Дэви Силен, Арно Мейсман, Мохамед Али

Основы Data Science и Big Data

Python и наука о данных

MANNING

Introducing Data Science

BIG DATA, MACHINE LEARNING, AND MORE, USING PYTHON TOOLS

DAVY CIELEN
ARNO D. B. MEYSMAN
MOHAMED ALI

MANNING
SHELTER ISLAND

Дэви Силен, Арно Мейсман, Мохамед Али

Основы Data Science и Big Data

Python и наука о данных

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

2017

ББК 32.973.233
УДК 004.62
С36

Силен Дэви, Мейсман Арно, Али Мохамед

С36 Основы Data Science и Big Data. Python и наука о данных. — СПб.: Питер, 2017. — 336 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-496-02517-1

Data Science — это совокупность понятий и методов, позволяющих придать смысл и понятный вид огромным объемам данных.

Каждая из глав этой книги посвящена одному из самых интересных аспектов анализа и обработки данных. Вы начнете с теоретических основ, затем перейдете к алгоритмам машинного обучения, работе с огромными массивами данных, NoSQL, потоковым данным, глубокому анализу текстов и визуализации информации. В многочисленных практических примерах использованы сценарии Python.

Обработка и анализ данных — одна из самых горячих областей IT, где постоянно требуются разработчики, которым по плечу проекты любого уровня, от социальных сетей до обучаемых систем. Надеемся, книга станет отправной точкой для вашего путешествия в увлекательный мир Data Science.

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

ББК 32.973.233
УДК 004.62

ISBN 978-1633430037 англ.
ISBN 978-5-496-02517-1

Оглавление

Предисловие

Анализировать данные умеют все люди. Способность нашего мозга видеть взаимосвязи, приходить к выводам на основании фактов и учиться на опыте — вот что делает человека человеком. Выживание человека в большей степени, чем любого другого биологического вида на планете, зависит от мозга; человечество сделало максимальную ставку на эту особенность, чтобы занять свое место в природе. Пока эта стратегия работает, и вряд ли мы захотим ее поменять в ближайшем будущем.

Однако в том, что касается тривиальной обработки чисел, возможности нашего мозга ограниченны. Он не справляется с объемом данных, который мы в состоянии воспринять за один раз, и с нашей любознательностью. По этой причине мы доверяем машинам часть своей работы: выявление закономерностей, формирование связей и получение ответов на многочисленные вопросы.

Стремление к знаниям заложено в наших генах. Применение компьютеров для выполнения части работы в наши гены не заложено, но без них не обойтись.

Благодарности

Огромное спасибо всем сотрудникам издательства Manning, которые участвовали в работе над книгой, за помощь и поддержку.

Мы благодарны Равишанкару Раджагопалану (Ravishankar Rajagopalan) за полное научное рецензирование рукописи, а также Джонатану Томсу (Jonathan Thoms) и Майклу Робертсу (Michael Roberts) за профессиональные замечания. Также в процессе рецензирования участвовали многие другие специалисты, высказавшие свое бесценное мнение: Элвин Радж (Alvin Raj), Артур Зубарев (Arthur Zubarev), Билл Марченко (Bill Martschenko), Крейг Смит (Craig Smith), Филип Правика (Filip Pravica), Хамиде Ирадж (Hamideh Iraj), Хизер Кэмпбелл (Heather Campbell), Гектор Куэста (Hector Cuesta), Иэн Стирк (Ian Stirk), Джефф Смит (Jeff Smith), Джоэл Котарски (Joel Kotarski), Джонатан Шарли (Jonathan Sharley), Джорн Динкла (Jörn Dinkla), Мариус Бутук (Marius Butuc), Мэтт Р. Коул (Matt R. Cole), Мэтью Хек (Matthew Heck), Мередит Годар (Meredith Godar), Роб Эгл (Rob Agle), Скотт Чосси (Scott Chaussee) и Стив Роджерс (Steve Rogers).

Прежде всего я хочу поблагодарить свою жену Филипу, которая вдохнула в меня решимость и помогла справиться со всеми трудностями, а также постоянно была рядом со мной на протяжении всей моей карьеры и во время работы над книгой. Она предоставила мне время, необходимое для реализации моих целей и амбиций, и взяла на себя все заботы по уходу за нашей маленькой дочерью. Я посвящаю ей эту книгу и в полной мере понимаю, на какие жертвы ей приходится идти для создания и поддержания нашей маленькой семьи.

Также хочу поблагодарить свою дочь Еву и моего еще не родившегося сына за чувство огромного счастья и мои улыбки. Это лучшие подарки, которые я мог получить от жизни, и лучшие из всех детей: веселые, любящие и дарящие радость.

Я особенно благодарен родителям, помогавшим мне все эти годы. Без их бесконечной любви и поддержки я не смог бы дописать эту книгу и продолжить свой жизненный путь.

Спасибо всем коллегам из моей компании, особенно Мо и Арно, за все наши совместные приключения. Мо и Арно очень помогали мне и давали полезные советы. Благодарю их за все время и усилия, потраченные ими на завершение книги. Это замечательные люди, и без них я вряд ли справился бы с книгой.

Наконец, я искренне благодарен всем друзьями, которые поддерживали меня и понимали, что у меня не хватает времени на них. Надеюсь, они по-прежнему будут любить меня и поддерживать, как это было на протяжении всей моей карьеры и во время написания книги.

Дэви Силен

Спасибо моей семье и друзьям, поддерживавшим меня во время написания книги. Мне не всегда легко было сидеть и работать, когда можно было выйти из дома и узнать что-то новое. Я особенно благодарен моим родителям, моему брату Яго и подруге Дельфине за то, что они всегда были рядом, какие бы безумные планы ни приходили мне в голову.

Спасибо моей крестной и моему крестному — его борьба с раком заставила меня вспомнить об истинных жизненных ценностях.

Спасибо друзьям, поившим меня пивом, чтобы немного отвлечь меня от работы, и родителям Дельфины, ее брату Карелу и его будущей жене Тэсс за их гостеприимство (и превосходные блюда, которыми меня угощали). Все они сделали мою жизнь намного лучше.

Наконец, я хочу поблагодарить моих соавторов Мо и Дэви за творческий вклад в книгу. Я разделяю с ними все повседневные успехи и неудачи как их коллега-предприниматель и специалист data science. Наша совместная работа была просто замечательной. Надеюсь, впереди у нас еще немало таких же дней.

Арно Д. Б. Мейсман

В первую очередь я благодарю свою невесту Мухубу за ее любовь, понимание, заботу и терпение. Также я многим обязан Дэви и Арно: во-первых, с ними было интересно, а во-вторых, они помогли мне осуществить мою предпринимательскую мечту. Их неизменная преданность делу стала жизненно важным фактором для создания книги.

Мохамед Али

О книге

Но я могу лишь указать дверь.
Пройти через нее должен ты сам.

Морфеус, «Матрица»

Добро пожаловать! Вероятно, при просмотре оглавления вы заметили, насколько разнообразные темы будут рассматриваться в книге. Мы постараемся дать немного информации обо всем сразу — достаточно, чтобы вы смогли действовать самостоятельно. Data science — чрезвычайно обширная область; настолько обширная, что изложить ее полностью не удалось бы даже в книге вдесятеро большего объема. Для каждой главы был выбран некоторый аспект, кажущийся нам интересным. Нам пришлось принять несколько трудных решений, чтобы ваша книжная полка не обрушилась под тяжестью этой книги!

Надеемся, книга станет отправной точкой для вашего путешествия — вашей дверью в увлекательный мир data science.

Структура книги

В главах 1 и 2 приводятся общие теоретические основы, необходимые для понимания других глав книги:

- Глава 1 знакомит читателя с data science и большими данными. Она завершается практическим примером Hadoop.
- Глава 2 посвящена процессу data science. В ней описаны шаги, присутствующие почти в каждом проекте data science.

В главах 3–5 описано применение принципов машинного обучения к наборам данных постепенно увеличивающихся размеров:

- В главе 3 рассматриваются относительно небольшие данные, легко помещающиеся в памяти среднего компьютера.
- В главе 4 задача усложняется: в ней рассматриваются «большие данные», которые могут храниться на вашем компьютере, но не помещаются в памяти, вследствие чего обработка таких данных без вычислительного кластера создает проблемы.

- В главе 5 мы наконец-то добираемся до настоящих больших данных, с которыми невозможно работать без многих компьютеров.

В главах 6–9 рассматриваются некоторые интересные вопросы data science, более или менее независимые друг от друга:

- В главе 6 рассматривается архитектура NoSQL и ее отличие от реляционных баз данных.
- В главе 7 data science применяется к потоковым данным. Здесь основная проблема связана не с размером, а со скоростью генерирования данных и потерей актуальности старых данных.
- Глава 8 посвящена глубокому анализу текста. Не все данные существуют в числовой форме. Глубокий анализ и аналитика текста начинают играть важную роль в текстовых форматах: электронной почте, блогах, контенте веб-сайтов и т. д.
- В главе 9 основное внимание уделяется последней части процесса data science (визуализации данных и построению прототипа приложения), для чего мы рассмотрим ряд полезных инструментов HTML5.

В приложениях А–Г рассматриваются процедуры установки и настройки систем Elasticsearch, Neo4j и MySQL, упоминаемых в главах книги, а также Anaconda — программного пакета Python, чрезвычайно полезного в data science.

Для кого написана эта книга

Эта книга знакомит читателя с областью data science. Опытные специалисты data science поймут, что по некоторым темам материал изложен в лучшем случае поверхностно. Другим читателям сообщим, что для извлечения максимальной пользы из книги потребуются некоторые предварительные условия: чтобы браться за практические примеры, желательно обладать хотя бы минимальными познаниями в SQL, Python, HTML5 и статистике или машинном обучении.

Условные обозначения и загружаемые файлы

В практических примерах мы решили использовать сценарии Python. За прошедшее десятилетие язык Python стал авторитетным и популярным языком в области data science.

Для обозначения кода используется моноширинный шрифт, многие листинги снабжены комментариями.

Материал книги поясняется многочисленными примерами. Многие из этих примеров включены в кодовую базу, которую можно загрузить на сайте книги: *https://www.manning.com/books/introducing-data-science.*

Об авторах

Дэви Силен — опытный предприниматель, автор книг и профессор. Вместе с Арно и Мо он является совладельцем *Optimately* и *Maiton* — двух компаний data science, базирующихся в Бельгии и Великобритании соответственно, а также одним из совладельцев еще одной компании data science в Сомалиленде. Все эти компании специализируются на стратегической обработке «больших данных»; многие крупные компании время от времени обращаются к ним за консультациями. Дэви является внештатным преподавателем школы менеджмента IESEG в Лилле (Франция), где он преподает и участвует в исследованиях в области теории «больших данных».

Арно Мейсман — целеустремленный предприниматель и специалист data science. Вместе с Дэви и Мо он является совладельцем Optimately и Maiton — двух компаний data science, базирующихся в Бельгии и Великобритании соответственно, а также одним из совладельцев еще одной компании data science в Сомалиленде. Все эти компании специализируются на стратегической обработке «больших данных»; многие крупные компании время от времени обращаются к ним за консультациями. Арно — специалист data science с широким кругом интересов, от розничной торговли до игровой аналитики. Он полагает, что информация, полученная в результате обработки данных, в сочетании с некоторым воображением, поможет нам улучшить этот мир.

Мохамед Али — предприниматель и консультант в области data science. Вместе с Арно и Мо он является совладельцем Optimately и Maiton — двух компаний data science, базирующихся в Бельгии и Великобритании соответственно. Его увлечения лежат в двух областях: data science и экологически рациональные проекты. Последнее направление воплотилось в создании третьей компании, базирующейся в Сомалиленде.

От издательства

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

Мы будем рады узнать ваше мнение!

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

1 Data science в мире больших данных

В этой главе:

- ✓ Определение data science и больших данных.
- ✓ Разные типы данных.
- ✓ Глубокий анализ процесса data science.
- ✓ Введение в область data science и больших данных.
- ✓ Работа с примерами использования Hadoop.

Под обобщающим термином «большие данные» принято понимать любые наборы данных, достаточно большие и сложные для того, чтобы их можно было обработать традиционными средствами работы с данными (например, РСУБД — реляционными системами управления базами данных). Широко распространенные РСУБД давно считаются универсальным инструментом, но спрос на обработку больших данных показывает иное. В концепцию data science входит использование методов анализа огромных объемов данных и извлечения содержащейся в них информации. Связь между большими данными и data science такая же, как между сырой нефтью и нефтеперерабатывающим заводом. Data science и большие данные развивались на базе статистики и традиционного управления данными, но сейчас считаются разными дисциплинами.

Характеристики больших данных часто называются «тремя V»:

- Объем (*Volume*) — сколько данных содержит набор?
- Разнообразие (*Variety*) — насколько отличаются друг от друга разные типы данных?
- Скорость (*Velocity*) — с какой скоростью генерируются новые данные?

Часто эти характеристики дополняются «четвертым V» — достоверностью (*Veracity*): насколько точны данные? Эти четыре свойства отличают большие дан-

ные от данных, встречающихся в традиционных средствах управления данными. Соответственно, привносимые ими изменения проявляются почти во всех аспектах: сборе данных, хранении и обслуживании данных, поиске, обмене, передаче и визуализации. Кроме того, большие данные требуют применения специализированных средств извлечения информации.

Data science — это расширение статистики, способное справляться с огромными объемами данных, производимыми в наши дни. Data science добавляет методы из computer science в репертуар статистики. В аналитической заметке Лейни и Карта «Emerging Role of the Data Scientist and the Art of Data Science» были проработаны сотни описаний рабочих обязанностей специалистов data science, статистиков и аналитиков бизнес-данных, чтобы выявить различия между этими должностями. Главное, что отличает специалиста data science от статистика, — это умение работать с большими данными и подготовка в области машинного обучения, организации вычислений и построения алгоритмов. Их инструментарии обычно тоже различаются; в описаниях работы специалистов data science чаще упоминается умение использовать Hadoop, Pig, Spark, R, Python и Java (среди прочего). Не огорчайтесь, если вас устрашило это перечисление; многие пункты будут постепенно раскрыты в этой книге, хотя основное внимание будет уделено Python. Python — замечательный язык для data science, потому что для него написано много библиотек data science и он широко поддерживается специализированными программами. Например, почти в каждой популярной базе данных NoSL существует программный интерфейс (API) для Python. Из-за этих особенностей, а также из-за возможности быстро строить прототипы на языке Python с сохранением приемлемой производительности его популярность в мире data science быстро растет.

По мере роста объема данных и потребностей в их использовании каждый специалист data science будет сталкиваться с проектами из области больших данных на протяжении своей карьеры.

1.1. Область применения data science и больших данных и их преимущества

Data science и большие данные сейчас встречаются почти повсеместно как в коммерческих, так и в некоммерческих средах. Количество потенциальных применений огромно, и примеры, приведенные в книге, едва ли дают даже поверхностное представление об их возможностях.

Коммерческие компании почти во всех промышленных областях используют data science и большие данные для получения информации о клиентах, процессах, персонале, конкурентах и товарах. Многие компании применяют data science, чтобы произвести хорошее впечатление на пользователя, для организации перекрестных и дополнительных продаж, а также для персонализации предложений. Хорошим примером служит сервис Google AdSense, собирающий информацию об интернет-пользователях для показа контекстной рекламы. MaxPoint

(*http://maxpoint.com/us*) — другой пример персонализированной рекламы в реальном времени. Профессионалы в области подбора кадров используют личностную аналитику (people analytics) и глубокий анализ текста для отбора кандидатов, отслеживания настроения работников и изучения неформальных связей среди коллег. People analytics является центральной темой книги «Moneyball: The Art of Winning an Unfair Game». В книге (и фильме[1]) показано, что традиционный процесс отбора спортсменов в бейсболе был случайным, и его замена коррелированными признаками изменила всё. Применение статистики позволило нанимать правильных игроков и выставлять их против тех соперников, над которыми возможно наибольшее преимущество. Финансовые учреждения используют data science для прогнозирования рынка ценных бумаг, вычисления риска предоставления ссуд и привлечения новых клиентов. На момент написания книги по меньшей мере 50% торговых сделок по всему миру выполнялось автоматически на основании алгоритмов, разработанных специалистами data science, с использованием больших данных и методов data science.

Правительственные организации также хорошо осведомлены о ценности данных. Многие правительственные организации не только используют собственных аналитиков для поиска ценной информации, но и выкладывают свои результаты в открытый доступ. Данные могут использоваться для глубокого анализа или построения приложений, управляемых данными. *Data.gov*, «дом» открытых данных правительства США, — всего лишь один пример. Специалисты data science в правительственных организациях работают над самыми разнообразными проектами, от выявления случаев мошенничества и других видов преступной деятельности до оптимизации финансирования проектов. Хорошо известный пример такого рода был предоставлен Эдвардом Сноуденом, опубликовавшим внутренние документы Агентства национальной безопасности США и Британского центра правительственной связи, из которых явно следует, что data science и большие данные применялись для слежения за миллионами граждан. Эти организации собрали 5 миллиардов записей данных из таких распространенных приложений и сервисов, как Google Maps, Angry Birds, электронная почта и текстовые сообщения (среди прочего). Затем методы data science были применены для фильтрации информации.

Неправительственные организации (НПО) тоже имеют опыт использования данных. Они используют эти методы для привлечения средств и отстаивания своих целей. Например, Международный фонд защиты природы пользуется услугами специалистов data science для повышения эффективности своих проектов по привлечению средств. Многие специалисты data science выделяют часть своего времени на помощь НПО, потому что у НПО часто не хватает ресурсов для того, чтобы собрать данные и нанять специалистов data science. DataKind — одна из таких групп, тратящих свое время на благо человечества.

[1] В российском прокате фильм шел под названием «Человек, который изменил всё». — *Примеч. пер.*

Университеты используют data science в своих исследованиях и для повышения качества учебного процесса. Бурное развитие массовых открытых дистанционных курсов (MOOC, Massive Open Online Course) породило большой объем данных, на основании которых университеты могут изучать, как этот тип обучения дополняет традиционные программы. MOOC — бесценный ресурс для людей, стремящихся стать профессионалами в области data science и больших данных, поэтому вам определенно стоит познакомиться с некоторыми из наиболее известных предложений: Coursera, Udacity и edX. Ситуация с большими данными и data science быстро меняется, и MOOCs позволяют следить за событиями и участвовать в учебных курсах ведущих университетов. Если вы еще не знакомы с курсами MOOC, не жалейте времени; они понравятся вам так же, как понравились нам.

1.2. Грани данных

В data science и области больших данных встречается много разных типов данных, для каждого из которых требуются свои инструменты и методы. Основные категории данных перечислены ниже.

- Структурированные.
- Неструктурированные.
- На естественном языке.
- Машинные.
- Графовые.
- Аудио, видео и графика.
- Потоковые.

Все эти типы данных представляют интерес, и их стоит рассмотреть подробнее.

1.2.1. Структурированные данные

Структурированные данные зависят от модели данных и хранятся в фиксированном поле внутри записи. Соответственно, структурированные данные часто бывает удобно хранить в таблицах, в базах данных или файлах Excel (рис. 1.1). SQL (Structured Query Language, язык структурированных запросов) является основным средством управления и обращения с запросами к данным, хранящимся в базах данных. Также иногда встречаются структурированные данные, которые достаточно трудно сохранить в традиционной реляционной базе данных (один из примеров — иерархические данные, например генеалогическое дерево).

Впрочем, мир не состоит из структурированных данных; просто это представление удобно для человека и машин. Чаще реальные данные хранятся в неструктурированном виде.

	Indicator ID	Dimension List	Timeframe	Numeric Value	Missing Value Flag	Confidence Inte
2	214390830	Total (Age-adjusted)	2008	74.6%		73.8%
3	214390833	Aged 18-44 years	2008	59.4%		58.0%
4	214390831	Aged 18-24 years	2008	37.4%		34.6%
5	214390832	Aged 25-44 years	2008	66.9%		65.5%
6	214390836	Aged 45-64 years	2008	88.6%		87.7%
7	214390834	Aged 45-54 years	2008	86.3%		85.1%
8	214390835	Aged 55-64 years	2008	91.5%		90.4%
9	214390840	Aged 65 years and over	2008	94.6%		93.8%
10	214390837	Aged 65-74 years	2008	93.6%		92.4%
11	214390838	Aged 75-84 years	2008	95.6%		94.4%
12	214390839	Aged 85 years and over	2008	96.0%		94.0%
13	214390841	Male (Age-adjusted)	2008	72.2%		71.1%
14	214390842	Female (Age-adjusted)	2008	76.8%		75.9%
15	214390843	White only (Age-adjusted)	2008	73.8%		72.9%
16	214390844	Black or African American only (Age-adjusted)	2008	77.0%		75.0%
17	214390845	American Indian or Alaska Native only (Age-adjusted)	2008	66.5%		57.1%
18	214390846	Asian only (Age-adjusted)	2008	80.5%		77.7%
19	214390847	Native Hawaiian or Other Pacific Islander only (Age-adjusted)	2008	DSU		
20	214390848	2 or more races (Age-adjusted)	2008	75.6%		69.6%

Рис. 1.1. Таблица Excel как пример структурированных данных

1.2.2. Неструктурированные данные

Неструктурированные данные трудно подогнать под конкретную модель данных, потому что их содержимое зависит от контекста или имеет переменный характер. Один из примеров неструктурированных данных — обычные сообщения электронной почты (рис. 1.2). Хотя сообщение содержит структурированные элементы (отправитель, заголовок, тело), одни и те же задачи могут решаться множеством разных способов, например, существует бесчисленное количество вариантов упоминания конкретного человека в сообщениях. Проблема дополнительно усложняется существованием тысяч языков и диалектов.

Сообщение электронной почты, написанное человеком (наподобие показанного на рис. 1.2), также является идеальным примером данных на естественном языке.

1.2.3. Данные на естественном языке

Данные на естественном языке составляют особую разновидность неструктурированных данных; обработка таких данных достаточно сложна, потому что она требует знания как лингвистики, так и специальных методов data science.

Сообщество обработки данных на естественном языке добилось успеха в области распознавания сущностей, распознавания тематических областей, обобщения, завершения текста и анализа эмоциональной окраски, но модели, адаптированные для одной предметной области, плохо обобщаются для других областей. Даже самые современные методы не смогут расшифровать смысл произвольного фрагмента текста. И этот факт вряд ли кого-то удивит: у людей также возникают проблемы с восприятием естественного языка. Он неоднозначен по своей природе. Сама концепция смысла выглядит спорно. Два человека слушают один разговор; вынесут

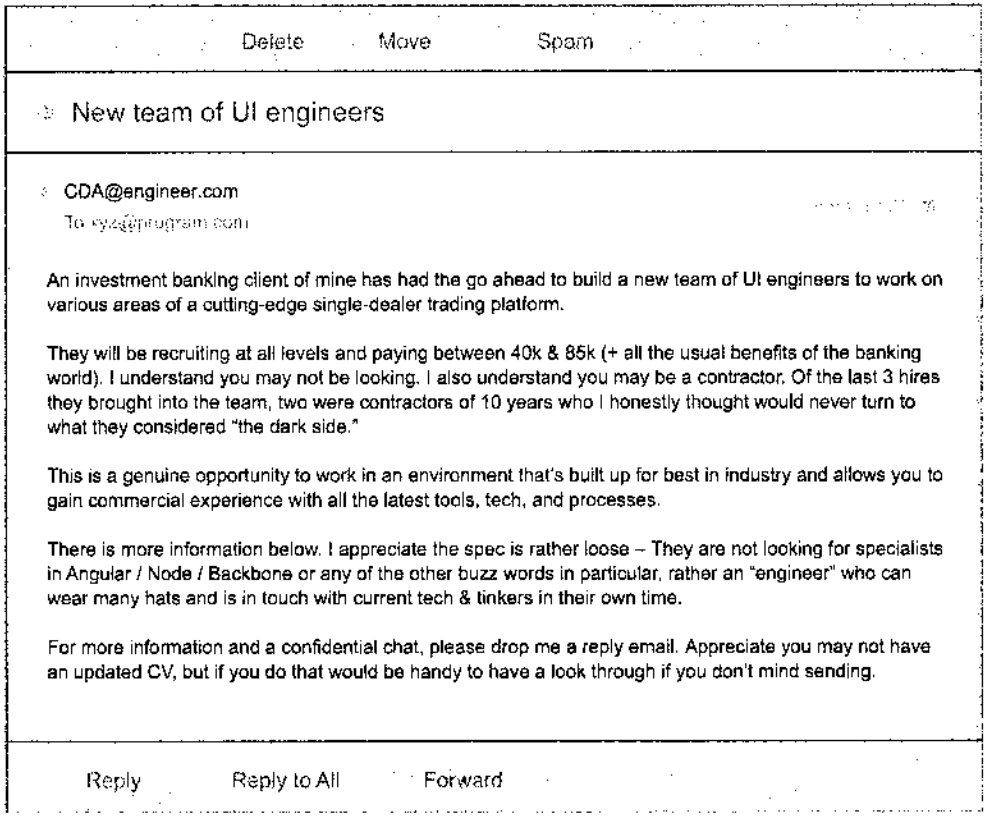


Рис. 1.2. Сообщение электронной почты одновременно является примером неструктурированных данных и данных на естественном языке

ли они одинаковый смысл из него? Даже смысл отдельных слов может изменяться в зависимости от настроения говорящего.

1.2.4. Машинные данные

К машинным данным относится информация, автоматически генерируемая компьютером, процессом, приложением или устройством без вмешательства человека. Машинные данные становятся одним из основных источников информации, и ситуация вряд ли изменится. Wikibon предсказывает, что рыночная стоимость промышленного Интернета (термин, предложенный компанией Frost&Sullivan для обозначения совокупности сложного физического оборудования с сетевыми датчиками и программным обеспечением) к 2020 году составит приблизительно 540 миллиардов долларов. По оценкам IDC (International Data Corporation), количество узлов сети к 2020 году в 26 раз превысит численность населения. Эта сеть часто называется *Интернетом вещей*.

Анализ машинных данных из-за их громадных объемов и скоростей сильно зависит от инструментов с высокой масштабируемостью. К примерам машинных данных относятся журналы веб-серверов, записи детализации звонков, журналы сетевых событий и телеметрии (рис. 1.3.)

```
CSIPERF:TXCOMMIT;313236
2014-11-28 11:36:13, Info                  CSI    00000153 Creating NT transaction (seq
69), objectname [6]"(null)"
2014-11-28 11:36:13, Info                  CSI    00000154 Created NT transaction (seq 69)
result 0x00000000, handle @0x4e54
2014-11-28 11:36:13, Info                  CSI    00000155@2014/11/28:10:36:13.471
Beginning NT transaction commit...
2014-11-28 11:36:13, Info                  CSI    00000156@2014/11/28:10:36:13.705 CSI perf
trace:
CSIPERF:TXCOMMIT;273993
2014-11-28 11:36:13, Info                  CSI    00000157 Creating NT transaction (seq
70), objectname [6]"(null)"
2014-11-28 11:36:13, Info                  CSI    00000158 Created NT transaction (seq 70)
result 0x00000000, handle @0x4e5c
2014-11-28 11:36:13, Info                  CSI    00000159@2014/11/28:10:36:13.764
Beginning NT transaction commit...
2014-11-28 11:36:14, Info                  CSI    0000015a@2014/11/28:10:36:14.094 CSI perf
trace:
CSIPERF:TXCOMMIT;386259
2014-11-28 11:36:14, Info                  CSI    0000015b Creating NT transaction (seq
71), objectname [6]"(null)"
2014-11-28 11:36:14, Info                  CSI    0000015c Created NT transaction (seq 71)
result 0x00000000, handle @0x4e5c
2014-11-28 11:36:14, Info                  CSI    0000015d@2014/11/28:10:36:14.106
Beginning NT transaction commit...
2014-11-28 11:36:14, Info                  CSI    0000015e@2014/11/28:10:36:14.428 CSI perf
trace:
CSIPERF:TXCOMMIT;375591
```

Рис. 1.3. Пример машинных данных

Машинные данные на рис. 1.3 хорошо укладываются в структуру классической базы данных. Это не лучший формат для данных с высокой степенью связности или «сетевых» данных, в которых достаточно значимую роль играют отношения между сущностями.

1.2.5. Графовые, или сетевые, данные

Термин «графовые данные» может сбить с толку, потому что любые данные могут быть представлены в виде графа. Под «графом» в данном случае имеется в виду понятие графа из математической теории графов — математическая структура для моделирования попарных отношений между объектами. Вкратце, в графовых, или сетевых, данных особое внимание уделяется связям или смежности объектов. Графовые структуры данных используют узлы, ребра и свойства для представления и хранения графических данных. Графовые данные естественным образом подходят для представления социальных сетей, а их структура позволяет вычислять такие специфические метрики, как влияние участников и кратчайший путь между двумя людьми.

Примеры графовых данных встречаются на многих веб-сайтах социальных сетей (рис. 1.4). Например, в LinkedIn можно увидеть, кого вы знаете в той или иной компании. Ваш список читателей в Твиттере также является примером графовых данных. Сила и мощь связанных данных проявляется при анализе нескольких перекрывающихся графов, построенных на одних и тех же узлах. Например, представьте, что ребра обозначают «друзей» на Facebook. А теперь возьмем другой граф с теми же людьми, но связывающий коллег по бизнесу через LinkedIn, и третий граф, основанный на интересе к фильмам на Netflix. Наложение этих трех графов позволит получить ответы на многие интересные вопросы.

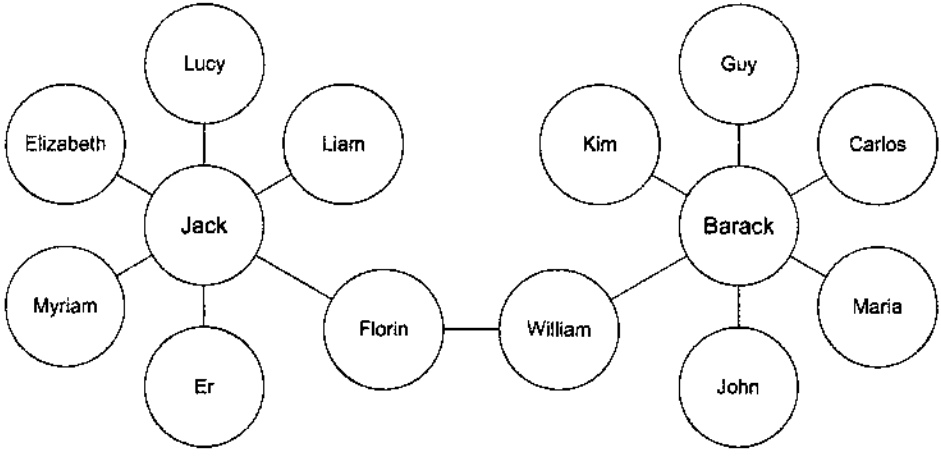


Рис. 1.4. Дружеские отношения в социальных сетях — пример графовых данных

Для хранения графовых данных используются графовые базы данных, а для построения запросов к ним — такие специализированные языки запросов, как SPARQL.

Работа с графовыми данными создает специфические проблемы, причем для компьютера эта задача становится еще сложнее.

1.2.6. Аудио, видео и графика

Аудио, видео и графика — типы данных, ставящие непростые задачи перед специалистом data science. Задачи, тривиальные с точки зрения человека (например, распознавание объекта на картинке), оказываются сложными для компьютера. В 2014 году компания MLBAM (Major League Baseball Advanced Media) объявила, что объем записываемых видеоматериалов для одного бейсбольного матча будет увеличен приблизительно до 7 Тбайт с целью проведения оперативного анализа. Высокоскоростные камеры на стадионах записывают движения мяча и спортсменов

для того, например, чтобы вычислять в реальном времени траекторию движения защитника.

Недавно компании DeepMind удалось создать алгоритм, который способен обучаться играть в видеоигры. Алгоритм получает на входе содержимое экрана и учится интерпретировать эти данные в сложном процессе *глубокого обучения*. Это замечательное достижение, и компания Google приобрела DeepMind для разработки искусственного интеллекта. Алгоритм обучения получает данные, генерируемые компьютерной игрой, т. е. потоковые данные.

1.2.7. Потоковые данные

Потоковые данные могут принимать почти любую из перечисленных форм, однако у них имеется одно дополнительное свойство. Данные поступают в систему при возникновении некоторых событий, а не загружаются в хранилище данных большими массивами. И хотя формально они не являются отдельной разновидностью данных, мы выделяем их в особую категорию, потому что вам придется приспособить свой рабочий процесс для работы с потоковой информацией.

Примерами потоковых данных могут служить раздел «Что происходит?» в Твиттере, прямые трансляции спортивных и музыкальных мероприятий и данные биржевых котировок.

1.3. Процесс data science

Процесс data science обычно состоит из шести шагов, как видно из схемы на рис. 1.5. Сейчас мы кратко рассмотрим основные пункты, а более подробное описание будет приведено в главе 2.

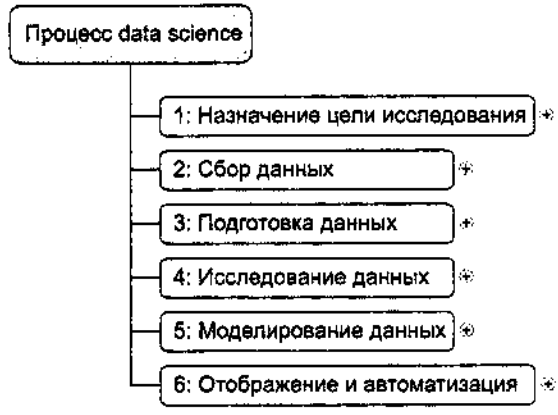


Рис. 1.5. Процесс data science

1.3.1. Назначение цели исследования

Дисциплина data science применяется в основном в контексте организаций. Когда бизнес предлагает вам взяться за проект data science, вы сначала готовите проектное задание. В проектном задании описано, что вы собираетесь исследовать, какую пользу это принесет компании, какие данные и ресурсы вам потребуются, представлен календарный план работ и описание выходных результатов.

В этой книге процесс data science будет применяться к примерам разного масштаба, и вы получите представление о возможных целях исследований.

1.3.2. Сбор данных

На втором этапе процесса происходит сбор данных. В проектном задании указано, какие данные вам потребуются и где их можно найти. На этом этапе вы принимаете меры к тому, чтобы данные могли использоваться в вашей программе, что означает проверку существования, качества и доступности данных. Данные также могут поставляться третьими сторонами и существовать в разных формах, от таблиц Excel до разного рода баз данных.

1.3.3. Подготовка данных

Процесс сбора данных подвержен ошибкам; в этой фазе исследователь повышает качество данных и готовит их к последующему использованию. Эта фаза состоит из трех подфаз: *очистка данных* удаляет некорректные значения из источника данных и устраняет расхождения между источниками, *интеграция данных* расширяет информацию посредством объединения информации из нескольких источников, а *преобразование данных* гарантирует, что данные находятся в подходящем формате для использования в ваших моделях.

1.3.4. Исследование данных

Исследование данных направлено на достижение более глубокого понимания данных. Вы пытаетесь понять, как переменные взаимодействуют друг с другом, оценить распределение данных и определить наличие выбросов. Для этого в основном используются описательные статистики, визуальные методы и простое моделирование. Этот шаг часто обозначается сокращением EDA от «Exploratory Data Analysis» («исследовательский анализ данных»).

1.3.5. Моделирование данных или построение модели

В этой фазе моделирования знания предметной области и информация о данных, полученная на предыдущих этапах, используются для получения ответа на вопрос исследования. В ходе моделирования используются методы из области статистики,

машинного обучения, исследования операций и т. д. Построение модели является итеративным процессом, в ходе которого исследователь выбирает переменные для модели, применяет модель и проводит диагностику модели.

1.3.6. Отображение и автоматизация

Наконец, результаты исследования должны быть представлены бизнес-стороне. Такие результаты могут существовать в разных формах, от презентаций до отчетов о научно-исследовательской работе. Иногда выполнение процесса приходится автоматизировать, потому что бизнес-сторона хочет использовать информацию, полученную в другом проекте, или задействовать рабочий процесс, использующий результаты вашей модели.

Итеративный характер процесса

Приведенное описание процесса data science может создать впечатление, что процесс имеет линейный характер, но в действительности вам придется часто возвращаться назад и что-то переделывать. Например, в фазе исследования данных могут обнаружиться выбросы, указывающие на ошибки импортирования данных. В ходе процесса data science вы получаете новую информацию, которая может породить новые вопросы. Чтобы вам не пришлось переделывать уже выполненную работу, проследите за тем, чтобы вопросы со стороны бизнеса с самого начала были определены достаточно ясно и подробно.

Итак, теперь суть процесса немного прояснилась и мы можем заняться рассмотрением методов.

1.4. Экосистема больших данных и data science

В настоящее время существует много разных инструментариев и инфраструктур для больших данных. В них легко запутаться, потому что новые технологии появляются очень быстро. Ситуация сильно упростится, если вы поймете, что экосистема больших данных может быть разбита на группы по технологиям с похожими целями и функциональностью; мы рассмотрим их в этом разделе. Специалисты data science применяют много разных технологий, но не все; важнейшим классам технологий data science в книге посвящена отдельная глава. На рис. 1.6 видны компоненты экосистемы больших данных и место разных технологий в этой структуре.

Рассмотрим разные группы инструментов на схеме и выясним, что делает каждая из них. Начнем с распределенных файловых систем.

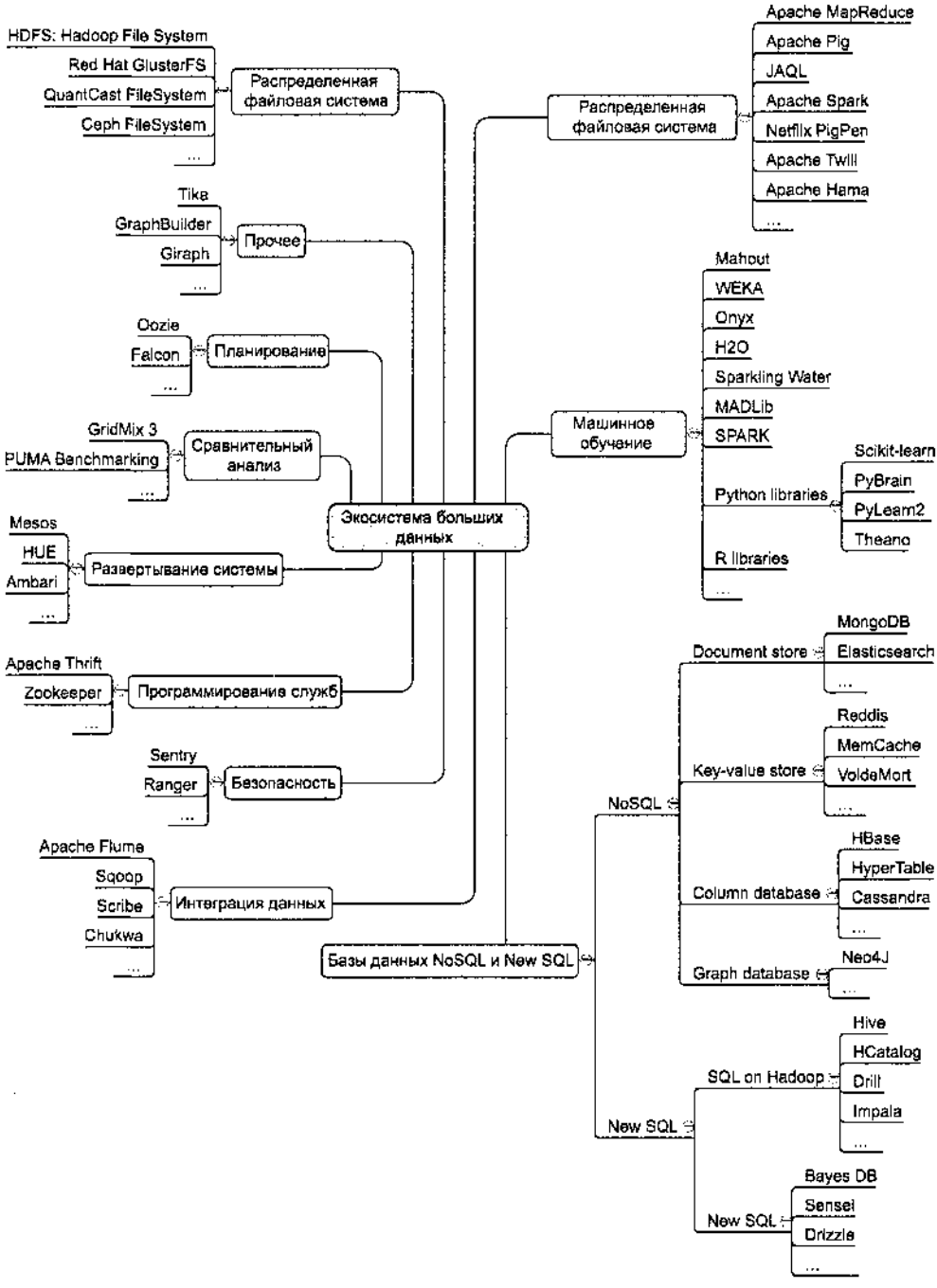


Рис. 1.6. Технологии больших данных можно разбить на несколько основных категорий

1.4.1. Распределенные файловые системы

Распределенная файловая система похожа на обычную файловую систему, но, в отличие от последней, она работает на нескольких серверах сразу. В ней можно делать почти все, что делается в обычных файловых системах. В основе любой файловой системы лежат такие действия, как хранение, чтение и удаление данных, а также реализация средств безопасности файлов; распределенные файловые системы не являются исключением. Распределенные файловые системы обладают рядом важных преимуществ:

- Они способны хранить файлы, размер которых превышает размер диска отдельного компьютера.
- Файлы автоматически реплицируются на нескольких серверах для создания избыточности или выполнения параллельных операций, при этом все сложности технической реализации скрываются от пользователя.
- Система легко масштабируется: пользователь не ограничивается объемом памяти или дискового пространства одного сервера.

Раньше масштабирование осуществлялось переводом всех систем на сервер с большим объемом памяти и дискового пространства и более быстрым процессором (вертикальное масштабирование). В наше время в систему можно добавить еще один малый сервер (горизонтальное масштабирование). Благодаря этому принципу потенциал масштабирования становится практически безграничным.

В настоящее время самой популярной распределенной файловой системой является *Hadoop File System* (*HDFS*). Она представляет собой реализацию Google File System с открытым кодом. В книге основное внимание будет уделено именно Hadoop File System, так как эта система чаще всего применяется на практике. Впрочем, существует много других распределенных файловых систем: *Red Hat Cluster File System*, *Ceph File System*, *Tachyon File System* и т. д.

1.4.2. Инфраструктура распределенного программирования

После того как данные будут сохранены в распределенной файловой системе, их нужно использовать. Один из важных аспектов работы с распределенным жестким диском заключается в том, что вы не перемещаете данные к программе, а скорее перемещаете программу к данным. Если вы начинаете «с нуля», имея только опыт работы на нормальном языке общего назначения (таком, как C, Python или Java), вам придется иметь дело со всеми сложностями, присущими распределенному программированию, например: перезапуском сбойных заданий, отслеживанием результатов из других субпроцессов и т. д. К счастью, в сообществе разработки открытого ПО было создано много инфраструктур, которые выполняют всю «черную работу» за вас. Эти инфраструктуры заметно упрощают работу с распределенными данными и решают многие из присущих ей проблем.

1.4.3. Инфраструктура интеграции данных

После создания распределенной файловой системы необходимо добавить данные. Вы должны переместить данные из одного источника в другой; именно здесь в полной мере проявляются достоинства таких инфраструктур интеграции данных, как Apache Sqoop и Apache Flume. Процесс похож на процесс извлечения, преобразования и загрузки в традиционных складах данных.

1.4.4. Инфраструктуры машинного обучения

Когда данные оказываются на своем месте, наступает время извлечения вожделенной скрытой информации. На этой стадии вам приходится использовать методы из области машинного обучения, статистики и прикладной математики. До Второй мировой войны все вычисления приходилось выполнять вручную, что существенно ограничивало возможности анализа данных. После Второй мировой войны появились компьютеры и технологии научных вычислений. Один компьютер мог выполнять огромные объемы вычислений, и перед исследователями открылся новый мир. Со времен этой научной революции человеку достаточно построить математические формулы, записать их в алгоритм и загрузить данные. При огромных объемах современных данных один компьютер уже не справляется с нагрузкой. Более того, некоторые алгоритмы, разработанные в предыдущем веке, никогда не завершатся до момента гибели Вселенной, даже если бы в вашем распоряжении оказались вычислительные ресурсы всех существующих компьютеров земли. Этот факт связан с их временной сложностью (*https://en.wikipedia.org/wiki/Time_complexity*). Пример такого алгоритма — попытка взлома пароля с проверкой всех возможных комбинаций (см. *http://stackoverflow.com/questions/7055652/real-world-example-of-exponential-time-complexity*). Один из самых серьезных недостатков старых алгоритмов заключался в том, что они плохо масштабировались. При тех объемах данных, которые приходится анализировать сегодня, это создает проблемы, и для работы с такими объемами данных требуются специализированные библиотеки и инфраструктуры. Самая популярная библиотека машинного обучения для Python называется *Scikit-learn*. Это отличный инструмент машинного обучения, и мы будем использовать ее в этой книге. Конечно, существуют и другие библиотеки для Python:

- *PyBrain для нейронных сетей* — термином «нейронные сети» обозначаются алгоритмы обучения, моделирующие человеческий мозг в отношении механики обучения и сложности. Нейронные сети принято считать областью сложной и недоступной для непосвященных.
- *NLTK* (Natural Language Toolkit) — как подсказывает название, библиотека предназначена для работы с данными на естественном языке. К этой обширной библиотеке прилагается ряд *текстовых корпусов*, упрощающих моделирование ваших данных.
- *Pylearn2* — другой инструментарий машинного обучения, немного уступающий Scikit-learn.

