и `l1_ratio`. Некоторые из SGD более устойчивы к выбросам, например, `modified_huber` для классификации или `huber` для регрессии.


ЗАПОМНИ!

SGD чувствителен к масштабу переменных, и это не только из-за регуляризации, но и из-за внутренней работы. Следовательно, вы всегда должны стандартизировать свои признаки (например, с помощью `StandardScaler`) или принудительно устанавливать их в диапазоне [0, + 1] или [–1, +1]. Невыполнение этого требования приведет к плохим результатам.


СОВЕТ

При использовании SGD вам всегда придется иметь дело с фрагментами данных, если вы не можете загрузить все учебные данные в память. Чтобы сделать обучение эффективным, вы должны стандартизировать данные, сделав так, чтобы `StandardScaler` вывел среднее значение и стандартное отклонение из первых доступных данных. Среднее значение и стандартное отклонение всего набора данных, скорее всего, отличаются, но преобразования по первоначальной оценке будет достаточно для выработки рабочей процедуры обучения.

```
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler

SGD = SGDRegressor(loss='squared_loss',
                   penalty='l2',
                   alpha=0.0001,
                   l1_ratio=0.15,
                   max_iter=2000,
                   random_state=1)
scaling = StandardScaler()
scaling.fit(polyX)
scaled_X = scaling.transform(polyX)
cv = cross_val_score(SGD, scaled_X, y,
                     scoring='neg_mean_squared_error',
                     cv=crossvalidation)
score = abs(np.mean(cv))
print('CV MSE: %.3f' % score)
```

Результирующая среднеквадратическая ошибка после запуска `SGDRegressor` такова:

```
CV MSE: 12.179
```

В предыдущем примере мы использовали метод подбора, который требует предварительной загрузки в память всех учебных данных. Вы можете обучать модель последовательными шагами, используя вместо этого метод `part_fit`, который осуществляет одну итерацию для предоставленных данных, а затем сохраняет ее в памяти и корректирует при получении новых данных. На этот раз код использует большее количество итераций:

```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

X_tr, X_t, y_tr, y_t = train_test_split(scaled_X, y,
                                        test_size=0.20,
                                        random_state=2)
SGD = SGDRegressor(loss='squared_loss',
                   penalty='l2',
                   alpha=0.0001,
                   l1_ratio=0.15,
                   max_iter=2000,
                   random_state=1)
improvements = list()
for z in range(10000):
    SGD.partial_fit(X_tr, y_tr)
    score = mean_squared_error(y_t, SGD.predict(X_t))
    improvements.append(score)
```

Отслеживая частичные улучшения алгоритма в течение `10000` итераций по одним и тем же данным, вы можете создать график и понять, как работают улучшения, что и показано в следующем коде. Обратите внимание, что на каждом этапе вы можете использовать разные данные.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 4))
plt.subplot(1,2,1)
range_1 = range(1,101,10)
score_1 = np.abs(improvements[:100:10])
plt.plot(range_1, score_1,'o--')
plt.xlabel('Iterations up to 100')
plt.ylabel('Test mean squared error')
plt.subplot(1,2,2)
range_2 = range(100,10000,500)
score_2 = np.abs(improvements[100:10000:500])
plt.plot(range_2, score_2,'o--')
plt.xlabel('Iterations from 101 to 5000')
plt.show()
```

Как показано на первой из двух панелей рис. 19.4, алгоритм изначально запускается с высокой частотой ошибок, но ему удается уменьшить ее всего за несколько итераций, обычно 5–10. После этого частота ошибок постепенно уменьшается на меньшую величину за каждую итерацию. На второй панели вы видите, что после 1500 итераций частота ошибок достигает минимума и начинает увеличиваться. В этот момент начинается переобучение, поскольку данные уже понимают правила, и вы фактически заставляете SGD учиться дальше, когда в данных уже не осталось ничего, кроме шума. Следовательно, он начинает учиться шуму и ошибочным правилам.

```
In [18]: import matplotlib.pyplot as plt
         plt.figure(figsize=(8, 4))
         plt.subplot(1,2,1)
         range_1 = range(1,101,10)
         score_1 = np.abs(improvements[:100:10])
         plt.plot(range_1, score_1,'o--')
         plt.xlabel('Iterations up to 100')
         plt.ylabel('Test mean squared error')
         plt.subplot(1,2,2)
         range_2 = range(100,10000,500)
         score_2 = np.abs(improvements[100:10000:500])
         plt.plot(range_2, score_2,'o--')
         plt.xlabel('Iterations from 101 to 5000')
         plt.show()
```

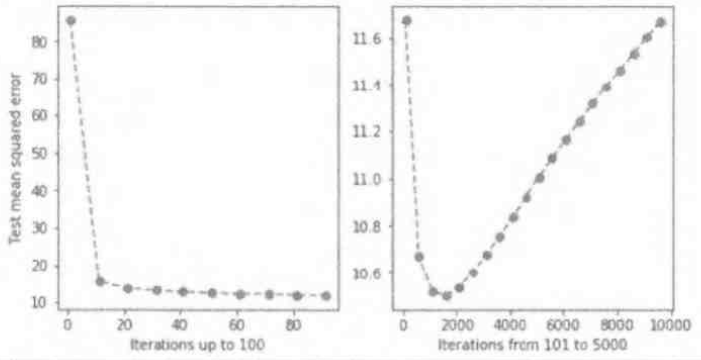


Рис. 19.4. Квадратичная ошибка оптимизации медленного спуска


СОВЕТ

Если вы работаете не со всеми данными в памяти, сеточный поиск и перекрестная проверка наилучшего количества итераций будут затруднены. Хитрость заключается в том, чтобы хранить в памяти или хранилище часть обучающих данных для использования при проверке отдельно. Проверив производительность на этой нетронутой части, вы можете увидеть, когда производительность обучения SGD начинает снижаться. В этот момент вы можете прервать итерацию данных (метод, известный как *ранняя остановка* (early stopping)).

Понятие метода опорных векторов

Специалисты по данным считают, что *метод опорных векторов* (Support Vector Machine — SVM) является одним из наиболее сложных и мощных методов машинного обучения в наборе их инструментов, поэтому обычно вы найдете эту тему только в руководствах не для начинающих. Тем не менее вы не должны отворачиваться от этого великолепного алгоритма обучения, только потому что библиотека Scikit-learn предлагает вам широкий и доступный диапазон контролируемых SVM классов для регрессии и классификации. Вы даже можете получить доступ к SVM без учителя, который описан в главе 16 (о выбросах). Оценивая, хотите ли вы попробовать алгоритмы SVM в качестве решения для машинного обучения, рассмотрите следующие основные преимущества.

- Обширное семейство методов для двоичной и многоклассовой классификации, регрессии и обнаружения новизны.
- Хороший генератор прогнозов, надежно справляющийся с переобучением, зашумленными данными и выбросами.
- Успешно справляется с наличием множества переменных.
- Эффективность в ситуациях, когда у вас больше переменных, чем примеров.
- Быстрота, даже если вы работаете с 10 000 учебных примеров.
- Автоматически обнаруживает нелинейность в ваших данных, поэтому вам не нужно применять сложные преобразования переменных.

Все это звучит великолепно. Тем не менее вы также должны учесть несколько важных недостатков, прежде чем приступить к импорту модуля SVM.

- Лучше работает применительно к бинарной классификации (которая и была первоначальной целью SVM), поэтому SVM не работает также хорошо с другими проблемами прогнозирования.
- Менее эффективен, если у вас гораздо больше переменных, чем примеров; вы должны искать другие решения, такие как SGD.
- Предоставляет только прогнозируемый результат; вы можете получить оценку вероятности для каждого ответа за счет более трудоемких вычислений.
- Удовлетворительно работает сразу, но если вы хотите получить наилучшие результаты, придется потратить время на эксперименты, чтобы настроить множество параметров.

Вычислительный метод

Владимир Наумович Вапник и его коллеги изобрели SVM в 1990-х годах, работая в лабораториях AT&T. SVM добился успеха благодаря своей высокой производительности во многих сложных задачах сообщества машинного обучения того времени, особенно когда он используется при чтении компьютером рукописных данных. Сегодня аналитики данных часто применяют SVM для решения множества проблем: от медицинской диагностики до распознавания изображений и классификации текста. Вы, вероятно, найдете SVM весьма полезным и для своих задач!

Код этого раздела относительно длинный и сложный. Он располагается в файле `P4DS4D2_19_Representing_SVM_boundaries.ipynb` вместе с выходными данными, описанными в этом разделе. Чтобы увидеть, как код генерирует рисунки этого раздела, обратитесь к исходному коду.

Идея SVM проста, но математическая реализация довольно сложная и требует много вычислений. Данный раздел поможет вам понять принципы, лежащие в основе этой технологии. Знание того, как работает инструмент, всегда помогает понять, где и как его лучше всего использовать. Начните рассматривать проблему разделения двух групп точек данных — звезд и квадратов, разбросанных по двум измерениям. Это классическая проблема двоичной классификации, в которой алгоритм обучения должен выяснить, как отделить один класс экземпляров от другого, используя информацию, предоставленную имеющимися данными. Первая панель на рис. 19.5 демонстрирует аналогичную проблему.

Если две группы отделены одна от другой, вы можете решать задачу разными способами, просто выбрав разные разделительные линии. Конечно, вы должны обратить внимание на детали и использовать точные измерения. Несмотря на то что это может показаться легкой задачей, вам нужно подумать о том, что происходит при изменении данных, например, последующем добавлении точек. Вы не можете быть уверены, что выбрали правильную разделительную линию.

Вторая панель на рис. 19.5 показывает два возможных решения, но могут существовать и другие. Оба выбранных решения слишком близки к существующим наблюдениям (как показывает близость линий к точкам данных), но нет оснований полагать, что новые наблюдения будут вести себя точно так же, как показано на рисунке. SVM сводит к минимуму риск выбора неправильной линии (как вы могли бы сделать, выбрав решение A или B на рис. 19.6), выбрав решение, характеризующееся наибольшим расстоянием от граничных точек

двух групп. Наличие такого большого пространства между группами (максимально возможное) должно уменьшить вероятность выбора неправильного решения!

```
ax = plt.subplot(1, 2, 2)
ax.scatter(X[:, 0], X[:, 1],
                edgecolor='black',
                facecolor=colors, s=70);
ax.plot([0.20, 0.25],[-2.3, 2.3],'k--')
ax.text(0.30, 2.1, "A")
ax.plot([-1.80, 1.80],[0.6, -0.7],'k--')
ax.text(-1.75, 0.7, "B")
ax.xlim = (-x_max, + x_max)
ax.ylim = (-y_max, + y_max)
ax.margins(0)
```

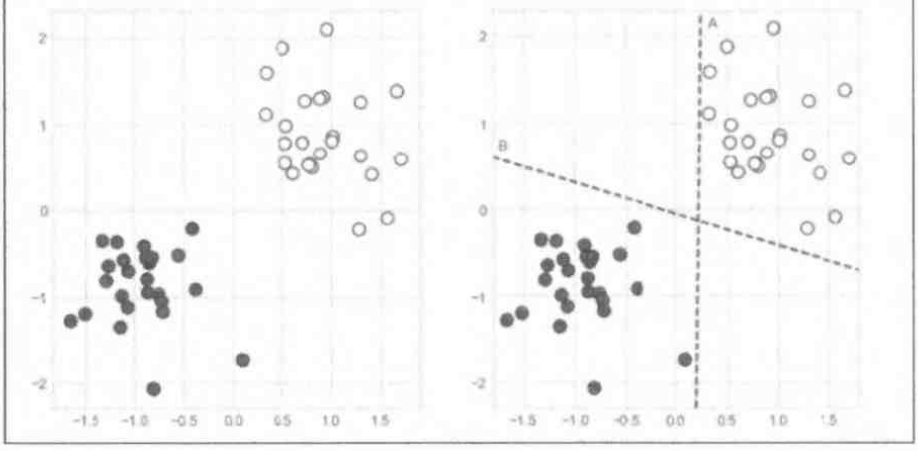


Рис. 19.5. Разделение двух групп

Наибольшее расстояние между двумя группами — это *зазор* (margin). Когда зазор достаточно велик, вы можете быть уверены, что он будет работать хорошо, даже если вам придется классифицировать невиданные ранее данные. Зазор определяется точками, которые расположены на его границе, — *опорными векторами* (support vector) (от них и получил свое название метод опорных векторов).

Вы можете увидеть решение SVM на первой панели рис. 19.6. На рисунке зазор показан в виде пунктирных линий, разделитель — в виде непрерывной линии, а опорные векторы — в виде точек данных, обведенных кружком.

Реальные проблемы не всегда обеспечивают аккуратно разделимые классы, как в этом примере. Тем не менее хорошо настроенный SVM может противостоять некой неоднозначности (некоторые неправильно классифицированные

точки). Алгоритм SVM с правильными параметрами действительно может творить чудеса.

```
ax = plt.subplot(1, 2, 2)
from sklearn.datasets import make_circles

X, y = make_circles(n_samples=50,
                    factor=.3,
                    noise=.1,
                    random_state=1)

X = scale(X)

clf = SVC(kernel='rbf', C=10)
clf.fit(X, y)

colors = ['white' if i==0 else 'blue' for i in y]
ax = plt.scatter(X[:, 0], X[:, 1], edgecolor='black',
                 facecolor=colors, s=70);

ax.xlim = (-x_max, + x_max)
ax.ylim = (-y_max, + y_max)

svc_decision_boundaries(clf);
```

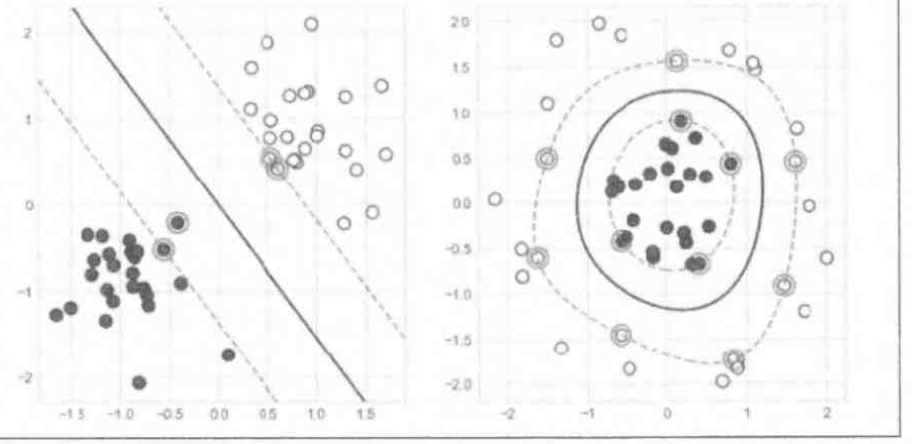


Рис. 19.6. Подходящее решение SVM для задачи двух и более групп

ЗАПОМНИ!

При работе с примерами данных проще искать правильные решения, чтобы точки данных могли лучше объяснить, как работает алгоритм, и вы смогли уяснить основные понятия. Однако с реальными данными вам нужны аппроксимации. Поэтому вы редко видите большие и четкие зазоры.

Помимо двоичных классификаций в двух измерениях, SVM может также работать со сложными данными. Вы можете считать данные сложными, если у вас более двух измерений или в ситуациях, похожих на показанную второй панелью рис. 19.6, когда разделение групп по прямой линии невозможно.

При наличии многих переменных SVM может использовать сложную разделяющую плоскость (*гиперплоскость* (hyperplane)). SVM также хорошо работает, когда вы не можете разделить классы по прямой или плоскости, поскольку он способен исследовать нелинейные решения в многомерном пространстве благодаря вычислительной технике, называемой *трюком ядра* (kernel trick).

Исправление многих новых параметров

Хотя SVM сложен, но это отличный инструмент. После того как вы найдете наиболее подходящую версию SVM для вашей задачи, вы должны применить ее к данным и немного поработать, чтобы оптимизировать некоторые из многих доступных параметров и улучшить свои результаты. Настройка работающей прогностической модели SVM включает в себя следующие общие этапы.

1. Выберите используемый класс SVM.
2. Обучите модель на данных.
3. Выполните проверку на ошибки и сделайте ее базовой линией.
4. Опробуйте разные значения для параметров SVM.
5. Проверьте, улучшается ли ваша проверка на ошибки.
6. Обучите модель снова, используя данные с лучшими параметрами.

Чтобы выбрать правильный класс SVM, вы должны подумать о своей задаче. Например, вы можете выбрать классификацию (угадать класс) или регрессию (угадать число). При работе с классификацией вы должны учитывать, нужно ли вам классифицировать только две группы (двоичная классификация) или более двух (многоклассовая классификация). Другим важным аспектом, который необходимо учитывать, является количество данных, которые вы должны обработать. После того как вы запишете все ваши требования в списке, быстрый взгляд на табл. 19.1 поможет вам сузить свой выбор.

Первый шаг — проверить количество примеров в ваших данных. Наличие более 10000 примеров может означать медленные и громоздкие вычисления, но вы все равно можете использовать SVM с приемлемой производительностью для задач классификации с помощью `sklearn.svm.LinearSVC`. При решении задачи регрессии вы можете обнаружить, что `LinearSVC` недостаточно

быстр, и в этом случае будете использовать стохастическое решение для SVM (как описано в следующих разделах).

Таблица 19.1. Модуль SVM обучающих алгоритмов

Класс	Характеристика	Основные параметры
`sklearn.svm.SVC`	Бинарная и многоклассовая классификация при количестве примеров менее 10000	`C`, `kernel`, `degree`, `gamma`
`sklearn.svm.NuSVC`	Похоже на SVC	`nu`, `kernel`, `degree`, `gamma`
`sklearn.svm.LinearSVC`	Бинарная и многоклассовая классификация при количестве примеров более 10000; разреженные данные	`Penalty`, `loss`, `C`
`sklearn.svm.SVR`	Задачи регрессии	`C`, `kernel`, `degree`, `gamma`, `epsilon`
`sklearn.svm.NuSVR`	Похоже на SVR	`Nu`, `C`, `kernel`, `degree`, `gamma`
`sklearn.svm.OneClassSVM`	Обнаружение выбросов	`nu`, `kernel`, `degree`, `gamma`

СОВЕТ

Модуль SVM Scikit-learn объединяет две мощные библиотеки, написанные на языке C, — `libsvm` и `liblinear`. При подборе модели существует поток данных между Python и двумя внешними библиотеками. Кеш сглаживает операции обмена данными. Но, если кеш слишком мал и у вас слишком много точек данных, кеш становится узким местом! Если у вас достаточно памяти, рекомендуется установить стандартный размер кеша больше 200 Мбайт (1000 Мбайт, если возможно) с помощью параметра `cache_size` класса `SVM`. Меньшее количество примеров требует только того, чтобы вы выбрали между классификацией и регрессией.

В каждом случае у вас будет два альтернативных алгоритма. Например, для классификации вы можете использовать `sklearn.svm.SVC` или `sklearn.svm.NuSVC`. Единственная разница с версией `Nu` — параметры, которые она получает, и использование немного другого алгоритма. В конце концов, версия `Nu`

получает в основном те же результаты, поэтому вы обычно выбираете версию, отличную от Nu.

Решив, какой алгоритм использовать, вы обнаружите, что у вас есть ряд параметров для выбора, и параметр C всегда среди них. Параметр C указывает, насколько алгоритм должен адаптироваться к учебным точкам. Когда значение C мало, SVM меньше адаптируется к точкам и имеет тенденцию принимать среднее направление, используя несколько доступных точек и переменных. Большие значения C приводят к тому, что процесс обучения стремится следовать большему количеству доступных учебных точек и вовлекаться во многие переменные.

Правильным значением C обычно является среднее значение, и вы можете найти его после небольшого количества экспериментов. Если ваше значение C слишком велико, вы рискуете *переобучением* (overfitting) — ситуация, когда ваш SVM слишком сильно адаптируется к данным и не может должным образом справиться с новыми проблемами. Если значение C слишком мало, ваш прогноз будет более грубым и неточным. Вы столкнетесь с ситуацией *недообучения* (underfitting), когда ваша модель слишком проста для задачи, которую вы хотите решить.

После выбора значения C важным блоком параметров для исправления являются kernel, degree и gamma. Все три взаимосвязаны, и их значения зависят от спецификации kernel (например, линейное ядро не требует degree или gamma, поэтому можете использовать любое значение). Спецификация kernel определяет, использует ли ваша модель SVM линию или кривую, чтобы определить класс или точечную меру. Линейные модели проще и имеют тенденцию хорошо угадывать новые данные, но иногда оказываются неэффективными, когда переменные в данных связаны друг с другом сложным образом. Поскольку вы не можете заранее знать, подходит ли линейная модель для вашей задачи, рекомендуется начать с линейного ядра, зафиксировать его значение C, использовать эту модель и ее производительность в качестве основы для последующего тестирования нелинейных решений.

Классификация с использованием SVC

Пришло время построить первую модель SVM. Поскольку SVM изначально так хорошо работал с рукописной классификацией, начало с аналогичной проблемы — отличная идея. Использование этого подхода может дать вам представление о том, насколько мощна эта техника машинного обучения. В этом примере используется набор данных digits (цифры), доступный из наборов данных модулей в пакете Scikit-learn. Набор данных digits содержит серию изображений 8×8 пикселей рукописных цифр в диапазоне от 0 до 9:

```
from sklearn import datasets
digits = datasets.load_digits()
X, y = digits.data, digits.target
```

После загрузки модуля наборов данных функция `load.digits` импортирует все данные, из которых в примере извлекаются предикторы (`digits.data`) как `X`, а прогнозируемые классы (`digits.target`) — как `y`.

Вы можете посмотреть, что находится в этом наборе данных, используя `matplotlib` функции `subplot` (для создания массива рисунков, расположенных в две строки по пять столбцов) и `imshow` (для нанесения значений пикселей в градациях серого на сетку 8×8). Код размещает информацию в `digits.images` в виде последовательности матриц, каждая из которых содержит данные пикселя числа:

```
import matplotlib.pyplot as plt
%matplotlib inline

for k,img in enumerate(range(10)):
    plt.subplot(2, 5, k+1)
    plt.imshow(digits.images[img],
               cmap='binary',
               interpolation='none')
plt.show()
```

Код отображает первые десять чисел в качестве примера данных, использованных в этом примере (рис. 19.7).

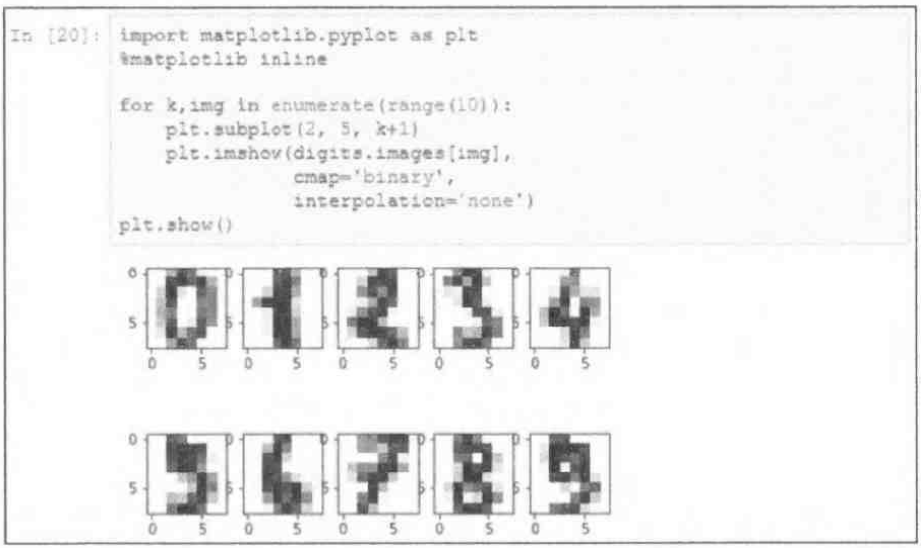


Рис. 19.7. Первые десять рукописных цифр из набора данных `digits`

Наблюдая за данными, вы также можете определить, что SVM смог выявить конкретную цифру, связав вероятность со значениями определенных пикселей в сетке. Число 2 может включать пиксели, отличные от числа 1, или, возможно, разные группы пикселей. Наука о данных включает в себя проверку многих подходов и алгоритмов программирования, прежде чем можно будет получить надежный результат, но это помогает проявить изобретательность и интуитивность, чтобы определить, какой подход опробовать в первую очередь. Фактически, если вы изучите X, то обнаружите, что он состоит ровно из 64 переменных, каждая из которых представляет значение оттенка серого для одного пикселя, и что у вас есть множество примеров — ровно 1797 случаев.

```
print(X[0])
```

Код возвращает вектор первого примера в наборе данных:

```
[ 0. 0. 5. 13.  9. 1. 0.  0.  0. 0. 13. 15. 10. 15.
  5. 0. 0.  3. 15. 2. 0. 11.  8. 0.  0.  4. 12.  0.
  0. 8. 8.  0.  0. 5. 8.  0.  0. 9.  8.  0.  0.  4.
 11. 0. 1. 12.  7. 0. 0.  2. 14. 5. 10. 12.  0.  0.
  0. 0. 6. 13. 10. 0. 0.  0.]
```

Если вы выводите тот же вектор, что и матрица 8×8, то увидите изображение нуля:

```
print(X[0].reshape(8, 8))
```

Вы интерпретируете нулевые значения как белый цвет, а более высокие значения как более темные оттенки серого:

```
[[ 0. 0.  5. 13.  9.  1. 0. 0.]
 [ 0. 0. 13. 15. 10. 15. 5. 0.]
 [ 0. 3. 15.  2.  0. 11. 8. 0.]
 [ 0. 4. 12.  0.  0.  8. 8. 0.]
 [ 0. 5.  8.  0.  0.  9. 8. 0.]
 [ 0. 4. 11.  0.  1. 12. 7. 0.]
 [ 0. 2. 14.  5. 10. 12. 0. 0.]
 [ 0. 0.  6. 13. 10.  0. 0. 0.]]
```

В этот момент вы можете спросить: "Что делать с метками?" Вы можете попробовать подсчитать количество меток, используя функцию `unique` из пакета NumPy:

```
np.unique(y, return_counts=True)
```

Выходные данные связывают метку класса (первое число) с его частотой и значением наблюдения (это вторая строка выходных данных):

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 array([178, 182, 177, 183, 181, 182, 181, 179, 174, 180],
 dtype=int64))
```

Все метки классов представляют примерно одинаковое количество примеров. Это означает, что ваши классы сбалансированы и что SVM не приведет к выводу, что один класс более вероятен, чем любой другой. Если один или несколько классов имели значительно различающееся количество случаев, вы встретились с проблемой несбалансированных классов. Сценарий несбалансированного класса требует выполнения оценки.

- Возьмите несбалансированный класс и получайте прогнозы, смещенные к наиболее частым классам.
- Установите равенство между классами, используя весовые коэффициенты, а значит, некоторые наблюдения могут учитываться больше.
- Используйте выбор, чтобы удалить некоторые случаи из классов, которые имеют слишком много случаев.


СОВЕТ

Проблема несбалансированных классов требует, чтобы вы установили некоторые дополнительные параметры. У `sklearn.svm.SVC` есть параметр `class_weight` и ключевое слово `sample_weight` в методе `fit`. Самый простой способ решить эту проблему — установить `class_weight='auto'` при определении SVC и позволить алгоритму все исправить самостоятельно.

Теперь вы готовы проверить SVC с линейным ядром. Но не забудьте разделить ваши данные на обучающие и тестовые наборы, иначе вы не сможете судить об эффективности работы модели. Всегда используйте для оценки производительности отдельную часть данных, иначе результаты будут хорошо выглядеть с самого начала, но ухудшаться при добавлении свежих данных.

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import MinMaxScaler
X_tr, X_t, y_tr, y_t = train_test_split(X, y,
                                        test_size=0.3,
                                        random_state=0)
```

Функция `train_test_split` разделяет `X` и `y` на обучающие и тестовые наборы, используя в качестве коэффициента разделения значение параметра `test_size`, равное 0,3:

```
scaling = MinMaxScaler(feature_range=(-1, 1)).fit(X_tr)
X_tr = scaling.transform(X_tr)
X_t = scaling.transform(X_t)
```

После разделения данных на обучающую и тестовую части рекомендуется отмасштабировать числовые значения, сначала получая параметры

масштабирования из обучающих данных, а затем применяя преобразование как к обучающим, так и к тестовым наборам.

Еще одним важным действием, которое необходимо выполнить перед подачей данных в SVM, является масштабирование. Масштабирование преобразует все значения в диапазон от –1 до 1 (или от 0 до 1, если вы предпочитаете). Преобразование масштабирования устраняет проблему влияния некоторых переменных на алгоритм (они могут обмануть его, думая, что они важны, поскольку имеют большие значения), и это делает вычисления точными, плавными и быстрыми.

Следующий код помещает обучающие данные в класс SVC с линейным ядром. Он также осуществляет перекрестную проверку и проверку точности результата (процент правильно угаданных чисел):

```
from sklearn.svm import SVC
svc = SVC(kernel='linear',
          class_weight='balanced')
```

Код инструктирует SVC использовать линейное ядро и автоматически взвешивать классы. Повторное взвешивание классов гарантирует, что они останутся равными по размеру после того, как набор данных будет разделен на наборы для обучения и тестирования.

```
cv = cross_val_score(svc, X_tr, y_tr, cv=10)
test_score = svc.fit(X_tr, y_tr).score(X_t, y_t)
```

Затем код присваивает две новые переменные. Эффективность перекрестной проверки записывается функцией `cross_val_score`, которая возвращает список со всеми десятью оценками после перекрестной проверки с десятью блоками (`cv=10`). Код получает результат теста, используя два последовательных метода в алгоритме обучения, — `fit`, который соответствует модели, и `score`, который оценивает результат в наборе тестов, используя среднюю точность (средний процент правильных результатов среди классов для прогнозирования).

```
print('CV accuracy score: %0.3f' % np.mean(cv))
print('Test accuracy score: %0.3f' % (test_score))
```

Наконец, код выводит две переменные и оценивает результат. Результат неплохой: на тестовом наборе 97,4% правильных прогнозов:

```
CV accuracy score: 0.983
Test accuracy score: 0.976
```

Вы можете задаться вопросом: что произойдет, если вы оптимизируете основной параметр `C` вместо использования стандартного значения 1,0? Ответ

дает следующий сценарий, использующий gridsearch для поиска оптимального значения параметра C:

```
from sklearn.model_selection import GridSearchCV
svc = SVC(class_weight='balanced', random_state=1)
search_space = {'C': np.logspace(-3, 3, 7)}
gridsearch = GridSearchCV(svc,
                          param_grid=search_space,
                          scoring='accuracy',
                          refit=True, cv=10)
gridsearch.fit(X_tr,y_tr)
```

Использование GridSearchCV немного сложнее, но позволяет последовательно проверять множество моделей. В первую очередь вы должны определить переменную пространства поиска, используя словарь Python, который содержит расписание исследования процедуры. Чтобы определить пространство поиска, вы создаете словарь (или, если имеется более одного словаря, список словарей) для каждой протестированной группы параметров. В словаре вы помещаете имя параметров в качестве ключей и связываете их со списком (или функцией, генерирующей список, как в данном случае), содержащим значения для тестирования.

СОВЕТ

Функция logspace NumPy создает список из семи значений C в диапазоне от 10^-3 до 10^3. Это вычислительно дорогостоящее количество значений для тестирования, но оно также всеобъемлющее, и вы можете всегда быть уверены, когда тестируете C и другие параметры SVM, используя такой диапазон.

Затем вы инициализируете GridSearchCV, определяя алгоритм обучения, пространство поиска, функцию оценки и количество блоков перекрестной проверки. Следующий шаг — после нахождения наилучшего решения дать указание процедуре подобрать наилучшую комбинацию параметров, чтобы вы имели прогностическую модель, готовую к использованию:

```
cv = gridsearch.best_score_
test_score = gridsearch.score(X_t, y_t)
best_c = gridsearch.best_params_['C']
```

Фактически теперь gridsearch содержит много информации о лучшей оценке (и наилучших параметрах, а также полный анализ всех оцененных комбинаций) и методах, таких как score, которые типичны для подходящих прогностических моделей в Scikit-learn.

```
print('CV accuracy score: %0.3f' % cv)
print('Test accuracy score: %0.3f' % test_score)
print('Best C parameter: %0.1f' % best_c)
```

Здесь код извлекает результаты перекрестной проверки и тестирования, а также выводит значение C, связанное с этими лучшими показателями:

```
CV accuracy score: 0.989
Test accuracy score: 0.987
Best C parameter: 10.0
```

Последний шаг выводит результаты и показывает, что использование C=100 повышает производительность по сравнению с предыдущим случаем, как для перекрестной проверки, так и для набора тестов.

Переходить на нелинейность легко

Теперь, определив простую линейную модель в качестве эталона для проекта рукописных цифр, вы можете проверить более сложную гипотезу, и SVM предлагает ряд нелинейных ядер.

- Полиномиальное (Polynomial — poly).
- Радиальная базисная функция (Radial Basis Function — rbf).
- Сигмовидная функция (sigmoid).
- Улучшенные пользовательские ядра (advanced custom kernel).

Несмотря на то что существует так много вариантов, вы редко используете что-то отличное от ядра радиальной базисной функции (для краткости rbf), поскольку оно быстрее, чем другие ядра, и может аппроксимировать практически любую нелинейную функцию.

ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Вот основное практическое объяснение того, как работает ядро rbf: он разделяет данные на множество кластеров, поэтому ответ легко связать с каждым кластером.

Ядро rbf требует, чтобы помимо настройки C вы установили параметры degree и gamma. Их легко установить (и хороший сеточный поиск всегда найдет правильное значение).

Параметр degree имеет значения, начинающиеся с 2. Он определяет сложность нелинейной функции, используемой для разделения точек. В качестве практического совета: не стоит слишком беспокоиться о параметре degree — проверьте при сеточном поиске значения 2, 3 и 4. Если заметите, что лучший результат имеет степень 4, попробуйте сместить диапазон сетки вверх и проверьте 3, 4 и 5. Продолжайте двигаться вверх по мере необходимости, но использование значения больше 5 редко.

Роль параметра gamma в алгоритме аналогична C (он обеспечивает компромисс между переобучением и недообучением). К ядру rbf это не относится.

Высокие значения gamma побуждают алгоритм создавать нелинейные функции, имеющие неправильную форму, поскольку они, как правило, более точно соответствуют данным. Более низкие значения создают более регулярные сферические функции, игнорируя большинство неровностей, присутствующих в данных.

Теперь, когда вы знаете детали нелинейного подхода, пришло время опробовать rbf на предыдущем примере. Имейте в виду, что, учитывая большое количество протестированных комбинаций, вычисления могут занять некоторое время, в зависимости от характеристик вашего компьютера.