- *TensorFlow* — библиотека Python для глубокого обучения, предоставленная компанией Google.

Конечно, варианты не ограничиваются библиотеками Python. Spark — новое ядро машинного обучения с лицензией Apache, специализирующееся на машинном обучении в реальном времени. Проект заслуживает внимания; дополнительную информацию можно найти по адресу *http://spark.apache.org/*.

1.4.5. Базы данных NoSQL

Для хранения огромных объемов данных требуется программное обеспечение, специализирующееся на управлении этими данными и формировании запросов к ним. Традиционно в этой области правили бал реляционные базы данных — такие, как Oracle SQL, MySQL, Sybase IQ и др. И хотя во многих ситуациях они все еще считаются предпочтительным решением, появились новые типы баз данных, объединенные в категорию баз данных NoSQL.

Название категории обманчиво, потому что «No» в этом контексте означает «не только». Нехватка функциональности в SQL не является основной причиной сдвига парадигмы, и многие базы данных NoSQL реализовали собственную версию SQL. Однако у традиционных баз данных существуют свои недостатки, которые усложняют их масштабирование. Решая некоторые проблемы традиционных баз данных, базы данных NoSQL обеспечивают возможность почти неограниченного роста данных. Эти недостатки проявляются во всем, что относится к большим данным: их память и вычислительный ресурс не масштабируются за пределы одного узла и в традиционных базах данных отсутствуют средства обработки потоковых, графовых и неструктурированных форм данных.

Существует много разновидностей баз данных, но их можно разделить на следующие типы:

- *Столбцовые базы данных* — данные организуются в столбцы, что позволяет алгоритмам существенно повышать скорость обработки запросов. В более современных технологиях используется принцип хранения по ячейкам. Табличные структуры продолжают играть важную роль в обработке данных.
- *Хранилища документов* — хранилища документов не используют таблицы, но хранят полную информацию о документе. Их отличительной особенностью является чрезвычайно гибкая схема данных.
- *Потоковые данные* — сбор, преобразование и агрегирование данных осуществляются не по пакетному принципу, а в реальном времени. Хотя мы выделили потоковые данные в категорию баз данных, чтобы упростить выбор инструмента, скорее они образуют особую разновидность задач, породившую такие технологии, как Storm.
- *Хранилища «ключ—значение»* — данные не хранятся в таблицах; с каждым значением связывается ключ: например, *org.marketing.sales.2015: 20000*. Такое

решение хорошо масштабируется, но реализация почти полностью возлагается на разработчика.

- *SQL в Hadoop* — пакетные запросы в Hadoop пишутся на SQL-подобном языке, во внутренней реализации которого используется инфраструктура отображения-свертки (map-reduce).
- *Обновленный SQL* — этот тип сочетает масштабируемость баз данных NoSQL с преимуществами реляционных баз данных. Все эти базы данных используют интерфейс SQL и реляционную модель данных.
- *Графовые базы данных* — не для всех задач табличный формат является оптимальным. Некоторые задачи более естественно подходят для представления в виде графа и хранения в графовых базах данных. Классический пример такого рода — социальная сеть.

1.4.6. Инструменты планирования

Инструменты планирования упрощают автоматизацию повторяющихся операций и запуск заданий по событиям (например, при появлении нового файла в папке). Они похожи на такие традиционные программы, как CRON в Linux, но разработаны специально для больших данных. Например, такие инструменты могут запускать задачу MapReduce при появлении нового набора данных в каталоге.

1.4.7. Инструменты сравнительного анализа

Этот класс инструментов был разработан для оптимизации установки больших данных за счет предоставления стандартизированных *профилей*. Профили строятся на основании представительного множества операций с большими данными. Задачи сравнительного анализа и оптимизации инфраструктуры больших данных и настройки конфигурации часто относятся к компетенции не специалистов data science, а профессионалов, специализирующихся на организации IT-инфраструктуры; по этой причине они не рассматриваются в этой книге. Использование оптимизированной инфраструктуры приводит к существенной экономии. Например, экономия 10% в кластере из 100 серверов равна стоимости 10 серверов.

1.4.8. Развертывание системы

Подготовка инфраструктуры больших данных — непростая задача. Инструменты развертывания системы проявляют себя при развертывании новых приложений в кластерах больших данных. Они в значительной степени автоматизируют установку и настройку компонентов больших данных. К числу основных задач специалиста data science эта область не относится.

1.4.9. Программирование служб

Предположим, вы создали приложение для прогнозирования результатов футбольных матчей и теперь хотите предоставить всем желающим доступ к прогнозам вашего приложения. Однако вы понятия не имеете, какие архитектуры или технологии будут использоваться в их системах. Инструменты *программирования служб* обеспечивают доступ к приложениям больших данных как к сервису. Специалистам data science иногда приходится открывать доступ к своим моделям через службы. Самым известным примером такого рода являются REST-службы; сокращение REST обозначает «Representational State Transfer», т. е. «передача состояния представления». Эти службы часто используются для передачи данных веб-сайтам.

1.4.10. Безопасность

Вы хотите, чтобы все желающие могли получить доступ к вашим данным? Вероятно, при этом нужно организовать механизм точного управления доступом к данным, но делать это на уровне каждого отдельного приложения не хочется. Средства безопасности больших данных позволяют создать централизованную и высокоточную систему управления доступом к данным. Безопасность больших данных превратилась в самостоятельную область исследований, и специалисты data science обычно сталкиваются с ней в роли потребителей данных, редко когда им приходится реализовывать средства безопасности самостоятельно. В этой книге тема безопасности больших данных не рассматривается, это работа для экспертов в области безопасности.

1.5. Вводный пример использования Hadoop

В завершение этой главы мы рассмотрим небольшое приложение в контексте больших данных. Для этого будет использоваться образ Hortonworks Sandbox — виртуальной машины, созданной Hortonworks для тестирования приложений больших данных на локальной машине. Далее в книге будет показано, как Juju упрощает установку Hadoop на нескольких машинах.

Для запуска первого примера будет использован небольшой набор данных, но запрос к большому набору с миллиардами записей выполняется так же просто. На первый взгляд язык запросов кажется похожим на SQL, но при этом незаметно для пользователя запускается задание MapReduce. Оно строит обычную таблицу результатов, которые затем можно представить на столбцовой диаграмме. В результате мы получим диаграмму наподобие изображенной на рис. 1.7.

Чтобы как можно быстрее и проще запустить приложение, мы воспользуемся средой Hortonworks Sandbox в VirtualBox. VirtualBox — инструмент виртуализации, позволяющий запустить другую операционную систему в вашей операционной системе. В данном случае он поможет вам запустить CentOS с существующей установкой Hadoop в вашей установленной операционной системе.

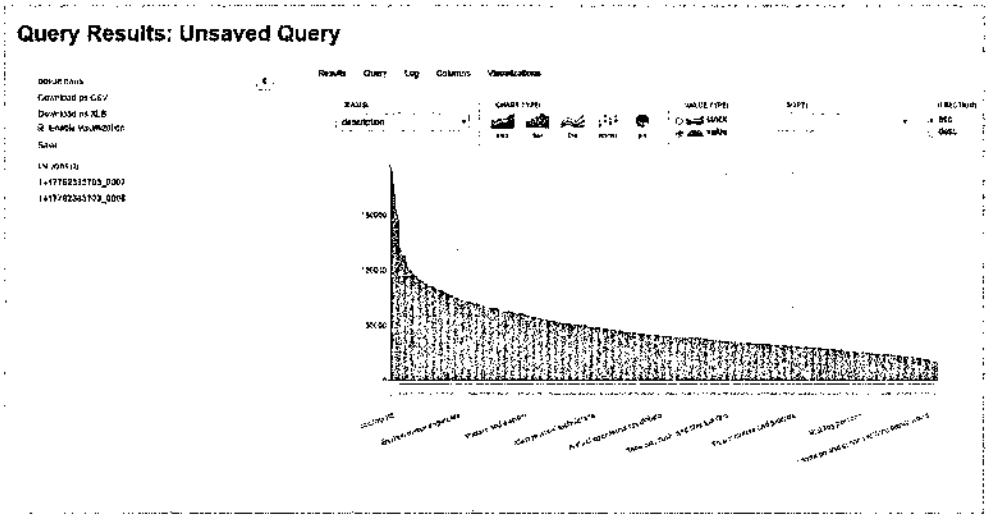


Рис. 1.7. Результат: зависимость среднего оклада от должности

Для запуска Sandbox в Virtualbox необходимо выполнить несколько простых шагов. Будьте внимательны: следующее описание актуально на момент написания этой главы (февраль 2015 года):

1. Загрузите виртуальный образ по адресу *http://hortonworks.com/products/hortonworks-sandbox/#install.*
2. Запустите управляющую виртуальную машину. VirtualBox можно загрузить по адресу *https://www.virtualbox.org/wiki/Downloads.*
3. Нажмите Ctrl+I и выберите виртуальный образ Hortonworks.
4. Щелкните на кнопке **Next**.
5. Щелкните на кнопке **Import**; через непродолжительное время ваш образ будет успешно импортирован.
6. Выберите свою виртуальную машину и щелкните на кнопке **Run**.
7. Немного подождите запуска CentOS с установкой Hadoop (рис. 1.8). Обратите внимание: на иллюстрации используется Sandbox версии 2.1. В других версиях возможны небольшие изменения.

На запущенную машину можно войти напрямую или воспользоваться SSH. Мы воспользуемся веб-интерфейсом. Введите в браузере адрес *http://127.0.0.1:8000*; на экране должна появиться заставка, показанная на рис. 1.9.

В поставку Hortonworks также включены два учебных набора данных, находящиеся на уровне HCatalog. Щелкните на кнопке **HCat**, чтобы просмотреть список доступных таблиц (рис. 1.10).

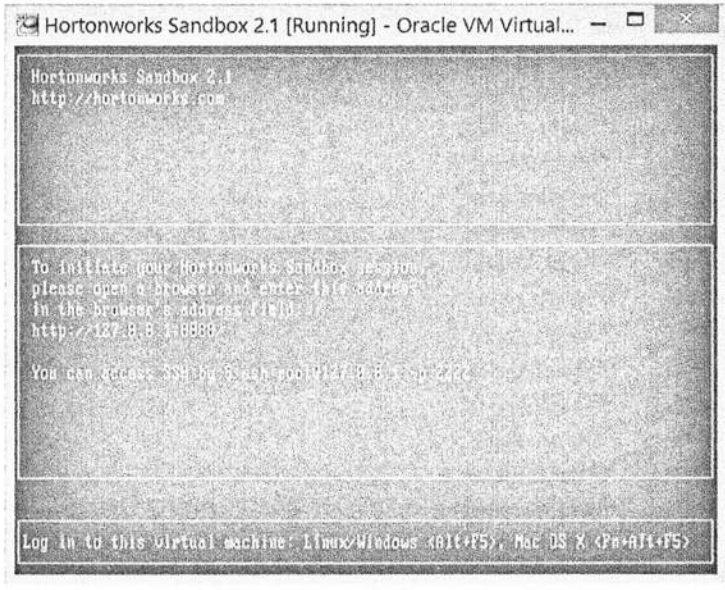


Рис. 1.8. Hortonworks Sandbox с VirtualBox

Рис. 1.9. Заставка Hortonworks Sandbox по адресу http://127.0.0.1:8000

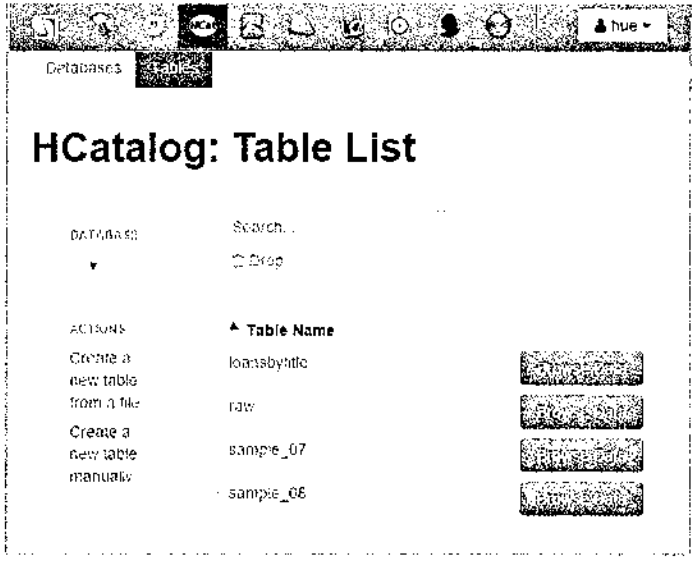


Рис. 1.10. Список доступных таблиц HCatalog

Чтобы просмотреть данные, щелкните на кнопке Browse Data рядом с пунктом sample_07 для перехода к следующему экрану (рис. 1.11).

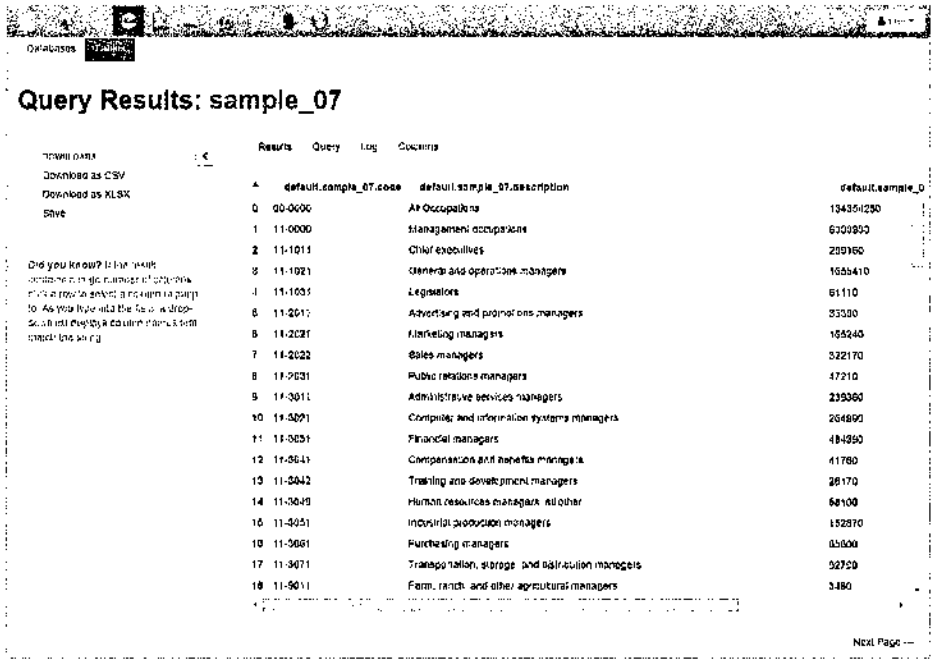


Рис. 1.11. Содержимое таблицы

На первый взгляд таблица кажется совершенно обычной. Hive — инструмент, позволяющий работать с этими данными как с обычной базой данных на языке SQL. Да, все верно: в Hive для получения результатов используется HiveQL — диалект традиционного языка SQL. Чтобы открыть редактор Beeswax HiveQL, щелкните на кнопке Beeswax в меню (рис. 1.12).

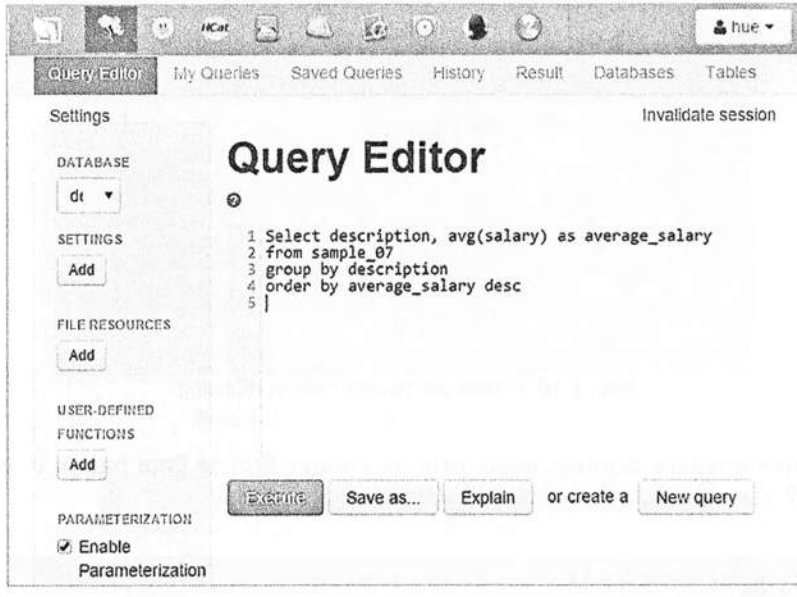


Рис. 1.12. Команды HiveQL выполняются в редакторе Beeswax HiveQL. Во внутренней реализации эти команды преобразуются в задания MapReduce

Для получения результатов выполните следующий запрос:

```
Select description, avg(salary) as average_salary from sample_07 group by
description order by average_salary desc.
```

Щелкните на кнопке Execute. Hive преобразует запрос HiveQL в задание MapReduce и выполняет его в среде Hadoop, как показано на рис. 1.13.

Впрочем, пока лучше не читать окно журнала; на этой стадии оно выглядит невразумительно. Если это ваш первый запрос, выполнение может занять до 30 секунд — Hadoop известен своей долгой инициализацией. Впрочем, мы еще вернемся к этой теме.

Через некоторое время на экране появляется результат. Отличная работа! Из полученной сводки (рис. 1.14) следует, что среди самых высокооплачиваемых профессий немало врачей. Кого-нибудь это удивляет?

Эта таблица завершает краткое введение в Hadoop.

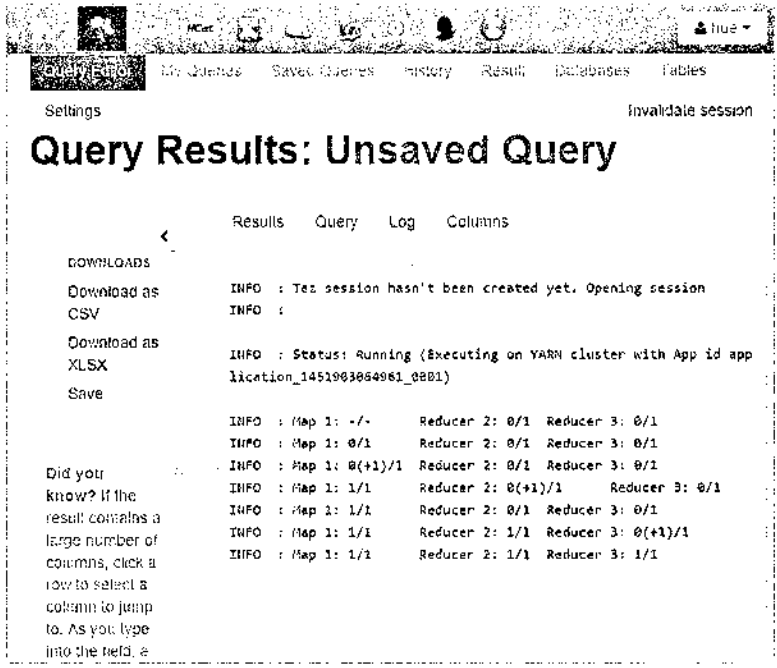


Рис. 1.13. Из журнала видно, что запрос HiveQL был преобразован в задание MapReduce. Примечание: этот журнал был сгенерирован в версии HDP за февраль 2015 года, в текущей версии он может выглядеть немного иначе

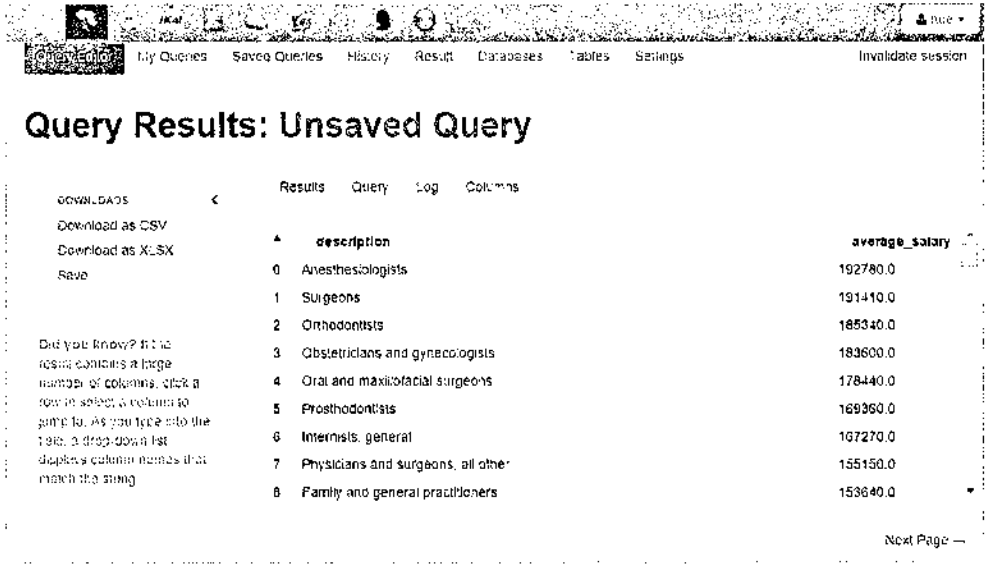


Рис. 1.14. Конечный результат: сводка средних окладов по профессиям

И хотя эта глава была всего лишь началом, возможно, порой что-то казалось непонятным. В таком случае оставьте все как есть и вернитесь после того, как все концепции будут подробно объяснены. Data science — обширная область, в которой существует обширная система понятий. Надеемся, мы сможем дать достаточно подробное представление об этой области за то время, которое проведем вместе с вами. В будущем вы сами выберете и отточите свои навыки в том направлении, которое интересует вас больше всего. Собственно, для этого и была написана книга; надеемся, что вы не пожалеете о потраченном времени.

1.6. Итоги

Основные положения этой главы:

- Большие данные — обобщающий термин для любых наборов данных, достаточно больших и сложных, чтобы их можно было обработать традиционными средствами работы с данными. Большие данные характеризуются «четырьмя V»: объемом, разнообразием, скоростью и достоверностью.
- Основным содержанием data science являются методы анализа наборов данных, от совсем небольших до гигантских.
- Процесс data science сам по себе нелинеен, но его можно разделить на несколько шагов:
  1. Назначение цели исследования.
  2. Сбор данных.
  3. Подготовка данных.
  4. Исследование данных.
  5. Моделирование.
  6. Отображение и автоматизация.
- Набор технологий больших данных вовсе не сводится к Hadoop. Он состоит из множества разных технологий, которые можно разбить на следующие категории:
  - Файловая система.
  - Инфраструктуры распределенного программирования.
  - Интеграция данных.
  - Базы данных.
  - Машинное обучение.
  - Безопасность.
  - Планирование.

- Сравнительный анализ.
- Развертывание.
- Программирование служб.

- Не все категории больших данных интенсивно используются специалистами data science. В основном они занимаются файловыми системами, инфраструктурами распределенного программирования, базами данных и машинным обучением. Конечно, им приходится иметь дело с другими компонентами, и все же эти предметные области относятся к сфере деятельности других профессий.
- Данные могут существовать во многих формах. Основные формы:
  - Структурированные данные.
  - Неструктурированные данные.
  - Данные на естественном языке.
  - Машинные данные.
  - Графовые данные.
  - Потоковые данные.

Процесс data science

В этой главе:

- ✓ Последовательность операций в процессе data science.
- ✓ Отдельные стадии процесса data science.

В этой главе дается общий обзор процесса data science без углубленного рассмотрения самих больших данных. О том, как работать с большими наборами данных, потоковыми данными и текстовыми данными, будет рассказано в последующих главах.

2.1. Обзор процесса data science

Структурированный подход к data science повышает вероятность успеха в проектах data science при минимальных издержках. Он также делает возможным коллективную работу над проектом, при которой каждый участник группы занимается той областью, в которой наиболее силен. Однако будьте внимательны: этот подход не годится для всех типов проектов и не является единственно правильным подходом к качественной обработке и анализу данных.

Типичный процесс data science состоит из шести последовательно выполняемых шагов (рис. 2.1).

На рис. 2.1 изображен процесс data science с основными этапами и действиями, предпринимаемыми в ходе проекта. Приведенный ниже список следует рассматривать лишь как краткую сводку; каждый этап будет более подробно описан далее в главе.

1. Процесс начинается с назначения *цели исследования*. На этой стадии необходимо прежде всего позаботиться о том, чтобы все ключевые участники понимали ответы на все вопросы «что», «как» и «почему», относящиеся к проекту. В любом серьезном проекте основным результатом данного этапа становится *проектное задание*.

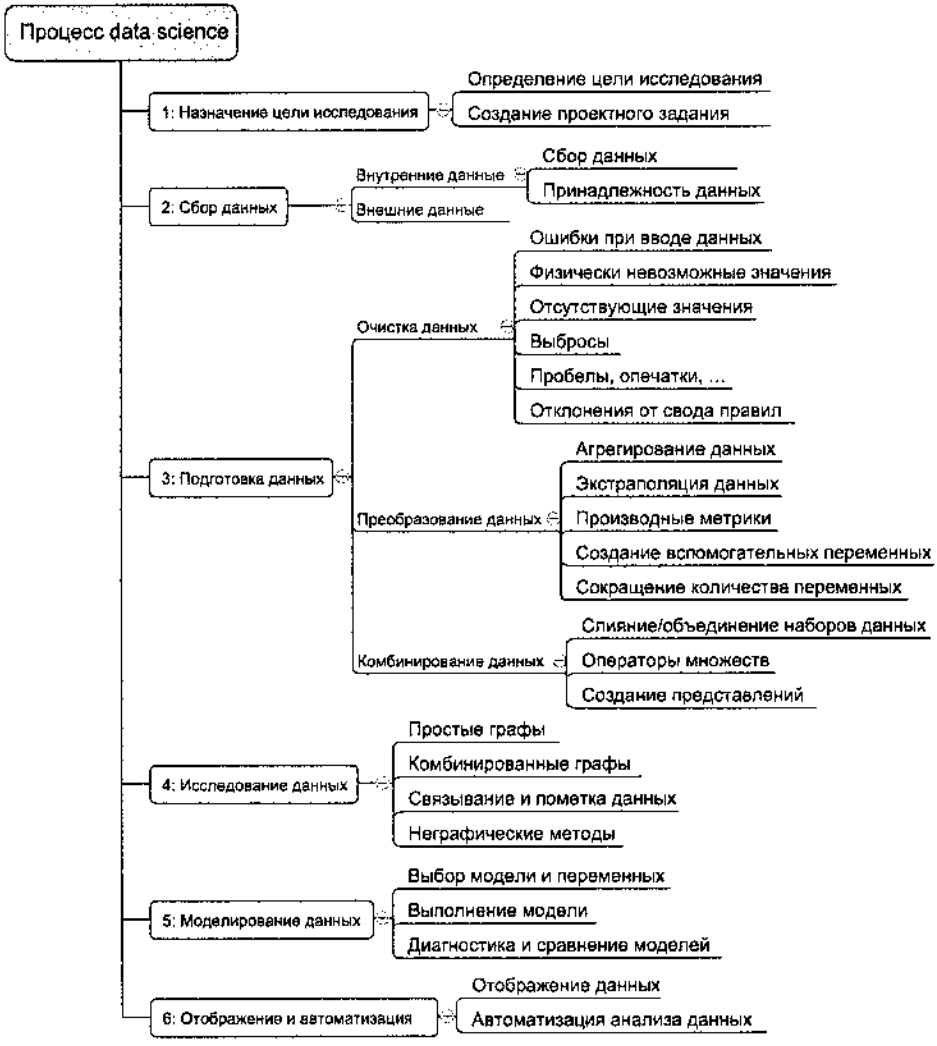


Рис. 2.1. Структура процесса data science

2. Во второй фазе происходит *сбор данных*. Для проведения анализа необходимы данные, поэтому этот шаг включает в себя поиск подходящих данных и получение доступа к ним у владельца. В результате второго этапа вы получаете необработанные, «сырые» данные; вероятно, для работы с ними потребуются некоторая доработка и преобразование.

3. Собранные необработанные данные необходимо *подготовить*. На этом этапе данные из низкоуровневой формы преобразуются в данные, которые могут напрямую использоваться в ваших моделях. Для этого в данных выявляются

и исправляются всевозможные ошибки, данные из разных источников объединяются и преобразуются. После того как данный этап будет успешно пройден, можно переходить к визуализации и моделированию данных.

4. На четвертом этапе выполняется *исследование данных*. Конечной целью этого этапа является глубокое понимание данных. Вы ищете закономерности, корреляции и отклонения, основанные на визуальных и описательных методах. Результаты, полученные в этой фазе, позволяют приступать к моделированию.

5. Наступает самая интересная часть: *построение модели* (в книге также часто используется термин «моделирование данных»). В этой фазе вы пытаетесь вникнуть в суть или сделать прогнозы, указанные в проектном задании. На этой стадии можно привлекать «тяжелую артиллерию», но помните: наш опыт исследований показывает, что комбинация простых моделей часто (хотя и не всегда) работает эффективнее одной сложной модели. Если вы справитесь с этой фазой, работа почти закончена.

6. На последнем этапе процесса data science вы демонстрируете полученные результаты и проводите автоматизацию анализа (если потребуется). Одной из целей проекта является изменение процесса и (или) принятие более правильных решений. Возможно, вам придется убедить заказчика в том, что полученные результаты действительно изменят бизнес-процессы так, как вы ожидаете. Именно здесь вы можете блеснуть в роли лица, влияющего на принятие решения. Важность этого шага в большей степени проявляется на стратегическом и тактическом уровнях проектов. Некоторые проекты требуют многократного выполнения бизнес-процессов, так что автоматизация проекта сэкономит вам время.

На практике линейное следование от этапа 1 к этапу 6 встречается редко. Чаще приходится отступать назад и повторять различные фазы в цикле.

Соблюдение этой структуры из шести этапов повышает коэффициент успеха проектов и влияние результатов исследования. Этот процесс гарантирует наличие четко определенного плана исследования, хорошее понимание бизнес-проблемы и четко сформулированные результаты еще до того, как вы хотя бы впервые увидите данные. Первые этапы процесса направлены на получение качественных данных, которые послужат входными данными для ваших моделей. Это гарантирует, что ваши модели продемонстрируют наиболее хорошие результаты в будущем. В области data science существует известная поговорка: «Мусор на входе — мусор на выходе».

Другое преимущество структурированного подхода заключается в том, что вы проводите больше времени в *режиме прототипа* в ходе поиска оптимальной модели. Скорее всего, в ходе построения *прототипа* вы опробуете несколько моделей, не уделяя первоочередного внимания таким проблемам, как скорость выполнения или соответствие кода существующим стандартам. Это позволит вам сосредоточиться на бизнес-возможностях.

Не каждый проект инициируется заказчиком. Результаты, полученные в ходе анализа или при поступлении новых данных, могут порождать новые проекты. Когда

группа data science выдает идею, часть работы по внесению предложения и поиску инвестора уже выполнена.

Разбиение проекта на стадии также упрощает коллективную работу над ним. Невозможно быть специалистом во всех областях сразу. Вы должны знать, как отправить данные в разные базы данных, как найти оптимальную схему данных, которая подходит не только для вашего приложения, но и для других проектов в вашей компании, а также отслеживать все разнообразные статистические методы и инструменты анализа данных, не говоря уже о средствах отображения и бизнес-политике. Найти такого специалиста непросто, именно поэтому все больше компаний поручают работу команде специалистов, а не ищут одного-единственного специалиста, который умеет все.

Процесс, описанный в этом разделе, лучше всего подходит для проектов data science с небольшим количеством моделей. Не для всех видов проектов он оптимален: например, для проекта с миллионами моделей реального времени потребуется подход, отличающийся от представленной последовательности операций. Впрочем, даже если начинающий специалист data science будет просто следовать этой схеме, он добьется достаточно весомых результатов.

2.1.1. Не будьте рабом процесса

Не каждый проект жестко подчиняется этой схеме, потому что процесс зависит от личных предпочтений специалиста data science, компании и природы проекта, над которым он работает. Одни компании требуют соблюдения жесткого протокола, тогда как в других принята менее формальная схема работы. В общем случае структурированный подход необходим в работе над сложным проектом или в проектах с большим количеством людей или ресурсов.

Альтернативой последовательному процессу с итерациями является *гибкая* (agile) модель проекта. Хотя эта методология завоевывает популярность в IT-подразделениях компаний, она также берется на вооружение сообществом data science. Гибкие методологии хорошо подходят для проектов data science, однако политика компаний во многих случаях отдает предпочтение более формальному подходу к data science.

Заранее спланировать процесс data science во всех подробностях не всегда возможно, и, как правило, различные этапы процесса приходится повторять. Например, после вводного совещания вы начинаете нормально работать, пока не входите в фазу исследования данных. На графиках проявляются различия в поведении двух групп — возможно, мужчин и женщин? Вы не уверены, потому что в данных нет переменной, отражающей пол участника. Для подтверждения гипотезы потребуется получить дополнительный набор данных. Для этого необходимо пройти процесс утверждения, а для этого вам (или заказчику), скорее всего, придется предоставить некое подобие проектного задания. В крупных компаниях сбор всех данных, необходимых для завершения проекта, может стать весьма непростым делом.

2.2. Этап 1: Определение целей исследования и создание проектного задания

Проект начинается с ответов на вопросы «что», «почему» и «как» (рис. 2.2). Что нужно компании? Почему руководство придает значение вашему исследованию? Является ли оно частью большей стратегической картины или это «разовый» проект, открытый потому, что кто-то заметил благоприятную возможность? Цель первой фазы — сформулировать ответы на эти три вопроса (что, почему, как), чтобы все знали, что делать, и могли согласовать оптимальный курс действий.

Рис. 2.2. Этап 1: Назначение цели исследования

Итогом этой фазы должна быть ясная цель исследования, хорошее понимание контекста, четко определенные результаты и календарный план действий. Эту информацию лучше всего включить в проектное задание. Конечно, длина и степень формализации результатов зависит от проектов и компаний. В ранней фазе проекта личные качества и деловое чутье играют более важную роль, чем техническая квалификация, поэтому в этой части направление нередко задается руководящим персоналом.

2.2.1. Выделите время на то, чтобы разобраться в целях и контексте исследования

Главный результат этой фазы — *цель исследования*, которая четко и предметно определяет смысл порученной вам работы. Понимание целей и контекста бизнес-стороны абсолютно необходимо для успеха проекта. Продолжайте задавать вопросы и изобретать примеры, пока вы не осознаете конкретные коммерческие ожидания, не определите место своего проекта в общей картине, не оцените, каким образом ваше исследование повлияет на коммерческую деятельность, и не поймете, как будут использоваться ваши результаты. Представьте самую неприятную из всех возможных ситуаций: вы проводите месяцы за исследованиями, на вас снисходит

озарение, вы решаете задачу и сообщаете о своем открытии организации — и тут внезапно выясняется, что вы неправильно поняли поставленный вопрос. К этой фазе нужно отнестись серьезно. Многие специалисты data science совершают эту ошибку: при всей своей математической и научной одаренности они не считают нужным разобраться в целях и контексте бизнеса.

2.2.2. Создайте проектное задание

Клиент хочет заранее знать, за что он платит деньги, поэтому после того, как у вас появится хорошее понимание бизнес-задачи, постарайтесь заключить формальное соглашение по поводу предъявляемых результатов. Всю эту информацию лучше всего собрать в проектном задании, а в любом сколько-нибудь серьезном проекте это просто обязательно.

Составление проектного задания — это командная работа, поэтому ваш вклад должен включать как минимум следующее:

- Четко сформулированная цель исследований.
- Предназначение и контекст проекта.
- Предварительное описание методики анализа.
- Ресурсы, которые вы намерены использовать.
- Доказательство практической реализуемости проекта (или возможности проверки концепции.)
- Предъявляемые результаты и критерий успеха.
- Календарный план.

На основании полученной информации ваш клиент может оценить затраты на проект, а также человеческие и информационные ресурсы, необходимые для его успешного завершения.

2.3. Этап 2: Сбор данных

На следующем этапе процесса происходит сбор необходимых данных (рис. 2.3). Иногда приходится засучивать рукава и проектировать процесс сбора данных самостоятельно, но в большинстве случаев вы не будете задействованы на этой стадии. Многие компании уже собрали и сохранили данные за вас, а все недостающее часто можно купить у третьих сторон. Не бойтесь искать данные за пределами своей организации, потому что все больше организаций открывают бесплатный доступ к высококачественным данным для общественного и коммерческого использования.

Данные могут храниться во многих форматах, от простых текстовых файлов до таблиц баз данных. На этом этапе следует собрать все необходимые данные. Это

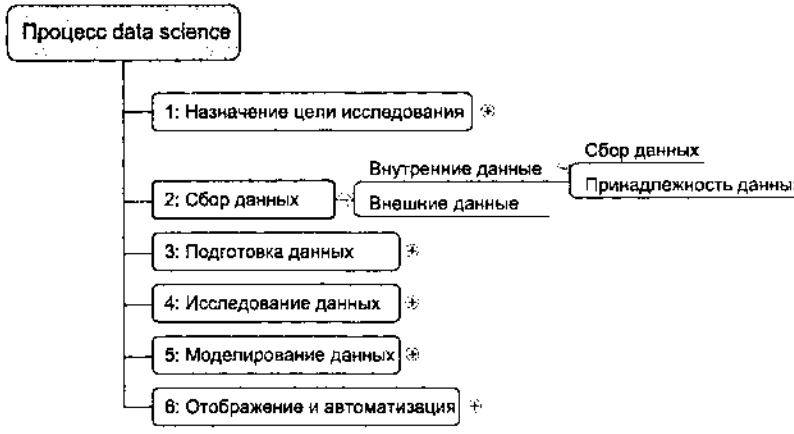


Рис. 2.3. Этап 2: Сбор данных

может быть непросто: даже когда данные собраны успешно, они часто напоминают необработанный алмаз: чтобы от них была хоть какая-то польза, им потребуется шлифовка и огранка.

2.3.1. Начните с данных, хранимых в компании

Прежде всего следует оценить актуальность и качество данных, уже доступных в компании. Во многих компаниях существуют специальные программы сопровождения ключевых данных, так что большая часть работы по очистке данных может быть уже выполнена. Эти данные могут храниться в официальных хранилищах (базах данных, *витринах данных* (data marts), *складах данных* (data warehouses) и *озерах данных* (data lakes)), которыми управляют группы IT-профессионалов. База данных предназначена прежде всего для хранения данных, тогда как склад данных предназначен для чтения и анализа этих данных. Витрина данных представляет собой подмножество склада данных, ориентированное на обслуживание конкретного структурного подразделения. Если в складах и витринах данных информация хранится в уже обработанном виде, то в озерах данных содержатся данные в естественном, необработанном формате. Впрочем, нельзя исключать вероятность того, что ваши данные все еще хранятся в файлах Excel на компьютере эксперта в предметной области.

Найти данные даже в пределах вашей собственной компании может быть достаточно сложно. В процессе роста компании ее данные оказываются рассеянными по многим местам. Данные могут быть разбросаны из-за того, что работники переходят на другие должности или уходят из компании. Документация и метаданные не всегда входят в число приоритетов руководства, так что, возможно, вам придется заняться расследованием в духе Шерлока Холмса для поиска всех потерянных фрагментов.

Получение доступа к данным — другая сложная задача. Организации понимают ценность и конфиденциальность данных, и в них часто устанавливаются правила, при которых любому работнику доступны данные, необходимые для его работы, — и никакие более. Эти правила превращаются в физические и электронные барьеры, иногда называемые «китайскими стенами» (chinese walls). В большинстве стран такие «стены» в отношении клиентских данных являются обязательными и строго регламентированными. На то есть веские причины; представьте, что в компании, занимающейся выпуском кредитных карт, любой желающий может получить доступ к информации о ваших текущих затратах. Словом, для получения доступа к данным тоже может потребоваться время, и здесь необходимо учитывать политику компании.

2.3.2. Не бойтесь покупок во внешних источниках

Если необходимые данные недоступны внутри вашей организации, поищите информацию во внешнем мире. Многие компании специализируются на сборе ценной информации. Например, Nielsen и GFK хорошо известны в этом отношении в сфере розничной торговли. Другие компании предоставляют данные для того, чтобы вы, в свою очередь, совершенствовали предоставляемые ими услуги и экосистемы. В частности, к этой категории относятся Twitter, LinkedIn и Facebook.

Хотя некоторые компании считают данные ресурсом, более ценным, чем нефть, в наши дни все больше правительственных учреждений и организаций бесплатно делится своими данными с миром. Это могут быть данные выдающегося качества; это зависит от учреждения, которое создает эти данные и управляет ими. Предоставляемая информация относится к самым разным областям: например, связи количества несчастных случаев или уровня употребления наркотиков в определенном регионе с его демографическими показателями. Информация может принести пользу как дополнение собственных данных компаний, но она также пригодится тем, кто занимается самообучением в области data science. В табл. 2.1 приведена небольшая подборка поставщиков открытых данных, которых с каждым днем становится все больше и больше.

Таблица 2.1. Начальная подборка поставщиков открытых данных

Сайт с открытыми данными	Описание
Data.gov	Центр открытых данных правительства США
https://open-data.europa.eu/	Центр открытых данных Европейской комиссии
Freebase.org	Открытая база данных, которая получает информацию с таких сайтов, как Википедия, MusicBrains и архив SEC
Data.worldbank.org	Проект открытых данных Всемирного банка
Aiddata.org	Открытые данные из области международного развития
Open.fda.gov	Открытые данные Администрации США по пищевым продуктам и лекарственным веществам

2.3.3. Проверьте качество данных сейчас, чтобы предотвратить проблемы в будущем

Скорее всего, на проверку и очистку данных будет потрачена значительная часть времени проекта — иногда до 80%. Первая проверка данных в процессе data science происходит на стадии сбора данных. Многие ошибки, с которыми вы столкнетесь в фазе сбора данных, легко обнаруживаются, но в случае невнимательности вы рискуете провести много часов за выявлением проблем с данными, которые можно было бы легко предотвратить в процессе импортирования.

Вы будете анализировать данные в ходе импортирования, в фазе подготовки и исследования данных. Впрочем, цели и глубина анализа в этих фазах различаются. В ходе *сбора данных* вы проверяете, совпадают ли данные с данными в исходном документе, и проверяете правильность типов данных. Проверка не должна занять слишком много времени; когда у вас будет достаточно уверенности в том, что полученные данные близки к данным в исходном документе, можно остановиться. В ходе *подготовки данных* выполняется более тщательная проверка. Если проверка в предыдущей фазе была выполнена тщательно, то ошибки, обнаруженные вами, также присутствуют в исходном документе. На этот раз основное внимание уделяется содержимому переменных: необходимо избавиться от опечаток и других ошибок ввода данных, а также привести данные разных наборов к общему стандарту (например, везде исправить United Kingdom на UK). В *фазе исследования* ваше внимание переключается на то, что можно узнать из данных. Вы предполагаете, что имеющиеся данные «чисты», и рассматриваете такие статистические характеристики, как распределения, корреляции и выбросы. Эти фазы часто повторяются. Например, выбросы, обнаруженные в фазе исследования, могут указывать на ошибку ввода данных. Теперь, когда вы понимаете, как качество данных повышается в ходе процесса, можно более глубоко изучить шаг подготовки данных.

2.4. Этап 3: Очистка, интеграция и преобразование данных

Данные, полученные в фазе сбора данных, с большой долей вероятности представляют собой «неотшлифованный алмаз». Ваша задача — убрать дефекты и подготовить данные для использования в фазах моделирования и представления результатов. Это очень важный момент, потому что ваши модели будут работать лучше и вы потратите меньше времени на исправление аномальных результатов. Сколько ни повторяй классическую поговорку, все равно будет мало: «мусор на входе — мусор на выходе». Ваша модель должна получать данные в конкретном формате, так что преобразование данных всегда будет играть важную роль. Привыкайте исправлять ошибки в данных на как можно более ранней стадии процесса. Впрочем, в реальной ситуации это не всегда возможно, так что вам придется вносить исправления в свою программу.

На рис. 2.4 показаны основные действия, выполняемые в фазе очистки, интеграции и преобразования данных.