```
from sklearn.model_selection import GridSearchCV
svc = SVC(class_weight='balanced', random_state=101)
search_space = [{'kernel': ['linear'],
                 'C': np.logspace(-3, 3, 7)},
                {'kernel': ['rbf'],
                 'degree':[2, 3, 4],
                 'C':np.logspace(-3, 3, 7),
                 'gamma': np.logspace(-3, 2, 6)}]
gridsearch = GridSearchCV(svc,
                          param_grid=search_space,
                          scoring='accuracy',
                          refit=True, cv=10,
                          n_jobs=-1)
gridsearch.fit(X_tr, y_tr)
cv = gridsearch.best_score_
test_score = gridsearch.score(X_t, y_t)
print('CV accuracy score: %0.3f' % cv)
print('Test accuracy score: %0.3f' % test_score)
print('Best parameters: %s' % gridsearch.best_params_)
```

Обратите внимание, что единственным отличием в этом сценарии является то, что пространство поиска сложней. Используя список, вы применяете два словаря: один содержит параметры для тестирования линейного ядра, а другой — для ядра rbf. Таким образом, вы можете сравнить производительность двух подходов одновременно. Запуск кода займет много времени. После этого он сообщит вам:

```
CV accuracy score: 0.990
Test accuracy score: 0.993
Best parameters: {'C': 1.0, 'degree': 2,
                  'gamma': 0.1, 'kernel': 'rbf'}
```

Результаты подтверждают, что rbf работает лучше. Тем не менее это предел небольшой победы над линейными моделями, получаемый за счет большей сложности и вычислительного времени. В таких случаях наличие большего количества данных может помочь в определении лучшей модели с большей

уверенностью. К сожалению, получение большего количества данных может быть дорогим с точки зрения денег и времени. Столкнувшись с отсутствием четкой модели выигрыша, лучше принять решение в пользу более простой модели. В этом случае линейное ядро намного проще, чем rbf.

Выполнение регрессии с помощью SVR

До сих пор вы имели дело только с классификацией, но SVM также может решать проблемы регрессии. Посмотрев, как работает классификация, вам не нужно знать намного больше того, что классом регрессии SVM является `SVR` и что есть новый параметр `epsilon`, который нужно исправить. Все остальные предыдущие разделы, рассмотренные для классификации, работают точно так же с регрессией.

В этом примере используется другой набор данных — для регрессии. Набор данных Boston о ценах на жилье, взятый из библиотеки StatLib, поддерживаемой университетом Карнеги–Меллона, встречается во многих статьях по машинному обучению и статистике, посвященных проблемам регрессии. Он насчитывает 506 случаев и 13 числовых переменных (одна из которых является двоичной переменной 1/0).

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVR
from sklearn import datasets

boston = datasets.load_boston()
X,y = boston.data, boston.target
X_tr, X_t, y_tr, y_t = train_test_split(X, y,
                                        test_size=0.3,
                                        random_state=0)
scaling = MinMaxScaler(feature_range=(-1, 1)).fit(X_tr)
X_tr = scaling.transform(X_tr)
X_t = scaling.transform(X_t)
```

Целью является среднее значение занимаемых владельцем домов, и вы попытаетесь выяснить его, используя SVR (векторную регрессию с поддержкой `epsilon`). В дополнение к `C`, `kernel`, `degree` и `gamma`, класс `SVR` имеет также параметр `epsilon`. *Эпсилон* (epsilon) — это мера количества ошибок, считаемая алгоритмом приемлемым. Высокое значение `epsilon` подразумевает меньшее количество опорных векторов, а низкое требует большего количества. Другими словами, `epsilon` предоставляет еще один способ компромисса между недообучением и переобучением.

В качестве пространства поиска для этого параметра возьмем последовательность [0, 0.01, 0.1, 0.5, 1, 2, 4]. Опыт подсказывает, что она работает вполне нормально. Начиная с минимального значения 0 (когда алгоритм не допускает никаких ошибок) и до максимального значения 4, вы должны увеличивать пространство поиска только в том случае, если заметите, что более высокие значения epsilon повышают производительность.

Включив epsilon в пространство поиска и назначив SVR в качестве алгоритма обучения, вы можете завершить сценарий. Имейте в виду, что, учитывая большое количество оцененных комбинаций, вычисления могут занять довольно много времени, в зависимости от характеристик вашего компьютера.

```
svr = SVR()
search_space = [{'kernel': ['linear'],
                 'C': np.logspace(-3, 2, 6),
                 'epsilon': [0, 0.01, 0.1, 0.5, 1, 2, 4]},
                {'kernel': ['rbf'],
                 'degree':[2, 3],
                 'C':np.logspace(-3, 3, 7),
                 'gamma': np.logspace(-3, 2, 6),
                 'epsilon': [0, 0.01, 0.1, 0.5, 1, 2, 4]}]
gridsearch = GridSearchCV(svr,
                          param_grid=search_space,
                          refit=True,
                          scoring= 'r2',
                          cv=10, n_jobs=-1)
gridsearch.fit(X_tr, y_tr)
cv = gridsearch.best_score_
test_score = gridsearch.score(X_t, y_t)
print('CV R2 score: %0.3f' % cv)
print('Test R2 score: %0.3f' % test_score)
print('Best parameters: %s' % gridsearch.best_params_)
```

Сеточный поиск может занять некоторое время на вашем компьютере. Несмотря на то что в примере используются все вычислительные мощности вашей системы (n_jobs=-1), компьютер должен протестировать довольно много комбинаций. Для каждого ядра вы можете выяснить, сколько моделей оно должно вычислить, умножив количество значений, которые оно должно протестировать для каждого параметра. Например, для ядра rbf оно имеет два значения: семь для C, шесть для gamma и семь для epsilon, что соответствует 2 * 7 * 6 * 7 = 588 моделям, каждая из которых реплицирована 10 раз (поскольку cv=10). Это 5880 моделей, протестированных только для ядра rbf (код также тестирует линейную модель, которая требует 420 тестов). Наконец, вы должны получить такие результаты:

```
CV R2 score: 0.868
Test R2 score: 0.834
Best parameters: {'C': 1000.0, 'degree': 2, 'epsilon': 2,
                  'gamma': 0.1, 'kernel': 'rbf'}
```

Обратите внимание, что мера ошибки регрессии рассчитывается с использованием R в квадрате, меры в диапазоне от 0 до 1, которая указывает на производительность модели (при этом 1 — наилучший возможный результат для достижения).

Создание стохастического решения с помощью SVM

Теперь, когда мы завершаем обзор семейства алгоритмов машинного обучения SVM, вы понимаете, что они являются фантастическим инструментом для исследователя данных. Конечно, даже у лучших решений есть проблемы. Например, вы можете подумать, что SVM имеет слишком много параметров в классе `SVM`. Разумеется, параметры мешают, особенно если приходится тестировать так много их комбинаций, которые могут занимать много процессорного времени. Однако ключевой проблемой является время, необходимое для обучения SVM. Возможно, вы заметили, что в примерах используются небольшие наборы данных с ограниченным количеством переменных, и выполнение некоторых обширных сеточных поисков все еще занимает много времени. Реальные наборы данных намного больше. Иногда может показаться, что на обучение и оптимизацию SVM на вашем компьютере уйдет вечность.

Возможное решение, если у вас слишком много случаев (рекомендуемый предел составляет 10 000 примеров), находится в том же модуле SVM, в классе `LinearSVC`. Этот алгоритм работает только с линейным ядром, и его цель — классифицировать (извините, без регрессии) большое количество примеров и переменных с более высокой скоростью, чем стандартное SVC. Такие характеристики делают `LinearSVC` хорошим кандидатом для текстовой классификации. В `LinearSVC` меньше параметров для исправления, и они немного другие, чем в обычном SVM (он похож на класс регрессии).

- **C**. Параметр `penalty`. Малые значения подразумевают большую регуляризацию (более простые модели с ослабленными или нулевыми коэффициентами).
- **`loss`**. Значение `l1` (так же, как в SVM) или `l2` (ошибки весят больше, поэтому труднее подобрать ошибочно классифицированные примеры).
- **`penalty`**. Значение `l2` (ослабление менее важных параметров) или `l1` (неважные параметры обнуляются).

» **dual**. Значение `true` или `false`. Относится к типу решаемой задачи оптимизации и, хотя и не сильно изменит полученный выигрыш, установка для параметра значения `false` приведет к более быстрым вычислениям, чем при установленном значении `true`.

Параметры `loss`, `penalty` и `dual` также связаны взаимными ограничениями, поэтому обратитесь к табл. 19.2, чтобы заранее спланировать, какую комбинацию использовать.

Таблица 19.2. Ограничения параметров `loss`, `penalty` и `dual`

Penalty	Loss	Dual
l1	l2	False
l2	l1	True
l2	l2	True; False

ЗАПОМНИ!

Алгоритм не поддерживает комбинацию `penalty='l1'` и `loss='l1'`. Но комбинация `penalty='l2'` и `loss='l1'` идеально повторяет подход оптимизации SVC.

Как упоминалось ранее, класс `LinearSVC` довольно быстр, и тест скорости на SVC демонстрирует уровень улучшения, ожидаемый при выборе этого алгоритма.

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
import numpy as np
X,y = make_classification(n_samples=10**4,
                          n_features=15,
                          n_informative=10,
                          random_state=101)
X_tr, X_t, y_tr, y_t = train_test_split(X, y,
                                        test_size=0.3,
                                        random_state=1)
from sklearn.svm import SVC, LinearSVC
svc = SVC(kernel='linear', random_state=1)
linear = LinearSVC(loss='hinge', random_state=1)

svc.fit(X_tr, y_tr)
linear.fit(X_tr, y_tr)
svc_score = svc.score(X_t, y_t)
libsvc_score = linear.score(X_t, y_t)
print('SVC test accuracy: %0.3f' % svc_score)
print('LinearSVC test accuracy: %0.3f' % libsvc_score)
```

Результаты очень похожи на SVC:

```
SVC test accuracy: 0.803
LinearSVC test accuracy: 0.804
```

После того как вы создадите искусственный набор данных с помощью `make_classfication`, код получает подтверждение того, как два алгоритма достигают почти одинаковых результатов. На этом этапе код проверяет скорость двух решений в искусственном наборе данных, чтобы понять, как они масштабируются для использования большего количества данных.

```
import timeit
X,y = make_classification(n_samples=10**4,
                          n_features=15,
                          n_informative=10,
                          random_state=101)
t_svc = timeit.timeit('svc.fit(X, y)',
                      'from __main__ import svc, X, y',
                      number=1)
t_libsvc = timeit.timeit('linear.fit(X, y)',
                         'from __main__ import linear, X, y',
                         number=1)
print('best avg secs for SVC: %0.1f' % np.mean(t_svc))
print('best avg secs for LinearSVC: %0.1f' % np.mean(t_libsvc))
```

Пример системы показывает следующий результат (вывод вашей системы может отличаться):

```
avg secs for SVC, best of 3: 16.6
avg secs for LinearSVC, best of 3: 0.4
```

Очевидно, что при одинаковом количестве данных `LinearSVC` намного быстрее, чем `SVC`. Вы можете рассчитать коэффициент производительности как `16.6 / 0.4 = 41.5`; он в разы быстрее, чем SVC. Но важно понимать, что происходит, когда вы увеличиваете размер выборки. Например, вот что происходит, когда вы утраиваете размер:

```
avg secs for SVC, best of 3: 162.6
avg secs for LinearSVC, best of 3: 2.6
```

Дело в том, что время, необходимое для SVC, увеличивается быстрее (в 9,8 раза), чем требуется `LinearSVC` (в 6,5 раза). Это связано с тем, что SVC требует пропорционально больше времени для обработки предоставленных данных, и время будет увеличиваться по мере увеличения размера выборки. Ниже приведены результаты, если у вас есть в пять раз больше данных, которые подчеркивают еще больше различий:

```
avg secs for SVC, best of 3: 539.1
avg secs for LinearSVC, best of 3: 4.5
```

Использование SVC с большими объемами данных вскоре становится невозможным; класс LinearSVC должен быть вашим выбором, если вам нужно работать с большими объемами данных. Тем не менее, даже если LinearSVC довольно быстро выполняет задачи, вам может потребоваться классифицировать или регрессировать миллионы примеров. Вы должны знать, является ли LinearSVC лучшим выбором. Ранее вы видели, как класс SGD, используя SGDClassifier и SGDRegressor, помогает реализовать алгоритм типа SVM в ситуациях с миллионами строк данных, не вкладывая слишком много вычислительных ресурсов. Все, что вам нужно сделать, — это установить для loss значение 'hinge' в классе SGDClassifier и 'epsilon_insensitive' в классе SGDRegressor (в этом случае нужно настроить параметр epsilon).

Еще один тест производительности и скорости проясняет преимущества и ограничения использования LinearSVC или SGDClassifier:

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.svm import LinearSVC
import timeit

from sklearn.linear_model import SGDClassifier
X, y = make_classification(n_samples=10**5,
                           n_features=15,
                           n_informative=10,
                           random_state=101)
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                        random_state=1)
```

Выборка сейчас довольно большая — 100 000 случаев. Если у вас достаточно памяти и много времени, вы можете даже увеличить количество обученных случаев или количество признаков и более тщательно проверить, как эти два алгоритма масштабируются с еще бо́льшими данными.

```
linear = LinearSVC(penalty='l2',
                   loss='hinge',
                   dual=True,
                   random_state=1)
linear.fit(X_tr, y_tr)
score = linear.score(X_t, y_t)
t = timeit.timeit("linear.fit(X_tr, y_tr)",
                  "from __main__ import linear, X_tr, y_tr",
                  number=1)
print('LinearSVC test accuracy: %0.3f' % score)
print('Avg time for LinearSVC: %0.1f secs' % np.mean(t))
```

На тестовом компьютере LinearSVC завершил вычисления по всем строкам примерно за семь секунд:

```
LinearSVC test accuracy: 0.796
Avg time for LinearSVC: 7.4 secs
```

Следующий код тестирует `SGDClassifier` с использованием той же процедуры:

```
sgd = SGDClassifier(loss='hinge',
                    max_iter=100,
                    shuffle=True,
                    random_state=101)
sgd.fit(X_tr, y_tr)
score = sgd.score(X_t, y_t)
t = timeit.timeit("sgd.fit(X_tr, y_tr)",
                  "from __main__ import sgd, X_tr, y_tr",
                  number=1)
print('SGDClassifier test accuracy: %0.3f' % score)
print('Avg time SGDClassifier: %0.1f secs' % np.mean(t))
```

`SGDClassifier` вместо этого занял около полутора секунд для обработки тех же данных и получения сопоставимой оценки:

```
SGDClassifier test accuracy: 0.796
Avg time SGDClassifier: 1.5 secs
```

СОВЕТ

Увеличение значения параметра `n_iter` может улучшить производительность, но пропорционально увеличит время вычислений. Увеличение количества итераций до определенного значения (которое вы должны выяснить при тестировании) увеличивает производительность. Однако после этого значения производительность начинает снижаться из-за переобучения.

Играя с нейронными сетями

Начиная с идеи воспроизведения того, как мозг обрабатывает сигналы, исследователи основали нейронные сети на биологических аналогиях и их компонентах, используя в качестве названий такие термины, как *нейроны* и *аксоны*. Но вы обнаружите, что нейронные сети напоминают не более чем сложный вид линейной регрессии, поскольку представляют собой сумму коэффициентов, умноженных на входные числа. Вы также обнаружите, что именно в нейронах происходят такие суммирования.

Даже если нейронные сети не имитируют мозг (он арифметический), эти алгоритмы чрезвычайно эффективны для сложных проблем, таких как распознавание образов, звука или машинный перевод. Они также быстро реализуются при прогнозировании, если вы используете правильное оборудование.

Хорошо разработанные нейронные сети используют термин *глубокое обучение* (deep learning) и стоят за такими мощными инструментами, как Siri, и другими цифровыми помощниками, наряду с более удивительными приложениями машинного обучения.

Для реализации глубокого обучения требуется специальное оборудование (компьютер с графическим процессором) и установка специальных сред, таких как Tensorflow (`https://www.tensorflow.org/`), MXNet (`https://mxnet.apache.org/`), Pytorch (`https://pytorch.org/`) или Chainer (`https://chainer.org/`). В этой книге рассматриваются не сложные нейронные сети, а более простая реализация, предлагаемая Scikit-learn, которая позволяет быстро создавать нейронные сети и сравнивать их с другими алгоритмами машинного обучения.

Понятие нейронных сетей

Основным алгоритмом нейронной сети является нейрон (называемый также *блоком* (unit)). Многие нейроны, расположенные во взаимосвязанной структуре, образуют слои нейронной сети, где каждый нейрон связан с входами и выходами других нейронов. Таким образом, нейрон может получать признаки из примеров или из результатов других нейронов, в зависимости от его расположения в нейронной сети.

В отличие от других алгоритмов, имеющих фиксированный конвейер, определяющий, как алгоритмы получают и обрабатывают данные, нейронные сети требуют, чтобы вы решали, как будет следовать информация, фиксируя количество блоков (нейронов) и их распределение по слоям. По этой причине настройка нейронных сетей — это скорее искусство, чем наука; вы узнаете из опыта, как расположить нейроны в слои и получить лучшие прогнозы. В более детальном представлении нейроны в нейронной сети получают в качестве входных данных множество взвешенных значений, суммируют их и предоставляют сумму в качестве результата.


ЗАПОМНИ

Нейронная сеть может обрабатывать только числовую непрерывную информацию и не может обрабатывать качественные переменные (например, метки, указывающие на качество, такое как красный, синий или зеленый). Вы можете обработать качественные переменные, преобразовав их в непрерывное числовое значение, такое как серия двоичных значений. Нейроны обеспечивают также более сложное преобразования при суммировании. Наблюдая за природой, ученые заметили, что нейроны получают сигналы, но не всегда испускают собственный сигнал. Это зависит от количества полученного сигнала. Когда нейрон в мозге приобретает достаточно стимулов, он дает

ответ; в противном случае молчит. Аналогичным образом нейроны в нейронной сети после получения взвешенных значений суммируют их и используют функцию активации для оценки результата, который преобразует его нелинейным образом. Например, функция активации может выдавать нулевое значение, если на входе не достигнуто определенное пороговое значение, или может ослаблять или усиливать значение в ходе нелинейного масштабирования, передавая таким образом измененный сигнал.

Каждый нейрон в сети получает входные данные от предыдущих слоев (при запуске он напрямую связывается с данными), взвешивает их, суммирует все и преобразует результат, используя функцию активации. После активации вычисленный вывод становится вводом для других нейронов или прогнозом сети. Следовательно, нейронная сеть, состоящая из определенного количества нейронов и слоев, является эффективной структурой для прогнозов, поскольку каждый нейрон использует веса для его входных данных. Такие весовые коэффициенты не отличаются от коэффициентов линейной регрессии, и сеть узнает их значение в ходе повторных проходов (итераций или эпох) по примерам набора данных.

Классификация и регрессия с нейронами

Scikit-learn предлагает две функции для нейронных сетей.

- **`MLPClassifier`**. Реализует *многослойный перцептрон* (Multilayer Perceptron — MLP) для классификации. Его вывод (один или несколько, в зависимости от того, сколько классов вы должны прогнозировать) предназначен для вероятностей примера того или иного класса.
- **`MLPRegressor`**. Реализует MLP для задач регрессии. Все его выводы (поскольку он может прогнозировать несколько целевых значений одновременно) предназначены для оценки мер при прогнозировании.

Поскольку обе функции имеют одни и те же параметры, пример используется в данном случае для классификации рукописных цифр в качестве примера многоклассовой классификации с использованием MLP. Пример начинается с импорта необходимых пакетов, загрузки набора данных в память и разделения его на обучающий и тестовый наборы (как это было сделано при демонстрации методов опорных векторов):

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn import datasets
from sklearn.neural_network import MLPClassifier
digits = datasets.load_digits()
X, y = digits.data, digits.target
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                        random_state=0)
```

Предварительная обработка данных для передачи в нейронную сеть является важным аспектом, поскольку внутренне выполняемые нейронными сетями операции чувствительны к масштабу и распределению данных. Следовательно, хорошей практикой является нормализация данных за счет установки их среднего значения в нуль и их дисперсии в единицу или изменения их масштаба так, чтобы минимальное и максимальное значения находились в диапазоне от –1 до +1 или от 0 до +1. Эксперимент покажет, какое преобразование работает лучше для ваших данных, хотя большинство людей считают, что масштабирование для диапазона –1–...+1 работает лучше. В этом примере все значения масштабируются от –1 до +1.

```
scaling = MinMaxScaler(feature_range=(-1, 1)).fit(X_tr)
X_tr = scaling.transform(X_tr)
X_t = scaling.transform(X_t)
```

СОВЕТ

Как обсуждалось ранее, эффективная практика заключается в том, чтобы определять преобразования предварительной обработки только для учебных данных, а затем применять изученную процедуру к тестовым данным. Только так вы сможете правильно проверить, как ваша модель работает с разными данными.

Чтобы определить MLP, вы должны учитывать, что существует довольно много параметров, и если вы не настроите их правильно, то результаты могут быть неутешительными. (MLP не является алгоритмом, который работает прямо из коробки.) Для правильной работы MLP необходимо сначала определить архитектуру нейронов, указав, сколько использовать для каждого слоя и сколько слоев создавать. (Вы указываете количество нейронов для каждого слоя в параметре `hidden_layer_sizes`.) Затем вы должны выбрать правильный *решатель* (solver) из следующих:

- **L-BFGS**. Используется для небольших наборов данных.
- **Adam**. Используется для больших наборов данных.
- **SGD**. Решает большинство проблем, если вы правильно установили некоторые специальные параметры. L-BFGS работает для небольших наборов данных, Adam — для больших, а SDG может справиться с большинством задач, если вы правильно установите его

параметры. Параметры SGD — это скорость обучения и импульс (или *импульс Нестерова* (Nesterov's momentum)), значение, которое помогает нейронной сети избегать менее полезных решений. При указании скорости обучения вы должны определить ее начальное значение (`learning_rate_init`, которое обычно составляет около 0,001, но может быть даже меньше), а также то, как скорость изменяется во время обучения (параметр `learning_rate`, значением которого может быть `'constant'`, `'invscaling'` или `'adaptive'`).


СОВЕТ

Учитывая сложность настройки параметров для решателя SGD, вы можете определить, как они работают с вашими данными, только протестировав их с оптимизацией гиперпараметров. Большинство людей предпочитают начинать с решателя L-BFGS или Adam.

Другим критическим гиперпараметром является `max_iter`, количество итераций, которое может привести к совершенно разным результатам, если вы установите его слишком низким или слишком высоким. Стандартно используется 200 итераций, но всегда лучше после исправления других параметров попытаться увеличить или уменьшить это значение. Наконец, важно также перемешать данные (`shuffle=True`) и установить `random_state` для воспроизводимости результатов. Пример кода устанавливает 512 узлов на одном слое, опирается на решатель Adam и использует стандартное количество итераций (200):

```
nn = MLPClassifier(hidden_layer_sizes=(512, ),
                   activation='relu',
                   solver='adam',
                   shuffle=True,
                   tol=1e-4,
                   random_state=1)
cv = cross_val_score(nn, X_tr, y_tr, cv=10)
test_score = nn.fit(X_tr, y_tr).score(X_t, y_t)
print('CV accuracy score: %0.3f' % np.mean(cv))
print('Test accuracy score: %0.3f' % (test_score))
```

Используя этот код, пример успешно классифицирует рукописные цифры, запустив MLP, оценки CV и тестов которого таковы:

```
CV accuracy score: 0.978
Test accuracy score: 0.981
```

Полученные результаты немного лучше, чем у SVC, но увеличение также включает в себя правильную настройку нескольких параметров. При использовании нелинейных алгоритмов вы не можете ожидать никакого легкого подхода, кроме нескольких решений на основе дерева решений, которые являются темой следующей главы.

Глава 20

Сила единения

В ЭТОЙ ГЛАВЕ...

- Как работает дерево решений
- Применение Random Forest и других методов упаковки
- Преимущества самых эффективных ансамблей

В этой главе мы выйдем за рамки моделей с одиночным машинным обучением, которые вы видели до сих пор, и исследуем силу *ансамблей* (ensemble), которые представляют собой группы моделей, способные превзойти отдельные модели. Ансамбли работают как коллективный разум толпы, используя объединенную информацию для выработки лучших прогнозов. Основная идея заключается в том, что группа не самых точных алгоритмов может давать лучшие результаты, чем одна хорошо обученная модель.

Возможно, вы участвовали в одной из тех игр, в которых вас просят угадать количество конфет в банке на вечеринках или ярмарках. Даже если у одного человека есть небольшая вероятность угадать правильное число, различные эксперименты подтвердили, что если вы берете неправильные ответы большого числа участников игры и усредняете их, то можете получить правильный ответ! Такое невероятное групповое знание (также известное как мудрость толпы) возможно потому, что неправильные ответы имеют тенденцию располагаться вокруг истинного. Взяв среднее из этих неправильных ответов, вы получите почти правильный ответ.

В проектах по науке о данных, включающих сложные прогнозы, вы можете использовать мудрость различных алгоритмов машинного обучения и стать более точным и аккуратным в прогнозировании, чем при использовании одного алгоритма. В этой главе описывается процесс, который вы можете

использовать для применения возможностей многих разных алгоритмов в процессе получения единственного лучшего ответа.


СОВЕТ

Вам не нужно вводить исходный код этой главы вручную используйте загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `P4DS4D2_20_Understanding_the_Power_of_the_Many.ipynb`.

Простое дерево решений

Деревья решений долгое время были частью инструментов интеллектуального анализа данных. Первые модели появились задолго до 1970-х годов. С тех пор деревья решений пользуются популярностью во многих областях благодаря их интуитивному алгоритму, понятному выводу и эффективности по сравнению с простыми линейными моделями. С введением более эффективных алгоритмов деревья решений на какое-то время медленно покидали сцену машинного обучения, их обвиняли в том, что их было слишком легко переобучить, но в последние годы они вернулись в качестве основного строительного блока алгоритмов ансамблей. Сегодня такие ансамбли древовидных моделей, как Random Forest (случайный лес) или Gradient Boosting Machine (метод градиентного бустинга), являются ядром многих приложений для обработки данных и считаются современными инструментами машинного обучения.

Понятие дерева решений

В основе деревьев решений лежит идея о том, что вы можете делить набор данных на все более мелкие и мелкие части, используя специальные правила, основанные на значениях признаков набора данных. При таком разделении набора данных алгоритм должен выбирать деления, которые увеличивают вероятность правильного прогноза целевого результата, либо в качестве класса, либо в виде оценки. Поэтому алгоритм должен пытаться максимизировать присутствие определенного класса или определенного среднего значения в каждом делении.

В качестве примера применения дерева решений попытайтесь предсказать вероятность выживания пассажиров на британском пассажирском лайнере RMS Titanic, который затонул в Северной Атлантике в апреле 1912 года после столкновения с айсбергом. В Интернете доступно несколько наборов данных, относящихся к этой трагедии. Наиболее заметным среди них является вебсайт Encyclopedia Titanica (`https://www.encyclopedia-titanica.org`), который содержит статьи, биографии и данные. Еще один — это конкурс Kaggle

по науке о данных, в котором приняли участие десятки тысяч энтузиастов (https://www.kaggle.com/c/titanic).

Многие наборы данных о трагедии "Титаника" отличаются содержащимися в них данными. Пример, приведенный в этой главе, основан на наборе данных Titanic, который бесплатно предоставляется Департаментом биостатистики при Медицинской школе Университета Вандербильта (Department of Biostatistics at the Vanderbilt University School of Medicine) и доступен для загрузки по адресу http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.csv. В этом наборе данных содержится 1309 зарегистрированных пассажиров с полной статистикой. Вы не найдете в наборе данных ни одного из членов экипажа, поскольку в записях основное внимание уделяется платным пассажирам, чтобы определить, является ли выживание при крушении вопросом удачи или местом, где пассажиры оказались на корабле во время столкновения. Коэффициент выживаемости среди пассажиров составил 38,2 процента (500 из 1309 пассажиров погибли). Исходя из характеристик пассажиров, дерево решений определяет следующее.

- У мужчин вероятность выживания снижается с 38,2 до 19,1%.
- У мальчиков младше 9,5 лет вероятность выживания увеличивается до 58,1%.
- У женщин, независимо от возраста, предположительная вероятность выживания 72,7%.

Используя такие знания, вы можете легко построить дерево, подобное изображенному на рис. 20.1. Такая визуализация (и визуализация набора данных Iris, приведенная далее в этой главе) возможна благодаря пакету dtreeviz, разработанному профессором Теренсом Парром из Университета Сан-Франциско (https://parrt.cs.usfca.edu) и Принсем Гровером с того же факультета. Если вы заинтересованы в создании визуализаций ваших деревьев решений, получите пакет и руководство по его установке по адресу https://github.com/parrt/dtreeviz и прочитайте о разработке и функционировании пакета в публикации блога профессора Парра "How to visualize decision trees" по адресу https://explained.ai/decision-tree-viz.Обратите внимание, что у визуализированного дерева корень расположен вверху, а ветви спускаются вниз. Пример начинается сверху. Затем он разделяется по половому признаку, создавая две *ветви* (branch), одна из которых превращается в лист. *Лист* (leaf) — это конечный сегмент. Диаграмма классифицирует случаи листьев, используя наиболее частый класс или вычисляя базовую вероятность случаев с теми же характеристиками, что и листовая вероятность. Вторая ветвь — разделение по возрасту.

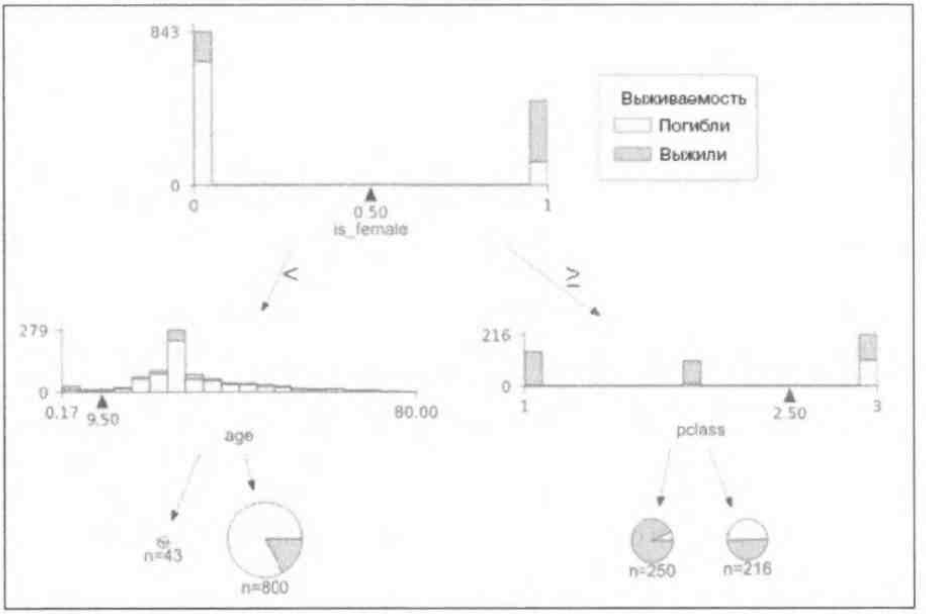


Рис. 20.1. Модель дерева выживаемости при катастрофе "Титаника"

При чтении узлов дерева, учтите, что самый верхний узел начинается с сообщения о правиле, используемом для разделения этого узла на все следующие узлы. Вы должны начать с вершины. Дерево, показанное на рис. 20.1, предполагает, что пол является лучшим предиктором, а на верхнем узле переменная `is_female` находится в вертикальной гистограмме с накоплением. Левый столбец предназначен для мужчин, правый — для женщин. Вы сразу заметите, что у женщин выживаемость была относительно более высокой, поскольку доля выживших (светло-зеленая область, которая не отображается в цвете на печатной книге) занимает почти весь столбец.

Дерево разделяет данный узел пополам, отделяя мужчин от женщин. Вы можете прочитать остальную часть истории, рассказанной деревом, наблюдая за тем, что происходит на следующем уровне. На втором уровне представления дерева, справа, вы найдете узел, состоящий только из женщин, а гистограммы с накоплением раскрывают ключевую информацию: выжили почти все пассажиры первого и второго класса, а около половины женщин-пассажиров третьего класса погибли. Понимание это позволяет дереву выработать первое правило: женщины в первом и втором классе могут быть классифицированы как выжившие, поскольку этот статус весьма вероятен. Что касается третьего класса, то выживаемость неясна, и дерево следует снова разделить, чтобы извлечь некое другое понимание, которое анализ не включает.

Что касается мужчин, то второй уровень показывает, что различающим критерием является возраст, поскольку находящиеся в возрасте до десяти лет, скорее всего, выжили, в то время как пожилые, скорее всего, погибли. Опять же, дерево останавливается, но дополнительные критерии могут предоставить более точный набор правил разделения, способные исследовать вероятность выживания в катастрофе "Титаника" на основе собственных характеристик. Из разделения дерева на верхнем уровне можно заметить, что большинство выживших — это женщины с детьми (согласно морскому правилу: женщины и дети — в первую очередь, применяемого в ситуациях, когда ресурсы для спасения жизни ограничены). Это правило полностью соответствует ситуации с "Титаником", поскольку из-за убежденности владельцев в непотопляемости корабля на борту оказалось недостаточно спасательных шлюпок. (Вы можете прочитать больше предположений о спасательных шлюпках на веб-сайте History по адресу `https://www.historyonthenet.com/thetitanic-lifeboats/`.)


ЗАПОМНИ!

В этом примере дерева каждое разделение является двоичным, но также возможны множественные разделения, в зависимости от алгоритма дерева. В модуле `sklearn.tree` библиотеки Scikit-learn реализованные классы `DecisionTreeClassifier` и `DecisionTreeRegressor` являются двоичными деревьями. Дерево решений может прекратить разделять данные, в следующих ситуациях.

- Больше нет случаев для разделения, поэтому все данные являются частью конечных узлов.
- Используемое для разделения листьев правило имеет меньшее предварительно определенное количество случаев. Это действие не позволяет алгоритму работать с листами, которые имеют небольшое представление в целом или являются более конкретными, чем данные, которые вы анализируете, тем самым предотвращая переобучение (см. главу 18) и дисперсию оценок.
- В одном из полученных листов меньше заданного количества случаев — еще одна проверка на исправность, позволяющая избежать вывода общих правил без уверенности, обеспечиваемой хорошим размером выборки.


СОВЕТ

Деревья решений имеют тенденцию переобучаться на данных. Установив правильное количество для разделений и конечных листьев, вы можете уменьшить дисперсию оценок. В зависимости от начального размера выборки ограничение 30-ю случаями обычно является хорошим выбором.

Помимо того, что деревья решений интуитивно понятны, просты для понимания и представления (в зависимости от количества ветвей и листьев на вашем дереве), они имеют еще одно мощное преимущество для аналитиков данных — не требуют никакой конкретной обработки или преобразования данных, поскольку они моделируют любую нелинейность, используя приближения. Фактически они получают любые переменные, даже категориальные, закодированные произвольными кодами для представленных классов. Кроме того, деревья решений обрабатывают пропущенные случаи. Все, что вам нужно сделать, — это присвоить недостающим случаям маловероятное значение, такое как экстремальное или отрицательное (в зависимости от распределения данных по присутствующим случаям). Наконец, деревья решений также невероятно устойчивы к выбросам.

Создание деревьев для разных целей

Аналитики данных называют деревья, специализирующиеся на прогнозировании *классов* (class) (атрибуты, качества или признаки, которые определяют группы), *деревьями классификации* (classification tree). Деревья, работающие с оценками, называются *деревьями регрессии* (regression tree). Ниже приведена проблема классификации, использующая набор данных Iris (мы использовали этот набор данных в разделе "Определение описательной статистики для числовых данных" главы 13).