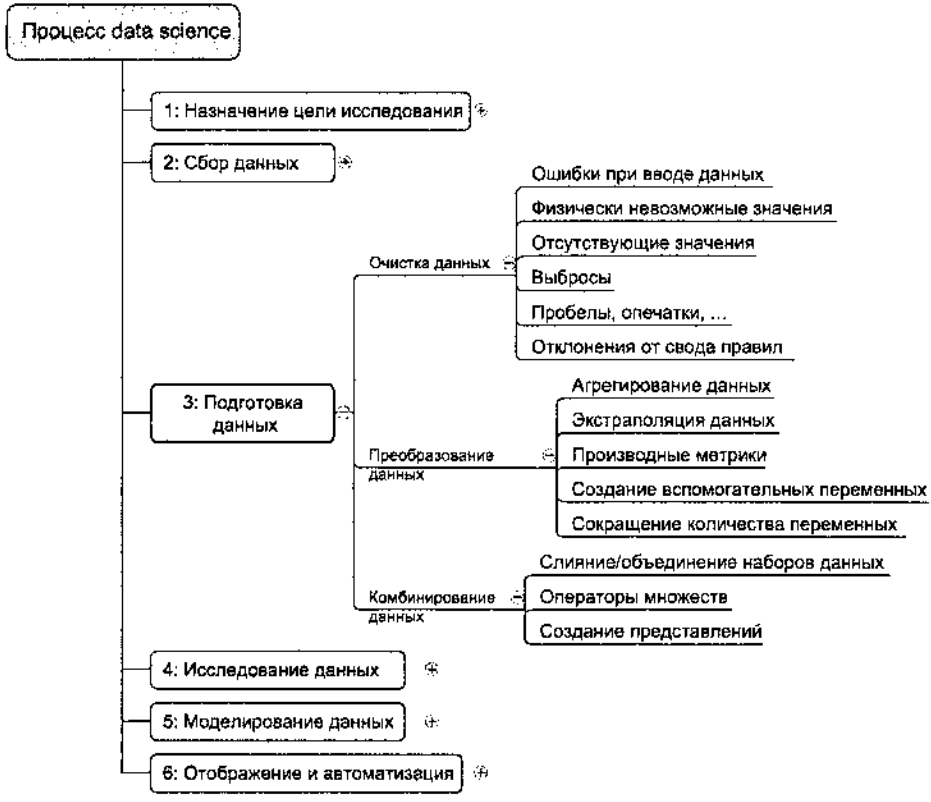


Рис. 2.4. Этап 3: Подготовка данных

На первый взгляд схема выглядит немного абстрактно, но все пункты списка будут более подробно описаны в следующих разделах. Вы увидите, что у всех этих действий много общего.

2.4.1. Очистка данных

Очистка данных представляет собой подпроцесс общего процесса data science, направленный на устранение ошибок в данных с тем, чтобы эти данные адекватно и последовательно представляли процесс, в результате которого они были получены.

«Адекватное и последовательное представление» означает, что существует как минимум два типа ошибок. К первому типу относятся *ошибки интерпретации*,

когда вы принимаете на веру значение в данных (пример: из данных следует, что возраст человека превышает 300 лет). Ошибки второго типа связаны с *расхождениями* между источниками данных или стандартизированными значениями вашей компании. Пример ошибки такого типа — включение строки «Female» в одну таблицу и «F» в другую, хотя они означают одно и то же: признак женского пола. Или другой пример: в одной таблице денежные суммы хранятся в фунтах, а в другой — в долларах. Разновидностей таких ошибок слишком много, чтобы дать полный список, но в табл. 2.2 приведена сводка типов ошибок, которые могут быть выявлены простыми проверками, — так сказать, «легкая добыча» в ходе проверки.

Таблица 2.2. Сводка распространенных ошибок

Общее решение	
Попробуйте исправить проблему на одном из ранних звеньев цепочки сбора данных или же исправьте ее в программе	
Описание ошибки	Возможное решение
Ошибки, указывающие на ложные значения в одном наборе данных	
Ошибки при вводе данных	Ручное переопределение
Лишние пропуски	Использование строковых функций
Невозможные значения	Ручное переопределение
Отсутствующие значения	Удаление выборки или значения
Выбросы	Проверка значения, и в случае ошибочности — интерпретация как отсутствующего значения (удаление или вставка)
Ошибки, указывающие на непоследовательность в данных	
Отклонения от свода правил	Сопоставление по ключу или ручное переопределение
Разные единицы измерения	Пересчет
Разные уровни агрегирования	Переход на единый уровень измерений посредством агрегирования или экстраполяции

Иногда для поиска и выявления ошибок данных применяются расширенные методы: например, простое моделирование; в этой ситуации диагностические диаграммы могут быть особенно полезны. Например, на рис. 2.5 применяется метрика для выявления точек данных, которые выглядят аномальными. Мы проводим регрессию, чтобы познакомиться с данными и обнаружить влияние отдельных наблюдений на регрессионную линию. Если одно наблюдение оказывает слишком заметное влияние, это может свидетельствовать об ошибке в данных, но нельзя исключать, что точка является действительной. Тем не менее на стадии очистки данных расширенные методы применяются редко, и некоторые специалисты data science считают их применение чрезмерным.

Итак, после краткой сводки пришло время рассмотреть ошибки более подробно.

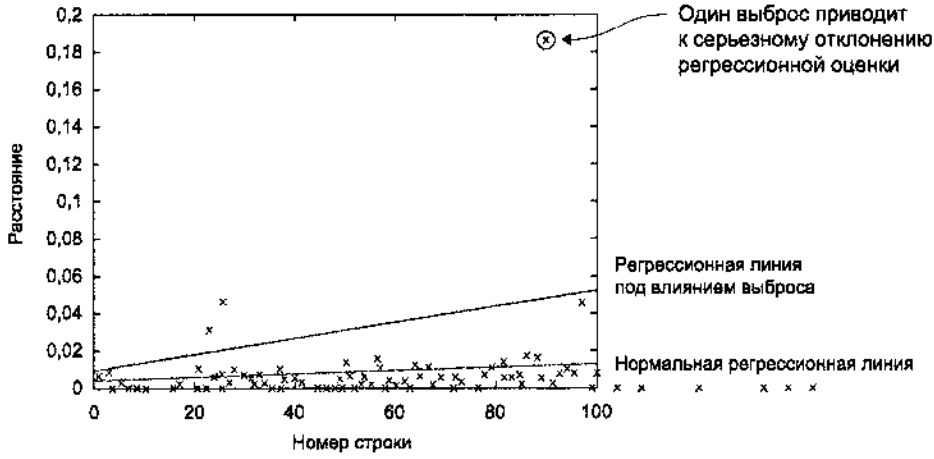


Рис. 2.5. Точка, выделенная кружком, сильно влияет на модель и заслуживает более внимательного рассмотрения, потому что она может указывать либо на область с недостаточными данными, либо на ошибку в данных, хотя она также может быть действительной точкой данных

Ошибки ввода данных

Процессы сбора и ввода данных подвержены ошибкам. Они часто требуют человеческого участия, а поскольку люди не идеальны, они допускают опечатки или отвлекаются и вносят ошибки в технологическую цепочку. Впрочем, данные, собранные машинами или компьютерами, тоже не застрахованы от ошибок. Одни ошибки появляются из-за человеческого несовершенства, другие обусловлены сбоями машин или оборудования. В частности, ко второй категории относятся ошибки, происходящие из ошибки передачи данных или ошибок в фазах извлечения, преобразования и загрузки (ETL, Extract-Transform-Load).

Для малых наборов данных все значения можно проверить вручную. Если количество классов в анализируемых переменных невелико, обнаружение ошибок в данных может осуществляться посредством группировки данных с подсчетом значений. Если некоторая переменная может принимать только два значения, «Good» и «Bad», можно создать частотную таблицу и посмотреть, действительно ли выборка состоит всего из двух значений. В табл. 2.3 значения «Godo» и «Bade» свидетельствуют о том, что как минимум в 16 случаях были обнаружены аномалии в данных.

Таблица 2.3. Обнаружение выбросов для простых переменных с использованием частотной таблицы

Значение	Количество
Good	1598647
Bad	1354468
Godo	15
Bade	1

Ошибки такого рода часто исправляются простыми командами присваивания и условными конструкциями:

```
if x == "Godo":
    x = "Good"
if x == "Bade":
    x = "Bad"
```

Избыточные пробелы

Пробелы (whitespaces) обычно трудно обнаружить при простом просмотре, но они порождают такие же ошибки, как и другие избыточные символы. Кому не приходилось терять несколько дней в проекте из-за ошибки, вызванной пробелами в конце строки? Вы приказываете программе объединить два ключа и замечаете, что в выходном файле отсутствует часть данных. После многодневных поисков ошибка наконец-то обнаруживается. И тогда наступает самое сложное: объяснить задержку руководству. Очистка в фазе ETL была проведена недостаточно тщательно, и ключи одной таблицы содержали пробелы в конце строки. Это привело к несовпадению таких ключей, как «FR » и «FR», и для некоторых наблюдений не было найдено совпадений.

К счастью, если вы знаете об этой проблеме, в большинстве языков программирования проблема решается достаточно легко. Во всех языках есть строковые функции для удаления начальных и конечных пробелов. Например, в Python для удаления начальных и конечных пробелов можно воспользоваться функцией `strip()`.

Расхождения в регистре символов

Ошибки несовпадения регистра символов встречаются достаточно часто. Во многих языках программирования `"Brazil"` и `"brazil"` считаются разными строками. В данном случае проблему можно решить применением функции, возвращающей версию строки в нижнем регистре, такой как `.lower()` в Python. Таким образом, выражение `"Brazil".lower() =="brazil".lower()` должно давать результат `true`.

Невозможные значения и проверка разумности

Проверки разумности (sanity checks) — еще одна категория проверки данных. Значение проверяется на соответствие физическим или теоретически невозможным критериям: например, человек вряд ли может иметь рост более 3 метров или возраст 299 лет. Проверка разумности может быть напрямую выражена следующими правилами:

```
check = 0 <= age <= 120
```

Выбросы

Выбросом (outlier) называется результат наблюдений, заметно отклоняющийся от других результатов, или более конкретно — результат наблюдений, который обусловлен иной логикой или иным порождающим процессом, чем другие результаты. Простейший способ поиска выбросов основан на использовании диаграмм или таблиц с минимумами и максимумами; пример представлен на рис. 2.6.

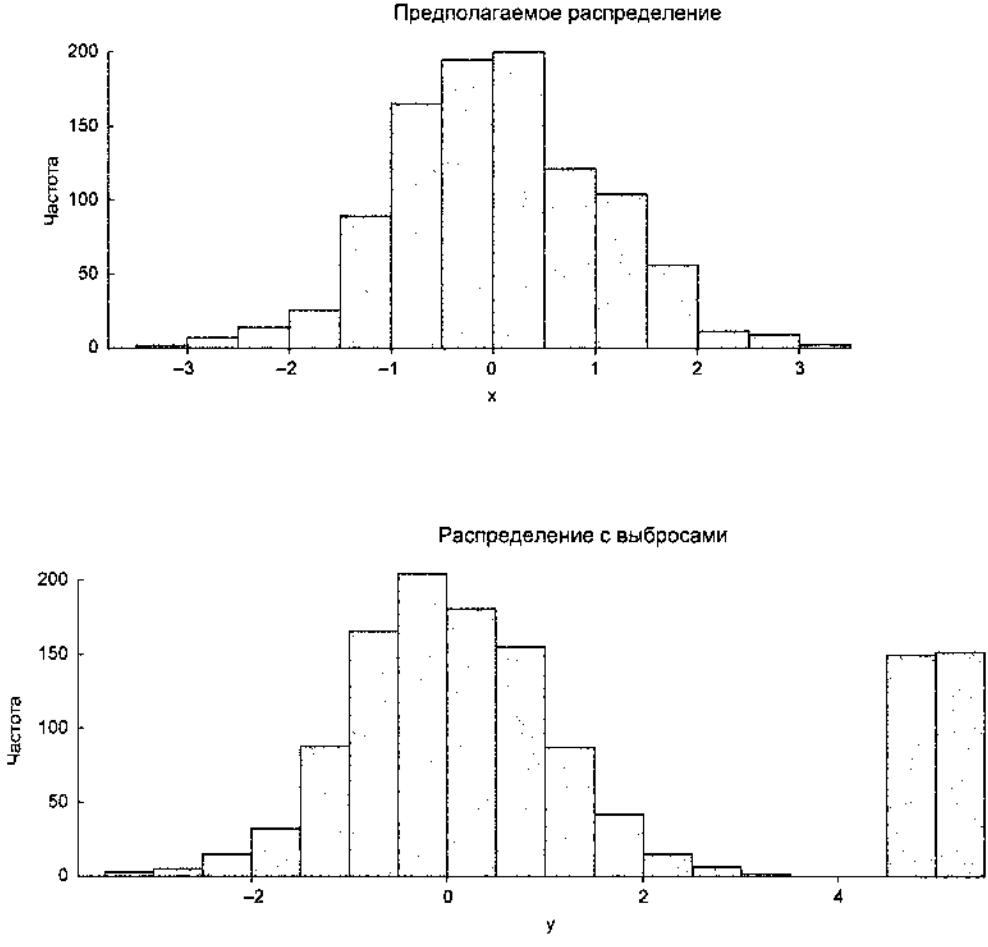


Рис. 2.6. Диаграммы распределения упрощают поиск выбросов и помогают лучше понять смысл переменных

На верхнем рисунке выбросы отсутствуют, тогда как на нижнем показана диаграмма с возможными выбросами там, где ожидается нормальное распределение (нормальное, или гауссово, распределение — самое распространенное распределение в статистике).

Из диаграммы видно, что большинство значений лежит близко к центру распределения, а по мере удаления от центра их количество падает. Если предположить нормальное распределение, высокие значения на нижнем графике могут указывать на наличие выбросов. Как было показано ранее на примере с регрессией, выбросы могут серьезно влиять на моделирование данных, поэтому начните с их расследования.

Отсутствующие значения

Отсутствующие значения не всегда создают проблему, но их следует обработать отдельно; некоторые методы моделирования не справляются с ними. Они могут свидетельствовать о том, что при сборе данных что-то пошло не так или ошибка произошла в процессе ETL. Некоторые методы, часто применяемые специалистами data science, представлены в табл. 2.4.

Таблица 2.4. Методы, используемые для обработки отсутствующих данных

Метод	Достоинства	Недостатки
Исключение пропущенных значений	Простота	Потеря информации, полученной в ходе наблюдений
Присваивание null	Простота	Не все методы моделирования и (или) реализации корректно обрабатывают null
Присваивание статического значения (например, 0 или среднего арифметического)	Простота Предотвращение потери информации от других переменных	Возможность формирования ложных оценок на основе модели
Вычисление значения на основании предполагаемого или теоретического распределения	Незначительное влияние на модель	Относительная сложность Необходимость допущений относительно данных
Моделирование значения (независимое)	Незначительное влияние на модель	Может потребовать слишком большой уверенности в модели Может создать искусственные зависимости между переменными Относительная сложность Необходимость допущений относительно данных

Выбор метода для каждого конкретного случая зависит от самого случая. Например, при небольшом количестве наблюдений вариант с исключением, пожалуй, не подойдет. Если переменная может быть описана устойчивым распределением, можно присвоить значение на основании этого распределения. Но может быть, отсутствие значения должно означать «нуль»? Например, подобная ситуация может встретиться на распродажах: если покупатель не ввел промокод, то информация о скидке отсутствует, но, скорее всего, ее можно интерпретировать как 0 (нулевая скидка).

Отклонения от свода правил

Обнаружение ошибок отклонения от свода правил или стандартизированных значений в больших наборах данных может осуществляться с помощью операций множеств. Свод правил представляет собой описание ваших данных, т. е. своего рода метаданные. В нем содержатся такие положения, как количество переменных на наблюдение, количество наблюдений и смысл специальных значений переменных. (Например, «0» может соответствовать отрицательной оценке, а «5» — крайне положительной). В своде правил также указывается тип данных, с которыми вы работаете: иерархия, граф, что-то еще.

Нужно найти значения, которые принадлежат множеству A, но не множеству B. Это те значения, которые необходимо исправить. Выбор множества как структуры данных для работы со сводом правил в программе не случаен. Старайтесь хорошо продумывать структуру данных; это сэкономит время и повысит эффективность вашей программы.

Если потребуется проверить несколько значений, лучше разместить их в таблице и воспользоваться оператором разности для проверки расхождений между таблицами. Такое решение позволит вам напрямую воспользоваться мощью базы данных (более подробная информация приводится в главе 5).

Разные единицы измерения

При слиянии двух наборов данных необходимо обращать внимание на соответствие единиц измерения. Представьте, что вы анализируете цены на бензин по всему миру. Для этого вы будете собирать данные от разных поставщиков. В одних наборах данных цена может быть указана за галлон, а в других — за литр. В данном случае проблема решается простым преобразованием.

Разные уровни агрегирования

Проблема разных уровней агрегирования отчасти напоминает проблему разных единиц измерения. Например, в одном наборе данных содержится информация за неделю, а в другом — за рабочую неделю. Ошибки такого рода обычно обнаружи-

ваются достаточно легко, и проблема решается *сведением* наборов данных (или, наоборот, *расширением*).

После устранения ошибок в данных можно переходить к комбинированию информации из разных источников. Но прежде, чем браться за эту тему, необходимо сделать небольшое отступление и указать, насколько важно провести очистку данных как можно раньше.

2.4.2. Исправляйте ошибки как можно раньше

Постарайтесь разобраться с ошибками данных на самой ранней возможной стадии сбора данных и исправлять как можно меньше ошибок в своей программе — исправления должны осуществляться на уровне источника проблем. Сбор данных — сложная задача, и организации тратят на нее миллионы долларов в надежде принять лучшие решения. Процесс сбора данных подвержен ошибкам, а в больших организациях он состоит из многих этапов и в нем задействовано много рабочих групп.

Очистку следует проводить при получении данных по многим причинам:

- Аномалии данных могут остаться незамеченными. Лица, принимающие решения, могут совершать дорогостоящие ошибки на основании неправильных данных, полученных от приложений, которые не исправили упущения в данных.
- Если ошибки не будут исправлены на ранней стадии процесса, очистку придется проводить во всех проектах, использующих эти данные.
- Ошибки в данных могут привести к формированию бизнес-процессов, которые работают не так, как было задумано. Например, двое авторов в прошлом работали в магазине розничной торговли и создали систему купонов для привлечения покупателей и повышения прибыли. В ходе проекта data science были обнаружены клиенты, которые воспользовались недочетами системы и зарабатывали деньги на покупке бакалейных товаров. Система купонов должна была стимулировать продажу сопутствующих товаров, а не бесплатную раздачу продуктов. Этот дефект наносил ущерб компании, и никто в компании о нем не знал. В данном случае данные формально были верны, но они привели к неожиданным результатам.
- Ошибки данных могут свидетельствовать о неисправности оборудования (например, разрыве линий передачи данных или неисправных датчиках).
- Дефекты программного обеспечения или процесса интеграции программного обеспечения могут привести к появлению ошибок, критических для компании. Однажды во время выполнения небольшого проекта для банка мы обнаружили, что два приложения используют разные настройки региональных стандартов, и это приводило к проблемам интерпретации чисел. Для одного приложения число 1.000 означало единицу, а для другого оно означало тысячу.

В идеале данные должны исправляться сразу же после их получения. К сожалению, специалист data science не всегда может участвовать в сборе данных, а просто попросить IT-отдел внести какие-то исправления может быть недостаточно. Если данные не были исправлены в источнике, вам придется решать проблему в своем коде. Обработка данных не завершается исправлением; возможно, вам придется заняться комбинированием поступающих данных.

И последнее замечание: всегда храните копию своих исходных данных (если это возможно). Иногда в процессе очистки данных происходят ошибки: переменные вычисляются неправильно, удаляются выбросы с интересной дополнительной информацией или в данные вносятся изменения в результате исходной неправильной интерпретации. Если у вас имеется резервная копия, можно начать заново. Для оперативно поступающих данных, обрабатываемых в момент поступления, это не всегда возможно, и вам придется согласиться на введение периода корректировки перед тем, как вы сможете использовать сохраненные данные. Впрочем, очистка отдельных наборов данных создает меньше трудностей, чем комбинирование нескольких источников в единое целое, которое содержит больше осмысленной информации.

2.4.3. Комбинирование данных из разных источников

Данные поступают из нескольких разных источников, и на этом этапе мы занимаемся интеграцией этих источников. Данные существуют в разных формах, с разным размером, типом и структурой, от баз данных и файлов Excel до текстовых документов.

Для краткости в этой главе основное внимание будет уделено данным в табличной структуре. По этой теме легко написать целые книги, и мы решили сосредоточиться на процессе data science вместо представления сценариев для всех типов данных. Однако следует помнить, что существуют и другие типы источников данных (хранилища «ключ—значение», хранилища документов и т. д.), о которых будет рассказано в других главах книги.

Разные способы комбинирования данных

Существует две основные операции, комбинирующие информацию из разных источников данных. Первая операция — *соединение* (joining): расширение наблюдений из одной таблицы информацией из другой таблицы. Вторая операция — *дополнение*: наблюдения из одной таблицы просто добавляются в другую таблицу.

При комбинировании данных можно выбрать между созданием новой физической таблицы и созданием виртуальной таблицы на основе *представления* (view). Преимущество представлений заключается в том, что они не требуют дополнительных затрат дискового пространства. Рассмотрим оба этих метода более подробно.

Соединение таблиц

Соединение таблиц позволяет скомбинировать информацию одного наблюдения, найденного в одной таблице, с информацией из другой таблицы. Основной целью является расширение одного наблюдения. Допустим, первая таблица содержит информацию о покупках клиента, а вторая — информацию о регионе, в котором ваш клиент живет. Соединение таблиц позволяет скомбинировать информацию, чтобы ее можно было использовать для вашей модели (рис. 2.7).

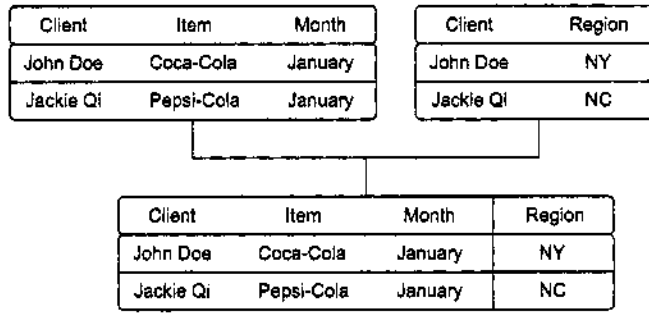


Рис. 2.7. Соединение двух таблиц

Для соединения таблиц используются переменные, представляющие один объект в двух таблицах: дату, название страниц или номер социального страхования. Эти общие поля называются *ключами*. Если ключи также однозначно определяют записи в таблице, они называются *первичными ключами*. В одной таблице хранится информация о покупках, а в другой — демографическая информация. На рис. 2.7 в обеих таблицах содержится имя клиента, и это обстоятельство позволяет легко дополнить информацию о покупках сведениями о месте жительства клиента. Люди, знакомые с Excel, заметят сходство с использованием уточняющих (lookup) функций.

Количество записей в выходной таблице зависит от конкретного типа соединения. Разные типы соединений будут представлены в книге позднее.

Дополнение таблиц

Дополнение таблиц фактически означает, что наблюдения из одной таблицы добавляются в другую таблицу. Пример дополнения представлен на рис. 2.8. Одна таблица содержит наблюдения за январь, а во второй таблице содержатся наблюдения за февраль. В результате дополнения будет получена большая таблица с данными как за январь, так и за февраль. Эквивалентной операцией из теории множеств будет объединение (union); кстати, такая команда существует в SQL, стандартном языке реляционных баз данных. В data science также часто используются другие операторы множеств, такие как разность и пересечение множеств.

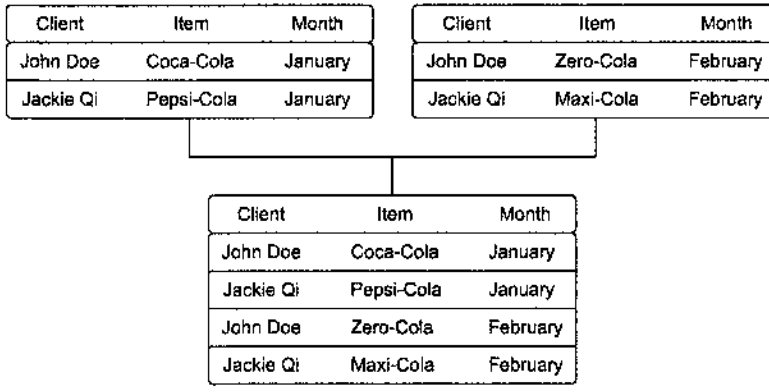


Рис. 2.8. Дополнение — стандартная операция, для которой необходимо полное совпадение структуры двух таблиц

Использование представлений для моделирования соединений и дополнений

Чтобы избежать дублирования данных, можно произвести виртуальное комбинирование данных в представлениях. В предыдущем примере мы берем данные за месяцы и комбинируем их в новую физическую таблицу. Проблема в том, что данные при этом дублируются, а следовательно, для их хранения требуется дисковое пространство. В текущем примере это не создаст проблем, но представьте, что каждая таблица содержит терабайты данных; в таком случае дублирование вряд ли возможно. По этой причине была изобретена концепция представления. Работа с представлением мало чем отличается от работы с таблицей, но само представление — не более чем виртуальная прослойка, связывающая таблицы за вас. На рис. 2.9 показано, как данные продаж по месяцам виртуально связываются в ежегодную таблицу продаж (вместо дублирования данных). Впрочем, у представлений есть и недостатки. Соединение таблиц выполняется всего один раз, а соединение, строящее представление, строится заново при каждом запросе, что требует бóльших вычислительных ресурсов, чем заранее построенная таблица.

Расширение сводных метрик

Расширение данных также может осуществляться добавлением в таблицу вычисленной информации, например общего объема продаж или процента товарных запасов в определенном регионе (табл. 2.5).

Дополнительные метрики такого рода способны открыть новую перспективу данных. Теперь табл. 2.5 дает нам сводный набор данных, который может использоваться для вычисления вклада каждого продукта в свою категорию. Новая перспектива может пригодиться в ходе исследования данных, но еще бóльшую пользу она принесет при создании моделей данных. Как обычно, все зависит от конкретной

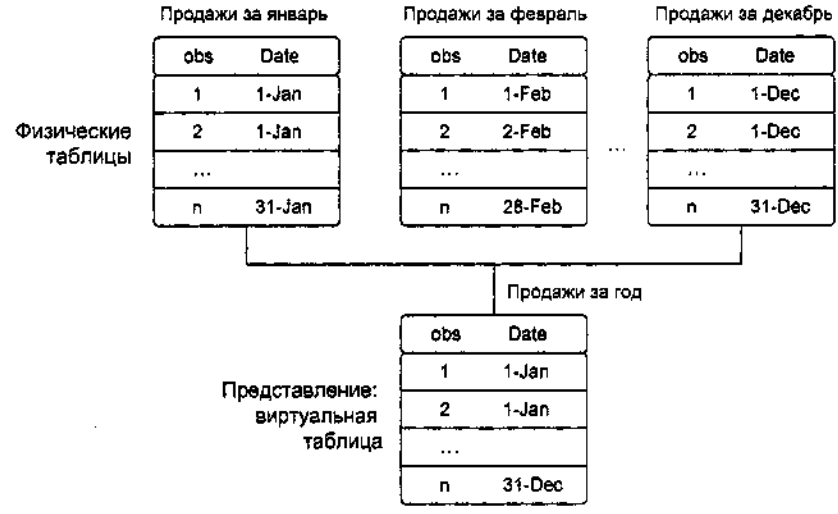


Рис. 2.9. Представление позволяет комбинировать данные без дублирования

Таблица 2.5. Прирост, объемы продаж по категориям товаров, ранг продаж — примеры производных и сводных метрик

Категория товаров	Продукт	Продажи в $	Продажи t-1 в $	Прирост	Продажи по категориям товаров	Ранг
Спорт	Спорт 1	95	98	–3,06%	215	2
Спорт	Спорт 2	120	132	–9,09%	215	1
Обувь	Обувь 1	10	6	66,67%	10	3

ситуации, но, по нашему опыту, модели с «относительными метриками» (проценты продаж, вычисленные как отношение объема продаж товара к общему объему продаж) работают эффективнее моделей, базирующихся на необработанных данных (объем продаж).

2.4.4. Преобразование данных

Некоторые модели требуют, чтобы их данные хранились в определенной форме. После очистки и интеграции данных это станет вашей следующей задачей: преобразование данных в форму, подходящую для моделирования данных.

Преобразование данных

Отношения между входной и выходной переменными не всегда линейны. Возьмем, к примеру, отношение вида $y = ae^{bx}$. Вычисление логарифма независимых переменных радикально упрощает задачу. На рис. 2.10 показано, как задача упрощается от

преобразования входных данных. В других ситуациях также две переменные могут комбинироваться в одну новую переменную.

x	1	2	3	4	5	6	7	8	9	10
log(x)	0.00	0.43	0.68	0.86	1.00	1.11	1.21	1.29	1.37	1.43
y	0.00	0.44	0.69	0.87	1.02	1.11	1.24	1.32	1.38	1.46

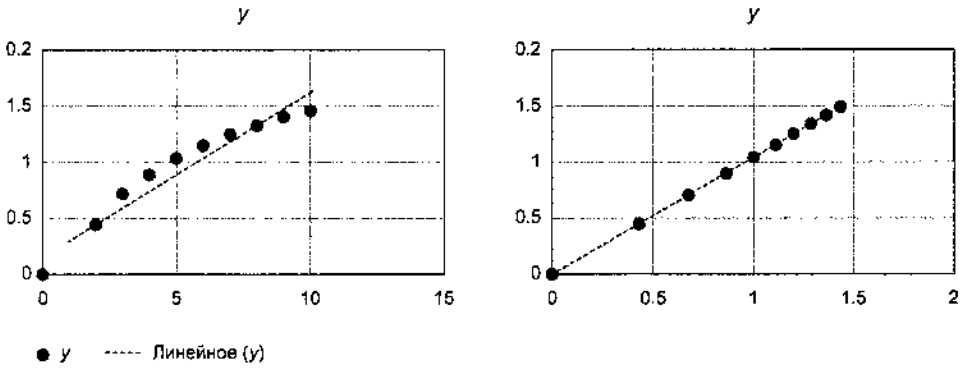


Рис 2.10. С переходом от x к log отношение между x и y становится линейным (справа) вместо нелинейного (слева)

Сокращение количества переменных

Иногда переменных оказывается слишком много и их количество приходится сокращать, потому что они не добавляют никакой новой информации в модель. Избыток переменных усложняет работу с моделью, а некоторые методы начинают хуже работать при слишком большом количестве входных переменных. Например, все методы, основанные на евклидовом расстоянии, нормально работают только для 10 и менее переменных.

Специалисты data science применяют специальные методы для сокращения количества переменных с минимальной потерей информации. Некоторые из этих методов рассматриваются в главе 3. На рис. 2.11 показано, как сокращение количества переменных упрощает понимание ключевых значений. Из графика также видно, что две переменные отвечают за 50,6% отклонения в наборе данных (component1 = = 27,8% + component2 = 22,8%). Обе эти переменные, называемые «component1» и «component2», представляют собой комбинации исходных переменных. Они являются *главными компонентами* (principal components) используемой структуры данных. Если сейчас что-то кажется вам непонятным, не огорчайтесь — *метод главных компонент* (PCA, Principal Components Analysis) будет более подробно описан в главе 3. Также из диаграммы видно, что присутствие третьей (неизвестной) переменной разбивает группу наблюдений надвое.

ЕВКЛИДОВО РАССТОЯНИЕ

Евклидово расстояние является расширением одной из теорем, с которых обычно начинается изучение «науки о треугольниках» (тригонометрии): теоремы Пифагора. Если вам известны длины двух сторон, прилегающих к прямому углу прямоугольного треугольника, длина третьей стороны (гипотенузы) вычисляется по простой формуле

$$\sqrt{(\text{side1}+\text{side2})^2}.$$

Евклидово расстояние между двумя точками на плоскости вычисляется по похожей формуле:

$$\sqrt{((x1-x2)^2+(y1-y2)^2)}.$$

Чтобы расширить формулу вычисления расстояния на большее количество измерений, добавьте в формулу координаты точек более высоких размерностей. Для трехмерного пространства формула принимает вид

$$\sqrt{((x1-x2)^2+(y1-y2)^2(z1-z2))^2}.$$

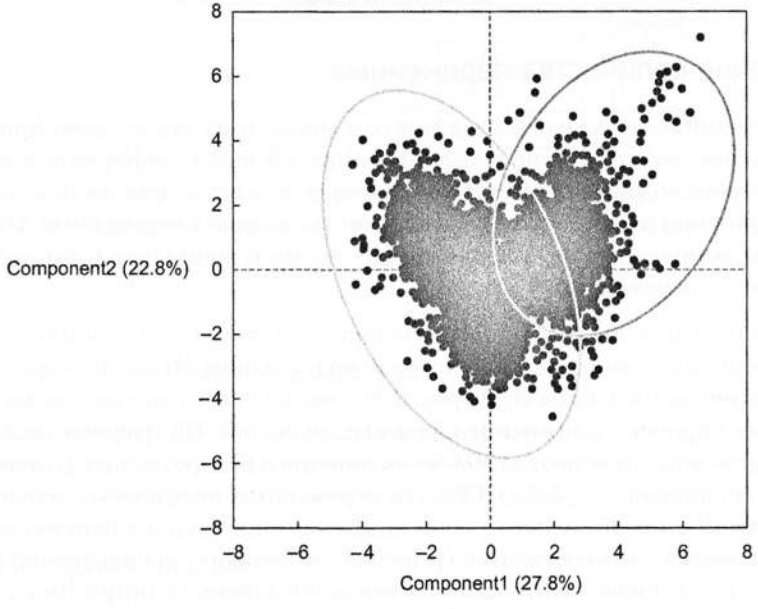


Рис. 2.11. Сокращение позволяет уменьшить количество переменных с минимальными потерями информации

Переход к вспомогательным переменным

Переменные можно преобразовать во вспомогательные (рис. 2.12). *Вспомогательные переменные* (dummy variables) принимают только одно из двух значений: `true`(1) или `false`(0). Они используются для обозначения отсутствия однозначного эффекта, который мог бы объяснить наблюдение. В данном случае мы создаем отдельные столбцы для классов, хранящихся в одной переменной; 1 обозначает присутствие класса, а 0 — его отсутствие. Например, один столбец «Дни недели» может быть преобразован в столбцы «Понедельник»—«Суббота». Значение-индикатор указывает, были ли данные наблюдения получены в понедельник; в столбец «Понедельник» заносится значение 1, а во все остальные столбцы заносится 0. Метод перехода к вспомогательным переменным применяется в моделировании и пользуется особенной популярностью у экономистов (хотя встречается и в других дисциплинах).

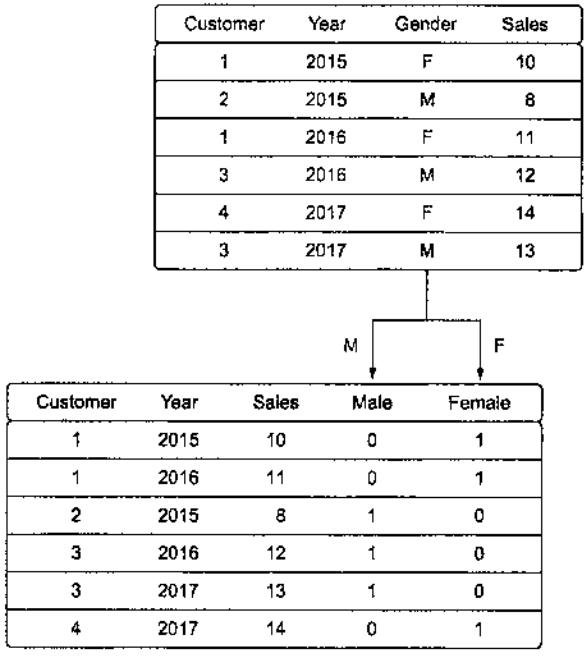


Рис. 2.12. Переход к вспомогательным переменным означает замену переменной с несколькими возможными значениями несколькими переменными, каждая из которых имеет только одно из двух возможных значений: 0 или 1

В этом разделе был представлен третий этап процесса data science — очистка, преобразование и интеграция данных, в результате которого «сырые» данные превращаются в полезную входную информацию для фазы моделирования. Целью следующего этапа процесса data science является лучшее понимание контента данных и отношений между переменными и наблюдениями; мы займемся этой темой в следующем разделе.

2.5. Этап 4: Исследовательский анализ данных

В фазе исследовательского анализа данных происходит углубленное изучение данных (рис. 2.13). В графическом виде информация воспринимается намного проще, поэтому для понимания данных и взаимодействий переменных применяются в основном графические методы. Целью этой фазы является исследование данных, поэтому в фазе исследовательского анализа данных необходимо сохранять объективность и смотреть в оба. Очистка данных непосредственной целью не является, однако в этой фазе нередко обнаруживаются аномалии, упущенные ранее; в таком случае отступите на шаг назад и исправьте их.

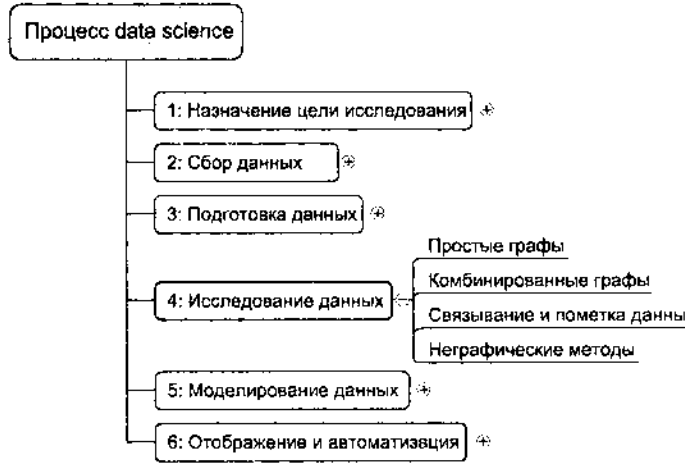


Рис. 2.13. Этап 4: Исследование данных

В этой фазе применяется широкий спектр методов визуализации, от простых графиков или столбцовых диаграмм, как на рис. 2.14, до более сложных диаграмм Сэнки и сетевых графов. Иногда бывает полезно составить из нескольких простых диаграмм одну сложную, чтобы еще лучше разобраться в сути данных. Также возможно построение анимированных или интерактивных диаграмм – с такими диаграммами работать проще (и, откровенно говоря, гораздо интереснее). Пример интерактивной диаграммы Сэнки доступен по адресу *http://bost.ocks.org/mike/sankey/*.

Майк Бостоk приводит интерактивные примеры почти для всех разновидностей диаграмм. Его сайт заслуживает внимания, хотя большинство примеров ориентировано скорее на отображение данных, нежели на их исследование.

Объединение этих диаграмм еще лучше раскрывает суть данных (рис. 2.15).

Наложение диаграмм также часто применяется на практике. На рис. 2.16 несколько простых диаграмм объединяются в диаграмму Парето («диаграмма 80/20»).

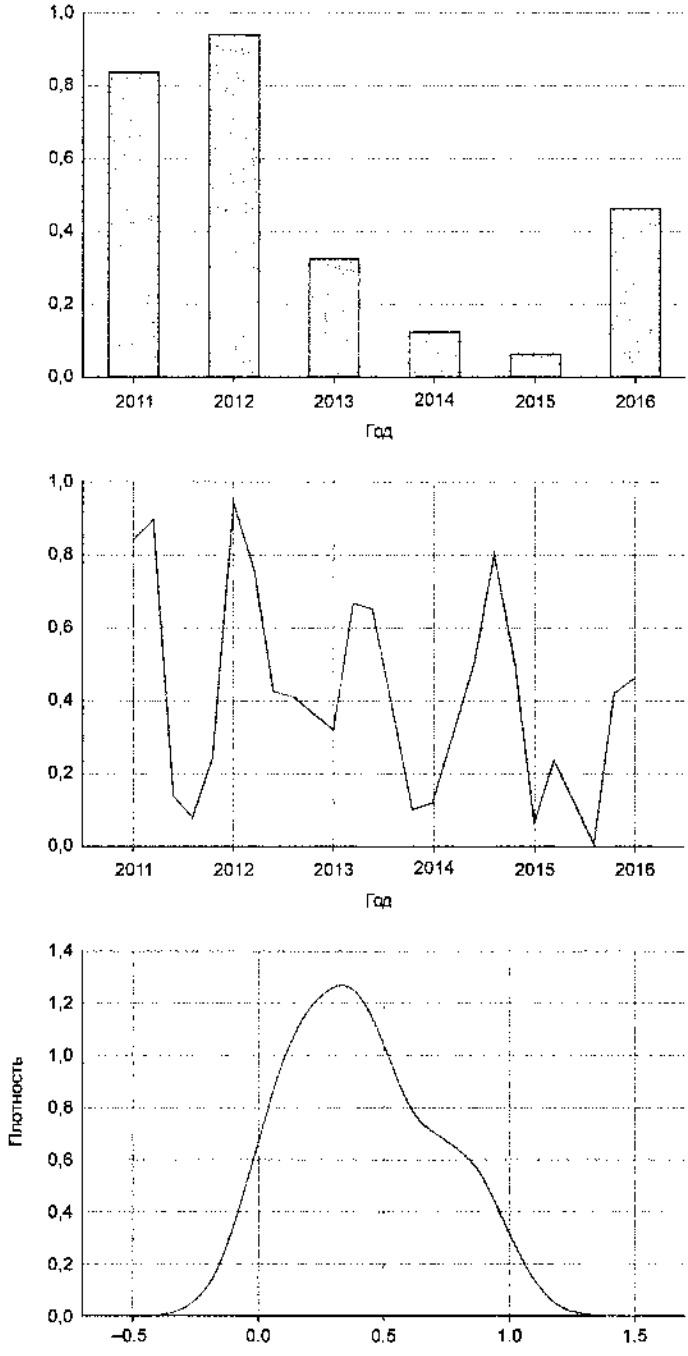


Рис. 2.14. Сверху вниз: столбцовая диаграмма, линейный график, кривая распределения — примеры диаграмм, используемых в исследовательском анализе

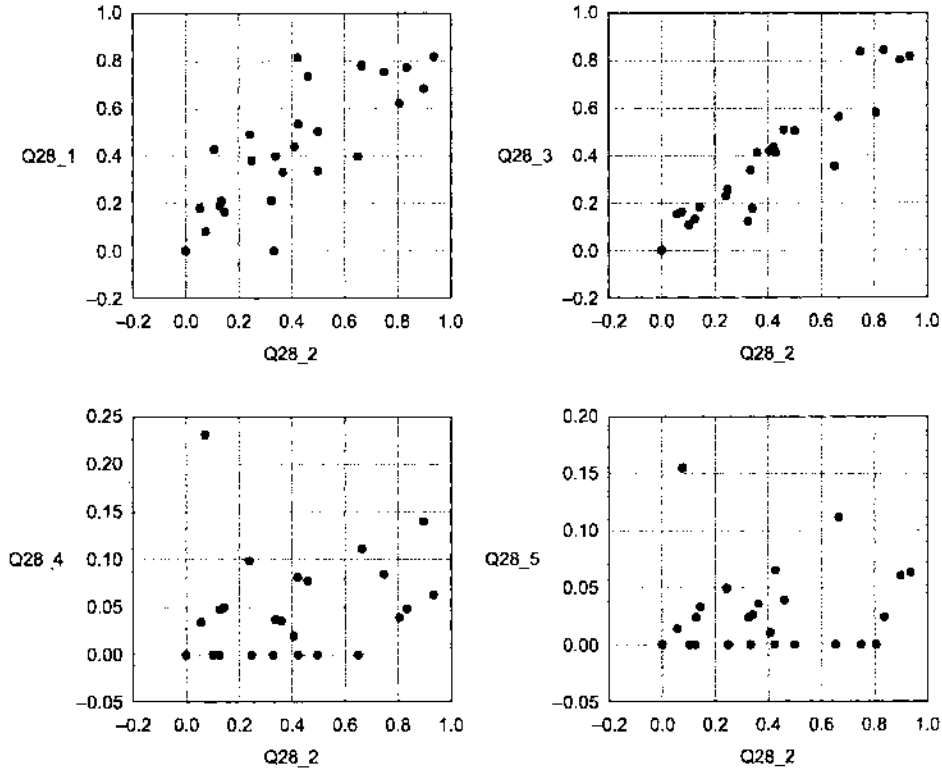


Рис. 2.15. Размещение диаграмм вблизи друг от друга помогает лучше понять структуру данных с несколькими переменными

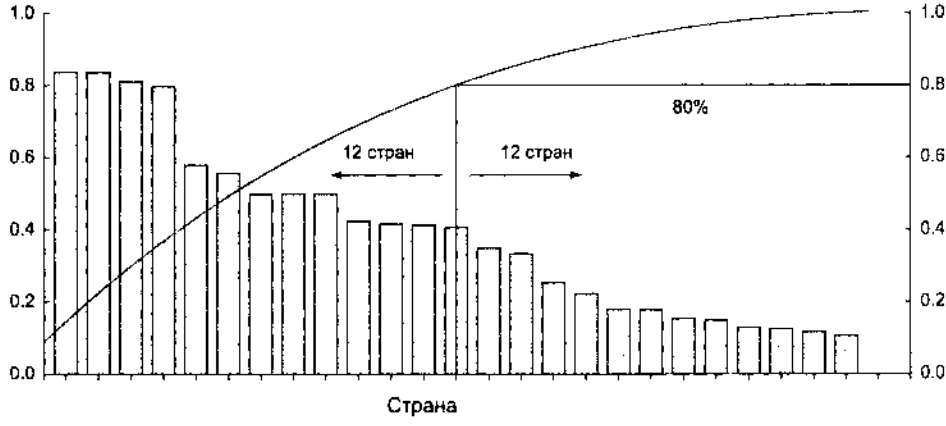


Рис. 2.16. Диаграмма Парето представляет собой комбинацию значений и кумулятивного распределения. Диаграмма наглядно показывает, что на первые 50% стран приходится чуть менее 80% общего вклада. Если бы на диаграмме была представлена покупательная способность, а вы занимались продажей дорогостоящих товаров, вероятно, тратить маркетинговый бюджет на все страны было бы неэффективно, разумнее начать с первых 50%

На рис. 2.17 представлен другой метод: *связывание и пометка данных* (brushing and linking). Разные диаграммы и таблицы (или представления) объединяются и связываются таким образом, что изменения в одной диаграмме автоматически переносятся на другие. Нетривиальный пример такого рода приведен в главе 9. Подобные интерактивные исследования данных упрощают выявление новых глубинных причин и взаимосвязей.

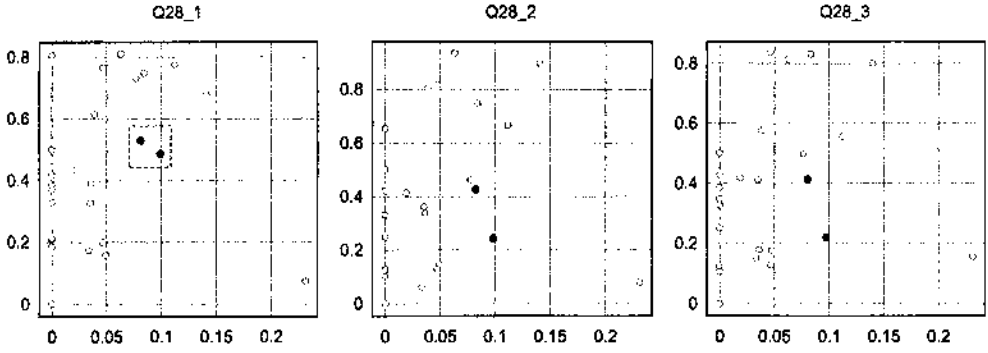


Рис. 2.17. Метод связывания и пометки данных позволяет выбрать наблюдения на одной диаграмме с выделением тех же наблюдений на другой диаграмме

На рис. 2.17 представлены средние баллы по странам. Диаграмма не только обозначает высокую степень корреляции между ответами, но и позволяет увидеть, что при выделении нескольких точек на одной диаграмме эти точки соответствуют похожим точкам на других диаграммах. В данном случае выделенные точки на левой диаграмме соответствуют точкам на средней и правой диаграммах, хотя между средней и правой диаграммами это соответствие более очевидно.

Две другие важные разновидности диаграмм — гистограмма на рис. 2.18 и коробчатая диаграмма на рис. 2.19.

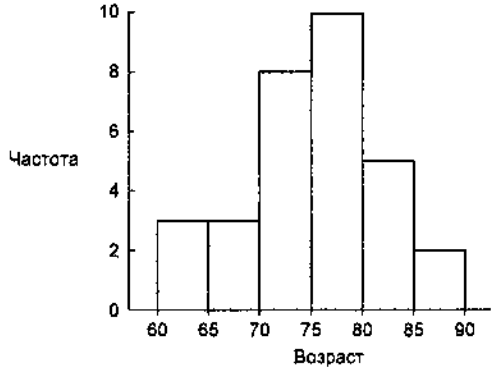


Рис. 2.18. Пример гистограммы: численность людей в возрастных группах с интервалом в 5 лет

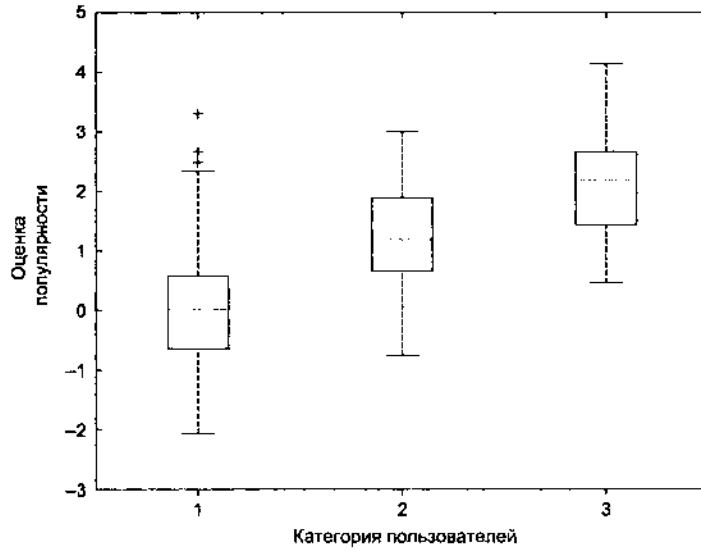


Рис. 2.19. Пример коробчатой диаграммы: у каждой категории пользователей существует распределение оценок, выставленных за определенное изображение на сайте фотографий

На гистограмме переменная делится на дискретные категории, количества вхождений в каждую категорию суммируются и отображаются на диаграмме. С другой стороны, коробчатая диаграмма не показывает количество наблюдений, но дает представление о распределении внутри категорий. На ней могут одновременно отображаться минимум, максимум, медиана и другие характеристики распределения.

Методы, упомянутые в этой фазе, в основном имеют визуальную природу, но на практике анализ не ограничивается методами визуализации. Сведение в таблицы, кластеризация и другие методы моделирования также могут быть частью исследовательского анализа. Даже построение простых моделей может быть частью этого шага.

Итак, фаза исследования данных завершена, а вы получили хорошее представление о своих данных. Пора переходить к следующей фазе: построению моделей.

2.6. Этап 5: Построение моделей

При наличии очищенных данных и хорошем понимании контента вы готовы к построению моделей с целью улучшения прогнозов, проведения классификации объектов или лучшего понимания моделируемой системы. Эта фаза является намного более целенаправленной, чем этап исследовательского анализа, потому что вы знаете, что ищете и каким должен быть результат. На рис. 2.20 представлены основные компоненты построения модели.

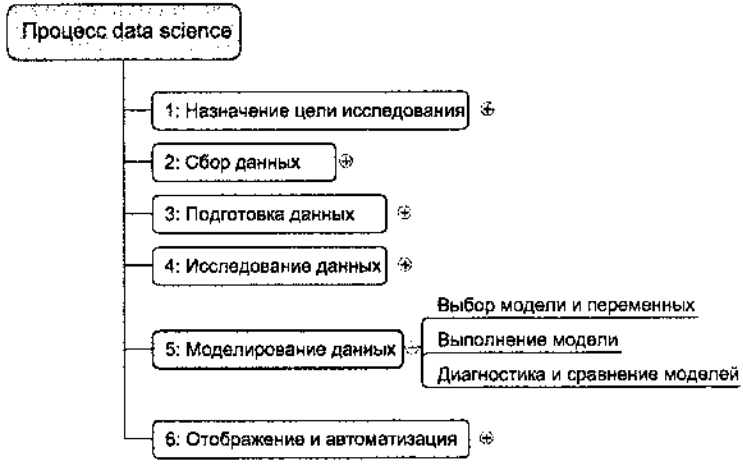


Рис. 2.20. Этап 5: Моделирование данных

Методы, которые будут использоваться, позаимствованы из области машинного обучения, обработки/анализа данных и статистики. В этой главе рассматривается лишь малая часть существующих методов, в главе 3 они будут представлены более подробно. Давать здесь нечто большее концептуального введения значило бы выйти за рамки книги, но и этой информации достаточно для начала; 20% методов помогут вам в 80% случаев, потому что методы перекрываются в отношении предполагаемой цели. Часто они достигают своих целей в основном похожими, но немного различающимися способами.

Построение модели является итеративным процессом. Способ построения модели зависит от того, принадлежите ли вы к школе классической статистики или же к несколько более современной школе машинного обучения, а также от типа применяемого метода. В любом случае процесс построения большинства моделей состоит из следующих шагов:

1. Выбор метода моделирования и переменных для включения в модель.
2. Выполнение модели.
3. Диагностика и сравнение моделей.

2.6.1. Выбор модели и переменных

Вам нужно выбрать переменные, которые должны быть включены в модель, и метод моделирования. Результаты, полученные в ходе исследовательского анализа, должны были уже дать достаточно четкое представление о том, какие переменные позволят построить хорошую модель. Известно много методов моделирования, и выбор правильной модели для задачи — ваша обязанность. Вы должны учесть

качество модели и соответствие проекта всем требованиям для использования модели, а также другие факторы:

- Должна ли модель быть вынесена в производственную среду, и, если должна, насколько просто она будет реализовываться?
- С какими трудностями связано сопровождение модели: долго ли она останется актуальной, если не менять ее?
- Должна ли модель быть простой для объяснения?

Когда предварительные размышления будут завершены, наступает время действовать.

2.6.2. Выполнение модели

После того как модель будет выбрана, ее необходимо реализовать в программном коде.

ПРИМЕЧАНИЕ

Здесь мы впервые будем заниматься выполнением кода Python, поэтому убедитесь в том, что виртуальная среда настроена и готова к использованию. Умение настраивать виртуальную среду относится к числу обязательных навыков, но если вы занимаетесь этим впервые, обратитесь к приложению Г.

Весь код этой главы можно загрузить по адресу *https://www.manning.com/books/introducing-data-science*. К этой главе прилагаются файлы ipython (.ipynb) и Python (.py).

К счастью, в большинстве языков программирования (таких, как Python) уже существуют специализированные библиотеки, например StatsModels или Scikit-learn. Эти пакеты поддерживают многие популярные методы моделирования. Программирование модели во многих случаях является делом нетривиальным, так что наличие таких библиотек ускорит процесс. Как видно из следующего кода, использовать линейную регрессию (рис. 2.21) с StatsModels или Scikit-learn до-

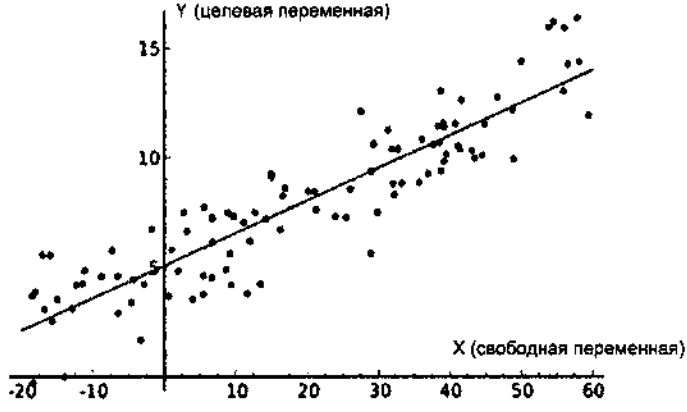


Рис. 2.21. Метод линейной регрессии пытается подобрать линию с минимальным расстоянием до каждой точки

статочно просто. Самостоятельная реализация потребует существенно больших усилий даже для простых методов. В листинге 2.1 приведен пример выполнения модели линейного прогнозирования на программном уровне.

Листинг 2.1. Выполнение модели линейного прогнозирования для полуслучайных данных

```
import statsmodels.api as sm
import numpy as np
predictors = np.random.random(1000).reshape(500,2)
target = predictors.dot(np.array([0.4, 0.6])) + np.random.random(500)
lmRegModel = sm.OLS(target,predictors)
result = lmRegModel.fit()
result.summary()
```

Импортирование необходимых модулей Python.

Создание случайных данных для свободных (x) и целевых переменных (y) модели. Прогностические параметры используются для создания целевых значений, чтобы создать корреляцию.

Подбор линейной регрессии для данных.

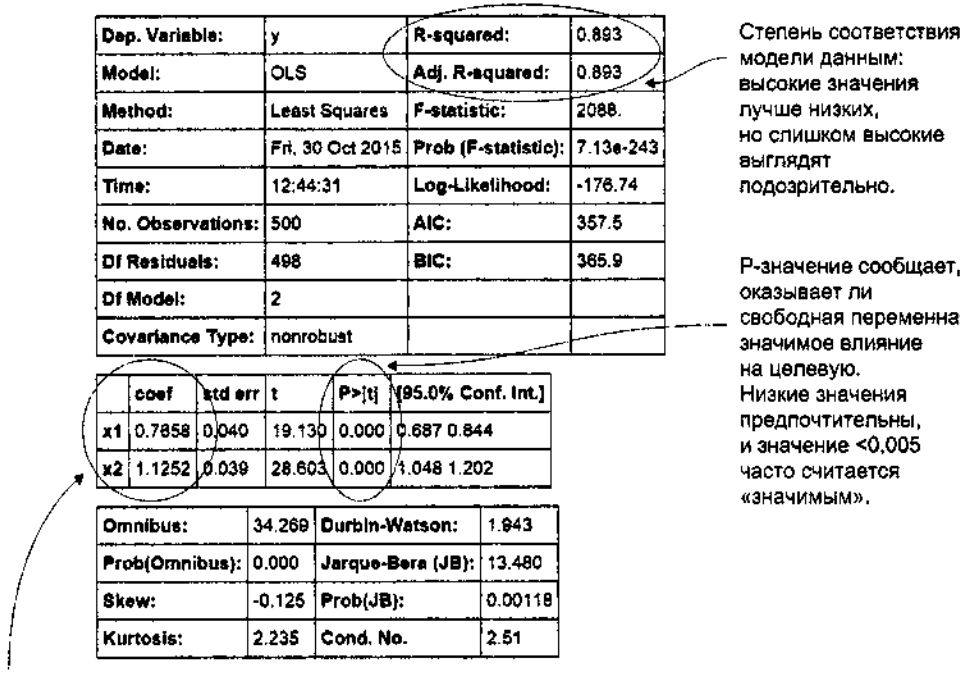
Вывод статистики соответствия модели.

Ладно, мы здесь смухлевали, и довольно основательно. Мы создали свободные значения, которые вроде бы должны прогнозировать поведение целевых переменных. Линейная регрессия предполагает линейное отношение между x (свободная переменная) и y (целевая переменная) (см. рис. 2.21).

Однако при этом мы создали целевую переменную на основании значения независимой, добавив к ней небольшую долю случайности. Разумеется, в результате получилась модель с высокой степенью соответствия. Вызов `results.summary()` выводит таблицу на рис. 2.22. Разумеется, точные результаты зависят от сгенерированных случайных значений.

Пока не будем обращать внимание на большую часть выводимых данных и сосредоточимся на самом важном:

- *Степень соответствия модели* — для оценки степени соответствия используется коэффициент детерминации (R-квадрат) или скорректированный (adjusted) коэффициент детерминации. Эта метрика обозначает степень разброса данных, отраженного в модели. Разность между скорректированным и простым коэффициентом детерминации в данном случае минимальна, потому что скорректированный коэффициент равен сумме простого коэффициента и штрафа за сложность модели. Модель усложняется с введением большого количества переменных. При наличии простой модели сложная модель не нужна, поэтому скорректированный коэффициент детерминации «наказывает» за излишнее усложнение. В любом случае значение 0,893 достаточно большое (так и должно быть, потому что мы смухлевали). Существуют разные эмпирические правила, но для экономических моделей значения свыше 0,85 обычно считаются хорошими. Если вы хотите однозначной победы, потребуются значения свыше 0,90. Впрочем, в ходе исследований часто встречаются модели с очень низкой степенью соответствия (даже <0,2). В данном случае важнее влияние введенных свободных переменных.



Dep. Variable:	y	R-squared:	0.893
Model:	OLS	Adj. R-squared:	0.893
Method:	Least Squares	F-statistic:	2088.
Date:	Fri, 30 Oct 2015	Prob (F-statistic):	7.13e-243
Time:	12:44:31	Log-Likelihood:	-176.74
No. Observations:	500	AIC:	357.5
Df Residuals:	498	BIC:	365.9
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
x1	0.7658	0.040	19.130	0.000	0.687 0.844
x2	1.1252	0.039	28.603	0.000	1.048 1.202

Omnibus:	34.269	Durbin-Watson:	1.943
Prob(Omnibus):	0.000	Jarque-Bera (JB):	13.480
Skew:	-0.125	Prob(JB):	0.00118
Kurtosis:	2.235	Cond. No.	2.51

Рис. 2.22. Выходная информация модели линейной регрессии

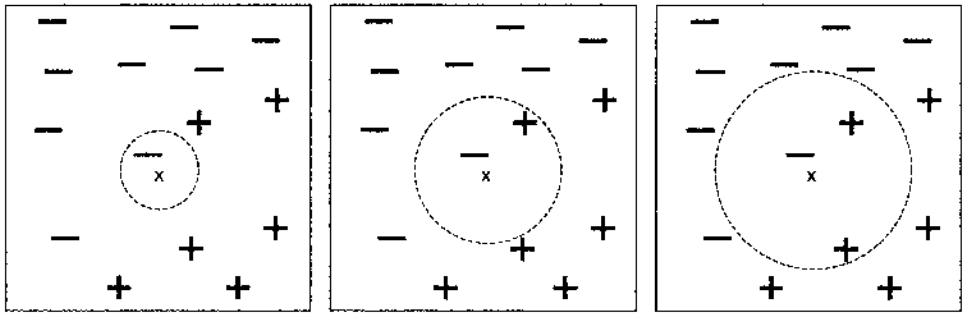
- *Коэффициенты свободных переменных* — в линейной модели эти значения интерпретируются легко. В нашем примере увеличение x_1 на 1 приводит к изменению y на 0,7658. Легко видеть, что удачный выбор свободной переменной может проложить путь к Нобелевской премии, даже если модель в целом никуда не годится. Если, например, вы определили, что некий ген является значимым фактором возникновения рака, эта информация крайне важна, даже если ген сам по себе не определяет, заболеет человек раком или нет. В данном примере мы имеем дело с классификацией, а не регрессией, но суть остается неизменной: обнаружить влияние в научных исследованиях важнее (не говоря уже о реалистичности), чем найти модель с идеальным соответствием. Но как определить, что ген оказывает влияние? Характеристика для его оценки называется значимостью.

- *Значимость свободных переменных* — коэффициенты удобны и понятны, но в некоторых случаях не существует убедительных доказательств наличия влияния. Для оценки этой величины применяется *p-значение* (p-value). Здесь можно было бы долго объяснять, что такое ошибки 1-го и 2-го типа, но вкратце ситуация выглядит так: если p-значение меньше 0,05, то переменная обычно считается значимой. Откровенно говоря, значение выбрано произвольно — оно

указывает на то, что с 5%-ной вероятностью свободная переменная не оказывает влияния. Вас устраивает 5%-ная вероятность ошибки? Дело ваше. Некоторые специалисты вводили понятие чрезвычайно значимых ($p<0{,}01$) и минимально значимых порогов ($p<0{,}1$).

Линейная регрессия работает, если нужно спрогнозировать значение, но что, если потребуется провести классификацию? Тогда на помощь приходят *модели классификации*, из которых наибольшей известностью пользуется модель *k ближайших соседей*.

Как видно из рис. 2.23, модель *k* ближайших соседей ищет помеченные точки рядом с непомеченной и на основании полученных результатов прогнозирует, какой должна быть метка.



(a) 1-ближайший сосед (b) 2-ближайший сосед (c) 3-ближайший сосед

Рис. 2.23. Метод k-ближайших соседей проверяет до k ближайших точек для формирования прогноза

Попробуем применить этот метод в коде Python при помощи библиотеки Scikit, как показано в следующем листинге.

Листинг 2.2. Выполнение классификации методом k ближайших соседей для полуслучайных данных

```
from sklearn import neighbors
predictors = np.random.random(1000).reshape(500,2)
target = np.around(predictors.dot(np.array([0.4, 0.6])) +
         np.random.random(500))
clf = neighbors.KNeighborsClassifier(n_neighbors=10)
knn = clf.fit(predictors,target)
knn.score(predictors, target)
```

Импортирование модулей.

Создание случайных свободных данных и полуслучайных целевых данных на основании свободных.

Классификация по модели 10 ближайших соседей.

Получение метрики соответствия модели: какой процент классификации был правильным?

Как и прежде, мы строим случайные коррелированные данные и — сюрприз, сюрприз! — получаем 85% случаев с правильной классификацией. Но чтобы рассмотреть результат более подробно, необходимо провести оценку модели, т. е. применить ее к данным для получения прогноза:

```
prediction = knn.predict(predictors)
```

Теперь используем прогноз и сравним его с реальной ситуацией при помощи *матрицы несоответствий* (confusion matrix):

```
metrics.confusion_matrix(target,prediction)
```

Мы получаем матрицу 3×3, показанную на рис. 2.24.

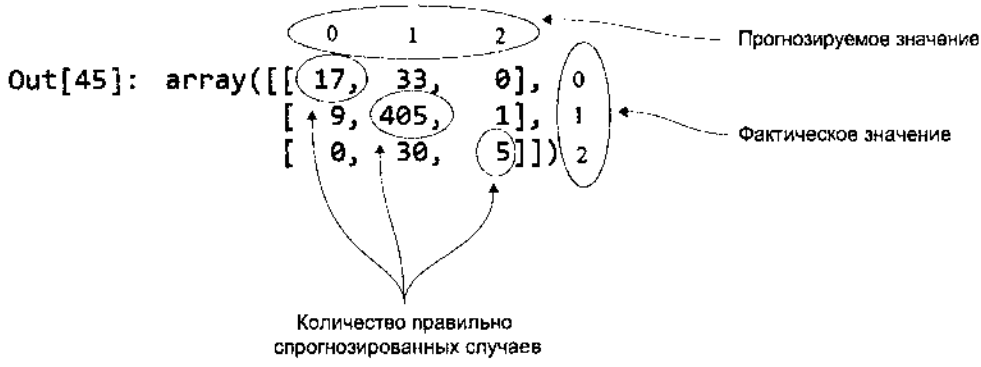


Рис. 2.24. Матрица несоответствий показывает, сколько случаев было классифицировано правильно и сколько неправильно, для чего прогноз сравнивается с реальными значениями. *Примечание*: классы (0, 1, 2) добавлены на иллюстрацию для ясности

Матрица несоответствий показывает, что мы правильно спрогнозировали 17+405+5 случаев, и это хорошо. Но стоит ли этому удивляться? Нет — по следующим причинам:

- Во-первых, у классификатора всего три варианта; пометка разности последним значением `np.around()` округляет данные до ближайшего целого. В данном случае это 0, 1 или 2. Всего с тремя вариантами трудно показать результат хуже 33% правильных классификаций при 500 предположениях, даже при случайном распределении (как при подбрасывании монетки).
- Во-вторых, мы снова смухлевали, связав целевую переменную со свободными. Из-за этого в большинстве наблюдений мы получаем 1. Просто предполагая «1» в каждом случае, мы уже получаем сходный результат.

- Прогноз сравнивался с реальными значениями, это верно, но мы никогда не пытались проводить классификацию на свежих данных. Прогноз строился на основе тех же данных, которые использовались для построения модели. Подыграть себе можно, но такой результат не позволяет судить о том, сработает ли модель на действительно новых данных. Для этого нам понадобится *контрольная выборка* (holdout sample), о которой будет рассказано в следующем разделе.

Не обманывайте себя — сам по себе этот код чудес не сотворит. Вероятно, вам придется потратить время на то, чтобы правильно настроить процесс моделирования и все его параметры.

Честно говоря, лишь небольшое подмножество методов имеет готовые реализации на Python. Однако вы можете относительно легко использовать в Python модели, доступные в R, с помощью библиотеки RPy. RPy предоставляет интерфейс из Python к R. R — бесплатная среда программирования, широко применяемая в статистических вычислениях. Если вы еще не знакомы с R, стоит по крайней мере взглянуть, потому что в 2014 году R оставался одним из самых популярных (если не самым популярным) языков программирования для data science. За дополнительной информацией обращайтесь по адресу *http://www.kdnuggets.com/polls/2014/languages-analytics-data-mining-data-science.html*.

2.6.3. Диагностика и сравнение моделей

В ходе анализа вы будете строить разнообразные модели, из которых затем будет выбираться лучшая модель на основании нескольких критериев. Работа с контрольной выборкой помогает выбрать самую эффективную модель. Контрольная выборка представляет собой часть данных, которая была исключена при построении модели, и может использоваться для последующей оценки модели. Принцип простой: модель должна работать для незнакомых данных. Вы используете малую часть данных для оценки модели, а другая часть — контрольная выборка — исключается из вычислений. Затем модель проверяется на незнакомых данных и вычисляются метрики погрешности для ее оценки. Существует много разных метрик погрешности; на рис. 2.25 представлена общая идея сравнения моделей. В этом примере в качестве метрики погрешности используется *среднеквадратичная погрешность*.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

Рис. 2.25. Формула для вычисления среднеквадратичной погрешности

Метрика среднеквадратичной погрешности проста: для каждого прогноза проверяется его отклонение от истинного значения, отклонение возводится в квадрат, а погрешности всех прогнозов суммируются.

На рис. 2.26 сравнивается эффективность двух моделей по прогнозированию зависимости размера заказа от цены. Первая модель `size=3*price`, а вторая `size=10`. Чтобы оценить модели, мы используем 800 случайно выбранных наблюдений из 1000 (или 80%), не показывая другие 20% данных модели. После того как модель пройдет обучение, мы прогнозируем значения других 20% переменных на основании тех, для которых уже известно истинное значение, и вычисляем погрешность модели. Затем выбирается модель с меньшей погрешностью. В данном случае выбирается модель 1, потому что она обеспечивает меньшую общую ошибку.

	n	Size	Price	Predicted model 1	Predicted model 2	Error model 1	Error model 2
	1	10	3				
	2	15	5				
	3	18	6				
	4	14	5				
	...	...					
80% train	800	9	3				
	801	12	4	12	10	0	2
	802	13	4	12	10	1	3
	...						
	999	21	7	21	10	0	11
20% test	1000	10	4	12	10	–2	0
				Total		5861	110225

Рис. 2.26. Контрольная выборка помогает сравнивать модели и проверяет, что результаты обобщаются для данных, которые пока неизвестны для модели

Многие модели делают сильные предположения (такие, как независимость входных факторов), и вы должны убедиться в том, что эти предположения действительно выполняются. Этот процесс называется *диагностикой модели*.

В этом разделе приводится краткое описание основных шагов построения действительной модели. После получения рабочей модели все готово для перехода к последней стадии.

2.7. Этап 6: Представление результатов и построение приложений на их основе

После того как вы успешно проанализировали данные и построили эффективную модель, полученные результаты можно представить миру (рис. 2.27). Это очень

важная часть; долгие часы усердной работы окупились, и вы можете объяснить полученные результаты ключевым участникам.

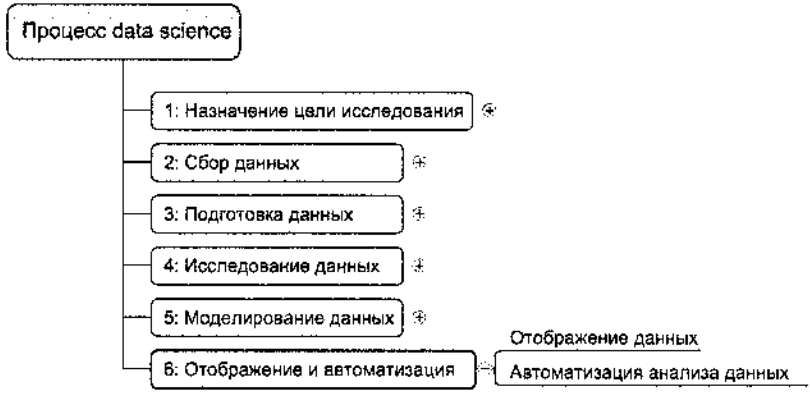


Рис. 2.27. Этап 6: Отображение и автоматизация

Иногда ваша работа производит такое впечатление, что вам приходится повторять ее снова и снова, потому что слушатели придают высокое значение прогнозам вашей модели или полученной информации. По этой причине стоит подумать об автоматизации ее модели. Это не всегда означает, что вам придется каждый раз заново повторять весь анализ. Иногда бывает достаточно реализовать только оценку модели; в других случаях приходится строить приложение, которое автоматически обновляет отчеты, таблицы Excel или презентации PowerPoint. Именно на последней стадии процесса data science ваши навыки работы с людьми принесут наибольшую пользу. Более того, мы рекомендуем обратиться к специализированной литературе и другой информации по теме и ознакомиться с ней; в конце концов, какой смысл проделывать всю эту тяжелую работу, если никто не будет слушать то, что вы говорите?

Если все было сделано правильно, к этому моменту у вас имеется работоспособная модель и довольные заказчики, и на этом главу можно считать законченной.

2.8. Итоги

В этой главе вы узнали, что процесс data science состоит из шести этапов:

- *Назначение цели исследования* — определение ответов на вопросы «что», «почему» и «как», относящихся к вашему проекту, в проектном задании.
- *Сбор данных* — поиск и получение доступа к данным, необходимым для проекта. Данные либо хранятся в компании, либо приобретаются у третьих сторон.

- *Подготовка данных* — проверка и исправление ошибок, расширение данных информацией из других источников и их преобразование в формат, подходящий для ваших моделей.
- *Исследование данных* — более глубокое изучение данных с использованием описательной статистики и визуальных средств.
- *Моделирование данных* — применение средств машинного обучения и статистических методов для достижения цели проекта.
- *Отображение и автоматизация* — представление результатов ключевым участникам проекта и адаптация процесса анализа для повторного использования и интеграции с другими инструментами.

Машинное обучение

В этой главе:

- ✓ Почему специалисты data science используют машинное обучение.
- ✓ Важнейшие библиотеки Python для машинного обучения.
- ✓ Процесс построения модели.
- ✓ Методы машинного обучения.
- ✓ Практический опыт машинного обучения.

А вы знаете, как компьютеры учатся защищать вас от злоумышленников? Они отфильтровывают более 60% сообщений электронной почты, а со временем обучаются и начинают защищать вас еще лучше.

Можно ли явно научить компьютер распознавать людей на фотографии? Запрограммировать все возможные способы распознавания людей — решение возможное, но непрактичное, потому что количество возможных вариантов почти безгранично. Чтобы успешно справиться с этой задачей, необходимо добавить в ваш инструментарий новую дисциплину — *машинное обучение*. Этой теме посвящена текущая глава.

3.1. Что такое машинное обучение и почему оно важно для вас?

> Машинное обучение — область исследований, которая наделяет компьютеры возможностью проявлять поведение, которое не было заложено в них явно.
>
> *Артур Сэмюэл, 1959 год*[1]

Определение машинного обучения, сформулированное Артуром Сэмюэлом, часто цитируется; оно гениально в своей широте, но вопрос о том, *как* учатся компьютеры, остается открытым. Чтобы реализовать концепцию машинного обучения, эксперты разрабатывают алгоритмы общего назначения, которые могут применяться к большим классам задач обучения. Если вам потребуется решить *конкретную задачу*, достаточно передать алгоритму более *конкретные данные*. В определенном смысле речь идет об обучении на примерах. В большинстве случаев компьютер использует данные как источник информации, сравнивает свой результат с желаемым, а затем вносит исправления. Чем больше данных («опыта») накапливает компьютер, тем лучше он справляется со своей работой — как и человек.

Если рассматривать машинное обучение как процесс, можно привести следующее содержательное определение:

> Машинное обучение — процесс, в котором точность работы компьютера повышается по мере сбора данных и извлечения из них информации.
>
> *Майк Роберте*[2]

Например, чем больше пользователь напишет текстовых сообщений на своем телефоне, тем больше телефон узнает о словаре и тем точнее и быстрее он может прогнозировать (автоматически завершать) вводимые слова.

В более широком научном контексте машинное обучение является подобластью искусственного интеллекта, тесно связанной с прикладной математикой и статистикой. Возможно, эти описания звучат немного абстрактно, но машинное обучение находит много применений в повседневной жизни.

---

[1] Хотя следующая статья часто указывается в качестве источника этой цитаты, она отсутствует в перепечатке статьи от 1967 года. Авторам не удалось проверить или найти точный источник цитаты. См. Arthur L. Samuel, «Some Studies in Machine Learning Using the Game of Checkers,» IBM Journal of Research and Development 3, no. 3 (1959): 210–229.

[2] Майк Роберте — научный редактор этой книги. Спасибо тебе, Майк.

3.1.1. Применение машинного обучения в data science

Задачи регрессии и классификации играют исключительно важную роль для специалиста data science. Одним из главных инструментов, применяемых специалистами data science для достижения этих целей, является машинное обучение. Среди практических применений регрессии и автоматической классификации можно выделить следующие:

- Поиск нефтяных месторождений, золотых рудников и мест археологических раскопок на основании информации о существующих местах (классификация и регрессия).
- Поиск имен людей или названий географических мест в тексте (классификация).
- Идентификация людей по фотографиям или по записи голоса (классификация).
- Распознавание птиц по голосам (классификация).
- Идентификация выгодных клиентов (регрессия и классификация).
- Активное выявление частей автомобиля, в которых с большой вероятностью произойдет поломка (регрессия).
- Идентификация опухолей и заболеваний (классификация).
- Прогнозирование суммы, которую человек потратит на продукт *X* (регрессия).
- Прогнозирование количества извержений вулкана за период времени (регрессия).
- Прогнозирование годового дохода компании (регрессия).
- Предсказание команды, которая победит в Лиге чемпионов по футболу (классификация).

Время от времени специалисты data science строят *модель* (абстракцию реальности), которая позволяет понять внутренние процессы явления. Если целью модели является не прогнозирование, а интерпретация, процесс называется *анализом первопричин*. Несколько примеров:

- Понимание и оптимизация бизнес-процесса (например, определение продуктов, которые повышают ценность линейки продуктов).
- Определение факторов, вызывающих диабет.
- Определение причин возникновения дорожных пробок.

Этот список практических применений машинного обучения – всего лишь затравка, потому что эта тема повсеместно встречается в области data science. Методы регрессии и классификации играют важную роль, но они не исчерпывают всего репертуара методов и их практических применений; другим примером полезного метода служит кластеризация. Методы машинного обучения могут применяться

на всех стадиях процесса data science, эта тема более подробно описана в следующем разделе.

3.1.2. Применение машинного обучения в процессе data science

Хотя машинное обучение в основном связано с шагом моделирования данных процесса data science, оно может применяться почти на каждом шаге. Чтобы освежить вашу память после предыдущих глав, мы снова приводим диаграмму процесса data science на рис. 3.1.

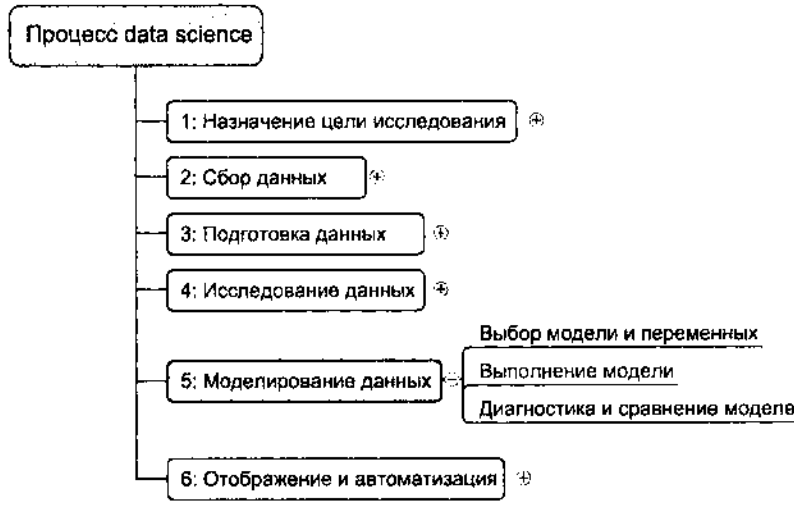


Рис. 3.1. Процесс data science

Фаза моделирования может начаться только после того, как вы получите качественные необработанные данные, смысл которых вам понятен. Но до этого методы машинного обучения могут принести пользу в фазе подготовки данных. В качестве примера можно привести список текстовых строк; методы машинного обучения могут группировать похожие строки, чтобы упростить исправление орфографических ошибок.

Машинное обучение также пригодится в ходе исследования данных. Алгоритмы помогают выявить в данных закономерности, которые трудно найти с использованием только диаграмм.

Итак, машинное обучение приносит большую пользу в процессе data science, поэтому не приходится удивляться большому количеству библиотек Python, разработанных для упрощения работы программиста.

3.1.3. Инструменты Python, используемые в машинном обучении

Python содержит множество пакетов, которые могут использоваться в среде машинного обучения. Экосистему машинного обучения Python можно условно разделить на пакеты трех типов (рис. 3.2).

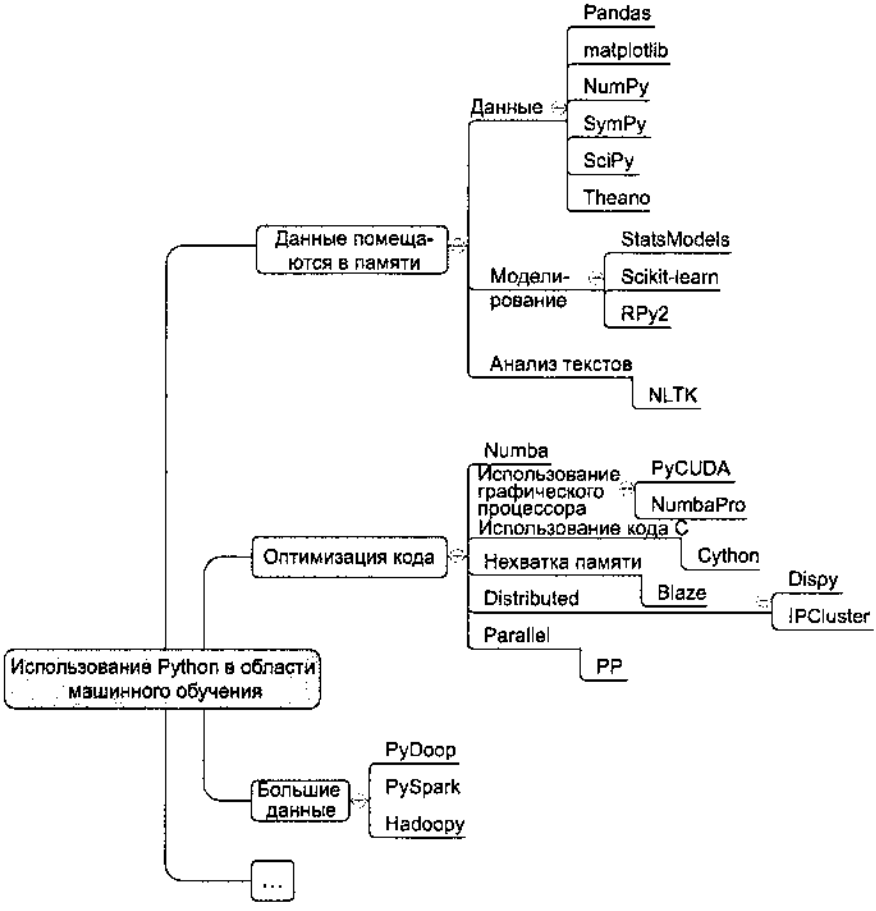


Рис. 3.2. Сводка пакетов Python, используемых в фазе машинного обучения

Пакеты первого типа на рис. 3.2 в основном используются для выполнения простых операций в тех случаях, когда данные помещаются в памяти. Второй тип используется для оптимизации кода, когда вы завершили построение прототипа и столкнулись с проблемами скорости или нехватки памяти. Пакеты третьего типа предназначены для использования Python с технологиями больших данных.

Пакеты для работы с данными в памяти

В ходе построения прототипа могут пригодиться следующие пакеты, предоставляющие расширенную функциональность в нескольких строках кода:

- SciPy — библиотека, интегрирующая фундаментальные пакеты, часто используемые в научных вычислениях (такие, как NumPy, matplotlib, Pandas и SymPy).
- NumPy предоставляет доступ к мощным функциям массивов и функциям линейной алгебры.
- Matplotlib — популярный пакет научной 2D-графики с ограниченной 3D-функциональностью.
- Pandas — высокопроизводительный, но простой в использовании пакет обработки данных. Он вводит в Python концепцию *фреймов данных*, своего рода таблиц данных в памяти. Вероятно, эта концепция покажется знакомой частым пользователям R.
- SymPy – пакет, используемый для решения задач символической математики и компьютерной алгебры.
- StatsModels – пакет статистических методов и алгоритмов.
- Scikit-learn — библиотека алгоритмов машинного обучения.
- RPy2 позволяет вызывать функции R из кода Python. R — популярная статистическая программа с открытым кодом.
- NLTK (Natural Language Toolkit) — инструментарий Python, ориентированный на анализ текстовых данных.

Эти библиотеки хорошо подходят для начала работы, но как только вы решаете, что вам потребуется часто запускать некоторую программу на языке Python, в дело вступает фактор производительности.

Оптимизация операций

После того как приложение перейдет в фазу реальной эксплуатации, перечисленные ниже библиотеки помогут обеспечить необходимую скорость выполнения. Иногда эта задача подразумевает подключение к инфраструктурам больших данных (таким, как Hadoop или Spark).

- *Numba* и *NumbaPro* — эти библиотеки используют компиляцию JIT (Just-In-Time) для ускорения приложений, написанных на Python с некоторыми аннотациями. NumbaPro также позволяет использовать мощь графического процессора (GPU).
- *PyCUDA* — позволяет писать код, который будет выполняться на графическом процессоре вместо центрального процессора, а следовательно, идеально подходит для интенсивных вычислений. Лучше всего годится для задач, которые

хорошо параллелизируются и требуют относительно малого объема входных данных по сравнению с необходимыми вычислительными ресурсами. Примером служит анализ надежности прогнозов, основанный на вычислении тысяч разных результатов на основании одного исходного состояния.

- *Cython*, или *C для Python* — открывает возможность использования языка программирования C из кода Python. Язык C относится к языкам более низкого уровня, поэтому код C ближе к машинному коду. Чем ближе код к битам и байтам, тем быстрее он выполняется. Компьютер также быстрее работает, если ему известен тип переменной (так называемая *статическая типизация*). При проектировании Python такая возможность не была предусмотрена, но Cython помогает преодолеть этот недостаток.
- *Blaze* — предоставляет структуры данных, размер которых может превышать объем памяти компьютера, и позволяет работать с большими объемами данных.
- *Dispy* и *IPCluster* — пакеты для написания кода, который может распределяться в компьютерном кластере.
- *PP* — по умолчанию Python выполняется в одном процессе. С помощью PP вы сможете параллелизировать вычисления на одном компьютере или в кластерах.
- *Pydoop* и *Hadoopy* — библиотеки связывают Python с Hadoop, популярной инфраструктурой больших данных.
- *PySpark* — связывает Python и Spark, инфраструктуру для работы с большими данными в памяти.

После краткого обзора доступных библиотек рассмотрим сам процесс моделирования.

3.2. Процесс моделирования

Фаза моделирования состоит из четырех шагов:

1. Планирование показателей и выбор модели.
2. Тренировка модели.
3. Проверка адекватности модели и выбор.
4. Применение тренированной модели к незнакомым данным.

Вероятно, хорошая модель будет найдена только после многократного повторения первых трех шагов.

Последний шаг присутствует не всегда, потому что иногда целью является не прогнозирование, а объяснение (анализ первопричин). Допустим, вы хотите найти причины вымирания некоторого вида, но при этом вам не всегда приходится прогнозировать, какой вид исчезнет следующим.

Несколько методов могут объединяться в цепочки или комбинироваться. При сцеплении вывод первой модели становится вводом для второй модели. При объединении нескольких моделей обучение производится независимо, а их результаты объединяются. Последний метод также называется *композиционным обучением* (ensemble learning).

Модель состоит из информационных конструкций, которые называются *показателями* или *свободными переменными*, и *целевой* переменной. Основной целью модели является прогнозирование целевой переменной, например максимальной температуры на следующий день. Переменные, которые используются для достижения этой цели, — те самые свободные переменные — (обычно) известны вам: сегодняшняя температура, движение облаков, текущая скорость ветра и т. д. Лучшими моделями считаются те, которые наиболее точно представляют реальность, желательно в сочетании с компактностью и хорошей интерпретируемостью. В этом контексте создание показателей является самой важной, а пожалуй, и наиболее интересной частью процесса моделирования. Например, важным показателем модели, пытающейся объяснить вымирание крупных сухопутных животных в Австралии за последние 60 000 лет, оказалась численность населения и распространение людей.