```
from sklearn.datasets import load_iris
iris = load_iris()
X, y = iris.data, iris.target
features = iris.feature_names
```

После загрузки в `X` данных, которые содержат предикторы, и в `y`, содержащих классификации, вы можете определить перекрестную проверку для результатов с использованием деревьев решений:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
crossvalidation = KFold(n_splits=5,
                        shuffle=True,
                        random_state=1)
```

Используя класс `DecisionTreeClassifier`, определим `max_depth` в итеративном цикле, чтобы поэкспериментировать с эффектом увеличения сложности результирующего дерева. Ожидается, что мы быстро достигнем идеальной точки, а затем станем свидетелями снижения производительности перекрестной проверки из-за переобучения:

```
import numpy as np
from sklearn import tree
for depth in range(1,10):
    tree_classifier = tree.DecisionTreeClassifier(
        max_depth=depth, random_state=0)
    if tree_classifier.fit(X,y).tree_.max_depth < depth:
        break
    score = np.mean(cross_val_score(tree_classifier,
                                    X, y,
                                    scoring='accuracy',
                                    cv=crossvalidation))
    print('Depth: %i Accuracy: %.3f' % (depth,score))
```

Код будет перебирать более глубокие деревья до тех пор, пока дерево не перестанет расширяться, а затем код сообщит оценку перекрестной проверки для точности:

```
Depth: 1 Accuracy: 0.580
Depth: 2 Accuracy: 0.913
Depth: 3 Accuracy: 0.920
Depth: 4 Accuracy: 0.940
Depth: 5 Accuracy: 0.920
```

Лучшее решение — это дерево с четырьмя разделениями, поскольку при дальнейшем росте дерево начинает переобучаться. На рис. 20.2 показана сложность результирующего дерева, которое предоставляет другую визуализацию, полученную с помощью пакета dtreeviz. Визуализация помогает показать, что вы можете легко отличить вид Ирис щетинистый (Setosa) от других. Различение между радужным (Versicolor) и виргинским (Virginica) ирисом требует тщательной сегментации по ширине и длине лепестка.

Чтобы получить эффективное сокращение и упрощение, можно установить `min_samples_split` равным 30 и избегать слишком маленьких конечных листьев, установив `min_samples_leaf` равным 10. Этот параметр сокращает небольшие конечные листья в новом результирующем дереве, уменьшая точность перекрестной проверки, но увеличивая простоту и обобщающая силу решения.

```
tree_classifier = tree.DecisionTreeClassifier(
    min_samples_split=30, min_samples_leaf=10,
    random_state=0)
tree_classifier.fit(X,y)
score = np.mean(cross_val_score(tree_classifier, X, y,
                                scoring='accuracy',
                                cv=crossvalidation))
print('Accuracy: %.3f' % score)
```

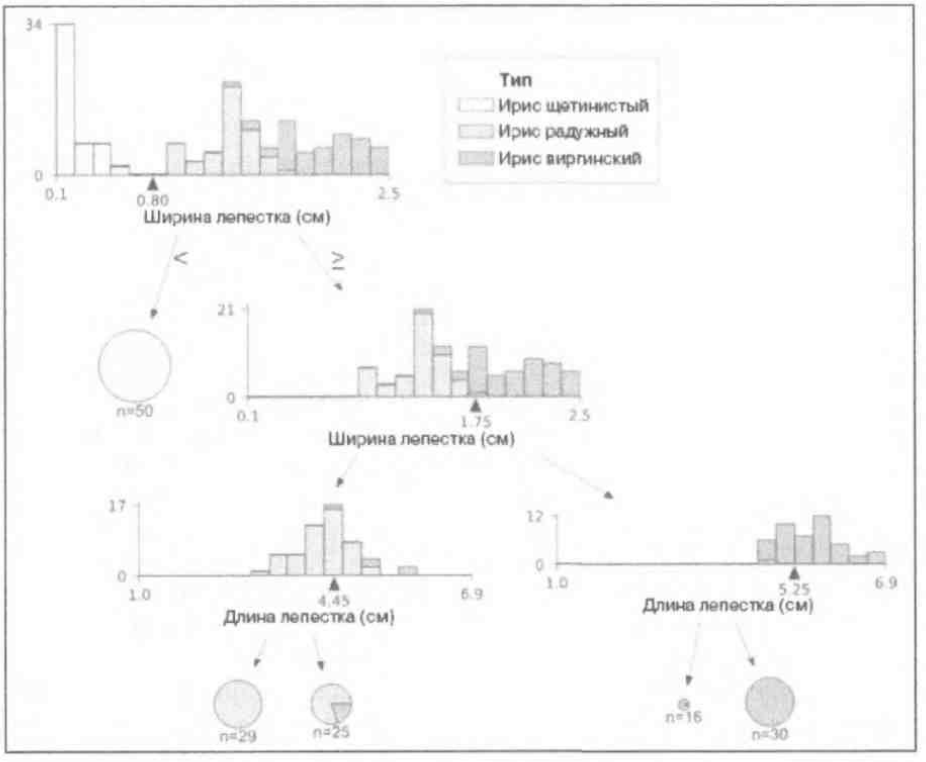


Рис. 20.2. Модель дерева набора данных Iris, использующая глубину четырех разделений

Сообщаемая точность перекрестной проверки меньше, чем ранее полученная оценка, потому что фокусировка на более простой древовидной структуре подразумевает необходимость некоторого подбора к проблеме данных:

```
Accuracy: 0.913
```

Аналогично, используя класс `DecisionTreeRegressor`, вы можете смоделировать регрессионную задачу, такую как набор цен на жилье в Бостоне (мы использовали этот набор данных в разделе "Определение приложений для науки о данных" главы 12). При работе с деревом регрессии конечные листья предлагают среднее значение наблюдений в качестве результата прогнозирования.

```
from sklearn.datasets import load_boston
boston = load_boston()
X, y = boston.data, boston.target
features = boston.feature_names

from sklearn.tree import DecisionTreeRegressor
regression_tree = tree.DecisionTreeRegressor(
```

```
    min_samples_split=30, min_samples_leaf=10,
    random_state=0)
regression_tree.fit(X,y)
score = np.mean(cross_val_score(regression_tree,
                                X, y,
                                scoring='neg_mean_squared_error',
                                cv=crossvalidation))
print('Mean squared error: %.3f' % abs(score))
```

Перекрестно проверенная среднеквадратическая ошибка для набора данных о ценах на жилье в Бостоне равна

```
Mean squared error: 22.593
```

Как сделать доступным машинное обучение

Случайный лес (Random Forest) — это алгоритм классификации и регрессии, разработанный Лео Брейманом и Адель Катлер, использующий большое количество моделей дерева решений для точных прогнозов, уменьшая как смещение, так и дисперсию оценок. Когда вы объединяете множество моделей в единое целое, вы получаете *ансамбль моделей* (ensemble of models). Случайный лес — это не просто ансамбль моделей, это также простой и эффективный алгоритм, который вполне готов для использования. Это делает машинное обучение доступным для неопытных разработчиков. Для выполнения прогнозов алгоритм Random Forest использует следующие этапы.

1. Создает большое количество деревьев решений, каждое из которых отличается от другого, на основе разных подмножеств наблюдений и переменных.
2. Для каждого дерева загружает набор данных наблюдений (выбираются из исходных данных с заменой). Одно и то же наблюдение может встречаться в одном наборе данных несколько раз.
3. Произвольно выбирает и использует только часть переменных для каждого дерева.
4. Оценивает производительность для каждого дерева, используя наблюдения, исключенные из выборки (оценка OOB (Out Of Bag — не вошедший в набор)).
5. После того как все деревья были подобраны и использованы для прогнозирования, получает окончательный прогноз, являющийся средним для регрессионных оценок или наиболее часто встречающимся классом.

Используя эти этапы, вы уменьшите смещение, поскольку деревья решений хорошо подходят для данных и, опираясь на сложные разбиения, могут аппроксимировать даже самые сложные отношения между предикторами и

прогнозируемым результатом. Деревья решений могут давать большую дисперсию оценок, но вы уменьшаете ее, усредняя множество деревьев. Прогнозы с шумом из-за дисперсии имеют тенденцию равномерно распределяться выше и ниже правильного значения, которое вы хотите предсказать, и при усреднении они, как правило, взаимно компенсируют друг друга, оставляя в результате более правильный средний прогноз.

Лео Брейман вывел идею случайного леса из техники упаковки. Scikit-learn имеет классы упаковки как для регрессии (`BaggingRegressor`), так и для классификации (`BaggingClassifier`), которые вы можете использовать с любым другим предиктором, который вы предпочитаете выбирать из модулей Scikit-learn. Параметры `max_samples` и `max_features` позволяют определить пропорцию случаев и переменных для выборки (не начальную загрузку, а выборочную, поэтому вы можете использовать случай только один раз) при построении каждого ансамбля моделей. Параметр `n_estimators` определяет общее количество моделей в ансамбле. Ниже приведен пример, который загружает набор данных рукописных цифр (использованный для демонстраций позже с другими алгоритмами ансамбля).

```
from sklearn.datasets import load_digits
digit = load_digits()
X, y = digit.data, digit.target

from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
tree_classifier = DecisionTreeClassifier(random_state=0)
crossvalidation = KFold(n_splits=5, shuffle=True,
                        random_state=1)
bagging = BaggingClassifier(tree_classifier,
                            max_samples=0.7,
                            max_features=0.7,
                            n_estimators=300)
scores = np.mean(cross_val_score(bagging, X, y,
                                 scoring='accuracy',
                                 cv=crossvalidation))
print ('Accuracy: %.3f' % scores)
```

Вот перекрестно проверенная точность для бэггинга, примененного к рукописному набору данных:

```
Accuracy: 0.967
```

В бэггинге, как и в алгоритме Random Forest, чем больше моделей в ансамбле, тем лучше. Вы рискуете переобучением, поскольку каждая модель отличается от других, и ошибки имеют тенденцию распространяться на реальное

значение. Добавление большего количества моделей просто придает больше стабильности результату.

Другой характеристикой алгоритма является то, что он позволяет оценивать значение переменной, учитывая наличие всех других предикторов. Таким образом, вы можете определить, какая функция важна для прогнозирования цели с учетом набора имеющихся у вас функций; Кроме того, вы можете использовать оценку важности в качестве ориентира для выбора переменных.

В отличие от отдельных деревьев решений, вы не можете легко визуализировать или понять случайный лес, заставляя его действовать как черный ящик (*черный ящик* (black box) — это преобразование, внутренняя работа которого не раскрывается; все, что вы видите, — это его входы и выходы). Учитывая его непрозрачность, оценка важности является единственным способом понять, как алгоритм работает в отношении признаков.

Оценка важности в случайном лесу получена простым способом. После построения каждого дерева код заполняет каждую переменную по очереди случайными данными, и пример записывает, насколько сильно снижается мощность прогнозирования. Если переменная важна, заполнение ее случайными данными наносит ущерб прогнозу; в противном случае прогнозы остаются практически без изменений, а переменная считается неважной.

Работа с классификатором Random Forest

Пример классификатора Random Forest продолжает использовать загруженный ранее набор цифр:

```
X, y = digit.data, digit.target
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
crossvalidation = KFold(n_splits=5, shuffle=True,
                        random_state=1)
RF_cls = RandomForestClassifier(n_estimators=300,
                                random_state=1)
score = np.mean(cross_val_score(RF_cls, X, y,
                                scoring='accuracy',
                                cv=crossvalidation))
print('Accuracy: %.3f' % score)
```

Перекрестно-проверенная точность, сообщаемая этим кодом для случайного леса, является улучшением по сравнению с методом бэггинга, проверенным в предыдущем разделе:

```
Accuracy: 0.977
```

Для большинства проблем, с которыми вы сталкиваетесь, достаточно установить количество оценщиков, а правильная их установка — это использование максимально возможного числа с учетом временных и ресурсных ограничений используемого компьютера. Вы можете продемонстрировать это в ходе расчета и построения кривой проверки алгоритма.

```
from sklearn.model_selection import validation_curve
param_range = [10, 50, 100, 200, 300, 500, 800, 1000, 1500]
crossvalidation = KFold(n_splits=3,
                        shuffle=True,
                        random_state=1)
RF_cls = RandomForestClassifier(n_estimators=300,
                                random_state=0)
train_scores, test_scores = validation_curve(RF_cls, X, y,
                                             'n_estimators',
                                             param_range=param_range,
                                             cv=crossvalidation,
                                             scoring='accuracy')
mean_test_scores = np.mean(test_scores, axis=1)

import matplotlib.pyplot as plt
plt.plot(param_range, mean_test_scores,
         'bD-.', label='CV score')
plt.grid()
plt.xlabel('Number of estimators')
plt.ylabel('accuracy')
plt.legend(loc='lower right', numpoints= 1)
plt.show()
```

На рис. 20.3 показаны результаты, предоставленные предыдущим кодом. Чем больше оценок, тем лучше результаты. Но в определенный момент выигрыш становится минимальным.

Работа с регрессором Random Forest

`RandomForestRegressor` работает аналогично случайному лесу для классификации, используя точно такие же параметры:

```
X, y = boston.data, boston.target
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
RF_rg = RandomForestRegressor (n_estimators=300,
                               random_state=1)
crossvalidation = KFold(n_splits=5, shuffle=True,
                        random_state=1)
score = np.mean(cross_val_score(RF_rg, X, y,
```

```
                scoring='neg_mean_squared_error',
                cv=crossvalidation))
print('Mean squared error: %.3f' % abs(score))
```

```
In [10]: import matplotlib.pyplot as plt
         %matplotlib inline
         plt.plot(param_range, mean_test_scores,
                  'bD-', label='CV score')
         plt.grid()
         plt.xlabel('Number of estimators')
         plt.ylabel('accuracy')
         plt.legend(loc='lower right', numpoints= 1)
         plt.show()
```

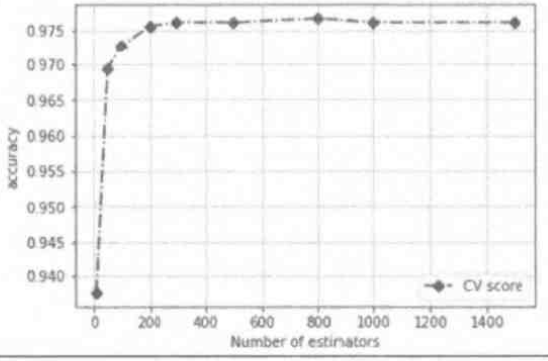


Рис. 20.3. Проверка влияния количества оценок на случайный лес

Вот итоги перекрестной проверки для среднеквадратичной ошибки:

```
Mean squared error: 12.028
```


ЗАПОМНИ!

Случайный лес использует деревья решений. При оценке значений регрессии деревья решений сегментируют набор данных на небольшие секции — *листья* (leave). Чтобы создать прогноз, случайный лес получает среднее значение в каждом листе. Использование этой процедуры приводит к тому, что экстремальные и слишком высокие значения исчезают из прогнозов из-за усреднения, используемого для каждого листа леса, что приводит к сглаживанию чрезвычайно высоких или низких значений.