3.2.1. Создание новых показателей и выбор модели

В ходе планирования показателей вы должны идентифицировать и определить возможные свободные переменные модели. Это один из важнейших этапов процесса, потому что модель объединяет эти показатели для построения прогнозов. Часто для создания осмысленных показателей приходится обращаться к услугам экспертов или тематической литературе.

Некоторые показатели соответствуют переменным, которые вы получаете из набора данных; это характерно для наборов данных, предоставляемых в учебных примерах. На практике вам приходится искать показатели самостоятельно, причем они могут быть разбросаны по разным наборам данных. В нескольких проектах нам приходилось сводить более 20 источников данных, чтобы получить необходимые исходные данные. Часто входные данные становятся хорошими свободными переменными только после применения к ним преобразования или объединения нескольких входных наборов. Одним из примеров объединения нескольких вариантов ввода служат *взаимодействующие переменные*: влияние каждой переменной само по себе мало, но в присутствии обоих факторов их влияние значительно возрастает. Это особенно типично для химии и медицины. Например, хотя уксус и отбеливатель сами по себе относительно безвредны, при их смешивании выделяется хлор — ядовитый газ, убивший тысячи людей во время Первой мировой войны.

В медицине клинической фармацией называется дисциплина, посвященная исследованию эффекта взаимодействия лекарственных препаратов. Это важная задача, причем для получения потенциально опасных результатов даже не всегда

необходимы два вида лекарств. Например, сочетание некоторых противогрибковых препаратов с грейпфрутом приводит к серьезным побочным эффектам.

Иногда для определения показателей приходится применять методы моделирования: вывод модели становится частью другой модели. Ситуация достаточно распространенная, особенно в области глубокого анализа текста. Например, в ходе предварительной разметки содержимое документа может быть разбито на категории, или предварительная обработка может подсчитать количество упоминаний географических мест или людей в тексте. Этот подсчет часто оказывается более сложным, чем может показаться; сначала применяются модели для распознавания некоторых слов (таких, как названия мест или имена людей). Вся новая информация затем передается модели, которую вы хотите построить. Одной из самых серьезных ошибок при построении модели оказывается *субъективное смещение*: аналитик выбирает только те показатели, которые он может легко получить, и в результате модель представляет одностороннюю «истину». Модели, страдающие субъективным смещением, часто дают сбой на стадии проверки адекватности — проверка очевидно показывает, что они не обеспечивают адекватного представления ситуации.

Так, во время Второй мировой войны после бомбежек германской территории многие английские самолеты возвращались с пулевыми пробоинами в крыльях, в носовой и хвостовой части самолета. У них почти никогда не бывало пробоин в кабине, хвостовом руле или блоке двигателя, поэтому инженерная служба решила, что на крылья следует установить дополнительную броню. Идея казалась вполне здравой, пока математик по имени Абрахам Уолд не указал на очевидную ошибку: инженеры принимали во внимание только вернувшиеся самолеты. Пулевые пробоины в крыльях оказывались наименьшей из проблем, потому что самолет с такими повреждениями хотя бы мог вернуться на базу для ремонта. Соответственно, броня была укреплена на тех местах, которые остались неповрежденными на возвращающихся самолетах. Исходные рассуждения страдали от субъективного смещения: инженеры игнорировали важную часть данных, потому что эти данные было сложнее получить. В данном случае им повезло, потому что их рассуждения можно было восстановить и получить искомый результат без получения данных со сбитых самолетов.

После определения показателей можно переходить к *тренировке* модели на данных.

3.2.2. Тренировка модели

При наличии правильных свободных переменных и запланированного метода моделирования наступает черед тренировки модели. В этой фазе модели передаются данные, на которых она могла бы учиться.

Самые распространенные методы моделирования имеют полноценные реализации почти во всех языках программирования, включая Python. Это позволяет организовать тренировку моделей всего в нескольких строках кода. Если же вам потребуются ультрасовременные методы data science, скорее всего, вам придется

выполнять математические вычисления и реализовывать их средствами современных технологий обработки данных.

После того как модель будет натренирована, необходимо проверить, как она экстраполируется на реальность: эта стадия называется проверкой адекватности модели.

3.2.3. Проверка адекватности модели

В области data science существует множество методов моделирования. Какой же из них следует выбрать? Хорошая модель обладает двумя свойствами: она обладает хорошей прогностической способностью и хорошо обобщается для данных, которые ей еще не встречались. Для оценки этих свойств определяется метрика погрешности (насколько сильно ошибается модель) и стратегия проверки адекватности.

Две распространенные метрики погрешности в машинном обучении — *частота ошибок классификации* для задач классификации и *среднеквадратичная погрешность* для регрессионных задач. Частотой ошибок классификации называется процент наблюдений тестового набора данных, которые были неправильно классифицированы вашей моделью; чем он ниже, тем лучше. Среднеквадратичная погрешность представляет собой величину средней погрешности прогнозирования. Возведение средней погрешности в квадрат имеет два последствия. Во-первых, неверное прогнозирование в одном направлении не может быть скомпенсировано ошибочным прогнозированием в другом направлении. Например, переоценка будущего оборота на следующий месяц на 5000 не компенсируется его недооценкой на 5000 в следующий месяц. Во-вторых, большие ошибки при возведении в квадрат имеют еще больший вес. Малые ошибки остаются малыми и даже могут уменьшаться (если <1), тогда как большие ошибки увеличиваются и определенно привлекут ваше внимание.

Существует много стратегий проверки адекватности, из которых наиболее распространенными являются следующие:

- *Разбиение данных на тренировочный набор с X% наблюдений и контрольную выборку с остальными данными* (набор данных, который никогда не используется при создании модели). Этот метод применяется чаще всего.
- *K-кратная перекрестная проверка* — в этой стратегии набор данных делится на k частей. Каждая часть однократно используется как тестовый набор данных, в то время как остальные части формируют тренировочный набор данных. Преимущество этого метода заключается в том, что все имеющиеся данные используются в наборе данных.
- *Исключение единицы* — этот метод идентичен k-кратной проверке, но с $k = 1$. Одно наблюдение всегда исключается, а для тренировки используются остальные данные. Этот метод используется только с малыми наборами данных, поэтому он более ценен для специалистов, оценивающих результаты лабораторных экспериментов, нежели для аналитиков, работающих с большими данными.

- Другой популярный термин в области машинного обучения — *регуляризация*. Применение регуляризации подразумевает внесение штрафа за каждую дополнительную переменную, используемую для построения модели. С *L1-регуляризацией* строится модель с минимально возможным количеством свободных переменных. Это важно для надежности модели: простые решения остаются работоспособными в большем количестве ситуаций. *L2-регуляризация* направлена на минимизацию расхождений между коэффициентами свободных переменных. Частично совпадающая дисперсия свободных переменных усложняет оценку реального влияния каждой свободной переменной. Предотвращая частично совпадающую дисперсию, вы улучшаете интерпретируемость модели. Проще говоря, регуляризация используется в основном для того, чтобы модель не использовала слишком много показателей, т. е. для предотвращения чрезмерно близкой подгонки (overfitting).

Проверка адекватности чрезвычайно важна, потому что она определяет работоспособность модели в реальной обстановке, т. е. стоит ли ваша модель потраченных усилий. Несмотря на это, люди сплошь и рядом присылают в уважаемые научные журналы статьи (которые иногда даже публикуют) с ошибочно проведенной проверкой адекватности. В результате статьи отклоняются или же их приходится отзывать, потому что они содержат полностью ложную информацию. Подобные ситуации плохо сказываются на нашем душевном здоровье, поэтому запомните раз и навсегда: тестируйте модели на данных, неизвестных построенной модели, и убедитесь в том, что эти данные являются истинным представлением результатов, которые будут получены на свежих наблюдениях, сделанных другими людьми. В классификационных моделях прекрасно работают такие инструменты, как матрица несоответствий (представленная в главе 2, но более подробно рассмотренная в этой главе); возьмите их на вооружение.

После того как вам удастся построить хорошую модель, вы сможете (по желанию) воспользоваться ею для прогнозирования.

3.2.4. Прогнозирование новых наблюдений

Если вы успешно реализовали три первых этапа, в вашем распоряжении появилась эффективная модель, которая хорошо обобщается для незнакомых данных. Процесс применения модели к новым данным называется *оценкой модели*; вы уже неявно выполняли оценку во время проверки данных, только сейчас вы не знаете правильный результат. К настоящему моменту вы должны доверять модели достаточно, чтобы использовать ее в реальных условиях.

Оценка модели состоит из двух шагов. Сначала вы подготавливаете набор данных с показателями, точно соответствующими определению вашей модели. Задача сводится к повторению подготовки данных, выполненной на первом этапе процесса моделирования, для нового набора данных. Затем модель применяется к новому набору данных и полученный результат становится прогнозом.

А теперь рассмотрим разные виды методов машинного обучения: для разных проблем нужны разные методы.

3.3. Типы машинного обучения

Если не вдаваться в подробности, разные подходы к машинному обучению можно классифицировать по величине затрат человеческого труда, необходимых для их координации, и способу использования помеченных данных (т. е. данных, которым назначены категория или вещественное число, представляющие результат предшествующих наблюдений).

- Методы *контролируемого обучения* пытаются различать результаты и учиться, стараясь выявлять закономерности в наборе помеченных данных. Для пометки данных требуется участие человека.
- Методы *неконтролируемого обучения* не зависят от пометки данных и ищут в наборе данных закономерности без участия человека.
- Методы с *частичным контролем* требуют использования помеченных данных (а следовательно, человеческого участия) для поиска закономерностей в наборе данных, но они также могут продвигаться к результату и учиться даже при передаче непомеченных данных.

В этом разделе мы рассмотрим все три подхода, выясним, для каких задач лучше подходит каждый вариант, и воспользуемся парой библиотек Python, упоминавшихся ранее, чтобы дать вам представление о коде и решении задаче. В каждом из рассматриваемых примеров будет использоваться загружаемый набор данных, уже прошедший очистку; это позволит нам перейти сразу к этапу моделирования данных процесса data science, рассмотренного ранее в данной главе.

3.3.1. Контролируемое обучение

Как упоминалось ранее, метод контролируемого обучения может работать только с помеченными данными. Типичный пример такого рода — распознавание цифр на изображениях. Рассмотрим учебный пример.

Учебный пример: распознавание цифр на изображении

Один из многих стандартных подходов для предотвращения несанкционированного доступа к учетным записям пользователей в Интернете — *контрольные изображения* (captcha). Они представляют собой картинки с текстом и цифрами, которые пользователь должен рассмотреть и ввести в поле формы перед отправкой формы на веб-сервер. Наверняка вам не раз попадались изображения вроде рис. 3.3.

С помощью *наивного байесовского классификатора* — простого, но мощного алгоритма для разбиения наблюдений на классы (см. далее во врезке) — ваша про-

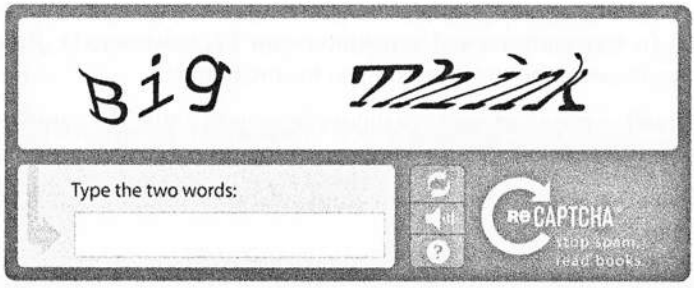


Рис. 3.3. Простое контрольное изображение используется для предотвращения автоматизированной рассылки спама через веб-форму

грамма может распознавать цифры на изображениях. Эти изображения отдаленно похожи на контрольные изображения на веб-сайтах, которые проверяют, что вы не компьютер, пытающийся подобрать пароль к чужой учетной записи. А теперь посмотрим, насколько хорошо компьютер справляется с распознаванием графических изображений чисел.

Цель нашего исследования (шаг 1 процесса data science) — добиться того, чтобы компьютер распознавал числа.

При этом будет использоваться набор данных MNIST, который часто применяется в литературе data science для обучения и сравнительного анализа.

НАИВНЫЕ БАЙЕСОВСКИЕ КЛАССИФИКАТОРЫ В КОНТЕКСТЕ СПАМ-ФИЛЬТРОВ

Не каждое сообщение, которое вы получаете по электронной почте, написано с честными намерениями. В вашем почтовом ящике могут оказаться несанкционированные сообщения коммерческих или массовых рассылок — проще говоря, спам. Мало того что спам раздражает — он часто используется для мошенничества и для распространения вирусов. По оценке лаборатории Касперского[1], свыше 60% всех сообщений электронной почты составляет спам. Для защиты пользователей от спама во многих почтовых клиентах выполняется фоновая программа, которая делит сообщения на проверенные и на спам.

Популярный метод построения спам-фильтров основан на применении классификатора, использующего слова в сообщении в качестве прогностических признаков. Классификатор выдает вероятность того, что конкретное сообщение является спамом, вычисленную на основании содержащихся

[1] Kaspersky 2014 Quarterly Spam Statistics Report, http:/ /usa.kaspersky.com/internet-security-center/threats/spam-statistics-report-q1-2014#.VVym9blViko.

в нем слов (в математической терминологии P(спам|слова)). Для получения этого результата используются три вычисления:

- P(спам) — средняя частота спама без учета слов. По данным лаборатории Касперского, спам в электронной почте составляет около 60%.
- P(слова) — частота использования этой комбинации слов независимо от спама.
- P(слова|спам) — частота использования этих слов в тренировочных сообщениях, помеченных как спам.

Вероятность того, что новое сообщение является спамом, вычисляется по следующей формуле:

$$P(\text{спам}|\text{слова}) = P(\text{спам})P(\text{слова}|\text{спам}) / P(\text{слова}).$$

Это практическое применение правила $P(B|A) = P(B)\,P(A|B) / P(A)$, известного как правило Байеса (отсюда и название классификатора). «Наивность» происходит от предположения классификатора о том, что присутствие одного признака не несет никакой информации о другом признаке (независимость признаков, также называемая отсутствием мультиколлинеарности). На практике признаки часто оказываются взаимосвязанными, особенно в тексте. Например, за словом «купить» часто следует слово «сейчас». Несмотря на нереалистичное предположение, наивный классификатор на удивление хорошо работает на практике.

После небольшой теоретической подготовки, данной во врезке, вы готовы к самостоятельному моделированию. Проследите за тем, чтобы весь код, приведенный ниже, выполнялся в одной области видимости, потому что для каждого фрагмента необходим предыдущий фрагмент. Файл IPython для этой главы можно загрузить со страницы книги на сайте Manning.

Изображения MNIST можно взять из пакета наборов данных Scikit-learn в уже нормализованном виде (все изображения масштабированы до единого размера 64×64 пиксела), так что этап подготовки данных (этап 3 процесса data science) сводится к минимуму. Но сначала необходимо получить данные на этапе 2 процесса data science (листинг 3.1).

Листинг 3.1. Шаг 2 процесса data science: выборка данных цифровых изображений

```
from sklearn.datasets import load_digits  <-- Импортирование базы данных цифр.
import pylab as pl
digits = load_digits()                    <-- Загрузка цифр.
```

Работа с графикой мало чем отличается от работы с другими наборами данных. Для изображения в оттенках серого в каждый элемент матрицы помещается зна-

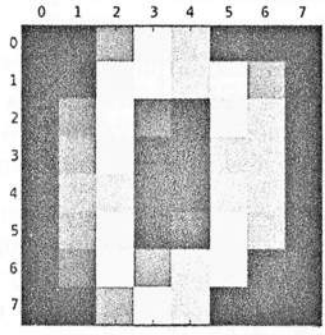
чение, представляющее интенсивность серого цвета. Следующий код (листинг 3.2), демонстрирующий этот процесс, относится к этапу 4 процесса data science: исследованию данных.

Листинг 3.2. Этап 4 процесса data science: использование Scikit-learn

```
pl.gray()                                  ← Изображение преобразуется в набор значений в оттенках серого.
pl.matshow(digits.images[0])               | Сначала выводится изображение.
pl.show()
digits.images[0]                           ← Вывод соответствующей матрицы.
```

На рис. 3.4 показано, как размытое изображение «0» преобразуется в матрицу данных.

На рис. 3.4 представлен непосредственный результат выполнения кода, но возможно, рис. 3.5 немного прояснит картину; из него видно, как каждый элемент вектора становится частью изображения.




```
rray([[  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.]])
```

Рис. 3.4. Размытое изображение 0 в оттенках серого преобразуется в соответствующую матрицу. Чем больше значение, тем ближе элемент к белому цвету, а чем меньше, тем он ближе к черному

Пока все просто, верно? Естественно, это еще не все. Наивный классификатор Байеса ожидает получить список значений, а `pl.matshow()` возвращает двумерный массив (матрицу), соответствующий геометрической форме изображения. Чтобы преобразовать матрицу в список, необходимо вызвать `reshape()` для `digits.images`. В результате будет получен одномерный массив, который выглядит примерно так:

```
array([[ 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.]])
```

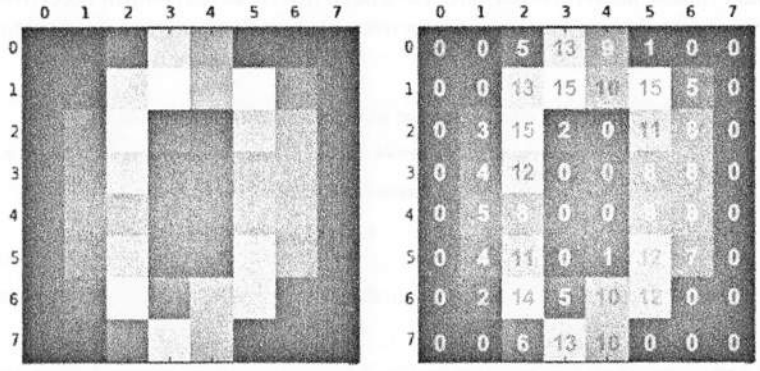


Рис. 3.5. Изображение преобразуется в формат, который может использоваться наивным классификатором Байеса. Для этого мы берем значения интенсивности всех пикселов (справа) и помещаем их в список

Предыдущий фрагмент кода демонстрирует, как матрица на рис. 3.5 преобразуется в список Python (т. е. размерность структуры данных уменьшается с 2 до 1). С этого момента все сводится к стандартной задаче классификации, и мы естественным образом приходим к этапу 5 процесса data science: построению модели.

Теперь, когда у нас появился механизм передачи содержимого изображения классификатору, необходимо передать ему тренировочный набор данных, чтобы он начал учиться прогнозировать числа на изображениях. Как уже упоминалось ранее, Scikit-learn содержит подмножество базы данных MNIST (1800 изображений), которым мы воспользуемся. Каждое изображение помечено числом, которое на нем присутствует. Так строится вероятностная модель наиболее вероятной цифры, представленной на изображении, по набору ее значений в оттенках серого.

После того как программа обработает тренировочный набор и построит модель, мы передадим ей тестовый набор и посмотрим, насколько хорошо она научилась интерпретировать изображения по модели.

Листинг 3.3 показывает, как этот процесс реализуется в коде.

Листинг 3.3. Задача классификации изображений на примере распознавания цифр

```
from sklearn.cross_validation import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
import pylab as plt

y = digits.target

n_samples = len(digits.images)
X= digits.images.reshape((n_samples, -1))
```

Шаг 1: Выбор целевой переменной.

Шаг 2: Подготовка данных. Метод reshape преобразует матричную форму данных. Например, он может превратить матрицу 10×10 в 100 векторов.

```
print X

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
gnb = GaussianNB()
fit = gnb.fit(X_train,y_train)
predicted = fit.predict(X_test)
confusion_matrix(y_test, predicted)
```

Шаг 3: Разбиение данных на тестовый и тренировочный набор.

Шаг 4: Выбор наивного классификатора Байеса; для оценки вероятности применяется гауссово распределение.

Шаг 5: Подгонка данных.

Шаг 6: Прогнозирование по незнакомым данным.

Шаг 7: Создание матрицы несоответствий.

Конечный результат выполнения этого кода называется *матрицей несоответствий*; пример такой матрицы представлен на рис. 3.6. Этот двумерный массив показывает, как часто прогнозируемое число совпадало с правильным (элементы главной диагонали), а элемент матрицы (i, j) определяет, сколько раз определялось значение j, хотя на изображении было показано значение i. Из рис. 3.6 видно, что модель 17 раз правильно распознала число 2 (координаты 3,3), но при этом модель 15 раз обнаруживала число 8, тогда как на изображении было показано число 2 (координаты 9,3).

```
array([[37,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 39,  0,  0,  0,  0,  1,  0,  3,  0],
       [ 0,  9, 17,  3,  0,  0,  0,  0, 15,  0],
       [ 0,  0,  0, 38,  0,  0,  0,  2,  5,  0],
       [ 0,  1,  0,  0, 27,  0,  2,  8,  0,  0],
       [ 0,  1,  0,  1,  0, 43,  0,  3,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 52,  0,  0,  0],
       [ 0,  0,  0,  0,  1,  0,  0, 47,  0,  0],
       [ 0,  5,  0,  1,  0,  1,  0,  4, 37,  0],
       [ 0,  2,  0,  7,  1,  0,  0,  3,  7, 27]])
```

Рис. 3.6. Матрица несоответствий строится на основании прогнозов того, какое число представлено на нечетком изображении

МАТРИЦЫ НЕСООТВЕТСТВИЙ

Матрица несоответствий показывает, насколько точно (или неточно) работает модель и сколько раз результаты прогноза не соответствовали действительности. В простейшем варианте она имеет вид таблицы 2×2 для моделей, которые пытаются классифицировать наблюдения по одному из двух вариантов, *A* или *B*. Допустим, некая классификационная модель предсказывает, купит ли клиент новейший продукт: вишневый пудинг. Возможны всего два варианта прогноза: «Да, купит» и «Нет, не купит». После формирования прогноза для 100 клиентов мы сравниваем результаты с фактическим

поведением, которое показывает, сколько раз модель выдала правильный результат. Пример показан в табл. 3.1.

Таблица 3.1. Пример матрицы несоответствий

Матрица несоответствий	Прогноз «Да, купит»	Прогноз «Нет, не купит»
Клиент купил вишневый пудинг	35 (истинный позитив)	10 (ложный негатив)
Клиент не купил вишневый пудинг	15 (ложный позитив)	40 (истинный негатив)

Модель правильно сработала в (35+40) 75 случаях и выдала ошибочный прогноз в (15+10) 25 случаях, что дает (75 правильных прогнозов/100 наблюдений) 75% точность.

Все правильно классифицированные наблюдения суммируются по диагонали (35+40), а все остальные наблюдения (15+10) были классифицированы ошибочно. Если модель выдает прогнозы только двух классов (бинарный прогноз), правильные прогнозы делятся на две группы: истинные позитивы (модель предсказала, что покупатель купит пудинг, и он купил) и истинные негативы (модель предсказала, что покупатель не купит пудинг, и он не купил). Неправильные прогнозы тоже делятся на две группы: ложные позитивы (модель предсказала, что покупатель купит пудинг, а он не купил) и ложные негативы (модель предсказала, что покупатель не купит пудинг, а он купил). Матрица помогает понять, в какой области модель ошибается чаще всего. Похоже, в нашем случае модель слишком уверена в неотразимости продукта и слишком легко относит клиентов к потенциальным покупателям (ложные позитивы).

Из матрицы несоответствий можно сделать вывод о том, что для большинства изображений прогнозы весьма точны. В хорошей модели можно ожидать, что сумма чисел на главной диагонали (также называемой *следом матрицы*) будет высокой по сравнению с суммой остальных элементов матрицы; это будет означать, что прогнозы в основном оказывались правильными.

Предположим, мы хотим представить результаты в более наглядном виде или же проанализировать несколько изображений и прогнозы, сделанные нашей программой: следующий код может использоваться для вывода их рядом друг с другом. Это позволит нам понять, где программа ошибается и нуждается в дополнительном обучении. Если результат нас устраивает, то построение модели на этом завершается и в процессе наступает этап 6: отображение результатов (листинг 3.4).

Из рис. 3.7 видно, что все прогнозы правильны, кроме цифры 3, которая распознается как 8. Эту ошибку можно простить, так как 2 и 8 визуально достаточно близки. Левое нижнее изображение неоднозначно даже для человека; что это, 5 или 3? Вопрос спорный, но алгоритм полагает, что это 3.

Листинг 3.4. Сравнение прогнозов с реальными числами

```
images_and_predictions = list(zip(digits.images, fit.predict(X)))
for index, (image, prediction) in enumerate(images_and_predictions[:6]):
    plt.subplot(6, 3 ,index + 5)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Prediction: %i' % prediction)
plt.show()
```

Добавление дополнительной поддиаграммы на сетке 6×3. Этот вызов можно было бы упростить до вида plt.subplot (3, 2 ,index), но такое решение визуально более привлекательно.

Сохранение матрицы изображения и прогноза (в числовом виде) в одном массиве.

Перебор первых 7 изображений.

Ось не отображается.

Изображение выводится в оттенках серого.

Прогнозируемое значение выводится в заголовке изображения.

Вывод полной диаграммы, состоящей из 6 поддиаграмм.

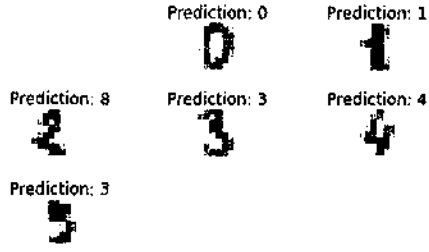


Рис. 3.7. Алгоритм пытается распознать число на каждом изображении; ошибочно распознано только число 2. Неоднозначное число распознано как 3, хотя это с таким же успехом может быть 5; результат неочевиден даже для человеческого глаза

Распознавая неправильно интерпретированные изображения, мы можем продолжить тренировку модели: каждое изображение помечается правильным числом и снова передается модели как новый тренировочный набор (этап 5 процесса data science). Это повысит точность модели, чтобы цикл обучения/прогнозирования/корректировки продолжился, а прогнозы стали более точными. Этот контролируемый набор данных используется в примере. Все изображения в примерах имеют одинаковый размер и содержат 16 оттенков серого. Расширьте задачу для переменного размера изображений со строками переменной длины и переменным количеством оттенков серого с алфавитно-цифровыми символами на контрольных изображениях, и вы поймете, почему достаточно точной модели для распознавания контрольных изображений еще не существует.

В этом примере обучения с частичным контролем вполне очевидно, что без меток, которые связываются с каждым изображением и сообщают программе, какое число представлено на изображении, не удастся ни построить модель, ни сделать прогнозы. С другой стороны, модель обучения без контроля не нуждается в пометке и может использоваться для формирования структуры в неструктурированном наборе данных.

3.3.2. Неконтролируемое обучение

Как правило, в больших наборах данных пометка отсутствует, и если вы не разберете все данные и не назначите им метки, метод контролируемого обучения к данным не сработает. Вместо этого необходимо выбрать метод, который будет работать с этими данными, потому что

- мы можем изучить *распределение данных* и сделать выводы относительно данных, находящихся в разных частях распределения;
- мы можем проанализировать *структуру и значения в данных* и сделать выводы относительно новых, более осмысленных данных и структуры.

В каждой из этих методологий *неконтролируемого обучения* существуют многочисленные практические приемы. Однако в реальном мире всегда приходится работать на достижение цели исследования, определенной на первом этапе процесса data science, поэтому вы будете комбинировать или пробовать разные методы, пока набор данных не будет размечен, что сделает возможным применение методов контролируемого обучения, или пока не будет достигнута сама цель исследования.

Выделение упрощенной скрытой структуры из данных

Не все поддается точному измерению. Встречая человека впервые, вы можете попытаться предположить, понравились вы ему или нет, по его поведению. Но что, если до встречи с вами у него был неудачный день? Может, его кошку сбила машина или у него недавно умер близкий человек? Суть в том, что некоторые переменные могут быть доступны сразу, тогда как другие могут быть только выведены логически, а следовательно, отсутствуют в наборе данных. Переменные первого типа называются *наблюдаемыми* (observable), а переменные второго типа — *скрытыми* (latent). В нашем примере эмоциональное состояние вашего нового знакомого является скрытой переменной. Она определенно влияет на его отношение к вам, но ее значение остается неясным.

Выведение скрытых переменных и их значений на основании фактического содержимого набора данных — весьма полезный навык, потому что

- скрытые переменные могут заменять несколько существующих переменных, уже включенных в набор данных;
- сокращение количества переменных в наборе данных упрощает работу с набором, может ускорить выполнение последующих алгоритмов и повысить точность прогнозов;
- так как скрытые переменные проектируются с учетом определенной цели исследования, их использование практически не приводит к потере ключевой информации.

Скажем, если набор данных удастся сократить от 14 наблюдаемых переменных на строку до 5–6 скрытых переменных, вероятность достижения цели исследования

повышается из-за упрощения структуры набора данных. Как видно из следующего примера, дело не сводится к сокращению существующего набора данных к минимальному количеству скрытых переменных. Необходимо найти оптимум, при котором несколько производных скрытых переменных приносят максимальную пользу. Воплотим эти рассуждения на практике в небольшом учебном примере.

Учебный пример: выявление скрытых переменных в наборе данных качества вина

В этом коротком примере мы воспользуемся *методом главных компонент* (PCA, Principal Components Analysis) для поиска в наборе данных скрытых переменных, описывающих качество вина. Затем эффективность скрытых переменных в прогнозировании качества вина сравнивается с эффективностью исходного наблюдаемого набора. Вы узнаете:

1. Как выявлять и как выводить эти скрытые переменные.
2. Как анализировать положение оптимума (количество новых переменных, приносящих наибольшую пользу) посредством генерирования и интерпретации *графика каменистой осыпи* (scree plot), сгенерированных PCA. (Вскоре мы расскажем о графиках каменистой осыпи более подробно.)

Рассмотрим основные компоненты этого примера.

- *Набор данных* — Калифорнийский университет в Ирвайне (UCI) предоставляет сетевой репозиторий с 352 наборами данных для упражнений машинного обучения (*http://archive.ics.uci.edu/ml/*). Мы воспользуемся набором данных качества красных вин, который создали П. Кортес, А. Сердейра, Ф. Алмейда, Т. Матос и Дж. Рейс[1]. Он состоит из 1600 строк и 11 переменных на строку (табл. 3.2).
- *Метод главных компонент* — метод выявления скрытых переменных в наборе данных с сохранением максимально возможной информации.

Таблица 3.2. Первые три строки набора данных качества красного вина

Постоянная кислотность	Летучая кислотность	Содержание лимонной кислоты	Остаточный сахар	Хлориды	Свободный диоксид серы	Общее содержание диоксида серы	Плотность	pH	Сульфаты	Алкоголь	Качество
7,4	0,7	0	1,9	0,076	11	34	0,9978	3,51	0,56	9,4	5
7,8	0,88	0	2,6	0,098	25	67	0,9968	3,2	0,68	9,8	5
7,8	0,76	0,04	2,3	0,092	15	54	0,997	3,26	0,65	9,8	5

[1] Полная информация о наборе данных качества вина доступна по адресу *https://archive.ics.uci.edu/ml/datasets/Wine+Quality*.

- *Scikit-learn* — мы используем эту библиотеку, потому что она уже содержит реализацию PCA и позволяет строить графики каменистой осыпи.

Процесс data science начинается с определения цели исследования: требуется объяснить субъективный показатель «качества вина» с использованием разных свойств.

Таким образом, нашей первой задачей будет загрузка набора данных (этап 2: сбор данных) в листинге 3.5 и подготовка их для анализа (этап 3: подготовка данных). Затем мы запустим алгоритм PCA и проанализируем результаты в поиске возможных вариантов решения.

Листинг 3.5. Сбор данных и стандартизация переменных

```
import pandas as pd
from sklearn import preprocessing
from sklearn.decomposition import PCA
import pylab as plt
from sklearn import preprocessing

url = http://archive.ics.uci.edu/ml/machine-learning-databases/wine-
quality/
winequality-red.csv
data = pd.read_csv(url, sep= ";")
X = data[[u'fixed acidity', u'volatile acidity', u'citric acid',
      u'residual sugar', u'chlorides', u'free sulfur dioxide',
      u'total sulfur dioxide', u'density', u'pH', u'sulphates',
      u'alcohol']]
y = data.quality
X= preprocessing.StandardScaler().fit(X).transform(X)
```

X — матрица свободных переменных, которыми в данном случае являются свойства вина (такие, как плотность или содержание алкоголя).

Адрес для загрузки набора данных качества вина.

Чтение данных в формате CSV, разделенных символом ";".

y – вектор, представляющий целевую переменную. В данном примере y содержит субъективное качество вина.

При стандартизации данных к каждой точке данных применяется следующая формула: z = (x-μ)/σ, где z – новое наблюдаемое значение, x – старое, μ – математическое ожидание, а σ – стандартное отклонение. Результаты PCA для матрицы данных проще интерпретируются, если столбцы были предварительно отцентрованы по средним арифметическим.

После исходной подготовки данных можно переходить к выполнению PCA. Полученный график каменистой осыпи (см. далее) показан на рис. 3.8. Так как метод PCA относится к исследовательским методам, мы приходим к этапу 4 процесса data science: исследованию данных (см. листинг 3.6).

Рассмотрим график каменистой осыпи на рис. 3.8.

На рис. 3.8 получен график, сгенерированный по данным набора качества вина. В данном случае мы рассчитываем найти на графике «перегиб», или «хоккейную клюшку». Такая форма графика показывает, что сокращенное количество пере-

Листинг 3.6. Проведение анализа главных компонент

```
model = PCA()
results = model.fit(X)
Z = results.transform(X)
plt.plot(results.explained_variance_)
plt.show()
```

Создание экземпляра класса анализа главных компонент.

Применение PCA к свободным переменным для поиска возможности свертки их в меньшее количество переменных.

Результат преобразуется в массив для использования вновь созданных данных.

График объясняемой дисперсии переменных; в данном случае используется график каменистой осыпи.

Отображение графика.

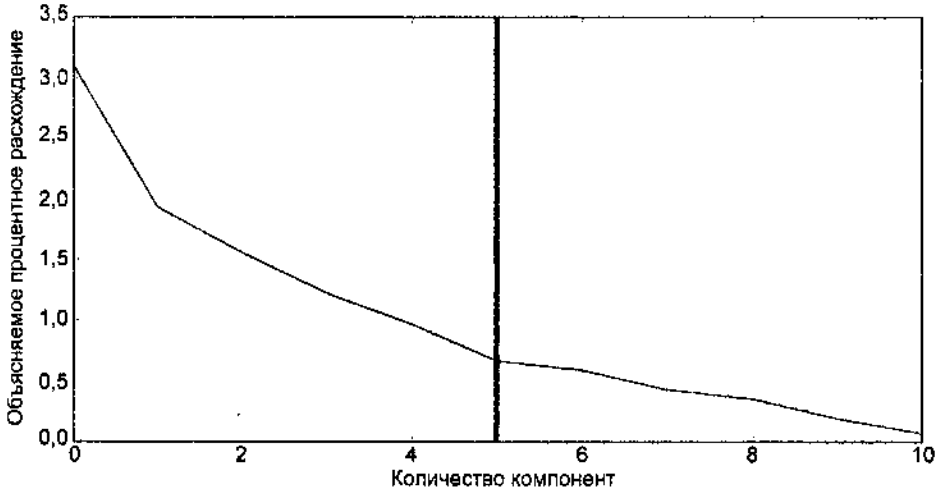


Рис. 3.8. График каменистой осыпи PCA демонстрирует предельную величину информации, которую может создавать каждая новая переменная. Первые переменные объясняют приблизительно 28% информации, вторая переменная добавляет еще 17%, третья около 15%, и так далее

менных может представлять бóльшую часть информации набора данных, тогда как остальные переменные добавляют минимум информации. На нашем графике метод PCA показывает, что сокращение всего набора до одной переменной может отразить приблизительно 28% общей информации набора (график строится с 0, поэтому первая переменная находится в позиции 0 оси x), две переменные отражают примерно на 17% больше информации (итого 45%), и т. д. Полные результаты приведены в табл. 3.3.

Перегиб на графике предполагает, что пять переменных могут содержать бóльшую часть информации, представленной в данных. Конечно, вы можете возразить и выбрать порог отсечения на шести или семи переменных, но мы ограничимся более

Таблица 3.3. Результаты PCA

Количество переменных	Дополнительная информация	Общий процент объясняемых данных
1	28%	28%
2	17%	45%
3	14%	59%
4	10%	69%
5	8%	77%
6	7%	84%
7	5%	89%
8–11	...	100%

простым набором данных по сравнению с набором, содержащим бóльшую часть информации из исходного набора.

На этой стадии можно сделать следующий шаг и посмотреть, будет ли исходный набор данных, перекодированный с пятью скрытыми переменными, достаточно хорошим для точного прогнозирования качества вина, но сначала нужно разобраться с тем, как определить, что представляют собой эти пять переменных.

Интерпретация новых переменных

После принятия исходного решения о сокращении набора данных с 11 исходных переменных до пяти скрытых мы проверим, можно ли интерпретировать или назвать эти переменные на основании их отношений с оригиналами. С реальными именами работать проще, чем с условными обозначениями типа lv1, lv2 и т. д. В листинг 3.7 добавляется строка кода для построения таблицы, демонстрирующей взаимосвязь между двумя наборами переменных.

Листинг 3.7. Вывод компонент PCA во фрейме данных Pandas

```
pd.DataFrame(results.components_, columns=list(
➥  [u'fixed acidity', u'volatile acidity', u'citric acid', u'residual sugar',
➥   u'chlorides', u'free sulfur dioxide',  u'total sulfur dioxide', u'density',
➥   u'pH', u'sulphates',  u'alcohol']))
```

Строки полученной таблицы (табл. 3.4) описывают математическую корреляцию. А проще говоря, первая скрытая переменная lv1, отражающая примерно 28% общей информации в наборе, вычисляется по следующей формуле.

```
Lv1 = (fixed acidity * 0.489314) + (volatile acidity * -0.238584) + … +
(alcohol * -0.113232)
```

Назначить разумное имя каждой новой переменной немного сложнее. Вероятно, для этого потребуется консультация эксперта в области вин. Так как у нас под рукой такого эксперта нет, мы присвоим им имена из табл. 3.5.

Таблица 3.4. Вычисление корреляции 11 исходных переменных с 5 скрытыми переменными методом PCA

	Постоянная кислотность (Fixed acidity)	Летучая кислотность (Volatile acidity)	Содержание лимонной кислоты (Citric acid)	Остаточный сахар (Residual sugar)	Хлориды (Chlorides)	Свободный диоксид серы (Free sulfur dioxide)	Общее содержание диоксида серы (Total sulfur dioxide)	Плотность (Density)	pH	Сульфаты (Sulfates)	Алкоголь (Alcohol)
0	0,489314	−0,238584	0,463632	0,146107	0,212247	−0,036158	0,023575	0,395353	−0,438520	0,242921	−0,113232
1	−0,110503	0,274930	−0,151791	0,272080	0,148052	0,513567	0,569487	0,233575	0,006711	−0,037554	−0,386181
2	0,123302	0,449963	−0,238247	−0,101283	0,092614	−0,428793	−0,322415	0,338871	−0,057697	−0,279786	−0,471673
3	−0,229617	0,078960	−0,079418	−0,372793	0,666195	−0,043538	−0,034577	−0,174500	−0,003788	0,550872	−0,122181
4	0,082614	0,218735	0,058573	−0,732144	−0,246501	0,159152	0,222465	−0,157077	−0,267530	−0,225962	−0,350681

Таблица 3.5. Интерпретация переменных качества вина, созданных методом PCA

Скрытая переменная	Возможная интерпретация
0	Долговременная кислотность
1	Сульфиды
2	Летучая кислотность
3	Хлориды
4	Отсутствие остаточного сахара

Теперь можно перекодировать исходный набор данных всего с пятью скрытыми переменными. Эта задача снова заключается в подготовке данных, поэтому мы вернемся к этапу 3 процесса data science: подготовке данных. Как упоминалось в главе 2, процесс data science имеет рекурсивную природу, и это утверждение особенно справедливо для стадии между этапами 3 (подготовка данных) и 4 (исследование данных).

В табл. 3.6 изображены первые три строки после завершения перекодировки.

Таблица 3.6. Первые три строки набора данных качества вина, пять скрытых переменных

	Долговременная кислотность	Сульфиды	Летучая кислотность	Хлориды	Отсутствие остаточного сахара
0	–1,619530	0,450950	**1,774454**	0,043740	–0,067014
1	–0,799170	1,856553	0,911690	0,548066	0,018392
2	**2,357673**	–0,269976	–0,243489	–0,928450	**1,499149**

Мы уже видим высокие значения летучей кислотности для вина 0, а в вине 2 особенно высока долговременная кислотность. От такого вина ничего хорошего ждать не приходится!

Сравнение точности исходного набора данных со скрытыми переменными

Итак, мы решили, что набор данных должен быть закодирован в пяти скрытых переменных вместо 11 исходных. Теперь пора посмотреть, насколько хорошо работает новый набор данных для прогнозирования качества вина по сравнению с оригиналом. Мы воспользуемся наивным байесовским классификатором, представленным в предыдущем примере.

Для начала выясним, насколько хорошо 11 исходных переменных предсказывают качество вина. Код для решения этой задачи представлен в листинге 3.8.

Листинг 3.8. Прогнозирование качества вина до применения анализа главных компонент

```
from sklearn.cross_validation import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
import pylab as plt

gnb = GaussianNB()
fit = gnb.fit(X,y)
pred = fit.predict(X)
print confusion_matrix(pred,y)//
print confusion_matrix(pred,y).trace()
```

Для оценки используется наивный классификатор Байеса с гауссовым распределением.

Подгонка данных.

Прогнозирование для неизвестных данных.

Изучение матрицы несоответствий.

Подсчет правильно классифицированных случаев: после проверки матрицы несоответствий суммируются все элементы диагонали (следа матрицы). Мы видим, что наивный классификатор Байеса выдает 897 правильных прогнозов из 1599.

Теперь запустим тот же тест, но начнем всего с одной скрытой переменной вместо исходных 11. Затем мы добавим другую, посмотрим, на что она повлияла, добавим еще одну, и т. д., чтобы видеть, как улучшается точность прогнозирования. Листинг 3.9 показывает, как это делается.

Листинг 3.9. Прогнозирование качества вина с наращиванием количества главных компонент

```
predicted_correct = []
for i in range(1,10):
    model = PCA(n_components = i)
    results = model.fit(X)
    Z = results.transform(X)
    fit = gnb.fit(Z,y)
    pred = fit.predict(Z)
    predicted_correct.append(confusion_matrix(pred,y).trace())
    print predicted_correct
plt.plot(predicted_correct)
plt.show()
```

Массив будет заполнен правильно спрогнозированными наблюдениями.

Перебор первых 10 обнаруженных главных компонент.

Создание экземпляра модели PCA с разным количеством компонентов, от 1 (1-я итерация) до 10 (10-я итерация).

Подгонка модели PCA по x-переменным (показателям).

Z содержит результат в форме матрицы (на самом деле массив, заполненный другими массивами).

Применение наивного классификатора Байеса с гауссовым распределением для оценки.

Собственно прогнозирование с использованием подогнанной модели.

В конце каждой итерации добавляется количество правильно классифицированных наблюдений.

Выводя этот массив, мы видим, как после каждой итерации добавляется новое количество правильно классифицированных наблюдений.

Результат выглядит более наглядно в графической форме.

Вывод графика.

Полученный график изображен на рис. 3.9.

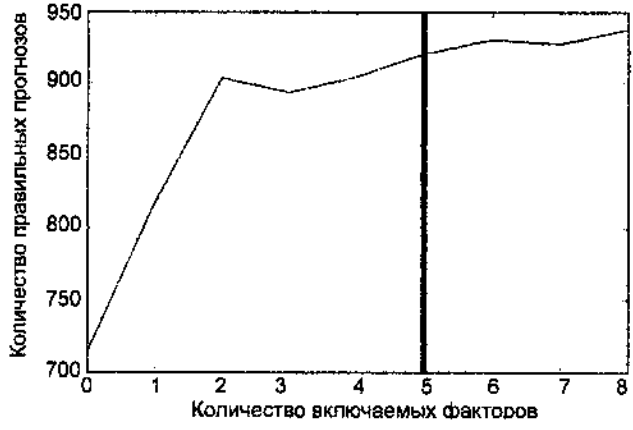


Рис. 3.9. Из графика видно, что добавление новых скрытых переменных в модель (ось x) до определенного момента сильно повышает прогностическую способность (ось y), но потом прирост замедляется

Из графика на рис. 3.9 видно, что всего с 3 скрытыми переменными классификатор лучше справляется с прогнозированием качества вина, чем с 11 исходными. Кроме того, добавление скрытых переменных свыше 5 не увеличивает прогностическую способность в такой мере, как первые 5. Это показывает, что выбор 5 переменных как порога отсечения тоже был удачным, как мы и рассчитывали.

Вы увидели, как группировать похожие переменные, но группировать также можно и наблюдения.