Оптимизация Random Forest

Модели случайного леса — это готовые алгоритмы, способные работать довольно хорошо без оптимизации и опасности переобучения. (Чем больше оценок вы используете, тем лучше будет результат, в зависимости от ваших ресурсов.) Вы всегда можете улучшить производительность, удалив избыточные

и менее информативные переменные, установив минимальный размер листа и определив количество выборок, что позволит избежать слишком большого количества коррелированных предикторов в выборке. В следующем примере показано, как выполнить эти задачи:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
X, y = digit.data, digit.target
crossvalidation = KFold(n_splits=5, shuffle=True,
                        random_state=1)
RF_cls = RandomForestClassifier(random_state=1)
scorer = 'accuracy'
```

Используя набор рукописных цифр и первый стандартный классификатор, вы можете оптимизировать как `max_features`, так и `min_samples_leaf`. При оптимизации `max_features` вы используете предварительно заданные параметры (`auto` для всех признаков, функции `sqrt` или `log2`, применяемые к номерам признаков) и интегрируете их, используя небольшие номера признаков и значение 1/3 признаков. Выбор правильного количества объектов для выборки, как правило, сокращает количество раз, когда коррелированные и похожие переменные выбираются вместе, увеличивая таким образом качество прогноза.

Есть статистическая причина для оптимизации `min_samples_leaf`. Использование листов с несколькими случаями зачастую соответствует подбору к очень конкретным комбинациям данных. Чтобы получить минимальную статистическую уверенность в том, что шаблоны данных соответствуют реальным и общим правилам, необходимо иметь не менее 30 наблюдений:

```
from sklearn.model_selection import GridSearchCV
max_features = [X.shape[1]//3, 'sqrt', 'log2', 'auto']
min_samples_leaf = [1, 10, 30]
n_estimators = [50, 100, 300]
search_grid = {'n_estimators':n_estimators,
               'max_features': max_features,
               'min_samples_leaf': min_samples_leaf}
search_func = GridSearchCV(estimator=RF_cls,
                           param_grid=search_grid,
                           scoring=scorer,
                           cv=crossvalidation)
search_func.fit(X, y)
best_params = search_func.best_params_
best_score = search_func.best_score_
print('Best parameters: %s' % best_params)
print('Best accuracy: %.3f' % best_score)
```

Затем сообщается о наилучших получаемых параметрах и наилучшей точности, подчеркивая, что параметром, на который нужно воздействовать, является количество деревьев:

```
Best parameters: {'max_features': 'sqrt',
                  'min_samples_leaf': 1,
                  'n_estimators': 100}
Best accuracy: 0.978
```

Бустинг прогнозов

Сбор различных моделей деревьев — это не единственная возможная техника ансамбля. Другая техника машинного обучения, называемая *бустингом* (boosting), также эффективно использует ансамбли. В бустинге вы выращиваете множество деревьев последовательно. Каждое дерево пытается построить модель, которая успешно предсказывает, какие деревья были построены до него. Метод объединяет последующие модели вместе и использует для окончательного прогноза взвешенное среднее значение или взвешенное большинство голосов.

В следующих разделах представлены два бустинговых приложения: *adaboost* и *метод градиентного бустинга* (gradient boosting machine). Вы можете использовать все алгоритмы бустинга как для регрессии, так и для классификации. Примеры в этих разделах начинаются с классификации. Хорошим местом для начала является набор рукописных цифр, как и в случае с Random Forest.

Если вы уже загрузили данные, используя `load_digits`, в переменную `digit`, вам просто нужно переприсвоить переменные `X` и `y` следующим образом:

```
X, y = digit.data, digit.target
```

Зная, что победят многие слабые предикторы

`AdaBoostClassifier` подходит для последовательных слабых предикторов. Он стандартно используется при работе с деревьями решений, но вы можете выбрать другие алгоритмы, изменив параметр `base_estimator`. Слабые предикторы, как правило, являются предикторами машинного обучения, которые работают плохо, поскольку имеют слишком большую дисперсию или смещение, поэтому они работают немногим лучше, чем случайный шанс. Классическим примером слабого ученика является *решающий пень* (decision stump), который представляет собой дерево решений, выросшее до одного уровня. Обычно деревья решений — это наиболее эффективный вариант бустинга, поэтому вы можете безопасно использовать стандартного ученика и сосредоточиться на

двух параметрах, важных для получения хороших прогнозов: `n_estimators` и `learning_rate`.

Параметр `learning_rate` определяет, как каждый слабый предиктор влияет на конечный результат. Высокая скорость обучения требует нескольких параметров `n_estimators`, прежде чем перейти к оптимальному решению, но это, вероятно, не будет наилучшим возможным решением. Низкая скорость обучения требует больше времени для обучения, потому что для достижения решения требуется больше предикторов. Кроме того, он медленнее переобучается.


СОВЕТ

В отличие от бэггинга, бустинг может привести к перегрузке, если вы используете слишком много оценок. Перекрестная проверка всегда помогает найти правильное число, имея в виду, что более низкие скорости обучения занимают больше времени, чем переобучение, поэтому проще выбрать почти оптимальное значение с помощью свободного сеточного поиска.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
ada = AdaBoostClassifier(n_estimators=1000,
                         learning_rate=0.01,
                         random_state=1)
crossvalidation = KFold(n_splits=5, shuffle=True,
                        random_state=1)
score = np.mean(cross_val_score(ada, X, y,
                                scoring='accuracy',
                                cv=crossvalidation))
print('Accuracy: %.3f' % score)
```

После запуска кода перекрестная проверка возвращает точность:

```
Accuracy: 0.754
```

В этом примере используется стандартный оценщик, представляющий собой полноценное дерево решений. Если вы хотите опробовать пень (который нуждается в дополнительных оценщиках), вам следует создать экземпляр `AdaBoostClassifier` с `base_estimator=DecisionTreeClassifier(max_depth=1)`.

Установка классификатора градиентного бустинга

Метод градиентного бустинга (Gradient Boosting Machine — GBM) работает намного лучше, чем метод бустинга Adaboost, — первый когда-либо созданный алгоритм бустинга. В частности, GBM использует расчет оптимизации для взвешивания последующих оценок. Как и в примере из предыдущего раздела,

в следующем примере используется набор данных цифр и рассматриваются некоторые дополнительные параметры, доступные в GBM:

```
X, y = digit.data, digit.target
crossvalidation = KFold(n_splits=5,
                        shuffle=True,
                        random_state=1)
```

Помимо скорости обучения и количества оценщиков, которые являются ключевыми параметрами для оптимального обучения без переобучения, вы должны предоставить значения для `subsample` и `max_depth`. `subsample` вводит в обучение *подвыборку* (subsampling) (так, чтобы обучение каждый раз проводилось на другом наборе данных), как при бэггинге. `max_depth` определяет максимальный уровень построенных деревьев. Обычно рекомендуется начинать с трех уровней, но для моделирования сложных данных может потребоваться больше уровней.

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import cross_val_score
GBC = GradientBoostingClassifier(n_estimators=300,
                                 subsample=1.0,
                                 max_depth=2,
                                 learning_rate=0.1,
                                 random_state=1)
score = np.mean(cross_val_score(GBC, X, y,
                                scoring='accuracy',
                                cv=crossvalidation))
print('Accuracy: %.3f' % score)
```

Для той же самой задачи, которую вы проверяли ранее, `GradientBoosting Classifier` дает следующий показатель точности после выполнения кода:

```
Accuracy: 0.972
```

Запуск регрессора градиентного бустинга

Создание регрессора градиентного бустинга не слишком отличается от создания классификатора. Основным отличием является наличие нескольких функций потерь, которые вы можете использовать (в отличие от `GradientBoostingClassifier`, который имеет только функцию потерь отклонения, аналогично функции стоимости логистической регрессии).

```
X, y = boston.data, boston.target
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
GBR = GradientBoostingRegressor(n_estimators=1000,
```

```
                                subsample=1.0,
                                max_depth=3,
                                learning_rate=0.01,
                                random_state=1)
crossvalidation = KFold(n_splits=5,
                        shuffle=True,
                        random_state=1)
score = np.mean(cross_val_score(GBR,
                                X, y,
                                scoring='neg_mean_squared_error',
                                cv=crossvalidation))
print('Mean squared error: %.3f' % abs(score))
```

После выполнения кода вы получите среднеквадратическую ошибку для регрессии, которая намного лучше, чем таковая у случайного леса:

```
Mean squared error: 10.094
```

В этом примере `GradientBoostingRegressor` обучается с использованием стандартного значения `ls` для параметра потерь, что аналогично линейной регрессии. Ниже приведено еще несколько вариантов.

- `quantile`. Прогнозирует конкретный квантиль, который вы указываете, используя параметр `alpha` (обычно это 0,5, что является медианой).
- `lad` (least absolute deviation — наименьшее абсолютное отклонение). Этот выбор очень устойчив к выбросам; но обычно правильно ранжирует прогнозы.
- `huber`. Комбинация `ls` и `lad`. Требуется исправить параметр `alpha`.

Использование гиперпараметров GBM

Модели GBM весьма чувствительны к переобучению, когда у вас слишком много последовательных оценок, и модель начинает приспосабливаться к шуму в данных. Важно проверить эффективность связанных значений количества оценок и скорости обучения. В следующем примере используется набор данных о ценах на жилье в Бостоне:

```
X, y = boston.data, boston.target
from sklearn.model_selection import KFold
crossvalidation = KFold(n_splits=5, shuffle=True,
                        random_state=1)
GBR = GradientBoostingRegressor(n_estimators=1000,
                                subsample=1.0,
                                max_depth=3,
                                learning_rate=0.01,
                                random_state=1)
```

Оптимизация может занять некоторое время из-за вычислительной нагрузки, требуемой алгоритмами GBM, особенно если вы решите проверить высокие значения `max_depth`.


СОВЕТ

Хорошая стратегия заключается в том, чтобы поддерживать фиксированную скорость обучения и пытаться оптимизировать `subsample` и `max_depth` по отношению к `n_estimators` (имея в виду, что высокие значения `max_depth` обычно подразумевают меньшее количество оценщиков). После того как вы найдете оптимальные значения для `subsample` и `max_depth`, можете начать поиск дальнейшей оптимизации `n_estimators` и `learning_rate`.

```
from sklearn import grid_search
from sklearn.model_selection import GridSearchCV
subsample = [1.0, 0.9]
max_depth = [2, 3, 5]
n_estimators = [500 , 1000, 2000]
search_grid = {'subsample': subsample,
               'max_depth': max_depth,
               'n_estimators': n_estimators}
search_func = GridSearchCV(estimator=GBR,
                           param_grid=search_grid,
                           scoring='neg_mean_squared_error',
                           cv=crossvalidation)
search_func.fit(X,y)

best_params = search_func.best_params_
best_score = abs(search_func.best_score_)
print('Best parameters: %s' % best_params)
print('Best mean squared error: %.3f' % best_score)
```

После запуска оптимизации вы можете проверить полученную в результате среднеквадратичную ошибку и заметить, как она улучшила ход выполнения алгоритма по стандартным параметрам. (Градиентный бустинг всегда требует некоторой настройки параметров для получения наилучших результатов.)