Группировка сходных наблюдений для извлечения информации из распределения данных

Представьте на секунду, что вы строите сайт, который рекомендует пользователям фильмы на основании введенных ими предпочтений и тех фильмов, которые они видели. Если пользователь смотрит много фильмов ужасов, скорее всего, он захочет знать больше о новых фильмах ужасов, а подростковая романтика для него интереса не представляет. Группируя пользователей, которые смотрят более или менее похожие фильмы и задают более или менее похожие предпочтения, можно получить достаточно основательную информацию о том, что еще им можно порекомендовать.

Обобщенный метод, который описывается в этом разделе, называется *кластеризацией* (clustering). В этом процессе набор данных делится на подмножества наблюдений, или *кластеры*; в пределах кластера наблюдения имеют много общего, но сильно отличаются от наблюдений из других кластеров. Рисунок 3.10 дает наглядное представление о том, чего пытается достичь кластеризация. Кружки в левой верхней части диаграммы явно расположены близко друг к другу, но при этом удалены от всех остальных. То же самое можно сказать о крестиках в правой верхней части.

Scikit-learn реализует в модуле sklearn.cluster несколько распространенных алгоритмов кластеризации данных, включая алгоритм *k*-средних, алгоритм распространения близости (affinity propagation) и спектральную кластеризацию. Для каждого алгоритма существует один-два случая, для которых он подходит лучше остальных[1], хотя метод *k*-средних является хорошим алгоритмом общего назначения на первых порах. Впрочем, как и во всех алгоритмах кластеризации, желательное количество кластеров должно задаваться заранее, что неизбежно приводит к процессу проб и ошибок перед получением хорошего результата. Также предполагается, что все данные, необходимые для анализа, доступны заранее. А если нет?

[1] Сравнительный анализ всех алгоритмов кластеризации Scikit-learn можно найти по адресу *http://scikit-learn.org/stable/modules/clustering.html*.

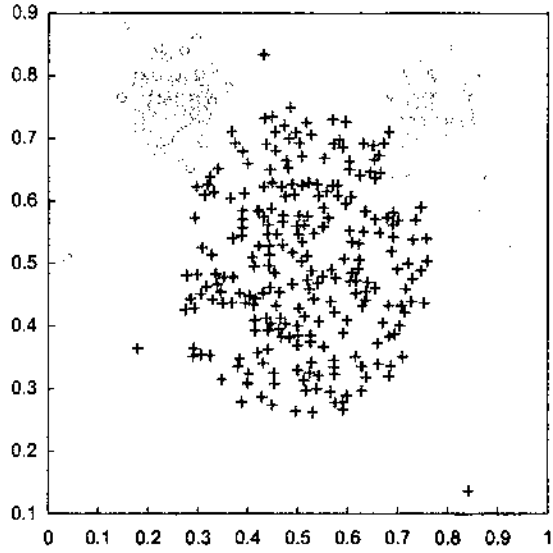


Рис. 3.10. Целью кластеризации является разбиение набора данных на «достаточно удаленные» подмножества. Например, на этой диаграмме набор данных был разделен на три кластера

Возьмем реальный пример кластеризации ирисов (цветы) по их свойствам (длина и ширина чашечки, длина и ширина лепестков и т. д.). В этом примере будет использоваться алгоритм *k*-средних. Этот алгоритм хорошо дает общее представление о данных, однако он чувствителен к начальным значениям, поэтому при каждом выполнении алгоритма вы можете получать разные кластеры, если только начальные значения не будут задаваться вручную посредством определения *затравки* (константы для генератора начальных значений). Если же вы хотите выявить иерархию, лучше воспользоваться алгоритмом из категории методов иерархической кластеризации.

Другой недостаток — необходимость задавать количество кластеров заранее. Часто это приводит к серии проб и ошибок, прежде чем вы придете к удовлетворительному решению.

Выполнение кода не создает особых проблем. Код имеет такую же структуру, как и во всех остальных примерах анализа, если не считать того, что вам не нужно передавать целевую переменную. Алгоритм сам должен найти интересные закономерности. Листинг 3.10 использует набор данных ирисов и пытается определить, может ли алгоритм сгруппировать ирисы по нескольким типам.

На рис. 3.11 показан результат классификации ирисов. Из таблицы видно, что даже без меток кластеры, сходные с официальной классификацией ирисов, обнаруживаются с результатом 134 (50+48+36) правильных классификаций из 150.

Листинг 3.10. Классификация ирисов

Вывод первых 5 наблюдений фрейма данных на экран; в данных хорошо видны 4 переменные: длина чашечки, ширина чашечки, длина лепестка, ширина лепестка.

Во фрейм данных включается новая переменная с именем "cluster". В ней для каждого цвета хранится информация о принадлежности его к кластеру.

Загрузка набора данных с информацией об ирисах из Scikit-learn.

Подгонка модели по данным. Все переменные считаются независимыми; в неконтролируемом обучении нет целевой переменной (y).

Данные ирисов преобразуются во фрейм данных Pandas.

```
import sklearn
from sklearn import cluster
import pandas as pd

data = sklearn.datasets.load_iris()
X = pd.DataFrame(data.data, columns = list(data.feature_names))
print X[:5]
model   = cluster.KMeans(n_clusters=3, random_state=25)
results = model.fit(X)
X["cluster"] = results.predict(X)
X["target"] = data.target //
X["c"] = "lookatmeIamimportant"
print X[:5]
classification_result = X[["cluster",
"target","c"]].groupby(["cluster","target"]).agg("count")
print(classification_result)
```

Наконец, во фрейм данных добавляется целевая переменная (y).

Код состоит из трех частей. Сначала мы выбираем столбцы cluster, target и c. После этого производится группировка по столбцам cluster и target. Наконец, строка группы вычисляется простым подсчетом.

Добавление переменной c – всего лишь трюк для упрощения подсчета в будущем. Значение выбрано произвольно, потому что нам нужен столбец для подсчета строк.

Инициализация кластерной модели k-средних с 3 кластерами. Значение random_state определяет случайную затравку; если его не указать, то затравка тоже будет случайной. Мы остановились на 3 кластерах, потому что, как видно из последнего листинга, это значение может обеспечить хороший компромисс между сложностью и быстродействием.

Матрица, представляющая этот результат классификации, позволяет увидеть, насколько успешно прошла кластеризация. Для кластера 0 все прошло идеально. В кластерах 1 и 2 встречается небольшая путаница, но в общем мы получаем всего 16 (14+2) неверных классификаций из 150.

Выбирать между контролируемым и неконтролируемым обучением приходится не всегда; в некоторых случаях эти два метода удается объединить.

		c
cluster	target	
0	0	50
1	1	48
	2	14
2	1	2
	2	36

Рис. 3.11. Результат классификации ирисов

3.4. Частично контролируемое обучение

Как бы нам ни хотелось получать только размеченные данные, чтобы применять более мощные методы контролируемого машинного обучения, на практике чаще приходится иметь дело с данными, имеющими минимальную разметку, а то и вовсе никакой. Мы можем воспользоваться методами неконтролируемого машинного обучения, чтобы проанализировать имеющиеся данные и, возможно, добавить метки в набор данных, но полная разметка становится непозволительно дорогой. В таком случае приходится тренировать модели на минимальной разметке данных. И тогда на помощь приходят методы частично контролируемого обучения — гибриды двух методов, которые вам уже знакомы.

Для примера возьмем диаграмму на рис. 3.12. В данных присутствует всего два помеченных наблюдения; обычно этого слишком мало для каких-либо обоснованных прогнозов.

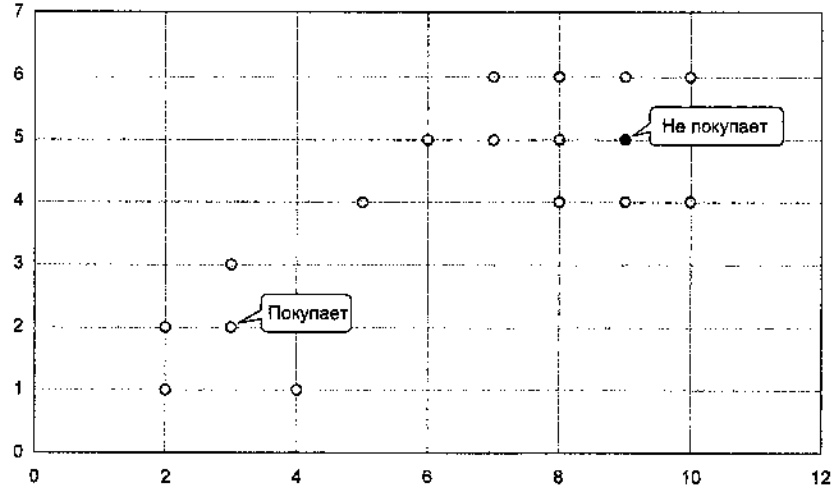


Рис. 3.12. На диаграмме всего два помеченных наблюдения — слишком мало для контролируемого анализа, но достаточно для начала неконтролируемого или частично контролируемого обучения

Популярным методом частично контролируемого обучения является *распространение меток*. В этом методе вы начинаете с размеченного набора данных и назначаете одну метку сходным точкам данных. Происходящее напоминает применение алгоритма кластеризации к набору данных и пометке каждого кластера в соответствии с содержащимися в нем метками. Применив этот метод к набору данных на рис. 3.12, мы получим нечто вроде рис. 3.13.

Среди методов частично контролируемого обучения стоит отдельно упомянуть метод *активного обучения*. При активном обучении программа сообщает, какие

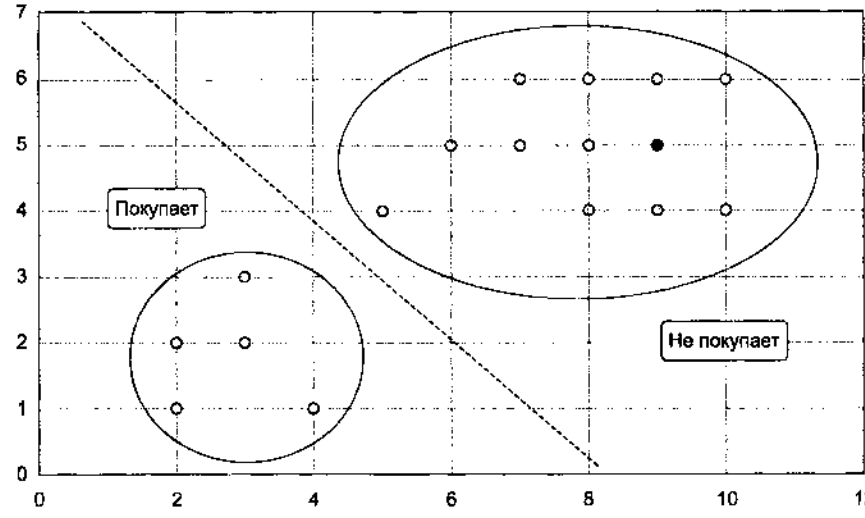


Рис. 3.13. На предыдущей диаграмме данные содержали всего два помеченных наблюдения — безусловно, для контролируемого обучения этого недостаточно. Диаграмма показывает, как можно воспользоваться структурой набора данных для получения более точных классификаторов, чем полученные только на основе помеченных данных. Данные разбиваются на два кластера методом кластеризации; помеченных значений всего два, но при определенной смелости можно предположить, что другие значения из того же кластера имеют ту же метку («Покупает» или «Не покупает»). Этот метод не идеален; лучше использовать реальные метки, если это возможно

наблюдения она хотела бы видеть помеченными для следующей итерации обучения, на основании заданного вами критерия. Например, можно настроить ее для пометки наблюдений, вызывающих у алгоритма наибольшие сомнения, или же воспользоваться несколькими моделями для создания прогноза и выбора точек, вызывающих наибольшее расхождение у моделей.

Итак, вы овладели основами машинного обучения, и в следующей главе будет рассмотрена тема машинного обучения в условиях одного компьютера. Эта задача может оказаться весьма нетривиальной, если набор данных слишком велик для одновременной загрузки в память.

3.5. Итоги

Основные положения этой главы:

- Специалисты data science в значительной мере полагаются на методы статистики и машинного обучения для проведения моделирования. Область машинного обучения имеет много практических применений, от классификации птичьего пения до прогнозирования извержений вулканов.

- Процесс моделирования состоит из четырех фаз:
  1. Планирование показателей, подготовка данных и параметризация модели — вы определяете входные параметры и переменные для своей модели.
  2. Тренировка модели — модель получает данные и изучает закономерности, скрытые в данных.
  3. Выбор и проверка адекватности модели — модель может работать хорошо или плохо; на основании ее эффективности выбирается модель, которая дает самый практичный результат.
  4. Применение тренированной модели к незнакомым данным — убедившись в том, что модель надежна, вы передаете ей новые данные. Если работа была выполнена качественно, модель сообщает дополнительную информацию или дает хороший прогноз того, что ждет вас в будущем.
- Две основные разновидности методов машинного обучения:
  1. *Контролируемые* — методы обучения, требующие помеченных данных.
  2. *Неконтролируемые* — методы обучения, не требующие помеченных данных, но обычно менее точные или надежные, чем контролируемые.
- *Частично контролируемые* методы занимают промежуточное положение и используются в тех случаях, когда помечена только небольшая часть данных.
- Для демонстрации методов контролируемого и неконтролируемого обучения были рассмотрены два учебных примера:
  - В первом примере наивный байесовский классификатор использовался для классификации изображений и распознавания цифр. Также была рассмотрена матрица несоответствий как способ оценки качества работы классификационной модели.
  - В примере неконтролируемых методов было продемонстрировано применение анализа первичных компонент (PCA) для сокращения количества входных переменных при сохранении большей части информации.

Работа с большими данными на одном компьютере

В этой главе:

- ✓ Работа с большими наборами данных на одном компьютере.
- ✓ Библиотеки Python для работы с большими наборами данных.
- ✓ Важность правильного выбора алгоритмов и структур данных.
- ✓ Возможности адаптации алгоритмов для работы в базах данных.

Что делать, если объем данных выходит за рамки ваших возможностей, а ваших методов для них оказывается недостаточно? Как вы поступите: сдадитесь или попробуете приспособиться?

К счастью, вы решили приспособиться, потому что вы читаете дальше. В этой главе представлены методы и инструменты для работы с большими объемами данных, которые можно обрабатывать на одном компьютере при применении правильных средств.

В этой главе представлены инструменты для выполнения классификации и регрессии с данными, не помещающимися в оперативную память вашего компьютера, тогда как глава 3 ориентировалась на наборы данных в памяти. Глава 5 пойдет еще дальше и научит вас работать с наборами данных, для которых необходимо несколько компьютеров. Говоря о больших данных в этой главе, мы имеем в виду данные, при работе с которыми возникают проблемы с памятью и скоростью, но которые все еще могут обрабатываться на одном компьютере.

Глава открывается кратким обзором проблем, встречающихся при работе с большими наборами данных. Затем предлагаются три типа решений для таких проблем: адаптация алгоритмов, правильный выбор структур данных и правильный выбор инструментов. Специалисты data science не единственные, кому приходится работать с большими объемами данных, поэтому вы можете использовать уже наработанные методы. Наконец, новые знания будут применены в двух

учебных примерах. Первый пример показывает, как распознавать вредоносные URL-адреса, а второй демонстрирует построение рекомендательной системы внутри базы данных.

4.1. Проблемы при работе с большими объемами данных

Большие объемы данных создают новые проблемы, такие как перегрузка памяти и алгоритмы, которые работают бесконечно. Они заставляют вас адаптироваться и расширять свой ассортимент методов. Но даже если вы можете провести анализ, необходимо учитывать такие проблемы, как нехватка ресурсов ввода/вывода и процессора, потому что они могут обернуться проблемами со скоростью выполнения.

На рис. 4.1 изображена общая схема, которая будет постепенно раскрываться по мере изучения основных блоков: проблем, решений и общих рекомендаций.

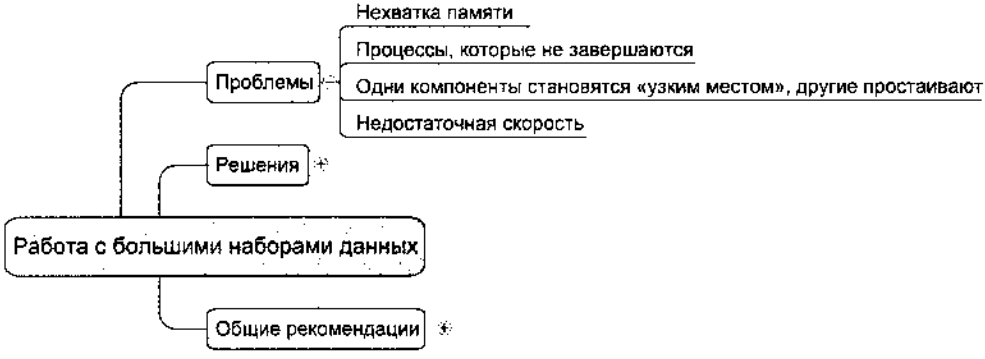


Рис. 4.1. Сводка проблем, встречающихся при работе с наборами данных, не помещающимися в памяти

Размер оперативной памяти на компьютере ограничен. Когда вы пытаетесь втиснуть в эту память больше данных, чем в ней может поместиться, ОС начинает выгружать блоки памяти на диск и скорость работы с данными резко падает. При этом количество алгоритмов, специально разработанных для больших наборов данных, невелико; чаще алгоритмы пытаются загрузить весь набор в память одновременно, что приводит к ошибке нехватки памяти. Другие алгоритмы хранят в памяти несколько копий данных или промежуточные результаты. Все это усугубляет проблему.

Даже если вы справились с проблемами памяти, возможно, придется разбираться с другим ограниченным ресурсом: *временем*. И хотя компьютер может думать, что

вы можете подождать миллион-другой лет, на практике это обычно не так (разве что вас заморозят до завершения работы программы). Некоторые алгоритмы не учитывают, сколько времени прошло; они просто продолжают работать бесконечно. Другие алгоритмы не могут завершиться за разумное время, когда нужно обработать всего несколько мегабайт данных.

Третья особенность работы с большими наборами данных — то, что некоторые компоненты вашего компьютера начинают создавать «узкие места», пока другие системы простаивают. И хотя эта проблема не настолько серьезна, как бесконечная работа программы или нехватка памяти, она все равно ведет к заметным затратам. Экономию следует рассматривать в контексте человеко-дней и затрат на вычислительную инфраструктуру. Некоторые программы не успевают достаточно быстро поставлять данные процессору, потому что им приходится читать данные с жесткого диска — одного из самых медленных компонентов компьютера. Проблема отчасти была решена с появлением твердотельных дисков (SSD, Solid State Drive), но диски SSD все еще стоят намного дороже более медленных и более распространенных жестких дисков (HDD).

4.2. Общие методы обработки больших объемов данных

Бесконечные вычисления, ошибки нехватки памяти и проблемы скорости — самые распространенные проблемы, возникающие при работе с большими данными. В этом разделе мы исследуем решения, направленные на преодоление или смягчение этих проблем.

Решения можно разделить на три категории: правильный выбор алгоритмов, правильный выбор структур данных и использование правильных инструментов (рис. 4.2).

Рис. 4.2. Обзор решений по обработке больших наборов данных

Между проблемами и решениями не существует однозначного соответствия, потому что многие решения направлены одновременно и на нехватку памяти, и на

производительность вычислений. Например, сжатие наборов данных помогает решить проблемы с памятью, потому что набор данных занимает меньше места. С другой стороны, оно также влияет на скорость вычислений: часть нагрузки перекладывается с медленного жесткого диска на быстрый процессор. В отличие от оперативной памяти, вся информация на жестком диске сохраняется даже при отключении питания, но запись на диск занимает больше времени, чем изменение информации в оперативной памяти (неспособной к долгосрочному хранению). Таким образом, при частом изменении информации оперативная память предпочтительнее более надежного жесткого диска. С распакованным набором данных производятся многочисленные операции ввода/вывода (чтения и записи), но процессор в основном простаивает, тогда как при работе со сжатым набором данных процессору достается повышенная часть нагрузки. Помните об этом, когда мы будем рассматривать разные решения.

4.2.1. Правильный выбор алгоритма

Правильный выбор алгоритма позволяет решить больше проблем, чем добавление или обновление оборудования. Алгоритм, хорошо подходящий для работы с большими данными, не обязан загружать весь набор данных в память для создания прогноза. В идеале алгоритм также должен поддерживать параллельные вычисления. В этом разделе будут рассмотрены три типа алгоритмов, обладающие этим свойством: *онлайновые алгоритмы*, *блочные алгоритмы* и *алгоритмы MapReduce* (рис. 4.3).

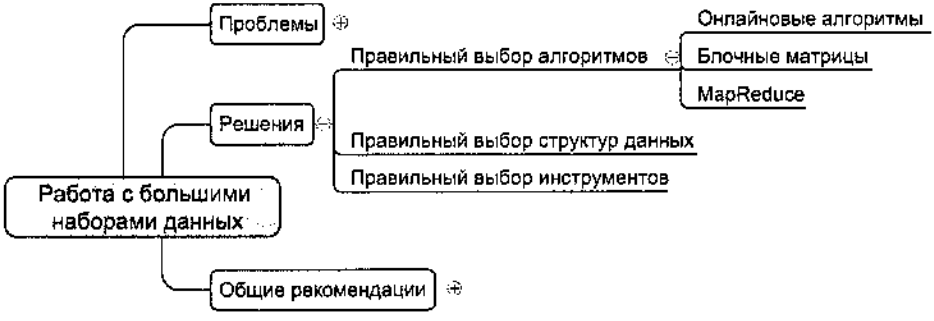


Рис. 4.3. Обзор решений по обработке больших наборов данных

Онлайновые алгоритмы

Некоторые, хотя и не все, алгоритмы машинного обучения могут тренироваться на одиночных наблюдениях вместо загрузки всех данных в память. При поступлении новой точки данных модель учитывает ее в тренировке, и это наблюдение можно отбросить: его эффект уже включен в параметры модели. Например, модель для

прогнозирования погоды может использовать разные параметры (такие, как атмосферное давление или температура) в разных регионах. Когда данные одного региона загружаются в алгоритм, он забывает об этих исходных данных и переходит к следующему региону. Принцип «использовать и забыть» идеально подходит для решения проблем с памятью, потому что одно наблюдение вряд ли когда-нибудь сможет заполнить всю память современного компьютера.

Листинг 4.1 демонстрирует применение этого принципа к перцептрону с онлайновым обучением. *Перцептрон* является одним из самых простых алгоритмов машинного обучения, предназначенных для бинарной классификации (0 или 1): например, купит клиент товар или нет.

Листинг 4.1. Тренировка перцептрона на наблюдениях

Скоростью обучения алгоритма называется величина поправки, вносимой им при каждом поступлении нового наблюдения. Если поправка высока, то модель быстро адаптируется к новым наблюдениям, но может произойти "перелет", мешающий получению точного результата. Сильно упрощенный пример: оптимальный (и неизвестный) вес переменной x=0,75. Текущая оценка равна 0,4 со скоростью обучения 0,5; поправка = 0,5 (скорость обучения) * 1(размер ошибки) * 1 (значение x) = 0,5. 0,4 (текущий вес) + 0,5 (поправка) = 0.9 (новый вес) вместо 0,75. Поправка оказалась слишком большой для получения правильного результата.

Создание класса перцептрона.

Метод __init__ любого класса Python всегда выполняется при создании экземпляра класса. В нем инициализируются некоторые значения по умолчанию.

```
import numpy as np
class perceptron():
    def __init__(self, X,y, threshold = 0.5,
learning_rate = 0.1, max_epochs = 10):
        self.threshold = threshold
        self.learning_rate = learning_rate
        self.X = X
        self.y = y
        self.max_epochs = max_epochs
```

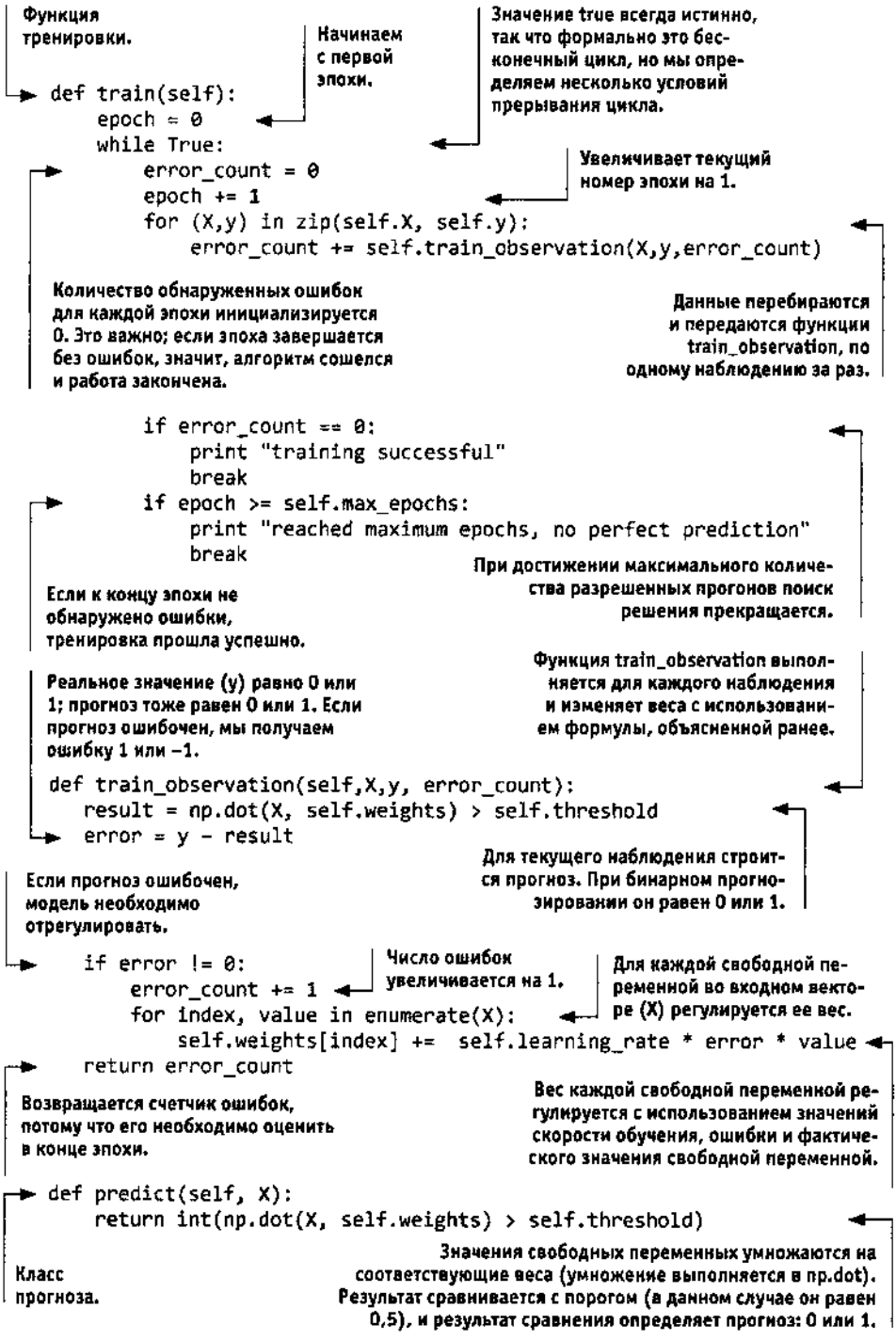
Переменная threshold определяет точку отсечения между 0 и 1. Она решает, какой прогноз будет выдан: 0 или 1. Часто точка отсечения располагается в середине (0,5), но выбор зависит от конкретной ситуации.

Под "эпохой" понимается один прогон по всем данным. Мы разрешаем до 10 прогонов, после которых перцептрон останавливается.

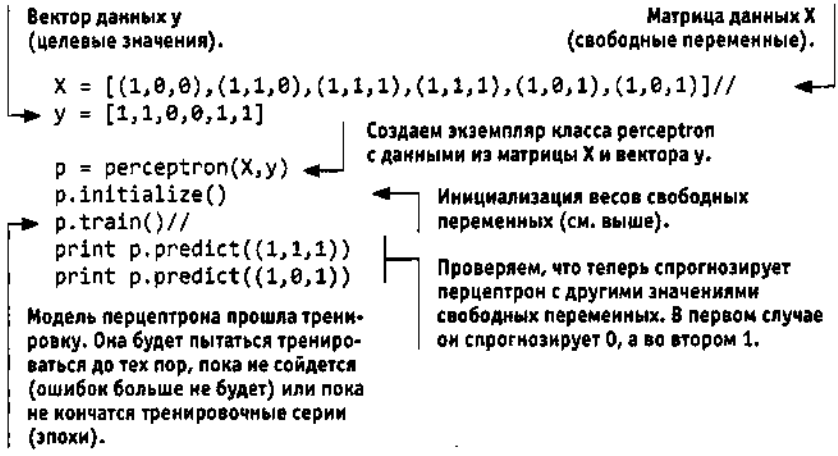
В классе присваиваются переменные x и y.

Каждому наблюдению будет назначен вес. Функция initialize назначает веса для всех входящих наблюдений. Мы допускаем один из двух вариантов: либо все веса начинаются с 0, либо им присваиваются небольшие (от 0 до 0,05) случайные значения.

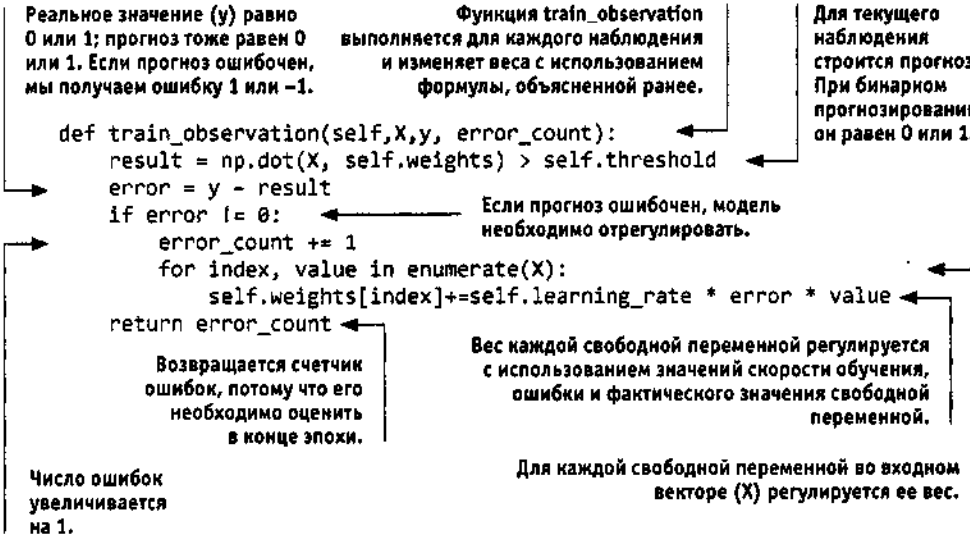
```
    def initialize(self, init_type = 'zeros'):
        if init_type == 'random':
            self.weights = np.random.rand(len(self.X[0])) * 0.05
        if init_type == 'zeros':
            self.weights = np.zeros(len(self.X[0]))
```




```
def train(self):
    epoch = 0
    while True:
        error_count = 0
        epoch += 1
        for (X,y) in zip(self.X, self.y):
            error_count += self.train_observation(X,y,error_count)

        if error_count == 0:
            print "training successful"
            break
        if epoch >= self.max_epochs:
            print "reached maximum epochs, no perfect prediction"
            break

def train_observation(self,X,y, error_count):
    result = np.dot(X, self.weights) > self.threshold
    error = y - result

    if error != 0:
        error_count += 1
        for index, value in enumerate(X):
            self.weights[index] +=  self.learning_rate * error * value
    return error_count

def predict(self, X):
    return int(np.dot(X, self.weights) > self.threshold)
```



Обратимся к частям кода, которые могут быть не столь очевидны без объяснения. Начнем с описания того, как работает функция `train_observation()`. Эта функция состоит из двух больших частей. Первая часть вычисляет прогноз наблюдения и сравнивает его с фактическим значением, а вторая изменяет веса, если прогноз оказался ошибочным.



Прогноз (y) вычисляется умножением входного вектора независимых переменных и суммированием слагаемых (как при линейной регрессии.) Затем это значение сравнивается с порогом. Если оно больше порога, алгоритм выдает 1, а если меньше, то алгоритм выдает 0. Выбор порога — дело субъективное, зависящее от бизнес-

сценария. Предположим, вы предсказываете, болен ли пациент смертельным заболеванием; 1 — болен, 0 — здоров. В этом случае лучше установить более низкий порог: обнаружить несуществующее заболевание и провести повторное обследование лучше, чем пропустить болезнь и обречь пациента на смерть. Вычисляется ошибка, которая определяет направление изменения весов:

```
result = np.dot(X, self.weights) > self.threshold
error = y - result
```

Веса изменяются в соответствии со знаком ошибки. Обновление производится по правилу обучения перцептронов. Для каждого веса в векторе весов его значение обновляется по формуле

$$\Delta w_i = \alpha \varepsilon x_i,$$

где Δw_i — величина изменения веса, α — скорость обучения, ε — ошибка, а x_i — i-е значение во входном векторе (i-я свободная переменная). В переменной `error_count` хранится количество неверно спрогнозированных наблюдений в текущей эпохе; это значение возвращается вызывающей функции. Если исходный прогноз был ошибочным, переменная увеличивается на 1. Напомним, что эпохой называется один тренировочный прогон по всем наблюдениям.

```
if error != 0:
       error_count += 1
            for index, value in enumerate(X):
                self.weights[index] +=  self.learning_rate * error * value
```

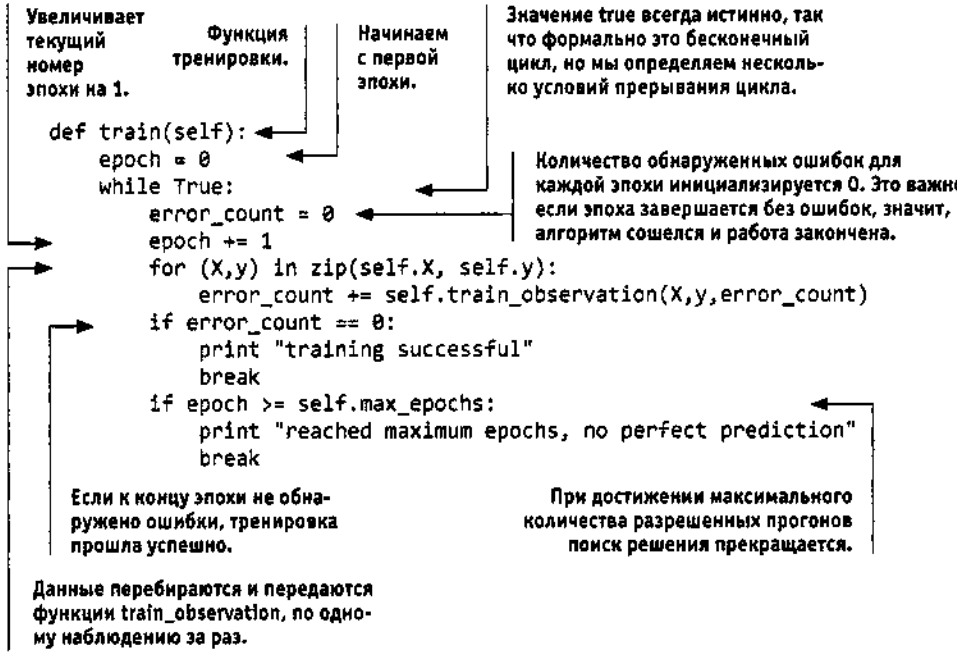
Вторая функция, которая будет рассмотрена более подробно, — `train()`. Эта функция содержит внутренний цикл, который продолжает обучать перцептрон до тех пор, пока тот не начнет выдавать идеальные прогнозы или не достигнет некоторого количества тренировочных циклов (эпох), как показано в листинге 4.2.

Многие онлайновые алгоритмы также могут работать с мини-пакетами; другими словами, вы можете передавать им пакеты от 10 до 1000 наблюдений одновременно, используя скользящее окно для перебора данных. Возможны три варианта:

- *Полнопакетное обучение* (также называемое *статистическим обучением*) — все данные передаются алгоритму одновременно. Именно этот способ использовался в главе 3.
- *Мини-пакетное обучение* — алгоритму передается небольшая часть наблюдений (100, 1000, ... в зависимости от возможностей вашего оборудования).
- *Онлайновое обучение* — наблюдения передаются алгоритму по-одному.

Онлайновые методы обучения связаны с *потоковыми алгоритмами*, которые «видят» каждую точку данных только один раз. Представьте входящий поток сообщений Твиттера: они загружаются в алгоритмы, после чего наблюдение (сообщение)

Листинг 4.2. Использование функций train



отбрасывается, потому что огромное количество входящих сообщений может создать непосильную нагрузку на оборудование. Онлайновые алгоритмы обучения отличаются от потоковых тем, что они могут «увидеть» одно и то же наблюдение несколько раз. Правда, и онлайновые и потоковые алгоритмы могут обучаться на одиночных наблюдениях. Различаются они тем, что *онлайновые алгоритмы* могут использоваться как со статическими, так и с потоковыми источниками данных, поставляющими данные небольшими пакетами (вплоть до одного наблюдения), что позволяет проходить по данным многократно. У *потоковых алгоритмов*, у которых вычисления обычно должны выполняться немедленно при поступлении данных в систему, такая возможность отсутствует.

Разбиение большой матрицы на несколько меньших

Хотя в предыдущей главе нас практически не интересовало, как именно алгоритм оценивает параметры, иногда этот вопрос стоит рассмотреть более подробно. Например, при разбиении большой таблицы данных на меньшие матрицы все равно можно выполнить линейную регрессию. Логика, на которой базируется разбиение матриц, и способ вычисления линейной регрессии с матрицами описаны на врезке. Пока достаточно знать, что библиотеки Python, которые мы будем использовать, позаботятся о разбиении матриц, а переменные веса линейной регрессии могут быть вычислены средствами матричного исчисления.

БЛОЧНЫЕ МАТРИЦЫ И МАТРИЧНАЯ ФОРМУЛА ОЦЕНКИ КОЭФФИЦИЕНТОВ ЛИНЕЙНОЙ РЕГРЕССИИ

Некоторые алгоритмы могут быть преобразованы в алгоритмы, использующие блоки матриц вместо полных матриц. При разбиении матрицы в блочную матрицу полная матрица делится на части, а вы работаете с меньшими частями вместо полной матрицы. В этом случае меньшие матрицы можно загрузить в память и выполнить с ними вычисления, избежав тем самым ошибки нехватки памяти. На рис. 4.4 показано, как переписать матричное сложение A+B в форме с подматрицами.

$$A+B=\begin{bmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,m} \end{bmatrix}+\begin{bmatrix} b_{1,1} & \cdots & b_{1,m} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,m} \end{bmatrix}$$

$$=\left[\begin{array}{ccc} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{j,1} & \cdots & a_{j,m} \\ \hline a_{j+1,1} & \cdots & a_{j+1,m} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,m} \end{array}\right]+\left[\begin{array}{ccc} b_{1,1} & \cdots & b_{1,m} \\ \vdots & \ddots & \vdots \\ b_{j,1} & \cdots & b_{j,m} \\ \hline b_{j+1,1} & \cdots & b_{j+1,m} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,m} \end{array}\right]=\begin{bmatrix} A_1 \\ \hline A_2 \end{bmatrix}+\begin{bmatrix} B_1 \\ \hline B_2 \end{bmatrix}$$

Рис. 4.4. Блочные матрицы могут использоваться для вычисления суммы матриц A и B

Формула на рис. 4.4 показывает, что нет никаких различий между сложением матриц A и B за один шаг или сложением верхних половин матриц с последующим сложением нижних половин.

Все основные операции с матрицами и векторами, такие как умножение, инверсия и сингулярное разложение (метод сокращения количества переменных вроде PCA), могут быть записаны в контексте блочных матриц. Операции с блочными матрицами экономят память за счет разбиения задачи на меньшие, легко параллелизуемые блоки.

Хотя числовые пакеты обычно содержат высокооптимизированный код, они работают только с матрицами, помещающимися в памяти, и используют блочные матрицы в памяти там, где это приносит пользу. Если матрицы не помещаются в памяти, пакеты не будут выполнять оптимизацию за вас и вам придется самостоятельно выполнить разбиение и реализовать версию с блочными матрицами.

Линейная регрессия используется для прогнозирования непрерывных переменных линейной комбинацией свободных переменных; один из простейших способов выполнения вычислений основан на применении так называемого обычного метода наименьших квадратов. Формула в матричной форме имеет вид

$$\beta=(X^TX)^{-1}X^Ty,$$

где β — коэффициенты, которые требуется получить, X — свободные переменные, y — целевая переменная.

Для решения этой задачи Python предоставляет в ваше распоряжение следующие инструменты:

- bcolz — библиотека Python, которая обеспечивает компактное хранение массивов данных и использует жесткий диск, если данные не помещаются в основной памяти.
- Dask — библиотека, которая позволяет оптимизировать последовательность вычислений и упрощает их параллелизацию. Dask не включается в стандартную конфигурацию Anaconda, поэтому перед выполнением приведенного ниже кода в виртуальной среде должна быть выполнена команда conda install dask. Внимание: при импортировании Dask в 64-разрядной версии Python иногда возникают ошибки. Dask зависит от нескольких других библиотек (например, toolz), но pip или conda должны разрешать эти зависимости автоматически.

В листинге 4.3 продемонстрированы вычисления с блочными матрицами с использованием этих библиотек.

Листинг 4.3. Вычисления с блочными матрицами с использованием библиотек bcolz и Dask

```
import dask.array as da
import bcolz as bc
import numpy as np
import dask

n = 1e4

ar = bc.carray(np.arange(n).reshape(n/2,2)  , dtype='float64',
     rootdir = 'ar.bcolz', mode = 'w')
y  = bc.carray(np.arange(n/2), dtype='float64', rootdir =
     'yy.bcolz', mode = 'w')


dax = da.from_array(ar, chunks=(5,5))
dy = da.from_array(y,chunks=(5,5))

XTX = dax.T.dot(dax)
Xy  = dax.T.dot(dy)
```

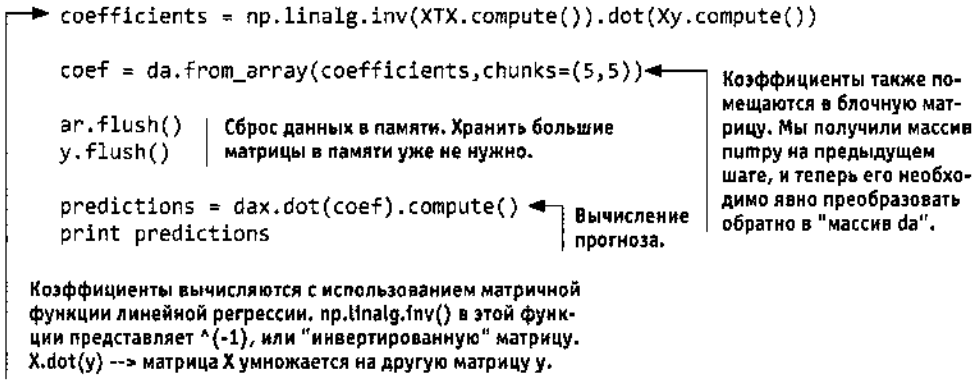
Количество наблюдений (в экспоненциальной записи): 1e4 = 10.000. При желании вы можете изменить это значение.

Создание фиктивных данных: np.arange(n). reshape(n/2,2) создает матрицу 5000 на 2 (потому что мы присвоили n значение 10.000). bc.carray = numpy - расширение массива, которое может выгружаться на диск. Данные в нем хранятся в сжатом виде. rootdir = 'ar.bcolz' --> создает файл на диске при нехватке оперативной памяти. Вы можете найти его в своей файловой системе рядом с файлом ipython или в любом другом каталоге, из которого был запущен код. mode = 'w' --> режим записи. dtype = 'float64' --> тип хранения данных (вещественные числа).

Блочные матрицы создаются для свободных (ar) и целевых (y) переменных. Блочная матрица представляет собой матрицу, разделенную на части (блоки). da.from_array() читает данные с диска или из оперативной памяти (в зависимости от того, где они сейчас находятся). chunks=(5,5): каждый блок является матрицей 5×5 (кроме ситуации, когда осталось < 5 наблюдений или переменных).

XTX определяется как произведение матрицы на ее транспонированную версию. Это структурный элемент формулы для вычисления линейной регрессии средствами матричного исчисления.

Xy — вектор y, умноженный на транспонированную матрицу X. И снова матрица только определяется, но еще не вычисляется. Это еще один структурный элемент формулы для вычисления линейной регрессии средствами матричного исчисления (см. формулу).



Обратите внимание: использовать инверсию блочных матриц не нужно, потому что XTX является квадратной матрицей со стороной, равной количеству свободных переменных. Это удобно, потому что Dask еще не поддерживает инверсию блочных матриц. Дополнительную информацию о работе с матрицами можно найти в Википедии по адресу *https://en.wikipedia.org/wiki/Matrix_(mathematics)*.