```
Best parameters: {'max_depth': 3,
                  'n_estimators': 2000,
                  'subsample': 0.9}
Best mean squared error: 9.324
```

6 Великолепные десятки

В ЭТОЙ ЧАСТИ...

- Поиск необходимых источников данных
- Повышение квалификации с использованием сетевых источников
- Поиск актуальных задач и их использование для своей пользы
- Участие в решении актуальных задач

Глава 21

Десять основных источников данных

В ЭТОЙ ГЛАВЕ...

- Поиск хорошей отправной точки
- Получение необходимых учебных материалов
- Отслеживание авторитетных источников
- Получение необходимого ресурса разработчика

Читая эту книгу, вы многое узнаете о науке о данных и языке Python. Прежде чем ваша голова взорвется от всех новых знаний, которые вы приобретаете, важно понять, что эта книга только верхушка айсберга. Да, информации действительно больше, и именно об этом данная глава. В следующих разделах вы познакомитесь со множеством коллекций ресурсов по науке о данных, которые вам нужны, чтобы наилучшим образом использовать новые знания.

В данном случае коллекция ресурсов — это просто список действительно интересных ссылок на текст, который объясняет, почему они так хороши. В некоторых случаях вы получаете доступ к статьям по науке о данных, в других — знакомитесь с новыми инструментами. На самом деле, наука о данных — это такая огромная тема, что вы можете легко найти куда больше ресурсов, чем обсуждается здесь, но следующие разделы дают хорошее начало.

Как и все остальное в Интернете, ссылки устаревают, сайты перестают работать, но на их место приходят новые. Если вы обнаружите, что ссылка не работает, сообщите мне об этом по адресу John@JohnMuellerBooks.com.

Поиск новостей в Subreddit

Отрасль науки о данных постоянно меняется по ряду причин, включая добавление новых алгоритмов и методов, а также использование все более крупных наборов данных из все более разнообразного набора источников. Следовательно, вам нужен источник новостей, такой как Subreddit (https://www.reddit.com/r/datascience/), чтобы получать самую свежую информацию и опережать своих конкурентов. Эти сообщения в блогах также часто содержат новейшие методики, гарантирующие, что после того, как вы освоите науку о данных, вы сможете оставаться таковым. Кроме того, вы найдете темы, которые важны для вашей карьеры, например, поиск диапазона зарплат в аналитике данных. Этот сайт предоставляет также информацию о языке Python по адресу https://www.reddit.com/r/Python/ и новости науки о данных по адресу https://www.reddit.com/r/datasciencenews/.

Хорошее начало с KDnuggets

Изучение интеллектуального анализа данных и науки о данных — это процесс. По адресу https://www.kdnuggets.com/faq/learning-data-mining-datascience.html KDnuggets разделяет процесс обучения на ряд этапов. Каждый этап дает обзор того, что вы должны делать и почему. Вы также найдете ссылки на различные ресурсы в Интернете, чтобы значительно облегчить процесс обучения. Несмотря на то что сайт подчеркивает использование для выполнения задач по науке о данных языков R, Python и SQL (в таком порядке), эти этапы на самом деле будут работать для любого из подходов, которые вы можете предпринять.

Как и в случае с любым другим опытом обучения, процедура, подобная показанной на сайте KDnuggets, для одних людей будет работать, а для других нет. Каждый учится немного по-своему. Не бойтесь импровизировать. Ресурсы на этом сайте могут дать представление и о других вещах, которые вы можете сделать, чтобы облегчить процесс обучения.

Поиск бесплатных учебных материалов с помощью Quora

Противостоять слову "бесплатно" действительно сложно, особенно когда речь идет об образовании, которое обычно стоит много тысяч долларов. Сайт Quora по адресу `https://www.quora.com/What-are-the-best-free-resources-to-learndata-science` предоставляет список лучших бесплатных учебных ресурсов для науки о данных.

Большинство ссылок имеют формат вопросов, например "Как мне стать аналитиком данных?" Формат вопросов и ответов полезен, поскольку вы можете задавать вопросы, на которые отвечает сайт. Полученный список сайтов, курсов и ресурсов по большей части является вводным, но это хороший способ начать работу в области науки о данных.


СОВЕТ

Несколько ссылок указывают на такие престижные учреждения, как Гарвард. Ссылка дает доступ к материалам курса, таким как лекционные видео и доски. Тем не менее вы не получите фактический курс бесплатно. Если хотите воспользоваться преимуществами курса, вам все равно придется платить за него. Тем не менее, просто просматривая материалы курса, вы можете получить много полезных знаний в области науки о данных.

Получение знаний на блоге Oracle Data Science

Крупные поставщики могут предложить значительное количество полезной информации. Конечно, вы должны помнить об источнике этой информации, поскольку она может быть весьма предвзятой. Блог Oracle Data Science Blog (`https://www.datascience.com/blog`) предоставляет значительный объем информации — от самых последних методов анализа данных до методов, которые вы можете использовать для сокращения затрат. Кроме того, вы найдете категоризированную информацию, основанную на

- лучших практиках;
- образовании науки о данных;
- случаях применения;
- науке о данных как о платформе.

Доступ к огромному списку ресурсов на Data Science Central

Многие ресурсы, которые вы найдете в Интернете, охватывают основные темы. Data Science Central (`https://www.datasciencecentral.com/`) предоставляет доступ к относительно большому количеству экспертов в науке о данных, которые рассказывают о самых неясных фактах. Одно из наиболее интересных сообщений в блоге размещено по адресу `https://www.datasciencecentral.com/profiles/blogs/huge-trellolist-of-great-data-science-resources`.

Список Trello (`https://trello.com/`), содержащий некоторые по-настоящему удивительные ресурсы. Навигация по огромному списку может быть немного трудной, но этому помогает древовидная структура, которую Trello предоставляет для организации информации. Просматривать этот список имеет смысл, когда у вас есть время, или вы просто хотите посмотреть, что доступно (возможно, к тому времени, когда вы прочитаете эту книгу). Категории включают в себя следующее.

- Новостные данные
- Данные деловых людей
- Журналистские данные
- Научные данные
- Статистика
- R
- Python
- Большие данные и другие инструменты
- Данные
- Другое

Изучение новых трюков на Aspirational Data Scientist

Блог Aspirational Data Scientist (`https://newdatascientist.blogspot.com/`) предлагает удивительный набор очерков на различные темы науки о данных. Автор разбивает публикации на следующие темы: комментарий по науке о данных, обзоры сетевых курсов, как стать аналитиком данных.

Наука о данных привлекает практиков из всех существующих видов отраслей. Сайт, кажется, в основном посвящен удовлетворению потребностей социологов, работающих в области науки о данных. На самом деле наиболее интересная публикация, расположенная по адресу `https://newdatascientist.blogspot.com/p/useful-links.html`, содержит список ресурсов, которые помогут социологу перейти в область аналитиков данных. Список ресурсов организован по авторам, поэтому вы можете найти имена, которые уже наметили в качестве потенциальных информационных ресурсов.

Как и с любым другим ресурсом, даже если статья предназначена для одной аудитории, она нередко одинаково хорошо подходит для другой. Даже если вы не социолог, вы можете обнаружить, что статьи содержат полезную информацию, когда вы продвигаетесь по пути к полному открытию чудес науки о данных.

Наиболее авторитетные источники на Udacity

Даже при хорошем подключении к Интернету и наличии поисковой системы попытка найти нужный ресурс может оказаться сложной задачей. U Climb Higher опубликовал список из 24 научных ресурсов по адресу `https://blog.udacity.com/2014/12/24-data-science-resources-keep-finger-pulse.html`, которые гарантированно помогут вам держать руку на пульсе новых стратегий и технологий. Этот ресурс затрагивает следующие темы: тенденции и события; места, где можно больше узнать о науке о данных; как присоединиться к сообществу; новости науки о данных; люди, которые действительно хорошо знают науку о данных; все последние исследования.

Получение справки о передовых темах в Conductrics

Сайт Conductrics (`https://conductrics.com/`) в целом посвящен продаже продуктов, помогающих решать различные задачи по обработке данных. На сайте есть также блог, содержащий несколько полезных публикаций, отвечающих на сложные вопросы, на которые вам может быть сложно получить ответы в других местах. Эти два сообщения расположены по адресам `https://conductrics.com/datascience-resources/` и `https://conductrics.com/data-scienceresources-2`.

Автор публикаций на блоге, Мэтт Гершофф, ясно дает понять, что эти списки являются результатом ответов на вопросы. Список огромен и расположен в двух публикациях, а не в одной, поэтому Мэтт должен ответить на многие вопросы. Список фокусируется в основном на машинном обучении, а не на аппаратном обеспечении или конкретных проблемах кодирования. Поэтому можно ожидать появления записей по таким темам, как *латентное семантическое индексирование* (Latent Semantic Indexing — LSI); *разложение по сингулярным значениям* (Single Value Decomposition — SVD); *линейный дискриминантный анализ* (Linear Discriminant Analysis — LDA); *непараметрические байесовские подходы* (nonparametric Bayesian approach); *статистический машинный перевод* (statistical machine translation); *обучение с подкреплением* (Reinforcement Learning — RL); *временные различия* (Temporal Difference — TD) обучения.

СОВЕТ

У этого списка нет конца. Многие из этих записей не будут иметь особого смысла для вас прямо сейчас, если вы еще не сильно вовлечены в науку о данных. Однако авторы пишут многие статьи таким образом, чтобы помочь вам подобрать информацию, даже если вы не совсем знакомы с ней. В большинстве случаев вам лучше хотя бы просмотреть статью, чтобы понять, можете ли вы ее понять. Если статья начинает обретать смысл, прочтите ее подробно. В противном случае сохраните ссылку на статью для последующего использования. Вы можете быть удивлены, обнаружив, что статья, которую вы не можете до конца понять сегодня, станет тем, что вы легко поймете завтра.

Получение фактов науки о данных с открытым исходным кодом от мастеров

Многие организации в настоящее время сосредоточены на решениях науки о данных с открытым исходным кодом. Внимание к этой столь велико, что теперь вы можете получить образование в области Open Source Data Science Masters (OSDSM) по адресу `http://datasciencemasters.org/`. Акцент делается на предоставлении материалов, которые обычно отсутствуют в чисто академическом образовании. Другими словами, сайт предоставляет ссылки на курсы, которые заполнят пробелы в вашем образовании, чтобы вы стали более конкурентоспособными в современной компьютерной среде. Различные ссылки предоставляют доступ к сетевым курсам, книгам и другим ресурсам, которые помогут лучше понять, как работает OSDSM.

Как сосредоточиться на ресурсах для разработчиков с Джонатаном Бауэром

На сайте GitHub (`https://github.com/`), посвященном совместной работе, обзору кода и управлению кодом, появилось множество интересных ресурсов. Одним из сайтов, которые вам нужно посетить, является список ресурсов по науке о данных Джонатана Бауэра (Jonathan Bower) по адресу `https://github.com/jonathanbower/DataScienceResources`. Большая часть этих ресурсов адресована разработчику, но пользу из них может извлечь любой. Вы найдете ресурсы, разделенными на следующие темы.

- Наука о данных, начало работы.
- Конвейер данных и инструменты.
- Продукт.
- Кадровые ресурсы.
- Ресурсы о данных с открытым исходным кодом.

Иерархическое форматирование различных тем облегчает поиск необходимой информации. Каждая основная категория делится на список тем. В каждой теме вы найдете список ресурсов, относящихся к этой теме. Например, в Data Pipeline & Tools (Конвейер данных и инструменты) есть раздел Python, содержащий ссылку на Anyone Can Code (Программировать может каждый). Это один из самых полезных сайтов в списке.

Глава 22

Десять задач, которые вы должны решить

В ЭТОЙ ГЛАВЕ...

- Определение начальных задач
- Работа с конкретными видами данных
- Анализ, распознавание шаблонов и классификация
- Работа с огромными сетевыми наборами данных

Наука о данных — это все, что связано с работой с данными. Читая эту книгу, вы использовали несколько наборов данных, в том числе игрушечные наборы данных, которые поставляются с библиотекой Scikit-learn. Конечно, все эти наборы данных отлично подходят для начала работы, но бегун не остановится после победы на коротком забеге, и вам нужно начинать тренироваться для марафонов по науке о данных, работая с большими наборами данных.

В этой главе вы познакомитесь с рядом сложных наборов данных, которые помогут вам стать специалистом по данным мирового уровня. Комбинируя то, что вы обнаружите в этой книге, с новыми наборами данных, вы можете научиться делать удивительные вещи. На самом деле некоторые люди могут посчитать вас волшебником, когда вы вытаскиваете из своей головы, казалось бы, невозможные шаблоны данных. Каждый из следующих наборов данных предоставляет определенные навыки и помогает достигать различных целей.

ЗАПОМНИ!

В Интернете вы можете найти множество наборов данных. Однако не все они созданы равными, и вам необходимо тщательно выбирать задачи. Следующие десять наборов данных предоставляют хорошо известные функциональные возможности, их часто предоставляют учебные пособия, и они появляются в научных статьях. Это делает данные наборы данных отличными от конкурентов. Да, доступны и другие хорошие наборы данных, но эти десять наборов данных предоставляют навыки, необходимые для решения еще более сложных задач, таких как та база данных, которая скрывается на сервере вашей компании.

Знакомство с конкурсом Data Science London + Scikit-Learn

В этой книге вы использовали библиотеку Scikit-Learn довольно много, так что, возможно, уже немного ее поняли. Конкурс Kaggle по адресу `https://www.kaggle.com/c/data-science-london-Scikit-learn` (текущий конкурс завершился в декабре 2014 года, но должны быть и другие) предоставляет практическую площадку для попыток общения и создания примеров с использованием алгоритмов классификации Scikit-learn. Все инструменты для предыдущего конкурса все еще в силе, и их все стоит изучить. Цель заключается в том, чтобы попробовать создать и поделиться примерами использования возможностей классификации Scikit-learn. Вы можете найти данные, использованные для конкурса, по адресу `https://www.kaggle.com/c/data-science-london-scikit-learn/data`. Правила находятся по адресу `https://www.kaggle.com/c/data-science-londonscikit-learn/rules`, а то, как Kaggle оценивает ваши материалы, вы можете узнать по адресу `https://www.kaggle.com/c/data-science-london-scikitlearn/details/evaluation`.

Конечно, у вас может и не быть желания соревноваться. Просмотр таблицы лидеров (`https://www.kaggle.com/c/data-science-london-scikitlearn/leaderboard`) может помешать вам всерьез задуматься о реальной конкуренции, поскольку в конкурсе принимали участие серьезные аналитики данных. Тем не менее вы можете получить удовольствие, пытаясь понять, как решить сложную задачу с данными, и в то же время узнать что-то новое, не отправляя свое решение с целью попасть в таблицу лидеров.

ЗАПОМНИ!

Поскольку этот сайт основан на знаниях, которые вы уже получили из книги, на самом деле это лучшее место, где можно начать развивать новые навыки. Вот почему этот сайт рассматривается первым в

этой главе: он позволяет начать использовать другие наборы данных с методами, которые вы уже знаете.

Прогнозирование выживания на "Титанике"

Вы уже в некоторой степени работали с данными "Титаника" в этой книге (главы 6 и 20), используя `Titanic.csv` и `Titanic3.csv` из материалов Медицинской школы Университета Вандербильта. На самом деле эта задача намного проще, чем описанная в предыдущем разделе, поскольку конкурс Kaggle разработал ее для начинающих. Вы можете найти ее на `https://www.kaggle.com/c/titanic`. Модель данных по адресу `https://www.kaggle.com/c/titanic/data` отличается от той, что приведена в книге, но концепции совпадают. Правила для этого конкурса приведены по адресу `https://www.kaggle.com/c/titanic/rules`, а метод оценки по адресу `https://www.kaggle.com/c/titanic#evaluation`.

Список лидеров этого конкурса представлен на `https://www.kaggle.com/c/titanic/leaderboard`. Количество людей, которые уже достигли того, что составляет идеальный результат, должно внушать вам уверенность.

СОВЕТ

Самая большая проблема в этом случае заключается в том, что набор данных довольно мал и требует создания новых признаков для получения точного результата. Конкурс поможет вам применить навыки, которые вы изучили в разделе "Искусство создания признаков" главы 9.

Как находить конкурсы Kaggle, соответствующие вашим потребностям

Конкурсы помогают вам продумывать решения в среде, в которой другие делают то же самое. В реальном мире вы регулярно будете сталкиваться с конкуренцией, поэтому соревнования дают хороший опыт критического и быстрого мышления. Они также позволяют учиться у других. Лучшее место для поиска таких конкурсов — на сайте Kaggle по адресу `https://www.kaggle.com/competitions`.

Этот сайт поможет найти любой прошлый или настоящий конкурс Kaggle. Чтобы найти текущий конкурс, перейдите по ссылке Active Competitions

(Текущие конкурсы). Чтобы найти прошлые конкурсы, перейдите по ссылке All Competitions (Все конкурсы). Все наборы данных находятся в свободном доступе, поэтому у вас есть шанс опробовать свои навыки в любом реальном сценарии, который вы захотите выбрать. Сообщество Kaggle предоставит вам множество учебных пособий, тестов и публикаций с превосходными результатами.


ЗАПОМНИ!

Вам не нужно выбирать текущий конкурс. Например, вы можете увидеть прошлые конкурсы, которое удовлетворяют вашим потребностям, и попробовать вместо текущего их (используя опубликованные решения). Если вы принимаете активное участие в конкурсе, вы можете опубликовать свои вопросы на форуме, и некоторые из самых опытных в мире аналитиков данных ответят на ваши вопросы и сомнения. Из-за большого количества конкурсов на этом сайте, вероятно, вы найдете такой, который будет полностью отвечать вашим интересам!

Интересно отметить, что в конкурсах Kaggle участвуют компании, которые обычно не имеют доступа к аналитикам данных, поэтому вы действительно работаете в реальных условиях. Вы также можете использовать этот сайт, чтобы найти потенциальную работу. Просто перейдите на `https://www.kaggle.com/jobs`, щелкнув на ссылке Jobs (Работа) на главной странице.

Как оттачивать свои стратегии

Набор данных Madelon (`https://archive.ics.uci.edu/ml/datasets/Madelon`) — это искусственный набор данных, содержащий задачу классификации для двух классов с непрерывными входными переменными. Это задача выбора признаков NIPS 2003 серьезно проверит ваши навыки в перекрестной проверке моделей. Основной упор в этой задаче делается на том, чтобы разработать стратегии, позволяющие избежать переобучения, — задача, с которой вы впервые встретились в разделе "Что еще может пойти не так" главы 16. Проблемы переобучения, упомянутые в главах 18–20, приведены ниже. Чтобы получить набор данных, свяжитесь с Изабель Гюйон (Isabelle Guyon) по адресу, указанному в разделе Source (Источник) на странице `https://archive.ics.uci.edu/ml/datasets/Madelon`.


СОВЕТ

Этот конкретный набор данных привлек внимание ряда людей, которые написали статьи об этом. Лучшие работы представлены в книге *Feature Extraction, Foundations and Applications* (`https://www.springer.com/us/book/9783540354871`). Вы также можете

загрузить соответствующий технический отчет по адресу `https://clopinet.com/isabelle/Projects/ETH/TM-fextractclass.pdf`. Публикация *Advances in Neural Information Processing Systems 17* (NIPS 2004) по адресу `https://papers.nips.cc/book/advances-in-neuralinformation-processing-systems-17-2004` также содержат полезные ссылки на статьи, которые помогут вам с этим конкретным набором данных.

Пробираясь через набор данных MovieLens

Сайт MovieLens (`https://movielens.org/`) поможет найти фильм, который вам понравится. В конце концов, благодаря миллионам фильмов поиск чего-то нового и интересного может занять время, которое вы не хотите тратить. Система работает, запрашивая у вас оценки для фильмов, о которых вы уже знаете. Затем сайт MovieLens дает вам рекомендации на основе ваших рейтингов. Короче говоря, ваши рейтинги учат алгоритм тому, что искать, а затем сайт применяет этот алгоритм ко всему набору данных.

Вы можете получить набор данных MovieLens по адресу `https://grouplens.org/datasets/movielens/`. Интересной особенностью этого сайта является то, что вы можете загрузить весь набор данных или его часть, в зависимости от того, как вы хотите с ним взаимодействовать. Вы можете найти загрузки в следующих размерах.

- 100000 оценок от 1000 пользователей по 1700 фильмам.
- 1 миллион оценок от 6000 пользователей по 4000 фильмов.
- 10 миллионов оценок и 100 000 тегов, примененных к 10 000 фильмов 72 000 пользователей.
- 20 миллионов оценок и 465 000 тегов, примененных к 27 000 фильмов 138 000 пользователей.
- Последний набор данных MovieLens в небольших или полных размерах (полный размер содержал 21 000 000 оценок и 470 000 меток, примененных к 27 000 фильмов 230 000 пользователей на момент написания статьи, но со временем размер будет увеличиваться).

Этот набор данных позволяет работать с пользовательскими данными, используя методы как с учителем, так и без учителя. Большие наборы данных представляют особые проблемы, которые могут обеспечить только большие данные. Некоторую начальную информацию для работы с методами с учителем и без учителя см. в главах 15 и 19.