MapReduce

Алгоритмы MapReduce легко понять по аналогии: представьте, что вам предложено подсчитать голоса на национальных выборах. В вашей стране 25 партий, 1500 избирательных участков и 2 миллиона людей. Как это можно сделать? Можно собрать все избирательные бюллетени со всех избирательных участков и подсчитать их централизованно, а можно приказать избирательным участкам подсчитать голоса по каждой из 25 партий и передать вам результаты, после чего объединить их по партиям.

Технология отображения-свертки (MapReduce) работает по второму принципу. Сначала выполняется отображение набора значений на ключ, а затем в фазе свертки производится объединение по этому ключу. Взгляните на псевдокод в листинге 4.4, чтобы лучше понять суть происходящего.

Листинг 4.4. Пример псевдокода MapReduce

```
Для каждого человека на избирательном участке:
    Yield (voted_party, 1)
Для каждого голоса на избирательном участке:
    add_vote_to_party()
```

Одно из преимуществ алгоритмов отображения-свертки — простота параллелизации и распределения. Это объясняет их успех в таких распределенных средах, как Hadoop, но они также могут использоваться на отдельных компьютерах. Они будут более подробно рассмотрены в следующей главе, а пример реализации (JavaScript)

представлен в главе 9. При реализации вычислений MapReduce на Python вам не придется начинать «с нуля». Существуют библиотеки, которые выполняют большую часть работы за вас, — Hadoopy, Octopy, Disco или Dumbo.

4.2.2. Правильный выбор структуры данных

Алгоритмы могут стать определяющим фактором успеха или неудачи ваших программ, но способ хранения данных не менее важен. Структуры данных предъявляют разные требования к хранению данных, но они также влияют на производительность операций создания/чтения/обновления/удаления (CRUD, Create/Read/Update/Delete) и других операций набора данных.

Как видно из рис. 4.5, существует много разных структур данных, три из которых рассматриваются ниже: разреженные данные, древовидные данные (деревья) и хеши. Начнем с разреженных наборов данных.

Рис. 4.5. Структуры данных, часто применяемые в теории обработки данных при работе с большими данными

Разреженные данные

Разреженный набор данных содержит относительно мало информации по сравнению с количеством элементов (наблюдений). Взгляните на рис. 4.6: почти все элементы равны «0», и только одно значение «1» присутствует в переменной 9 второго наблюдения.

Такие данные выглядят нелепо, но часто вы получаете именно такие данные при преобразовании текстовых данных в двоичные. Представьте себе набор из 100 000 совершенно несвязанных сообщений в Твиттере. Вероятно, большая часть сообщений содержит менее 30 слов, но вместе они могут содержать сотни и тысячи разных слов. В главе, посвященной глубокому анализу текста, мы займемся процессом разбиения текстовых документов на слова и сохранением их в векторах. А теперь представьте, что получится, если каждое слово преобразуется в двоичную переменную: «1» означает «слово присутствует в сообщении», а «0» — «слово отсутствует в сообщении». Так вы получите типичный набор разреженных данных.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рис. 4.6. Пример разреженной матрицы: почти все данные равны 0, другие значения в разреженных матрицах можно считать аномалиями

Полученная большая матрица может создать нехватку памяти, даже если она содержит относительно мало реальной информации.

К счастью, такие данные могут храниться в сжатом виде. В случае рис. 4.6 сжатие может выглядеть так:

```
data = [(2,9,1)]
```

Строка 2, столбец 9 содержит значение 1.

Поддержка работы с разреженными матрицами в Python расширяется. Многие современные алгоритмы поддерживают или возвращают разреженные матрицы.

Древовидные структуры

В категории *древовидных структур данных* выборка информации выполняется намного быстрее, чем сканирование по таблице. Дерево всегда содержит корневой элемент и дочерние элементы, каждый из которых может иметь свои дочерние элементы, и т. д. Простые примеры древовидных структур — генеалогическое древо или обычное биологическое дерево, разделяющееся на ветви, веточки и листья. Простые правила позволяют быстро найти дочернее дерево, в котором находятся ваши данные. На рис. 4.7 показано, как древовидная структура ускоряет поиск нужной информации.

На рис. 4.7 поиск начинается от корня дерева. Сначала выбирается возрастная категория — очевидно, этот фактор отсечет большую часть альтернатив. Поиск продолжается до тех пор, пока не будет найдено искомое значение. Для тех, кто еще не знаком с программой Акинатор, мы рекомендуем заглянуть на сайт *ru.akinator.com/*. Акинатор — джинн из волшебной лампы, который пытается угадать загаданного вами персонажа, задавая вопросы о нем. Попробуйте, и вы сильно удивитесь... или рассмотрите за волшебством поиск по дереву.

Деревья также часто применяются в базах данных. Базы данных обычно не просматривают содержимое таблиц от первой строки до последней, а используют *индекс* для ускорения поиска. Индексы часто строятся на базе таких структур данных, как деревья и хеш-таблицы. Применение индекса радикально сокращает время выборки данных. Посмотрим, что собой представляют хеш-таблицы.

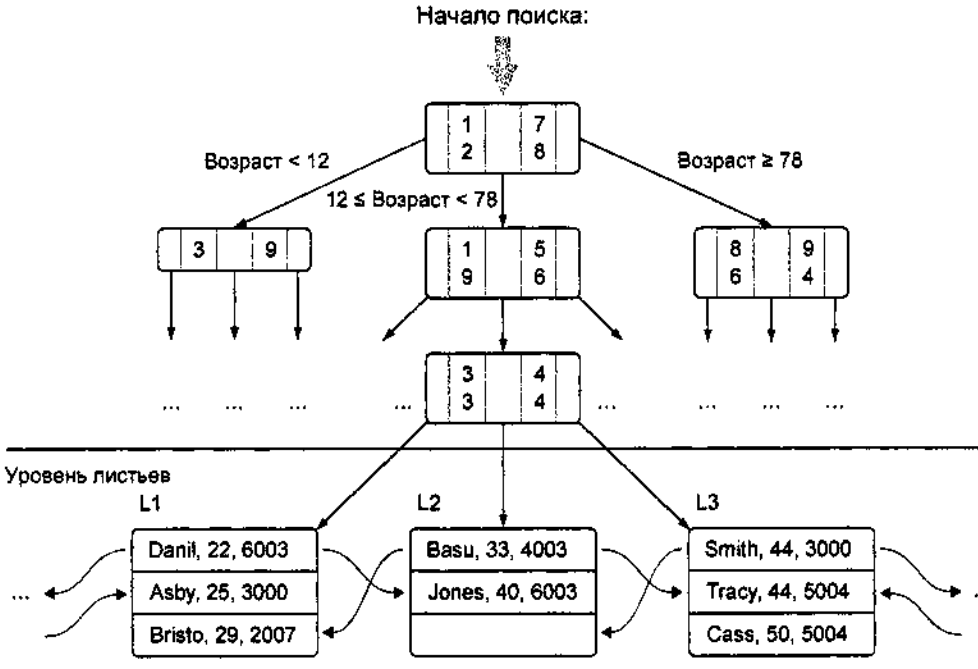


Рис. 4.7. Пример древовидной структуры данных: различные правила (например, возрастные категории) ускоряют поиск нужной информации в генеалогическом древе

Хеш-таблицы

Хеш-таблицы представляют собой структуры данных, которые вычисляют ключ для каждого значения в данных и объединяют ключи в *гнезда* (buckets). Это позволяет быстро извлечь необходимую информацию из гнезда, соответствующего искомым данным. В частности, словари в языке Python реализованы на основе хеш-таблиц; это близкие родственники хранилищ «ключ—значение». Мы встретимся с хеш-таблицами в последнем примере этой главы, когда займемся построением рекомендательной системы в базе данных. Хеш-таблицы широко применяются в базах данных для создания индексов, ускоряющих выборку информации.

4.2.3. Правильный выбор инструментов

После того как вы выберете правильный класс алгоритмов и структур данных, следует выбрать правильный инструмент. Таким инструментом может быть библиотека Python или по крайней мере инструмент, которым можно управлять из Python (рис. 4.8). Полезных инструментов очень много, поэтому мы рассмотрим лишь небольшое подмножество.

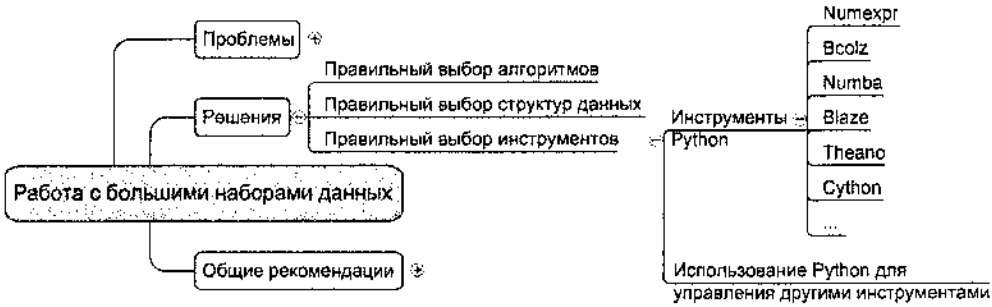


Рис. 4.8. Инструменты, используемые при работе с большими данными

Инструменты Python

Python содержит несколько библиотек, упрощающих работу с большими данными. Библиотеки предоставляют разную функциональность, от умных структур данных и оптимизаторов кода до JIT-компиляторов. Ниже перечислены библиотеки, которые мы предпочитаем использовать при работе с большими данными:

- *Cython* — чем ближе вы работаете к физическому уровню, тем важнее компьютеру знать тип обрабатываемых данных. Для компьютера сложение 1+1 отличается от сложения 1.00+1.00. В первом примере складываются целые числа, а во втором — вещественные, и эти вычисления выполняются разными частями процессора. В языке Python разработчик не указывает используемые типы данных, поэтому компилятор Python должен определять их неявно. Однако определение типа данных — медленная операция, и отчасти это объясняет, почему Python не входит в число самых быстрых языков. Cython, надмножество Python, решает эту проблему, заставляя программиста указывать тип данных во время написания программы. Если компилятор располагает этой информацией, программы выполняются намного быстрее. За дополнительной информацией о Cython обращайтесь по адресу *http://cython.org/*.
- *Numexpr* — Numexpr занимает центральное место в работе многих пакетов больших данных (как NumPy в пакетах для работы с данными в памяти). Numexpr — быстрый вычислитель числовых выражений для NumPy, который может работать во много раз быстрее исходной версии NumPy. Для этого Numexpr переписывает ваше выражение и использует внутренний JIT-компилятор. За подробностями о Numexpr обращайтесь по адресу *https://github.com/pydata/numexpr*.
- *Numba* — для повышения скорости выполнения кода Numba компилирует код непосредственно перед выполнением (так называемая JIT-компиляция). В результате разработчик пишет высокоуровневый код, который выполняется со скоростью, сходной со скоростью кода C. Использовать Numba достаточно просто; см. *http://numba.pydata.org/*.

- *Bcolz* — Bcolz помогает решить проблемы нехватки памяти, которые могут возникнуть при использовании NumPy. Bcolz позволяет сохранять массивы и работать с ними в оптимальной сжатой форме. При этом не только сокращаются затраты памяти для хранения данных, но и незаметно используется библиотека Numexpr для ускорения вычислений с массивами bcolz. См. *http://bcolz.blosc.org/*.

- *Blaze* — Blaze идеально подходит для тех случаев, когда вы хотите воспользоваться всей мощью баз данных, но предпочитаете работать с данными «в стиле Python». Blaze преобразует код Python на SQL, но не ограничивается реляционными базами данных и может работать с множеством других форматов хранения — CSV, Spark и т. д. Blaze предоставляет унифицированный механизм для работы со многими базами и библиотеками данных. Впрочем, Blaze находится в стадии разработки, так что многие возможности еще не реализованы. За дополнительной информацией обращайтесь по адресу *http://blaze.readthedocs.org/en/latest/index.html*.

- *Theano* — Theano позволяет работать напрямую с графическим процессором (GPU) и применять символические упрощения там, где это возможно; при этом используется превосходный JIT-компилятор. Вдобавок эта библиотека отлично подходит для работы с нетривиальной, но полезной математической концепцией — *тензорами*. См. *http://deeplearning.net/software/theano/*.

- *Dask* — Dask позволяет оптимизировать последовательность вычислений и выполнять их более эффективно. Кроме того, у вас появляется возможность выполнять распределенные вычисления. См. *http://dask.pydata.org/en/latest/*.

Эти библиотеки в основном ориентированы на обработку данных непосредственно в Python (если не считать библиотеки Blaze, которая подключается к базам данных). Для достижения высокого быстродействия Python также может использоваться для взаимодействия с разными базами данных или другими программными продуктами.

Использование Python для управления другими инструментами

Многие разработчики ПО поддерживают интерфейс к своим продуктам для языка Python. Это позволяет вам легко пользоваться возможностями специализированных программных продуктов в сочетании с простотой и эффективностью, свойственными Python. В этом отношении Python отличается от других популярных языков обработки данных, таких как R и SAS. Не упускайте эту замечательную возможность и используйте мощь специализированных программ в полной мере. В главе 6 рассматривается учебный пример, в котором Python используется для подключения к базе данных NoSQL, а в главе 7 рассмотрен пример работы с графами.

А теперь перейдем к общим рекомендациям при работе с большими данными.

4.3. Общие рекомендации для программистов при работе с большими наборами данных

Приемы, работающие в контексте общего программирования, актуальны и для data science. Возможно, какие-то формулировки будут выглядеть несколько иначе, но принципы остаются, по сути, неизменными для всех программистов. В этом разделе перечислены рекомендации, играющие особенно важную роль в контексте data science.

Эти рекомендации можно разделить на три группы, как показано на рис. 4.9.

Рис. 4.9. Инструменты, используемые при работе с большими данными

- *Не повторяйте уже выполненную работу.* Пользуйтесь библиотеками и инструментами, написанными другими разработчиками.
- *Используйте все возможности оборудования.* Ваша машина никогда не работает на максимуме своих возможностей; иногда даже простые изменения заставляют ее трудиться более напряженно.
- *Экономьте вычислительные ресурсы.* Сократите затраты памяти и потребности в вычислительных операциях настолько, насколько возможно.

«Не повторяйте уже выполненную работу» применительно к конкретной задаче обычно проще сказать, чем сделать, но вашей первой мыслью должно быть: «Наверняка кто-то уже сталкивался с такой задачей до меня».

4.3.1. Не повторяйте уже выполненную работу

Вероятно, этот принцип лучше простого «не повторяйтесь». Повышайте эффективность своих действий, позаботьтесь о том, чтобы они были действительно значимыми. Решение уже решенной задачи — напрасная трата времени. Специалист data science должен руководствоваться двумя основными правилами, которые помогают ему справиться с большими объемами данных и делают его работу более эффективной.

- *Используйте возможности баз данных.* Первое стремление многих специалистов data science при работе с большими наборами данных — подготовить таблицы своей аналитической базы в базе данных. Этот способ работает, если ваши потребности достаточно просты. Если же подготовка требует нетривиального моделирования, выясните, можно ли применить функции и процедуры, определяемые пользователем. Последний пример этой главы посвящен интеграции базы данных в ваш рабочий процесс.
- *Используйте оптимизированные библиотеки.* Создание таких библиотек, как Mahout, Weka и других алгоритмов машинного обучения, требует времени и знаний. Эти библиотеки высокооптимизированны, в них задействованы передовые методы и ультрасовременные технологии. Расходуйте свое время на решение задач, а не на повторение работы, выполненной другими людьми (если только вы не пытаетесь разобраться в том, как что-то работает).

Затем необходимо принять во внимание аппаратные ограничения.

4.3.2. Используйте все возможности оборудования

Одни ресурсы компьютера могут простаивать, тогда как другие работают в условиях повышенной нагрузки. Это замедляет работу программ, а иногда даже вызывает в них сбои. Иногда возможно (и необходимо) сместить нагрузку с чрезмерно загруженного ресурса на другой, менее востребованный, при помощи следующих приемов:

- *Передавайте процессору сжатые данные.* Простой прием для предотвращения простоев процессора — передача процессору сжатых данных вместо исходных (необработанных) данных. В результате более заметная часть работы передается с жесткого диска на процессор, а это именно то, что нужно, потому что жесткий диск отстает от процессора в большинстве современных компьютерных архитектур.
- *Используйте графический процессор.* Иногда «узким местом» системы оказывается процессор, а не память. Если ваши вычисления можно параллелизировать, переключение на графический процессор может принести ощутимую пользу. Он обладает существенно бо́льшим вычислительным потенциалом, чем центральный процессор. Графический процессор невероятно эффективно работает с параллелизируемыми задачами, но размер кэша у него меньше, чем у центрального процессора. С другой стороны, бессмысленно переключаться на графический процессор, если проблема кроется в жестком диске. Некоторые пакеты Python (такие, как Theano и NumbaPro) используют графический процессор без особых усилий со стороны программиста. Если их возможностей окажется недостаточно, используйте пакет CUDA (Compute Unified Device Architecture), например PyCUDA. Также это хорошо известный способ добычи биткойнов, если вас интересует тема создания собственных денег.

- *Используйте многопоточные вычисления.* Параллельные вычисления возможны и на центральном процессоре, причем задача решается при помощи обычных потоков (threads) Python.

4.3.3. Экономьте вычислительные ресурсы

«Ум + упорство = результат». Это правило также относится и к написанным вами программам. Чтобы избежать проблем при работе с большими данными, проще всего заранее исключить большую часть работы и поручить компьютеру только ту работу, без которой не обойтись. Ниже перечислены некоторые методы для достижения этой цели.

- *Проведите профилирование своего кода и оптимизируйте медленные части.* Не все фрагменты кода нуждаются в оптимизации; при помощи профайлера найдите неэффективные части своих программ и решите проблемы.
- *По возможности используйте откомпилированный код, особенно в том, что касается циклов.* Старайтесь использовать функции из пакетов, оптимизированных для числовых вычислений, вместо того чтобы реализовывать все собственными силами. Часто такие пакеты содержат высокооптимизированный и откомпилированный код.
- *Откомпилируйте код самостоятельно.* Если вы не можете использовать существующий пакет, воспользуйтесь JIT-компилятором или напишите самые медленные части своего кода на языке более низкого уровня (таком, как C или Fortran) и интегрируйте их со своей кодовой базой. Если вы решите перейти на низкоуровневые языки (языки, приближенные к машинному коду), научитесь работать с такими вычислительными библиотеками, как LAPACK, BLAST, Intel MKL и ATLAS. Эти библиотеки высокооптимизированны, и добиться аналогичного быстродействия своими силами будет нелегко.
- *Избегайте загрузки данных в память.* Если вы работаете с данными, не помещающимися в памяти, избегайте загрузки всех данных в память. Самое простое решение — читать данные блоками и обрабатывать их по мере загрузки. Такой способ возможен не во всех алгоритмах, но он позволяет проводить вычисления с очень большими наборами данных.
- *Используйте генераторы для отказа от промежуточного хранения данных.* Генераторы помогают возвращать данные на уровне отдельных наблюдений, а не в виде пакетов. В этом случае вам не придется хранить промежуточные результаты.
- *Используйте как можно меньший объем данных.* Если крупномасштабный алгоритм недоступен и вы не желаете реализовать его самостоятельно, существует возможность тренировки модели на подмножестве исходных данных.

- *Используйте свои знания математики для упрощения вычислений.* Для примера возьмем формулу $(a + b)^2 = a^2 + 2ab + b^2$. Левая сторона вычисляется намного быстрее правой; даже в таком тривиальном примере это может быть существенно при работе с большими объемами данных.

4.4. Пример 1: Прогнозирование вредоносных URL-адресов

Интернет — одно из величайших изобретений современности. Интернет стремительно ускоряет развитие человечества, но не все используют это великое изобретение с благородными целями. Многие компании (Google, например) пытаются защитить нас от мошенников, отыскивая вредоносные сайты за нас. Это непростая задача, потому что для этого нужно просканировать миллиарды веб-страниц в Интернете. В этом примере мы покажем, как работать с набором данных, который не помещается в памяти.

Что нам для этого понадобится:

- *Данные* — в этом примере будут использованы данные, опубликованные в исследовательском проекте. Проект содержит данные за 120 дней, каждое наблюдение содержит приблизительно 3 200 000 показателей. Целевая переменная содержит 1, если сайт является вредоносным, или –1 в противном случае. За дополнительной информацией обращайтесь к материалам доклада «Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs»[1].
- *Библиотека Scikit-learn* — к настоящему моменту эта библиотека должна быть установлена в вашей среде программирования Python, потому что мы использовали ее в предыдущей главе.

Как видите, нужно не так уж много, так что не будем откладывать.

4.4.1. Этап 1: Определение цели исследования

Цель этого проекта — определение того, можно ли доверять некоторым URL-адресам или нет. При таком огромном объеме данных придется подумать об экономии памяти. На следующем шаге мы сначала посмотрим, что произойдет, если не обращать внимания на проблемы, связанные с памятью.

[1] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker, «Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs,» Proceedings of the ACM SIGKDD Conference, Paris (июнь 2009), 1245–53.

4.4.2. Этап 2: Сбор данных URL

Загрузите данные по адресу *http://sysnet.ucsd.edu/projects/url/#datasets* и поместите их в каталог. Выберите данные в формате SVMLight — текстовом формате с одним наблюдением на строку. Для экономии места нули не приводятся.

Листинг 4.5 и рис. 4.10 показывают, что произойдет при попытке прочитать всего 1 файл из 120 и создать обычную матрицу, как того ожидает большинство алгоритмов. Метод `todense()` преобразует данные из специального файлового формата в обычную матрицу, в которой каждый элемент содержит значение.

Листинг 4.5. Генерирование ошибки нехватки памяти

```
import glob
from sklearn.datasets import load_svmlight_file
files = glob.glob('C:\Users\Gebruiker\Downloads\
url_svmlight.tar\url_svmlight\*.svm')
files = glob.glob('C:\Users\Gebruiker\Downloads\
url_svmlight\url_svmlight\*.svm')
print "there are %d files" % len(files)
X,y = load_svmlight_file(files[0],n_features=3231952)
X.todense()
```

Ссылка на файлы (Linux).

Ссылка на файлы (Windows: tar-файл необходимо предварительно распаковать).

Количество файлов.

Загрузка файлов.

Данные представляют собой большую, но разреженную матрицу. Преобразование их в плотную матрицу (каждый 0 представлен в файле) приводит к ошибке нехватки памяти.

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-532-d196c05088ce> in <module>()
      5 print "there are %d files" % len(files)
      6 X,y = load_svmlight_file(files[0] ,n_features=3500000)
----> 7 X.todense()
```

Рис. 4.10. Ошибка при попытке загрузить большой объем данных в память

Сюрприз — ошибка нехватки памяти! Если, конечно, не запустить этот код на очень мощной машине. Впрочем, после нескольких дальнейших приемов ошибка исчезнет и вы будете успешно обнаруживать 97% вредоносных сайтов.

Инструменты и методы

Ошибка памяти возникла даже при загрузке одного файла — остается еще 119. К счастью, для этого у нас в запасе найдется несколько полезных приемов:

- использование разреженного представления данных;
- передача алгоритму сжатых данных вместо необработанных;
- применение онлайнового алгоритма для прогнозирования.

Каждый прием будет более подробно описан тогда, когда мы им воспользуемся. Данные доступны локально, с ними можно работать. Этап 3 процесса data science — подготовка и очистка данных — в данном случае не нужен, потому что URL-адреса проходят предварительную очистку. Впрочем, прежде чем запускать алгоритм обучения, необходимо провести небольшое исследование.

4.4.3. Этап 4: Исследование данных

Чтобы понять, можно ли применить первый прием (разреженное представление данных), необходимо сначала выяснить, действительно ли данные содержат множество нулей. Для этого можно воспользоваться следующим фрагментом кода:

```
print "number of non-zero entries %2.6f"  % float((X.nnz)/(float(X.shape[0])
* float(X.shape[1])))
```

Результат выглядит так:

```
number of non-zero entries 0.000033
```

Данные, содержащие минимум полезной информации по сравнению с нулями, называются *разреженными*. Такие данные можно хранить более компактно, если использовать формат `[(0,0,1),(4,4,1)]` вместо

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

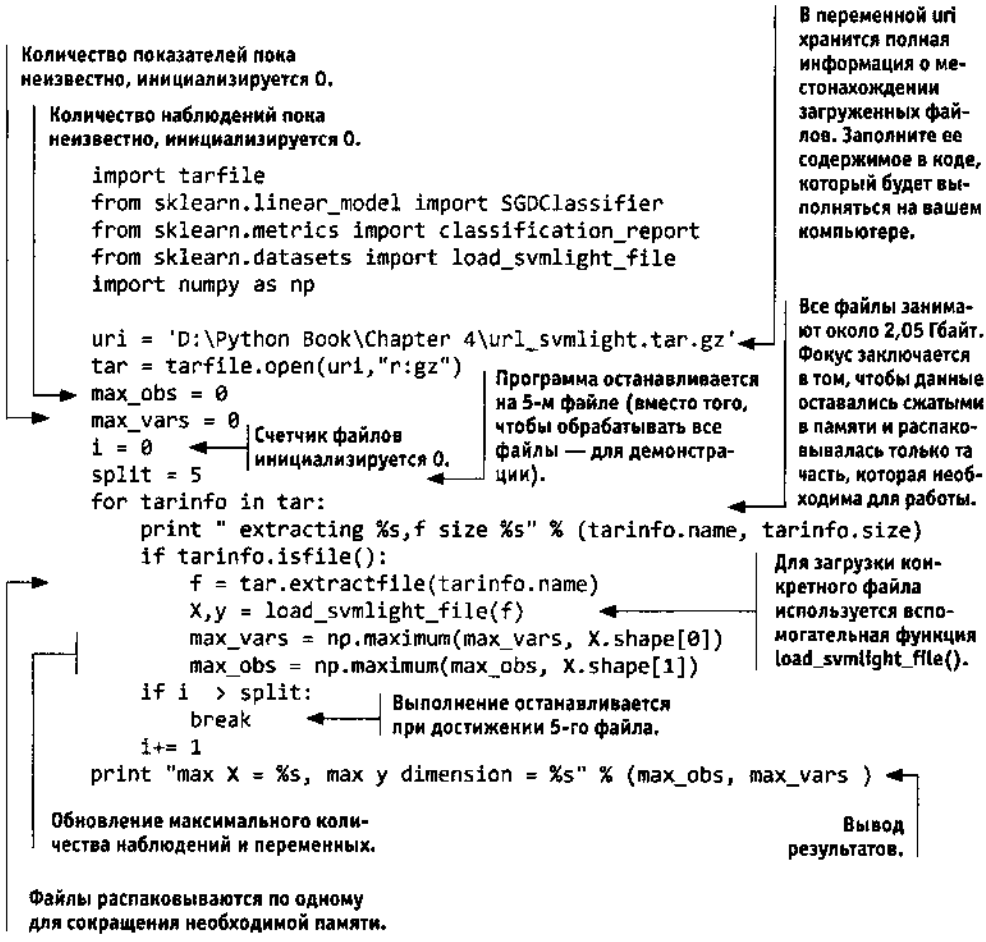
SVMLight входит в число файловых форматов, в которых реализован этот принцип; именно поэтому мы загрузили данные в нем. Однако работа еще не закончена, потому что нужно составить представление о размерах данных.

Для получения этой информации данные уже должны храниться в сжатом виде при проверке максимального количества наблюдений и переменных. Также данные должны читаться последовательно, файл за файлом — таким образом вы будете расходовать еще меньше памяти. Второй прием заключается в передаче процессору сжатых файлов. В нашем примере данные уже упакованы в формате tar.gz. Файл распаковывается только тогда, когда это необходимо, без записи на жесткий диск (самая медленная часть компьютера).

В листинге 4.6 обрабатываются только первые 5 файлов, однако при желании вы можете использовать их все.

Часть кода заслуживает дополнительных пояснений. В этом коде мы перебираем файлы svm в архиве tar. Файлы распаковываются по одному для сокращения затрат

Листинг 4.6. Определение размера данных




```
import tarfile
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import classification_report
from sklearn.datasets import load_svmlight_file
import numpy as np

uri = 'D:\Python Book\Chapter 4\url_svmlight.tar.gz'
tar = tarfile.open(uri,"r:gz")
max_obs = 0
max_vars = 0
i = 0
split = 5
for tarinfo in tar:
    print " extracting %s,f size %s" % (tarinfo.name, tarinfo.size)
    if tarinfo.isfile():
        f = tar.extractfile(tarinfo.name)
        X,y = load_svmlight_file(f)
        max_vars = np.maximum(max_vars, X.shape[0])
        max_obs = np.maximum(max_obs, X.shape[1])
    if i  > split:
        break
    i+= 1
print "max X = %s, max y dimension = %s" % (max_obs, max_vars )
```

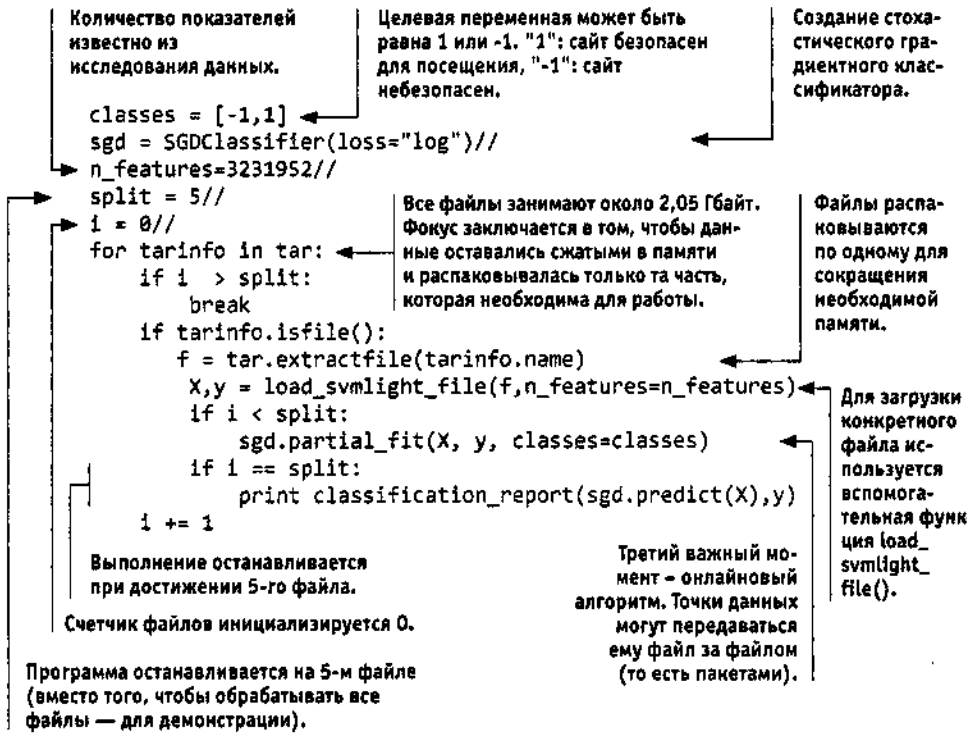
памяти. Так как файлы хранятся в формате SVM, для загрузки конкретного файла используется вспомогательная функция `load_svmlight_file()`. Чтобы узнать, сколько наблюдений и переменных содержит этот конкретный файл, достаточно проверить размеры полученного набора данных.

Вооружившись этой информацией, можно перейти к построению модели.

4.4.4. Этап 5: Построение модели

Теперь, когда мы знаем размеры данных, можно применить те же два приема (разреженное представление и сжатые файлы), добавив к ним третий (использование онлайнового алгоритма), как показано в листинге 4.7. За дело — найдем вредоносные веб-сайты!

Листинг 4.7. Создание модели для выявления вредоносных сайтов



Этот код в целом похож на листинг 4.6, если не считать классификатор статического градиентного спуска `SGDClassifier()`.

Тренировка алгоритма осуществляется в итеративном режиме, с представлением наблюдений из одного файла функцией `partial_fit()`.

При переборе только первых 5 файлов будет получен результат, приведенный в табл. 4.1. В таблице представлены диагностические метрики классификации: точность (precision), полнота (recall), F1-метрика и поддержка (support).

Таблица 4.1. Задача классификации: можно ли доверять веб-сайту?

	Точность	Полнота	F1-метрика	Поддержка
–1	0,97	0,99	0,98	14045
1	0,97	0,94	0,96	5955
avg/total	0,97	0,97	0,97	20000

Всего 3% (1 – 0,97) вредоносных сайтов не были обнаружены (*точность*), а 6% (1 – 0,94) обнаруженных сайтов были обвинены незаслуженно (*полнота*). Это вполне

достойный результат, и мы можем заключить, что методология работает. Если провести анализ заново, результат может быть несколько иным, потому что алгоритмы могут сходиться несколько иначе. Если вы согласны немного подождать, попробуйте полный набор данных. Теперь вы можете обработать все данные без проблем. В этом учебном примере шестой этап (отображение или автоматизация) отсутствует.

Рассмотрим второй пример использования этих приемов; на этот раз мы построим рекомендательную систему внутри базы данных. Хорошо известный пример рекомендательных систем можно найти на сайте Amazon. Во время просмотра вы вскоре столкнетесь с рекомендациями вида «Люди, купившие этот товар, также купили...»

4.5. Пример 2: Построение рекомендательной системы внутри базы данных

На практике большая часть данных, с которыми вы будете работать, хранится в реляционной базе данных, однако многие базы данных не подходят для глубокого анализа. Но, как показывает этот пример, применяемые методы можно адаптировать так, что большая часть анализа будет выполняться в самой базе данных; таким образом вы будете использовать преимущества оптимизатора запросов базы данных, который будет оптимизировать код за вас. В этом примере продемонстрировано использование структуры данных хеш-таблицы, а также управление другими инструментами из кода Python.

4.5.1. Необходимые инструменты и методы

Прежде чем браться за написание кода, стоит в общих чертах представить, какие инструменты и теория нам при этом понадобятся.

Инструменты

- *База данных MySQL* — прежде всего вам понадобится база данных MySQL для хранения информации. Если вы еще не установили сервер MySQL Community Server, загрузите его на сайте *www.mysql.com*. Процесс установки подробно описан в приложении B: «Установка сервера MySQL».
- *Библиотека Python для работы с базой данных MySQL* — для подключения к серверу из Python вам также потребуется SQLAlchemy или другая библиотека, способная взаимодействовать с MySQL; мы будем использовать MySQLdb. В системе Windows вы не сможете сразу воспользоваться Conda для ее установки. Сначала установите Binstar (другая программа управления пакетами) и найдите подходящий пакет mysql-python для своей конфигурации Python.

```
conda install binstar
binstar search -t conda mysql-python
```

- Следующая команда, введенная в командной строке Windows, сработала в нашей системе (после активизации среды Python):

```
conda install --channel https://conda.binstar.org/krisvanneste mysql-python
```

- Впрочем, ничто не мешает вам воспользоваться библиотекой SQLAlchemy, если какие-то из ее возможностей покажутся вам более удобными.
- Также нам понадобится библиотека pandas, но она уже должна быть установлена в вашей системе.

Когда вся необходимая инфраструктура будет готова, можно переходить к рассмотрению методов.

Методы

Простая рекомендательная система ищет клиентов, которые брали напрокат примерно такие же фильмы, как вы, и рекомендует те, которые видели эти клиенты, но еще не видели вы. Этот метод в машинном обучении называется методом *k ближайших соседей*.

Клиент с похожим поведением не всегда оказывается *самым* похожим клиентом. Метод, которым мы воспользуемся, гарантирует поиск похожих клиентов (локальные оптимумы) без гарантий нахождения самого похожего клиента (глобальный оптимум). Для решения этой задачи часто применяется метод *локально-чувствительного хеширования*. Хороший обзор статей по этой теме можно найти по адресу *http://www.mit.edu/~andoni/LSH/*.

В основе локально-чувствительного хеширования лежит простая идея: построение функций, которые размещают похожих клиентов поблизости (в гнезде с одной меткой), а также следят за тем, чтобы разные объекты размещались в разных гнездах.

Естественно, в этой концепции главная роль отводится функции, выполняющей отображение. Такие функции, отображающие произвольный диапазон входных данных на фиксированный набор выходных значений, называются *хеш-функциями*. Простейшая хеш-функция выполняет конкатенацию значений из нескольких произвольно выбранных столбцов. Количество столбцов несущественно (масштабируемый ввод); результат сводится к одному столбцу (фиксированный вывод).

В нашем примере будут определены три функции для поиска похожих клиентов. Эти три функции получают флаги трех фильмов:

- Первая функция получает флаги фильмов 10, 15 и 28.
- Вторая функция получает флаги фильмов 7, 18 и 22.
- Третья функция получает флаги фильмов 16, 19 и 30.

Такое решение гарантирует, что клиенты, находящиеся в одном гнезде, взяли хотя бы несколько одинаковых фильмов; однако клиенты в одном гнезде могут различаться по фильмам, не включенным в функции хеширования. Чтобы решить эту проблему, необходимо иметь возможность сравнивать клиентов из одного гнезда друг с другом. Для этого необходимо создать новую метрику расстояния.

Метрика расстояния, используемая для сравнения клиентов, называется *расстоянием Хэмминга*. Расстояние Хэмминга описывает степень различия двух строк и определяется как количество различающихся символов в строках. Некоторые примеры расстояния Хэмминга представлены в табл. 4.2.

Таблица 4.2. Примеры вычисления расстояния Хэмминга

Строка 1	Строка 2	Расстояние Хэмминга
Hat	Cat	1
Hat	Mad	2
Tiger	Tigre	2
Paris	Rome	5

Сравнение нескольких столбцов — весьма затратная операция, поэтому нам понадобится трюк, ускоряющий ее выполнение. Так как столбцы содержат двоичную переменную (0 или 1), которая указывает, брал клиент этот фильм или нет, все эти флаги можно объединить в один новый столбец. В табл. 4.3 приведена переменная `movies`, которая содержит ту же информацию, что и все объединенные столбцы с флагами.

Таблица 4.3. Объединение информации из разных столбцов в столбец movies (как в молекуле ДНК: вся информация хранится в одной длинной строке)

Столбец 1	Фильм 1	Фильм 2	Фильм 3	Фильм 4	movies
Клиент 1	1	0	1	1	1011
Клиент 2	0	0	0	1	0001

Слияние позволит вычислять расстояние Хэмминга намного эффективнее. При работе с двоичными данными можно воспользоваться оператором XOR (^), который работает по следующей схеме:

```
1^1 = 0
1^0 = 1
0^1 = 1
0^0 = 0
```

После такой подготовки процедура поиска похожих клиентов реализуется очень просто. Сначала рассмотрим ее в виде псевдокода.

Предварительная обработка:

1. Определить p (например, 3) функций, выбирающих k (например, 3) значений из вектора фильмов.
2. Эти функции применяются к каждой точке, а результаты сохраняются в отдельном столбце. (В литературе функции называются хеш-функциями, а в каждом столбце хранится одно гнездо.)

Получение информации о точке q:

1. Применить те же p функций к точке (наблюдению) q, о которой вы хотите получить информацию.
2. Получить для каждой функции точки, соответствующие результату из соответствующего гнезда.
3. Процесс останавливается тогда, когда будут получены все точки из гнезд или до достижения 2p точек (например, 10 для 5 функций).
4. Вычислить расстояние до каждой точки и вернуть точки с наименьшим расстоянием.

Рассмотрим реализацию на языке Python, чтобы описание стало более понятным.

4.5.2. Этап 1: Вопрос исследования

Представьте, что вы работаете в видеопрокате и директор спрашивает, можно ли на основании информации о том, какие фильмы берут клиенты, предсказать, какие фильмы им могут понравиться. Данные хранятся в базе данных MySQL, а вам поручено провести анализ. Фактически речь идет о рекомендательной системе — автоматизированной системе, которая изучает предпочтения клиентов и рекомендует фильмы и другие продукты, которые клиент еще не пробовал. Цель нашего учебного примера — создать рекомендательную систему, экономно расходующую память. Для этого мы воспользуемся базой данных и еще несколькими приемами. Мы создаем данные для учебного примера самостоятельно, поэтому этап сбора данных можно пропустить и перейти сразу к подготовке данных. А после этого можно пропустить этап исследования данных и перейти сразу к построению модели.

4.5.3. Этап 3: Подготовка данных

Данные, собранные директором, приведены в табл. 4.4. Мы самостоятельно создадим эти данные в демонстрационных целях.

Таблица 4.4. Фрагмент базы данных клиентов и фильмов, взятых клиентами

Клиент	Фильм 1	Фильм 2	Фильм 3	...	Фильм 32
Jack Dani	1	0	0		1
Wilhelmson	1	1	0		1
...					
Jane Danc	0	0	1		0
Xi Liu	0	0	0		1
Eros Mazo	1	1	0		1
...					

Для каждого клиента в базе данных хранится информация о том, брал он напрокат этот фильм (1) или нет (0). Посмотрим, что еще необходимо для того, чтобы вы могли представить директору ту рекомендательную систему, которую он хочет увидеть.

Сначала необходимо связать Python с MySQL для создания данных. Подключитесь к MySQL с именем пользователя и паролем. В листинге 4.8 используется база данных с именем «test». Замените имя пользователя, пароль и имя базы данных значениями, соответствующими вашей конфигурации, получите объект подключения и курсор. *Курсор* базы данных представляет собой управляющую структуру, которая отслеживает текущую позицию в базе данных.

Листинг 4.8. Создание клиентов в базе данных

```
import MySQLdb
import pandas as pd

user = '****'
password = '****'
database = 'test'
mc = MySQLdb.connect('localhost',user,password,database)
cursor = mc.cursor()

nr_customers = 100
colnames = ["movie%d" %i for i in range(1,33)]
pd.np.random.seed(2015)
generated_customers = pd.np.random.randint(0,2,32 *
     nr_customers).reshape(nr_customers,32)

data = pd.DataFrame(generated_customers, columns = list(colnames))
data.to_sql('cust',mc, flavor = 'mysql', index = True, if_exists =
     'replace', index_label = 'cust_id')
```

Сначала создаем подключение: укажите свое имя пользователя, пароль и имя схемы (переменная "database").

Затем создается фиктивная база данных с клиентами и несколькими наблюдениями.

Данные сохраняются во фрейме данных Pandas, а фрейм данных сохраняется в таблице MySQL с именем "cust". Если таблица уже существует, она заменяется.

Мы создаем 100 клиентов и случайным образом устанавливаем признаки того, видели клиенты тот или иной фильм или нет. Всего в базе данных 32 фильма. Сначала данные создаются во фрейме данных Pandas, но затем они преобразуются в код SQL. Внимание: при выполнении этого кода может быть выдано предупреждение: «Режим "mysql" для подключений DBAPI устарел и будет исключен в будущих версиях. Дальнейшая поддержка MySQL будет обеспечиваться ядром SQLAlchemy». При желании вы можете перейти на SQLAlchemy или другую библиотеку прямо сейчас. Мы будем использовать SQLAlchemy в других главах, но в этом примере работаем с MySQLdb для полноты картины.

Чтобы обеспечить эффективное выполнение будущих запросов к базе данных, данные необходимо дополнительно подготовить:

- *Создать битовые строки* — сжатые версии содержимого столбцов (значения 0 и 1). Сначала двоичные значения объединяются операцией конкатенации, после чего полученная битовая строка заново интерпретируется как число. Пока описание выглядит абстрактно, но в коде все прояснится.
- *Определить хеш-функции.* Собственно, битовые строки будут создаваться именно хеш-функциями.
- *Добавить в таблицу индекс для ускорения выборки данных.*

Создание битовых строк

Итак, мы создаем промежуточную таблицу для запросов, применения хеш-функций и представления последовательности битов в виде десятичного числа. Наконец, данные помещаются в таблицу.

Сначала необходимо создать битовые строки. Сперва строка «11111111» преобразуется в двоичное или числовое значение для работы функции расстояния Хэмминга. Мы выбрали числовое представление, как видно из листинга 4.9.

Преобразуя информацию 32 столбцов в четыре числа, мы сжимаем ее для последующего поиска. Из рис. 4.11 видно, что мы получим при запросе первых двух наблюдений (история просмотра фильмов клиентом) в этом новом формате:

```
store[0:2]
```

На следующем шаге определяются хеш-функции, применяемые к данным для проверки сходства поведения двух пользователей.

	bit1	bit2	bit3	bit4
0	10	82	42	182
1	23	28	223	180

Рис. 4.11. Информация по всем 32 фильмам для первых двух клиентов после преобразования битовых строк в числовую форму

Листинг 4.9. Создание битовых строк

Строка строится конкатенацией нулей (0) и единиц (1), указывающих, смотрел клиент тот или иной фильм или нет. Затем последовательность интерпретируется как числовое значение. Пример: двоичное число 0011 эквивалентно 3. Функция createNum() берет 8 значений, сцепляет их и преобразует в строку, после чего преобразует двоичное представление строки в число.