Избавление от спама

Каждый хочет избавиться от спама в электронной почте, содержащем все, от приглашения присоединиться к фантастически новому предприятию до порнографии. Конечно, лучший способ решить эту задачу — создать алгоритм, который сделает сортировку за вас. Однако вам необходимо обучить алгоритм выполнять его работу, и именно здесь вступает в действие набор данных Spambase. Набор данных Spambase представлен по адресу `https://archive.ics.uci.edu/ml/datasets/Spambase`.

Эта коллекция спама поступила от почтовых служб и отдельных лиц, передавших сообщения о спаме. Он также включает в себя электронную почту без спама из разных источников, что позволяет создавать фильтры, пропускающие хорошие электронные письма. Это сложная задача, связанная с текстовыми данными и разными сложными целями.

СОВЕТ

Вы можете найти ряд статей, которые ссылаются на этот конкретный набор данных. В следующем списке приведен краткий обзор соответствующих документов и содержащих их сайтов:

- Los Alamos National Laboratory Stability of Unstable Learning Algorithms (`http://rexa.info/paper/a2734ae038cae7393159934e860c24a52dc2754d`)
- Modeling for Optimal Probability Prediction (`http://rexa.info/paper/631197638c7e0317c98e1a8d98e5fce8921aa758`)
- Visualization and Data Mining in an 3-D Immersive Environment: Summer Project 2003 (`http://rexa.info/paper/48d6beec2a36a87d9d88b6de85dd85a75e5ed24d`)
- Online Policy Adaptation for Ensemble Classifiers (`http://rexa.info/paper/3cb3fbd5512e3cd12111b598fece53fcb42c484b`)

Работа с рукописной информацией

Распознавание образов, особенно работа с рукописной информацией, является важной задачей науки о данных. Набор данных рукописных цифр на сайте `http://yann.lecun.com/exdb/mnist/` Смешанного национального института стандартов и технологий (Mixed National Institute of Standards and Technology — MNIST) содержит обучающий набор из 60000 примеров и набор

из 10 000 примеров. Это подмножество исходного набора данных Национального института стандартов и технологий (National Institute of Standards and Technology — NIST), который можно найти по адресу `https://srdata.nist.gov/gateway/gateway?keyword=handwriting+recognition`. Это хороший набор данных, который можно использовать, чтобы научиться работать с рукописными данными без необходимости выполнять большую часть предварительной обработки с самого начала.


СОВЕТ

Набор данных содержится в четырех файлах. Два обучающих и два тестовых файла содержат изображения и метки. Чтобы создать полный набор данных для работы с цифрами, вам нужны все четыре файла. Потенциальная проблема при работе с набором данных MNIST заключается в том, что файлы изображений не имеют определенного формата. Формат, используемый для хранения изображений, отображается в нижней части страницы. Конечно, на языке Python вы всегда можете создать собственное приложение для чтения, но использовать код, созданный кем-то другим, намного проще. В следующем списке представлены места, где вы можете получить код для чтения набора данных MNIST с использованием языка Python:

- `https://cs.indstate.edu/~jkinne/cs475-f2011/code/mnistHandwriting.py`
- `https://martin-thoma.com/classify-mnist-with-pybrain/`
- `https://gist.github.com/akesling/5358964`

На главной странице содержится также важный список методов, используемых для работы с обучающим и тестовым наборами. Список содержит внушительное количество классификаторов, которые должны дать некоторые идеи для ваших собственных экспериментов. Дело в том, что этот конкретный набор данных полезен для самых разных задач.


ЗАПОМНИ!

Вы работали с игрушечным набором цифр из Scikit-learn в нескольких главах книги. Чтобы использовать этот набор данных, импортируйте базу данных цифр, используя `from sklearn.datasets import load_digits`. Этот конкретный набор данных приведен в главах 12, 15, 17, 19 и 20.

Работа с изображениями

Наборы данных Канадского института перспективных исследований (Canadian Institute for Advanced Research — CIFAR) по адресу `https://www.cs.toronto.edu/~kriz/cifar.html` предоставляют графическое содержимое для обработки различными способами. Наборы данных CIFAR-10 и CIFAR-100 содержат помеченные подмножества набора данных с 80 миллионами крошечных изображений (о том, как набор данных работает с исходным набором изображений, см. в техническом отчете *Learning Multiple Layers of Features from Tiny Images* по адресу `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`). В наборе данных CIFAR-10 содержится 60 000 цветных изображений размером 32×32 в десяти классах (по 6000 изображений в каждом). Вот классы, которые вы найдете:

- Airplane (самолет)
- Automobile (автомобиль)
- Bird (птица)
- Cat (кот)
- Deer (олень)
- Dog (собака)
- Frog (лягушка)
- Horse (лошадь)
- Ship (корабль)
- Truck (грузовик)

Набор данных CIFAR-100 содержит больше классов. Вместо 10 вы получаете 100 классов, содержащих по 600 изображений в каждом. Размер набора данных такой же, но количество классов больше. В этом случае система классификации является иерархической. 100 классов делятся на 20 суперклассов. Например, в суперклассе водных млекопитающих имеются классы бобров, дельфинов, выдр, тюленей и китов (beaver, dolphin, otter, seal, whale).


ВНИМАНИЕ!

Оба набора данных CIFAR доступны в Python, MATLAB и двоичных версиях. Убедитесь, что загрузили правильную версию, и следуйте инструкциям по их использованию на странице загрузки. Да, вы можете использовать другие версии с Python, но для этого потребуется много дополнительного программирования, а поскольку у вас уже есть доступ к версии Python, вы ничего не получите от этого упражнения.

Это отличная задача, которую нужно решить после работы с набором цифр, описанным в предыдущем разделе. В ходе решения этой задачи вам придется иметь дело с красочными, сложными изображениями. Если вы работали с примерами из главы 14, у вас уже есть некоторый опыт работы с изображениями на примере игрушечного набора данных Olivetti Faces.

Анализ обзоров Amazon.com

Если хотите работать с действительно большим набором данных, попробуйте набор данных обзоров Amazon.com по адресу `https://snap.stanford.edu/data/web-Amazon.html`. Этот набор состоит из обзоров Amazon.com, полученных за 18 лет, в том числе приблизительно 35 миллионов обзоров до марта 2013 года. Обзоры включают информацию о продукте и пользователе, рейтинги и обзор в виде простого текста. Это набор данных имеет смысл использовать после работы с меньшими наборами данных, такими как MovieLens. Это поможет вам понять, как работать с пользовательскими данными в бизнес-контексте.

В отличие от многих наборов данных, приведенных в этой главе, набор данных Amazon.com представлен в нескольких формах. Вы можете скачать файл `all.txt.gz`, чтобы получить весь набор данных (11 Гбайт), или загружать только части набора данных. Например, вы можете загрузить только 184 887 обзоров, связанных с детскими товарами, получив файл `Baby.txt.gz` (42 Мбайт).


СОВЕТ

Обязательно ознакомьтесь с нижней частью страницы. Владелец сайта тщательно предоставил код Python, необходимый для интерпретации данных. Использование этой простой функции значительно облегчает работу с огромным набором данных. Даже если вы решите создать модифицированную версию функции, у вас, по крайней мере, будет хорошая отправная точка.

Взаимодействие с огромным графом

Представьте себе попытку проработать связи между 3,5 миллиарда веб-страниц. Вы можете сделать это, загрузив огромный набор данных по адресу `https://www.bigdatanews.com/profiles/blogs/big-data-set-3-5-billion-web-pages-made-available-for-all-of-us`. Самым большим, богатым и сложным набором данных является сам Интернет. Начните

с подвыборки, предлагаемой веб-корпусом Common Crawl 2012 (https://commoncrawl.org/), и узнайте, как извлекать и обрабатывать данные с веб-сайтов. Принцип использованный для этого набора данных таков:

- Алгоритмы поиска
- Методы обнаружения спама
- Алгоритмы анализа графов
- Научные исследования в Веб

Обратите особое внимание на раздел Contents (Содержание) в середине страницы. Щелкнув на ссылке, вы попадете на страницу http://webdatacommons.org/hyperlinkgraph/, которая более подробно объясняет набор данных. Вам нужна дополнительная информация для выполнения большинства задач по науке о данных. В нижней части страницы находятся ссылки для загрузки различных уровней всего графа (к счастью, вам не нужно загружать все, что составляет 45 Гбайт файла индекса и 331 Гбайт файла архива).

Не позволяйте мысли о проведении анализа такого большого набора данных напугать вас. Если вы работали с примерами в главе 7, значит, работали с данными простых графов. Этот набор данных является аналогичной задачей, но в значительно большем масштабе. Да, размер имеет значение в некоторой степени, но вы уже знаете некоторые из необходимых методов для выполнения работы.


СОВЕТ

Этот конкретный сайт предоставляет доступ к ряду других наборов данных. Ссылки на эти наборы данных находятся внизу страницы. Например, вы можете найти публикацию "Great statistical analysis: forecasting meteorite hits" по адресу https://www.analyticbridge.com/profiles/blogs/greatstatistical-analysis-forecasting-meteorite-hits. Короче говоря, если анализ всего Интернета вам не подходит, попробуйте один из других поразительных (и огромных) наборов данных.

Предметный указатель

Шпаргалка

Краткое описание методов Scikit-learn

Scikit-learn — это центральный элемент для работы с данными на языке Python, поэтому стоит знать, какие методы вам больше всего нужны. В приведенной ниже таблице приведен краткий обзор наиболее важных методов, используемых для анализа данных.

Синтаксис	*Использование*	*Описание*
`model_selection.cross_val_score`	Этап перекрестной проверки	Оценка перекрестной проверки
`model_selection.KFold`	Этап перекрестной проверки	Разделяет набор данных на *k* блоков для перекрестной проверки
`model_selection.StratifiedKFold`	Этап перекрестной проверки	Стратифицированная проверка, учитывающая распределение прогнозируемых вами классов
`model_selection.train_test_split`	Этап перекрестной проверки	Разделяет данные на обучающие и тестовые наборы
`decomposition.PCA`	Уменьшение размерности	Анализ основных компонентов (PCA)
`decomposition.RandomizedPCA`	Уменьшение размерности	Анализ основных компонентов (PCA) с использованием случайного SVD
`feature_extraction.FeatureHasher`	Подготовка данных	Трюк хеширования, позволяющий разместить большое количество признаков в наборе данных
`feature_extraction.text.CountVectorizer`	Подготовка данных	Преобразование текстовых документов в матрицу данных
`feature_extraction.text.HashingVectorizer`	Подготовка данных	Непосредственно преобразует ваш текст, используя трюк хеширования
`feature_extraction.text.TfidfVectorizer`	Подготовка данных	Создает набор данных признаков TF-IDF
`feature_selection.RFECV`	Выбор признаков	Автоматический выбор признаков

BESTSELLING
BOOK
SERIES

Python и наука о данных для чайников®

2-е издание

СЕРИЯ
КОМПЬЮТЕРНЫХ
КНИГ ОТ
ДИАЛЕКТИКИ

Шпаргалка

Синтаксис	*Использование*	*Описание*
`model_selection.GridSearchCV`	Оптимизация	Исчерпывающий поиск для максимизации алгоритма машинного обучения
`linear_model.LinearRegression`	Прогнозирование	Линейная регрессия
`linear_model.LogisticRegression`	Прогнозирование	Линейная логистическая регрессия
`metrics.accuracy_score`	Оценка решения	Оценка точности классификации
`metrics.f1_score`	Оценка решения	Вычисляет оценки F1, балансируя точность и отзыв
`metrics.mean_absolute_error`	Оценка решения	Средняя абсолютная ошибка регрессии
`metrics.mean_squared_error`	Оценка решения	Среднеквадратическая ошибка регрессии
`metrics.roc_auc_score`	Оценка решения	Вычисляет площадь под кривой (AUC) из результатов прогноза
`naive_bayes.MultinomialNB`	Прогнозирование	Полиномиальный наивный байесовский классификатор
`neighbors.KNeighborsClassifier`	Прогнозирование	Классификация k-соседей
`preprocessing.Binarizer`	Подготовка данных	Создает двоичные переменные (значения признаков 0 или 1)
`preprocessing.Imputer`	Подготовка данных	Внедрение пропущенных значений
`preprocessing.MinMaxScaler`	Подготовка данных	Создает переменные, связанные с минимальным и максимальным значением
`preprocessing.OneHotEncoder`	Подготовка данных	Преобразует категориальные целочисленные признаки в двоичные
`preprocessing.StandardScaler`	Подготовка данных	Стандартизация переменных за счет удаления среднего значения и масштабирования до дисперсии единиц

Шпаргалка

Общие магические функции IPython

Кто бы мог подумать, что IPython предоставляет магию, но именно это вы получаете с помощью магических функций. Магическая функция начинается со знака % или %%. Функции со знаком % работают в среде, а со знаком %% — на уровне ячеек.

Обратите внимание, что магические функции лучше всего работают с Jupyter Notebook. Те, кто используют альтернативы, такие как Google Colab, могут обнаружить, что некоторые магические функции не могут обеспечить желаемый результат.

В следующем списке приведено несколько наиболее распространенных магических функций и их назначение. Чтобы получить полный список, введите **%quickref** и нажмите клавишу <Enter> в консоли IPython и ознакомьтесь с полным списком.

Магическая функция	*Отдельный ввод предоставляет статус?*	*Описание*
%%timeit	Нет	Вычисляет наилучшую временную производительность для всех инструкций в ячейке, кроме той, которая размещена в той же строке ячейки, что и магическая функция ячейки (которая, следовательно, может быть инструкцией инициализации)
%%writefile	Нет	Записывает содержимое ячейки в указанный файл
%alias	Да	Присваивает или отображает псевдоним для системной команды
%autocall	Да	Позволяет вызывать функции без включения скобок. Параметры **Off**, Smart (стандартно) и **Full**. Параметр Smart применяет скобки, только если вы включили аргумент в вызов
%automagic	Да	Позволяет вызывать магические функции, не включая знак %. Параметры: **False** (стандартно) и **True**
%cd	Да	Изменяет каталог нового места хранения. Вы также можете использовать эту команду для перемещения по истории каталогов или для помещения каталогов в закладки
%cls	Нет	Очищает экран
%colors	Нет	Задает цвета, используемые для отображения текста, связанного с подсказками, информационной системой и обработчиками исключений. Вы можете выбрать между **NoColor** (черный и белый), **Linux** (стандартно) и **LightBG**

Магическая функция	*Отдельный ввод предоставляет статус?*	*Описание*
`%config`	Да	Позволяет настроить IPython
`%dhist`	Да	Отображает список каталогов, посещенных во время текущего сеанса
`%file`	Нет	Выводит имя файла, содержащего исходный код объекта
`%hist`	Да	Отображает список команд магических функций, выполненных во время текущего сеанса
`%install_ext`	Нет	Устанавливает указанное расширение
`%load`	Нет	Загружает код приложения из другого источника, например, сетевого примера
`%load_ext`	Нет	Загружает расширение Python, используя имя его модуля
`%lsmagic`	Да	Отображает список доступных в настоящее время магических функций
`%matplotlib`	Да	Устанавливает внутренний процессор, используемый для графиков. Использование встроенного значения отображает график в ячейке для файла IPython Notebook. Возможные значения: `'gtk'`, `'gtk3'`, `'inline'`, `'nbagg'`, `'osx'`, `'qt'`, `'qt4'`, `'qt5'`, `'tk'` и `'wx'`
`%paste`	Нет	Вставляет содержимое буфера обмена в среду IPython
`%pdef`	Нет	Показывает, как вызвать объект (при условии, что объект может быть вызван)
`%pdoc`	Нет	Отображает строку документации для объекта
`%pinfo`	Нет	Отображает подробную информацию об объекте (зачастую больше, чем предоставлено только справкой)
`%pinfo2`	Нет	Отображает дополнительную подробную информацию об объекте (при наличии)
`%reload_ext`	Нет	Перезагружает установленное ранее расширение
`%source`	Нет	Отображает исходный код объекта (при условии, что источник доступен)
`%timeit`	Нет	Рассчитывает лучшее время выполнения для инструкции
`%unalias`	Нет	Удаляет ранее созданный псевдоним из списка
`%unload_ext`	Нет	Выгружает указанное расширение

BESTSELLING BOOK SERIES

Python и наука о данных для чайников®

2-е издание

СЕРИЯ КОМПЬЮТЕРНЫХ КНИГ ОТ ДИАЛЕКТИКИ

Шпаргалка

Стили линейных графиков

Всякий раз, когда вы создаете график, вам нужно идентифицировать источники информации, используя не только линии. Создание графика, в котором используются разные типы линий и символы точек данных, значительно упрощают его использование другими людьми. В следующей таблице перечислены стили линейного графика.

Код цвета	Цвет линии	Код маркера	Стиль маркера	Код линии	Стиль линии
b	Синий	.	Точка	—	Сплошная
g	Зеленый	o	Круг	:	Пунктирная
r	Красный	x	Метка x	-.	Штрихпунктирная
c	Голубой	+	Плюс	—	Штриховая
m	Фиолетовый	*	Звезда	(нет)	нет линии
y	Желтый	s	Квадрат		
k	Черный	d	Ромб		
w	Белый	v	Треугольник вниз		
		^	Треугольник вверх		
		<	Треугольник влево		
		>	Треугольник вправо		
		p	5-конечная звезда		
		h	6-конечная звезда		

СОВЕТ

Помните, что можно также использовать эти стили с другими видами графиков. Например, в точечной диаграмме эти стили могут использоваться для определения каждой из точек данных. Если сомневаетесь, опробуйте стили, чтобы увидеть, будут ли они работать с вашим конкретным графиком.

BESTSELLING
BOOK
SERIES

Python и наука о данных для чайников®

2-е издание

СЕРИЯ
КОМПЬЮТЕРНЫХ
КНИГ ОТ
ДИАЛЕКТИКИ

Шпаргалка

Восемь самых распространенных ошибок программирования на языке Python.

Каждый разработчик на планете делает ошибки. Однако знание о самых распространенных ошибках сэкономит вам время и силы в будущем. Ниже приведены наиболее распространенные ошибки, с которыми сталкиваются разработчики при работе с языком Python.

- **Неправильный отступ**. Многие функции Python зависят от отступов. Например, когда вы создаете новый класс, все в этом классе имеет отступ под объявлением класса. То же самое верно для решения, цикла и других структурных выражений. Если вы обнаружите, что код выполняет задачу, когда этого не должно быть, проверьте отступ.
- **Оператор присваивания вместо оператора равенства**. При сравнении двух объектов или значений вы используете оператор равенства (==), а не оператор присваивания (=). Оператор присваивания помещает объект или значение в переменную и ничего не сравнивает.
- **Расположение вызовов функций в неправильном порядке при создании сложных операторов**. Язык Python всегда выполняет функции слева направо. Таким образом, оператор `MyString.strip().center(21,"*")` дает результат, отличный от `MyString.center(21,"*").strip()`. Если вы сталкиваетесь с ситуацией, в которой вывод ряда объединенных функций отличается от ожидаемого, проверьте порядок функций, чтобы убедиться, что каждая функция находится в правильном месте.
- **Неправильная пунктуация**. Вы можете поставить пунктуацию не в том месте и получить совершенно другой результат. Помните, что вы должны ставить двоеточие в конце каждого структурного выражения. Кроме того, расположение скобок имеет решающее значение. Например, `(1 + 2) * (3 + 4)`, `1 + ((2 * 3) + 4)` и `1 + (2 * (3 + 4))` дают разные результаты.
- **Использование неверного логического оператора**. Большинство операторов не создают проблем разработчикам, но не логические операторы. Не забудьте использовать `and` для определения, когда оба операнда должны иметь значение `True`, и `or`, когда любой из операндов может быть `True`.
- **Ошибка отсчета с единицы в циклах**. Помните, что в цикле не учитывается последний номер, указанный вами в диапазоне. Таким образом, если вы укажете диапазон `[1:11]`, то получите выходные значения от 1 до 10.
- **Неправильной регистр**. Python чувствителен к регистру, поэтому `MyVar` отличается от `myvar` и `MYVAR`. Всегда проверяйте регистр символов, когда не можете получить доступ к нужному значению.
- **Орфографические ошибки**. Даже опытные разработчики время от времени страдают от орфографических ошибок. Помогает использование общего подхода к именованию переменных, классов и функций. Но даже последовательная схема именования не всегда препятствует набрать `MyVer` вместо `MyVar`.

Наука о данных — это вовсе не страшно!

Интересуетесь наукой о данных, но немного побаиваетесь? Не нужно! Эта книга покажет, как использовать язык Python для создания интересных вещей с помощью науки о данных. Вы увидите, как установить набор инструментов Anaconda, благодаря которому работа с Python станет очень простой. Здесь вы откроете для себя инструмент Google Colab, позволяющий писать код в облаке с помощью обычного планшета. Вы узнаете, как выполнять все виды вычислений, используя последнюю версию языка Python. Вы также научитесь использовать различные библиотеки, обеспечивающие научный статистический анализ, построение диаграмм, графиков и многое другое.

В книге...

- Настройка Python для науки о данных
- Работа с Jupyter Notebook
- Сбор и формирование данных
- Графика с использованием MatPlotLib
- Способы анализа данных
- Как получить больше от Python
- Полезные алгоритмы науки о данных
- Десять важных ресурсов данных

Джон Пол Мюллер — внештатный автор и технический редактор, автор более 100 книг по таким темам, как работа с сетями, домашняя безопасность, управление базами данных и программирование. Блог Джона находится по адресу `http://blog.johnmuellerbooks.com/`.

Лука Массарон — аналитик данных, специализирующийся на организации и интерпретации больших данных и их преобразовании в интеллектуальные данные. Является экспертом Google Developer Expert (GDE) в области машинного обучения.

Диалектика
www.dialektika.com

Изображение на обложке:
©Depositphotos.com/18398501
Автор: agsandrew

Python/наука о данных

ISBN 978-5-907203-47-1

для чайников®