```
def createNum(x1,x2,x3,x4,x5,x6,x7,x8)://
    return  [int('%d%d%d%d%d%d%d%d' % (i1,i2,i3,i4,i5,i6,i7,i8),2)
for (i1,i2,i3,i4,i5,i6,i7,i8) in zip(x1,x2,x3,x4,x5,x6,x7,x8)]

assert int('1111',2) == 15
assert int('1100',2) == 12
assert createNum([1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,0],[1,0])
     == [255,252]

store = pd.DataFrame()
store['bit1'] = createNum(data.movie1,
     data.movie2,data.movie3,data.movie4,data.movie5,
data.movie6,data.movie7,data.movie8)
store['bit2'] = createNum(data.movie9,
     data.movie10,data.movie11,data.movie12,data.movie13,
data.movie14,data.movie15,data.movie16)
store['bit3'] = createNum(data.movie17,
     data.movie18,data.movie19,data.movie20,data.movie21,
data.movie22,data.movie23,data.movie24)
store['bit4'] = createNum(data.movie25,
     data.movie26,data.movie27,data.movie28,data.movie29,
data.movie30,data.movie31,data.movie32)
```

Столбцы преобразуются в 4 битовые строки в числовой форме. Каждая битовая строка представляет 8 фильмов, 4*8=32 фильма. Примечание: для сокращения длины кода можно использовать 32-разрядную строку вместо 4*8.

Тестирование правильности работы функции. Двоичный код 1111 эквивалентен 15 (=1*8+1*4+1*2+1*1). Если условие assert ложно, инициируется ошибка; в противном случае не происходит ничего.

Создание хеш-функции

Хеш-функции, которые мы создадим, получают флаги просмотра фильмов клиентом. В теоретической части примера мы решили создать 3 хеш-функции: первая функция объединяет фильмы 10, 5 и 18; вторая — фильмы 7, 18 и 22; и третья — фильмы 16, 19 и 30. Если вам захочется выбрать другие фильмы, это абсолютно нормально; их вообще можно выбрать случайно. Листинг 4.10 показывает, как это делается.

Хеш-функция объединяет флаги разных фильмов в двоичное значение (по аналогии с тем, как это делалось ранее в функции `createNum()`), но на этот раз данные не преобразуются в числа и вместо 8 фильмов на входе всего 3. Функция `assert` показывает, как происходит конкатенация трех значений для каждого наблюдения. Если клиент брал фильм 10, но не фильмы 15 и 28, для гнезда 1 возвращается значение `b'100'`. Если клиент брал фильмы 7 и 18, но не 22, для гнезда 2 возвращается

Листинг 4.10. Создание хеш-функций

```
def hash_fn(x1,x2,x3):
    return [b'%d%d%d' % (i,j,k) for (i,j,k) in zip(x1,x2,x3)]

assert hash_fn([1,0],[1,1],[0,0]) == [b'110',b'010']//

store['bucket1'] = hash_fn(data.movie10, data.movie15,data.movie28)
store['bucket2'] = hash_fn(data.movie7, data.movie18,data.movie22)
store['bucket3'] = hash_fn(data.movie16, data.movie19,data.movie30)
store.to_sql('movie_comparison',mc, flavor = 'mysql', index = True,
          index_label = 'cust_id', if_exists = 'replace')
```

Определение хеш-функции (она очень похожа на функцию createNum(), только без завершающего преобразования в число и с 3 столбцами вместо 8).

Тестирование функции (если обходится без ошибок, значит, функция работает). Используется лишь часть столбцов, но выбираются все наблюдения.

Сохранение информации в базе данных.

Хеширование информации о фильмах, просмотренных клиентом, — соответственно, [10,15, 28], [7,18, 22] и [16,19, 30].

значение `b'110'`. Просматривая текущий результат, вы видите 4 переменные, созданные ранее (`bit1`, `bit2`, `bit3`, `bit4`) из 9 отобранных фильмов (рис. 4.12).

	bit1	bit2	bit3	bit4	bucket1	bucket2	bucket3
0	10	62	42	182	011	100	011
1	23	28	223	180	001	111	001

Рис. 4.12. Информация из сжатых битовых строк и 9 фильмов

Последний прием, который мы применим, — индексирование таблицы клиентов для ускорения поиска.

Добавление индекса в таблицу

Теперь необходимо добавить индексы для ускорения выборки, как это следует делать в системах реального времени. Листинг 4.11 показывает, как это делается.

Листинг 4.11. Создание индекса

```
def createIndex(column, cursor):
    sql = 'CREATE INDEX %s ON movie_comparison (%s);' % (column, column)
    cursor.execute(sql)

createIndex('bucket1',cursor)
createIndex('bucket2',cursor)
createIndex('bucket3',cursor)
```

Определение функции упрощает создание индексов. Индексы ускоряют выборку данных.

Создание индексов по гнездам.

После того как данные будут проиндексированы, можно переходить к шагу «построение модели». В данном примере никакое машинное обучение или статистическая модель не реализуются. Вместо этого используется более простой прием: вычисление расстояния между строками. Для сравнения двух строк можно воспользоваться расстоянием Хэмминга так, как объяснялось ранее в теоретической вводной части этого примера.

4.5.4. Этап 5: Построение модели

Чтобы использовать метрику расстояния Хэмминга с базой данных, необходимо определить ее вычисление в функции.

Создание функции вычисления расстояния Хэмминга

Вычисление расстояние Хэмминга будет реализовано в виде *пользовательской функции*. Эта функция, вычисляющая расстояние для 32-разрядного целого числа (на самом деле 4*8), приведена в листинге 4.12.

Листинг 4.12. Создание функции вычисления расстояния Хэмминга

```
Sql = '''
CREATE FUNCTION HAMMINGDISTANCE(
  A0 BIGINT, A1 BIGINT, A2 BIGINT, A3 BIGINT,
  B0 BIGINT, B1 BIGINT, B2 BIGINT, B3 BIGINT
)

RETURNS INT DETERMINISTIC
RETURN
  BIT_COUNT(A0 ^ B0) +
  BIT_COUNT(A1 ^ B1) +
  BIT_COUNT(A2 ^ B2) +
  BIT_COUNT(A3 ^ B3); '''

cursor.execute(Sql)

Sql = '''Select hammingdistance(
    b'11111111',b'00000000',b'11011111',b'11111111'
,b'11111111',b'10001001',b'11011111',b'11111111'
)''' //
pd.read_sql(Sql,mc)
```

Функция сохраняется в базе данных. Этот код может быть выполнен только один раз; при попытке повторного выполнения выдается ошибка: OperationalError: (1304, 'FUNCTION HAMMINGDISTANCE already exists').

Определение функции. Функция получает 8 входных аргументов: 4 строки длины 8 для первого клиента и еще 4 строки длины 8 для второго клиента. Это позволяет провести параллельное сравнение 2 клиентов по 32 фильмам.

Чтобы проверить эту функцию, необходимо выполнить эту команду SQL с 8 фиксированными строками. Обратите внимание на префикс "b" перед каждой строкой — это признак двоичных значений. Этот конкретный тест должен вернуть 3; результат означает, что строки различаются только в 3 местах.

Выполнение запроса.

Если все прошло нормально, программа должна вывести 3.

Теперь, когда функция вычисления расстояния Хэмминга готова к работе, мы можем воспользоваться ею для поиска клиентов с такими же предпочтениями, как у заданного клиента, что, собственно, и требовалось изначально. А теперь перейдем к последней части: оформлению результатов в виде приложения.

4.5.5. Этап 6: Отображение и автоматизация

После всего, что было сделано в предыдущем разделе, наше приложение должно выполнять для любого конкретного клиента две операции:

- Находить клиентов с похожими предпочтениями.
- Рекомендовать фильмы, которые еще не видел пользователь, на основании списка просмотренных им фильмов и истории просмотра похожих клиентов.

Впрочем, обо всем по порядку.

Поиск похожего клиента

Пора заняться выполнением запросов в реальном времени. В листинге 4.13 клиент 27 оказывается тем счастливчиком, для которого будет составляться рекомендация фильмов. Но сначала нужно выбрать клиентов с похожей историей просмотра.

Листинг 4.13. Поиск клиентов с похожей историей просмотра

Мы проведем двухфазный отбор. В первой фазе выбираются клиенты, индекс которых точно совпадает с индексом выбранного клиента (проверка основана на 9 фильмах.) Выбираемые клиенты видели (или не видели) те же 9 фильмов, что и наш клиент. Второй отбор представляет собой ранжирование, основанное на 4-битовых строках. В нем учитываются все фильмы в базе данных.

Выбор клиента из базы данных.

```
customer_id = 27
sql = "select * from movie_comparison where cust_id = %s" % customer_id
cust_data = pd.read_sql(sql,mc)
sql =  """ select cust_id,hammingdistance(bit1,
bit2,bit3,bit4,%s,%s,%s,%s) as distance
          from movie_comparison where bucket1 = '%s' or bucket2 ='%s'
or bucket3='%s' order by distance limit 3""" %
     (cust_data.bit1[0],cust_data.bit2[0],
cust_data.bit3[0], cust_data.bit4[0],
      cust_data.bucket1[0], cust_data.bucket2[0],cust_data.bucket3[0])
shortlist = pd.read_sql(sql,mc)
```

Вывод 3 клиентов, самых близких к клиенту 27. В начале списка находится сам клиент 27.

Из табл. 4.5 следует, что самыми близкими к клиенту 27 оказались клиенты 2 и 97. Не забывайте, что данные были сгенерированы случайным образом, поэтому при попытке повторить этот пример вы можете получить другие результаты.

И теперь мы наконец-то можем выбрать фильм для клиента 27.

Таблица 4.5. Клиенты, наиболее похожие на клиента 27

	cust_id	расстояние
0	27	0
1	2	8
2	97	9

Подбор нового фильма

Нужно найти фильмы, которые клиент 27 еще не видел, но видел ближайший к нему клиент. В листинге 4.14 показано, как это делается. Кстати, заодно это хорошая проверка правильности функции вычисления расстояния. Возможно, это и не самый близкий клиент, но результат достаточно близок к вкусам клиента 27. Применение хешированных индексов обеспечивает огромный выигрыш в скорости при обработке запросов к большим базам данных.

Листинг 4.14. Поиск фильма для рекомендации

```
cust = pd.read_sql('select * from cust where cust_id in (27,2,97)',mc)
dif = cust.T
dif[dif[0] != dif[1]]
```

Выбор фильмов, которые видели клиенты 27, 2 и 97.

Транспонирование для удобства.

Выбор фильмов, которые клиент 27 еще не видел.

Из табл. 4.6 видно, что на основании поведения клиента 2 можно порекомендовать фильмы 12, 15 и 31.

Таблица 4.6. Фильмы клиента 2 могут использоваться в рекомендациях для клиента 27

	0	1	2
Cust_id	2	27	97
Movie3	0	1	1
Movie9	0	1	1
Movie11	0	1	1
Movie12	1	0	0
Movie15	1	0	0
Movie16	0	1	1
Movie25	0	1	1
Movie31	1	0	0

Готово - наш счастливый киноман может порадоваться новому фильму, подобранному с учетом его предпочтений.

В следующей главе мы рассмотрим данные еще большего размера. Вы увидите, как работать с ними в среде Horton Sandbox, загруженной в главе 1.

4.6. Итоги

Основные положения этой главы:

- Главные *проблемы*, возникающие при работе с большими наборами данных:
  - Нехватка памяти.
  - Слишком долгое выполнение программ.
  - Ресурсы, образующие «узкие места» и вызывающие проблемы со скоростью.
- *Решения* этих проблем делятся на три основных типа:
  - Адаптация алгоритмов.
  - Выбор других структур данных.
  - Использование инструментов и библиотек.
- Три основных приема, применяемые при *адаптации алгоритмов*:
  - Данные алгоритма представляются *по одному наблюдению* за раз (вместо одновременной загрузки в память всего набора данных).
  - *Разбиение матриц на матрицы меньшего размера*, которые используются при вычислениях.
  - Реализация алгоритма *отображения-свертки* (с использованием таких библиотек Python, как Hadoopy, Octopy, Disco или Dumbo).
- В области data science используются три основные структуры данных. Первая — *разреженная матрица* — представляет собой разновидность матрицы с относительно малым объемом полезной информации. Вторая и третья структуры обеспечивают быструю выборку информации в большом наборе данных: это хеш-*таблица* и *древовидная структура*.
- Python содержит многочисленные *инструменты* для работы с большими наборами данных. Одни инструменты помогают справиться с размером данных, другие обеспечивают параллельную обработку, а третьи решают проблему с относительно медленным выполнением кода Python. Также Python удобно использовать для управления другими инструментами data science, так как Python часто выбирается для реализации API.
- Многие *принципы* из теории обработки данных остаются актуальными в контексте data science, так что их применение поможет вам справиться с проблемами, с которыми вы сталкиваетесь в контексте больших данных.

Первые шаги в области больших данных

В этой главе:

- ✓ Две системы для работы с «большими данными»: Hadoop и Spark.
- ✓ Написание заданий для обработки «больших данных» на языке Python.
- ✓ Построение информационной панели, связанной с данными, хранящимися в базе данных.

В последних двух главах размер данных постепенно увеличивался. В главе 3 рабочие данные помещались в памяти компьютера. В главе 4 были представлены средства для работы с наборами данных, которые не помещались в памяти одновременно, но все еще могли быть обработаны на одном компьютере. В этой главе вы научитесь работать с технологиями, предназначенными для данных настолько больших, что они не помещаются на одном узле (компьютере). Собственно, эти данные могут не поместиться даже на сотне компьютеров. Задача усложняется, не так ли?

Мы постараемся придерживаться того же подхода, который использовался в предыдущих главах: читатель должен почувствовать достаточную уверенность для работы на платформе больших данных. По этой причине основная часть этой главы посвящена рассмотрению учебных примеров. Мы создадим информационную панель для исследования данных. К концу главы мы отработаем следующие операции:

- ❑ Загрузка данных в Hadoop, самую распространенную платформу «больших данных».
- ❑ Преобразование и очистка данных в Spark.
- ❑ Сохранение данных в базе данных Hive, предназначенной для больших данных.
- ❑ Интерактивная визуализация данных с использованием программы Qlik Sense.

Управление всеми этими задачами (кроме визуализации) будет осуществляться из сценария Python. Конечным результатом всех этих усилий станет информационная панель для исследования данных (рис. 5.1).

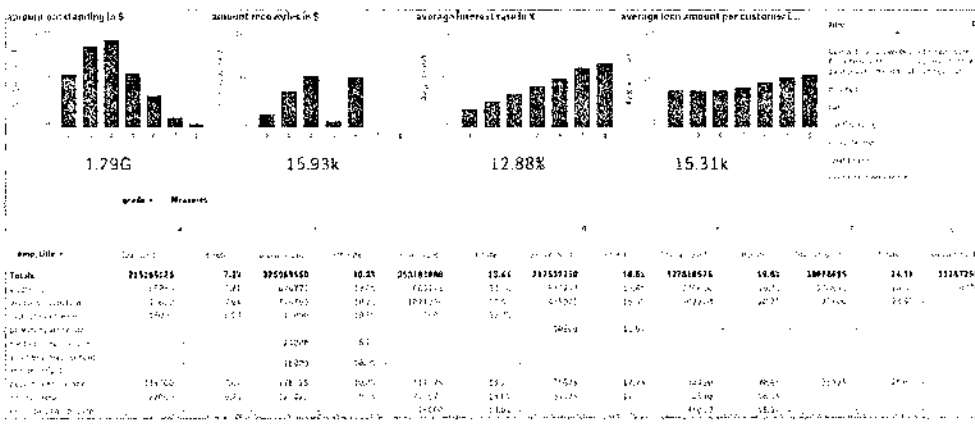


Рис. 5.1. Информационная панель Qlik

Помните, что эта вступительная глава дает в лучшем случае поверхностные знания о технологиях больших данных. В учебном примере затронуты три технологии больших данных (Hadoop, Spark и Hive), но только в отношении обработки данных, а не построения модели. Вам придется самостоятельно объединить технологии больших данных, которые будут представлены здесь, с методами построения модели, упоминавшимися в предыдущих главах.

5.1. Распределение хранения и обработки данных в инфраструктурах

Новые технологии больших данных, такие как Hadoop и Spark, упрощают работу с компьютерными кластерами и управление ими. Hadoop может масштабировать тысячи компьютеров, создавая кластеры с петабайтами пространства хранения. Это позволяет бизнесу извлечь пользу из гигантских объемов доступных данных.

5.1.1. Hadoop: инфраструктура для хранения и обработки больших объемов данных

Apache Hadoop -- инфраструктура, упрощающая работу с компьютерными кластерами. Hadoop пытается достичь следующих целей (и не только):

- *Надежность* -- достигается посредством создания нескольких копий данных и повторного применения логики обработки в случае сбоя.

- *Отказоустойчивость* — обнаружение сбоев и применение автоматического восстановления.
- *Масштабируемость* — данные и их обработка распределяются в компьютерных кластерах (горизонтальное масштабирование).
- *Портируемость* — возможность установки на всех видах устройств и операционных систем.

Базовая инфраструктура состоит из распределенной файловой системы, менеджера ресурсов и системы выполнения распределенных программ. На практике она позволяет работать с распределенной файловой системой почти так же просто, как с локальной файловой системой вашего домашнего компьютера. Однако на заднем плане эти данные могут быть разбросаны по тысячам серверов.

Основные компоненты Hadoop

В Hadoop центральное место занимают:

- Распределенная файловая система (HDFS).
- Метод крупномасштабного выполнения программ (MapReduce).
- Система управления ресурсами кластеров (YARN).

На этой базе сформировалась экосистема приложений (рис. 5.2), например базы данных Hive и HBase, и инфраструктуры машинного обучения (такие, как Mahout). В этой главе будет использоваться Hive. Для работы с информацией, хранящейся в базе данных, Hive использует язык, основанный на популярном языке SQL.

Популярный инструмент Impala может использоваться для ускорения обработки запросов Hive до 100 раз. В этой книге Impala не рассматривается, но дополни-

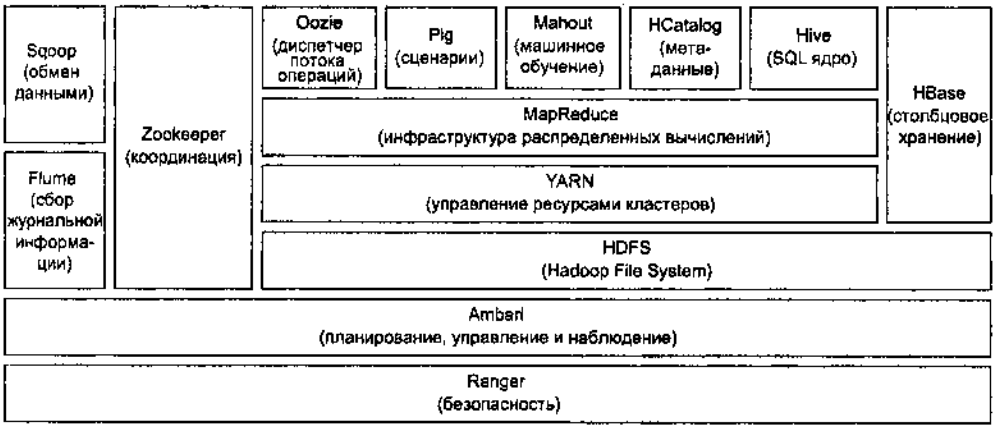


Рис. 5.2. Часть экосистемы приложений, сформировавшейся на основе базовой инфраструктуры Hadoop

тельную информацию можно найти по адресу *http://impala.io/*. Краткое введение в MapReduce было приведено в главе 4, но сейчас об этой технологии стоит рассказать немного подробнее, потому что она играет исключительно важную роль в Hadoop.

MapReduce: как в Hadoop реализуется параллелизм

Для реализации параллелизма Hadoop использует технологию программирования, называемую *MapReduce*. Алгоритм MapReduce выполняет разбивку данных, обрабатывает их параллельно, а затем сортирует, объединяет и обобщает результаты. Однако алгоритм MapReduce плохо подходит для интерактивного анализа или итеративных программ, потому что данные записываются на диск между этапами вычислений. При работе с большими наборами данных запись обходится достаточно дорого.

Рассмотрим работу MapReduce на небольшом вымышленном примере. Вы — директор компании, занимающейся производством игрушек. Каждая игрушка окрашена в два цвета; когда клиент заказывает игрушку с веб-страницы, файл заказа размещается в Hadoop с указанием цветов. Ваша задача — определить, сколько емкостей с краской нужно подготовить. Для подсчета цветов будет использоваться алгоритм в стиле MapReduce. Начнем с упрощенной версии на рис. 5.3.

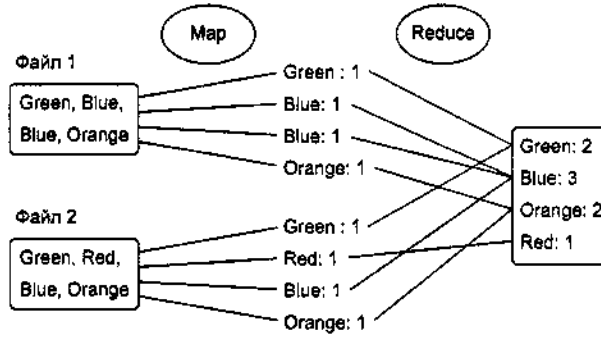


Рис. 5.3. Упрощенный пример организации в MapReduce подсчета цветов в текстовых данных

Как следует из названия, процесс делится приблизительно на две крупные фазы:

- *Фаза отображения* (Map) — документы делятся на пары «ключ—значение». До выполнения свертки в данных будет много дубликатов.
- *Фаза свертки* (Reduce) — в целом аналогична конструкции SQL GROUP BY. Разные уникальные экземпляры собираются воедино, и в зависимости от функции свертки могут быть получены разные результаты. В данном случае нужно получить количество вхождений каждого цвета, поэтому именно оно возвращается функцией свертки.

Впрочем, на практике все немного сложнее.

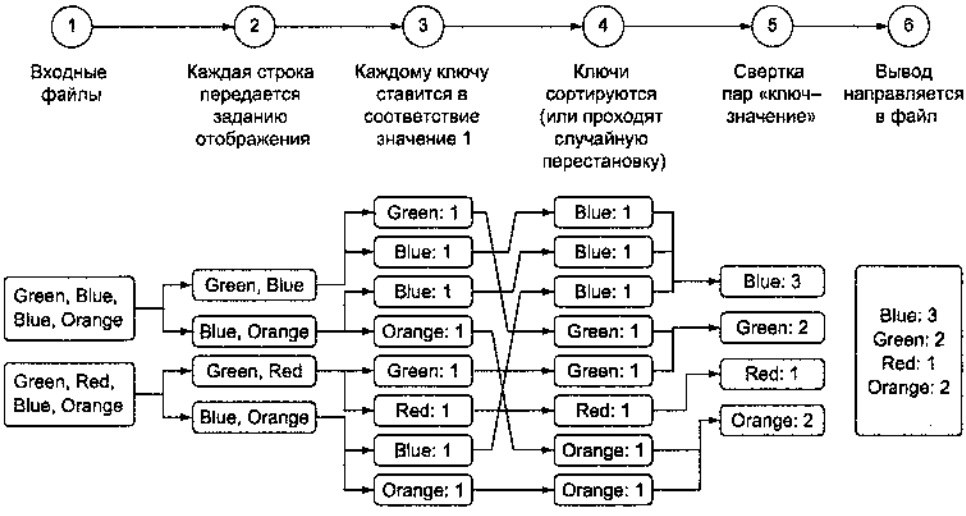


Рис. 5.4. Пример организации в MapReduce подсчета цветов в текстовых данных

Весь процесс описан в следующих шести шагах и изображен на рис. 5.4.

1. Чтение входных файлов.
2. Каждая строка передается заданию отображения (mapper job).
3. Задание читает из файла цвета (ключи) и выводит для каждого цвета файл с количеством его вхождений (значение). Или, в терминологии программирования, ключ (цвет) *отображается* на значение (количество вхождений).
4. Ключи сортируются, чтобы упростить обобщение данных.
5. Фаза свертки суммирует количество вхождений каждого цвета и выводит для каждого ключа один файл с общим количеством вхождений этого цвета.
6. Ключи собираются в выходной файл.

ПРИМЕЧАНИЕ

Хотя Hadoop упрощает работу с большими данными, создание хорошего рабочего кластера остается делом нетривиальным. Впрочем, менеджеры кластеров, такие как Apache Mesos, упрощают задачу. На практике многим компаниям (среднего размера) не хватает квалификации для управления жизнеспособной установкой Hadoop. По этой причине мы будем работать в Hortonworks Sandbox — заранее установленной и настроенной экосистеме Hadoop. Инструкции по установке приведены в разделе 1.5: «Вводный пример использования Hadoop».

Итак, вы получили некоторое представление о том, как работает Hadoop. Теперь обратимся к Spark.

5.1.2. Spark: замена MapReduce с повышенной производительностью

Специалисты data science часто проводят интерактивный анализ и используют алгоритмы, интерактивные по своей природе. Чтобы алгоритм пришел к решению, ему может потребоваться некоторое время; это слабое место инфраструктуры MapReduce, и для его преодоления была создана инфраструктура Spark. Spark может повысить производительность таких задач на порядок.

Что такое Spark?

Spark — инфраструктура кластерных вычислений, сходная с MapReduce. Однако Spark не занимается ни хранением файлов в (распределенной) файловой системе, ни управлением ресурсами. Для этого Spark пользуется услугами таких систем, как Hadoop File System, YARN или Apache Mesos. Таким образом, Hadoop и Spark являются взаимодополняющими системами. В процессе тестирования и разработки вы даже можете запускать Spark в своей локальной системе.

Как Spark решает проблемы MapReduce?

Spark создает своего рода общую оперативную память для компьютеров вашего кластера (хотя мы немного упрощаем ситуацию для наглядности.) Это позволяет разным рабочим процессам использовать общие переменные (и их состояние), а следовательно, снимает необходимость в записи промежуточных результатов на диск. Если же вы предпочитаете более научное и формальное объяснение, Spark использует RDD (Resilient Distributed Datasets) — распределенную абстракцию памяти, которая позволяет программистам выполнять вычисления в памяти в больших кластерах с защитой от ошибок[1].

Так как система работает в памяти, она избегает затратных дисковых операций.

Компоненты экосистемы Spark

Ядро Spark предоставляет среду NoSQL, хорошо подходящую для интерактивного, исследовательского анализа. Spark может работать как в пакетном, так и в интерактивном режиме и поддерживает Python.

Spark содержит еще четыре крупных компонента, представленные на рис. 5.5.

1. Spark Streaming — инструмент анализа в реальном времени.
2. Spark SQL предоставляет интерфейс SQL для работы с Spark.
3. MLLib — инструмент машинного обучения в инфраструктуре Spark.

[1] См. https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf.

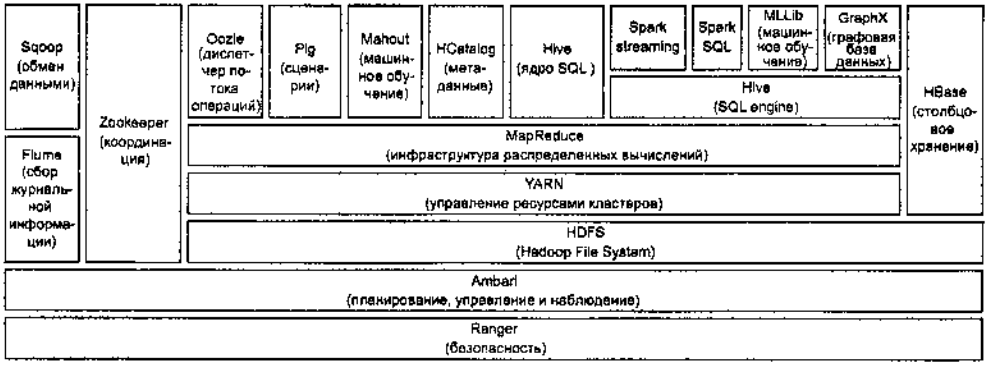


Рис. 5.5. Инфраструктура Spark, используемая в сочетании с инфраструктурой Hadoop

4. GraphX — графовая база данных для Spark. Графовые базы данных более подробно рассматриваются в главе 7.

А теперь рассмотрим практический пример использования Hadoop, Hive и Spark.

5.2. Учебный пример: Оценка риска при кредитовании

Вооружившись общим пониманием Hadoop и Spark, можно взяться непосредственно за большие данные. Цель этого учебного примера — получить первый опыт использования технологий, представленных ранее в этой главе, и убедиться в том, что в значительной мере с ними можно (хотя и не обязательно) работать так же, как и с другими технологиями. Примечание: здесь используется лишь небольшая часть данных, потому что для сбора всей информации потребуется канал с серьезной пропускной способностью, а для повторения примера необходимо несколько узлов.

Что мы используем:

- Horton Sandbox на виртуальной машине. Если вы еще не загрузили и не импортировали Horton Sandbox в виртуальную машину (например, VirtualBox), вернитесь к разделу 1.5 — там рассказано, как это делается. При написании этой главы была использована версия Horton Sandbox 2.3.2.
- Библиотеки Python: Pandas и pywebhdsf. Устанавливать их в локальной виртуальной среде не нужно; они потребуются непосредственно в Horton Sandbox. Следовательно, необходимо запустить Horton Sandbox (например, в VirtualBox) и выполнить ряд подготовительных действий.

Чтобы все это заработало, вам придется кое-что сделать в командной строке Sandbox, поэтому войдите в режим командной строки. Для этого можно вос-

пользоваться программой PuTTY. Если вы еще не знакомы с PuTTY — программа предоставляет интерфейс командной строки к серверам, и ее можно бесплатно загрузить по адресу *http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html*.

Конфигурация подключения к Horton Sandbox в PuTTY показана на рис. 5.6.

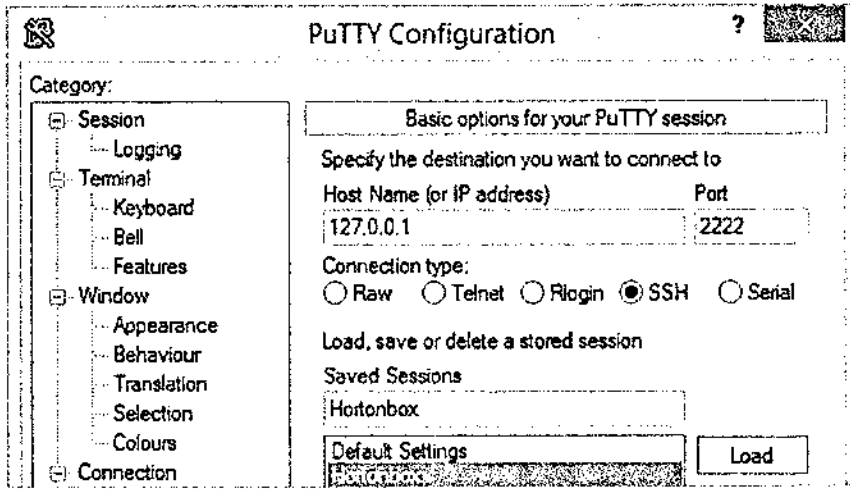


Рис. 5.6. Подключение к Horton Sandbox в PuTTY

Имя пользователя и пароль по умолчанию (на момент написания книги) – «root» и «hadoop» соответственно. Впрочем, пароль следует изменить при первом входе.

После подключения введите следующие команды:

- `yum -y install python-pip` – устанавливает pip, менеджер пакетов Python.
- `pip install git+https://github.com/DavyCielen/pywebhdfs.git –upgrade` — на момент написания книги в библиотеке pywebhdfs существовала проблема, которая исправляется этой командой. Надеемся, что к тому моменту, когда вы будете читать книгу, эта команда уже не будет обязательной; программисты, занимающиеся сопровождением пакета, знают о проблеме и исправят ее.
- `pip install pandas` — для установки Pandas. Обычно установка занимает некоторое время из-за зависимостей.

Имеется файл .ipynb, который вы можете открыть в Jupyter или (более старой) Ipython для повторения кода этой главы. В нем продублированы подготовительные инструкции Horton Sandbox; проследите за тем, чтобы код выполнялся непосредственно в Horton Sandbox. А теперь, разобравшись с предварительной подготовкой, посмотрим, что же нам предстоит сделать.

В этом упражнении мы пройдем несколько шагов процесса data science:

- Этап 1: Цель исследования. Состоит из двух частей:
  - Реализация информационной панели, которую от вас требует директор.
  - Подготовка данных, на основе которых другие люди смогут создать свои интерактивные панели.
- Этап 2: Сбор данных:
  - Загрузка данных с сайта финансовой организации.
  - Размещение данных в системе Hadoop File System в Horton Sandbox.
- Этап 3: Подготовка данных:
  - Преобразование данных средствами Spark.
  - Сохранение подготовленных данных в Hive.

Этапы 4 и 6: Исследование и создание отчета:

- Визуализация данных средствами Qlik Sense.

Этап построения модели в этом примере отсутствует, но вся необходимая инфраструктура у вас имеется, — проведите его самостоятельно, если хотите. Например, вы можете воспользоваться средствами машинного обучения Spark, чтобы спрогнозировать потенциальные банкротства.

5.2.1. Этап 1: Цель исследования

Lending Club — организация, которая сводит людей, нуждающихся в кредите, с потенциальными инвесторами. У вашего директора есть деньги для инвестиций, но он хочет получить информацию, прежде чем рисковать значительной суммой. Для этого вы создадите отчет с информацией о среднем рейтинге, риске и прибыли при выдаче кредита определенному клиенту. Для доступа к данным будет создана специальная информационная панель, что позволит работать с ними другим людям. В некотором роде это можно считать вторичной целью исследования: предоставление доступа к данным для самостоятельной бизнес-аналитики. Самостоятельная бизнес-аналитика часто применяется в организациях, работающих с данными, но не располагающих лишним штатом аналитиков. Любой сотрудник может провести несложную обработку данных, оставив более сложную аналитику для специалистов data science.

Наша работа возможна благодаря тому, что Lending Club публикует анонимные данные о существующих кредитах. К концу этого примера вы создадите отчет, напоминающий рис. 5.7.

Впрочем, обо всем по порядку; начнем с получения данных.

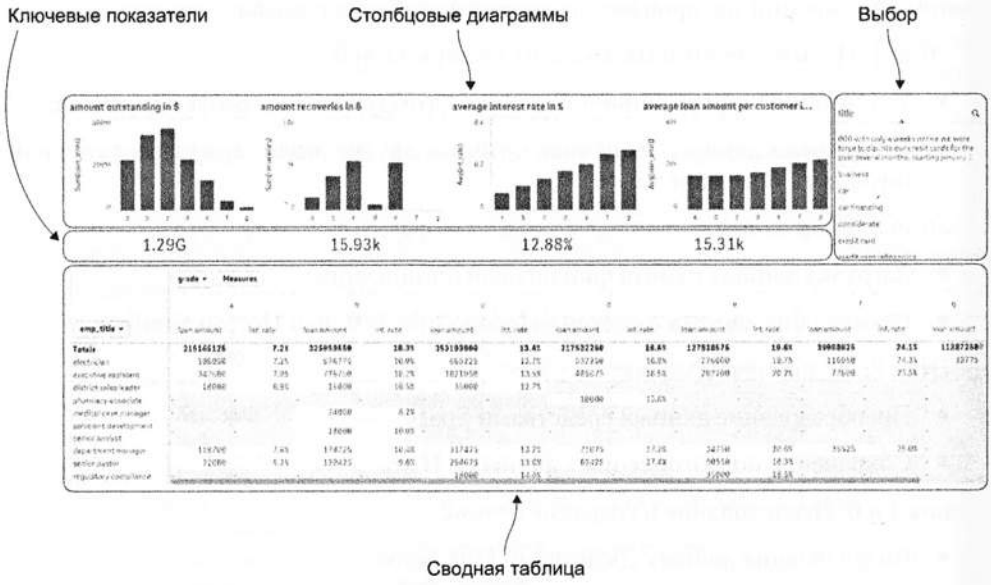


Рис. 5.7. Конечным результатом данного упражнения станет наглядная информационная панель для сравнения возможностей предоставления займов

5.2.2. Этап 2: Сбор данных

Пришло время поработать с системой Hadoop File System (или hdfs). Сначала команды будут передаваться из командной строки, а затем — з языка сценариев Python с помощью пакета pywebhdfs.

Файловая система Hadoop похожа на обычную файловую систему, если не считать того, что файлы и папки хранятся на разных серверах и физический адрес каждого файла вам неизвестен. Ситуация покажется знакомой читателям, работавшим с Dropbox или Google Drive. Файлы, которые вы сохраняете в этих системах, сохраняются где-то на сервере, причем вы точно не знаете на каком. Как и в обычной файловой системе, файлы и папки можно создавать, переименовывать и удалять.

Взаимодействие с HDFS из командной строки

Для начала получим текущий список каталогов и файлов в корневой папке Hadoop из командной строки. Введите в PuTTY команду `hadoop fs –ls /`.

Прежде чем пытаться создавать подключение, убедитесь в том, что вы включили свою виртуальную машину в Hortonworks Sandbox. Затем в PuTTY следует создать подключение по адресу 127.0.0.1:2222, как показано на рис. 5.6.

Выходные данные команды Hadoop представлены на рис. 5.8. Вы также можете добавить аргументы (например, hadoop fs –ls –R /) для получения рекурсивного списка всех файлов и подкаталогов.

```
[root@sandbox ~]# hadoop fs -ls /
Found 20 items
drwxrwxrwx   - admin  hadoop          0 2015-07-14 14:54 /LoanStats3c.cs
-rw-r--r--   1 root   hadoop  120834552 2015-07-14 14:47 /LoanStats3c.csv
drwxrwxrwx   - yarn   hadoop          0 2015-07-15 13:32 /app-logs
drwxr-xr-x   - hdfs   hdfs            0 2015-06-05 09:19 /apps
drwxr-xr-x   - admin  hadoop          0 2015-07-13 06:47 /book
drwxr-xr-x   - root   hadoop          0 2015-07-17 10:24 /chapter5
-rwxr-xr-x   1 hdfs   hadoop       4240 2015-07-14 19:32 /cout.json
```

Рис. 5.8. Вывод содержимого каталога Hadoop командой hadoop fs –ls /. Команда выводит содержимое корневой папки Hadoop

Создадим новый каталог с именем «chapter5» для этой главы. Следующие команды создают новый каталог и открывают полный доступ к нему:

```
sudo -u hdfs hadoop fs -mkdir /chapter5
sudo -u hdfs hadoop fs -chmod 777 /chapter5
```

Вероятно, вы уже заметили закономерность. Команды Hadoop очень похожи на команды локальной файловой системы (в стиле POSIX), но они начинаются с hadoop fs и перед каждой командой ставится дефис (-). В табл. 5.1 приведена сводка основных команд файловой системы Hadoop и их аналогов для локальной файловой системы.

Таблица 5.1. Список основных команд файловой системы Hadoop

Операция	Команда файловой системы Hadoop	Команда локальной файловой системы
Получение содержимого каталога (списка файлов и каталогов)	hadoop fs ls URI	ls UR
Создание каталога	hadoop fs –mkdir URI	mkdir URI
Удаление каталога	hadoop fs –rm r URI	rm –r URI
Изменение разрешений доступа к файлам	hadoop fs –chmod MODE URI	chmod MODE URI
Перемещение или переименование файла	hadoop fs –mv OLDURI NEWURI	mv OLDURI NEWURI

Есть еще две специальные команды, которые вы будете часто использовать:

- Отправка файла из локальной файловой системы в распределенную файловую систему (hadoop fs –put LOCALURI REMOTEURI).

- Загрузка файла из распределенной файловой системы в локальную файловую систему (hadoop –get REMOTEURI).

Поясним сказанное примером. Допустим, у вас имеется файл .CSV на виртуальной машине Linux, с которой вы подключаетесь к Linux-кластеру Hadoop. Чтобы скопировать файл .CSV с виртуальной машины Linux в кластер HDFS, используйте команду hadoop –put mycsv.csv /data.

При помощи PuTTY можно запустить сеанс Python в Horton Sandbox для получения данных с использованием сценария Python. Чтобы запустить сеанс, введите в командной строке команду pyspark. Если все было сделано правильно, появляется заставка, изображенная на рис. 5.9.

Рис. 5.9. Заставка Spark для интерактивного использования из Python

Теперь можно выполнить код Python, который получит необходимые данные (листинг 5.1.)

Листинг 5.1. Загрузка данных Lending Club

```
import requests
import zipfile
import StringIO
source = requests.get("https://resources.lendingclub.com/
    LoanStats3d.csv.zip", verify=False)
stringio = StringIO.StringIO(source.content)
unzipped = zipfile.ZipFile(stringio)
```

Загрузка данных. С префиксом https следует включить проверку, но мы отвлекаться не будем (verify=False).

Создание виртуального файла.

Распаковка данных.

Мы загружаем файл LoanStats3d.csv.zip с сайта Lending Club по адресу *https://resources.lendingclub.com/LoanStats3d.csv.zip* и распаковываем его. Методы из пакетов Python requests, zipfile и stringio используются для загрузки данных, создания виртуального файла и его распаковки соответственно. Здесь обрабатывается всего один файл; если вам нужны все данные, можно создать цикл, но для демонстрационных целей хватит и этого. Как упоминалось ранее, важной частью этого примера будет подготовка данных для технологий больших данных. Но прежде чем переходить к ней, необходимо поместить данные в файловую систему Hadoop. Пакет PyWebHdfs позволяет взаимодействовать с файловой системой Hadoop из Python;

он преобразует и передает команды вызовам интерфейса webhdfs. Это позволяет вам применить ваш любимый язык сценариев для автоматизации различных задач, как показано в листинге 5.2.

Листинг 5.2. Сохранение данных в Hadoop

Предварительная очистка данных с использованием Pandas: удаляется верхняя строка и две нижние, потому что они бесполезны (чтобы убедиться в этом, достаточно открыть исходный файл).

Локальное сохранение для пересылки в файловую систему Hadoop.

```
import pandas as pd
from pywebhdfs.webhdfs import PyWebHdfsClient
subselection_csv = pd.read_csv(unzipped.open('LoanStats3d.csv'),
skiprows=1,skipfooter=2,engine='python')
stored_csv = subselection_csv.to_csv('./stored_csv.csv')
hdfs = PyWebHdfsClient(user_name="hdfs",port=50070,host="sandbox")
hdfs.make_dir('chapter5')
with open('./stored_csv.csv') as file_data:
        hdfs.create_file('chapter5/LoanStats3d.csv',file_data,
      overwrite=True)
```

Открытие локально хранимых данных CSV.

Создание файла .csv в файловой системе Hadoop.

Создание папки "chapter5" в файловой системе Hadoop.

Соединение с Hadoop Sandbox.

Файл уже был загружен и распакован в листинге 5.1; теперь в листинге 5.2 мы создаем выборку данных средствами Pandas и сохраняем ее локально. Затем программа создает каталог в Hadoop и перемещает локальный файл в Hadoop. Загруженные данные хранятся в формате .CSV, а поскольку они относительно малы, мы можем воспользоваться библиотекой Pandas для удаления первой строки и последних двух строк из файла. Эти строки содержат комментарии и только создают неудобства при работе с файлом в среде Hadoop. Первая строка кода импортирует пакет Pandas, а вторая разбирает файл в памяти и удаляет первую и две последние строки. Третья строка сохраняет данные в локальной файловой системе для последующего использования и простоты просмотра.

Прежде чем двигаться дальше, проверьте файл следующей строкой кода:

```
print hdfs.get_file_dir_status('chapter5/LoanStats3d.csv')
```

На консоли PySpark должно появиться сообщение о том, что файл благополучно находится в системе Hadoop, как показано на рис. 5.10.

Итак, файл находится в Hadoop и готов к работе. Теперь можно переходить к подготовке данных с использованием Spark, потому что данные недостаточно хорошо очищены для сохранения непосредственно в Hive.

```
>>> print hdfs.get_file_dir_status('chapter5/LoanStats3d.csv')#A
{u'FileStatus': {u'group': u'hdfs', u'permission': u'755', u'blockSize': 1342177
28, u'accessTime': 1449236321223, u'pathSuffix': u'', u'modificationTime': 14492
36321965, u'replication': 3, u'length': 120997124, u'childrenNum': 0, u'owner':
u'hdfs', u'storagePolicy': 0, u'type': u'FILE', u'fileId': 17520}}
```

Рис. 5.10. Получение информации о файле в Hadoop через консоль PySpark

5.2.3. Этап 3: Подготовка данных

После того как данные будут загружены для анализа, мы воспользуемся Spark и очистим данные перед тем, как сохранить их в Hive.

Подготовка данных в Spark

Очистка данных часто проходит в интерактивном режиме: пользователь находит проблему и исправляет ее, и, скорее всего, для получения чистых и безупречных данных это придется проделать не раз и не два. Скажем, к «грязным» данным будет относиться строка «UsA» с неверным регистром символа. На этой стадии мы уже не работаем в jobs.py, а используем интерфейс командной строки PySpark для непосредственного взаимодействия с Spark.

Spark хорошо подходит для подобного интерактивного анализа, потому что данные не нужно сохранять после каждой корректировки, а модель Spark обмена данными между серверами (своего рода распределенная память) намного лучше модели Hadoop.

Преобразование состоит из четырех этапов:

1. Запустите PySpark (программа должна быть еще открытой после раздела 5.2.2) и загрузите контекст Spark и Hive.
2. Прочитайте и разберите файл .CSV.
3. Отделите строку заголовка от данных.
4. Проведите очистку данных.

В листинге 5.3 показана реализация кода в консоли PySpark.

Листинг 5.3. Подключение к Apache Spark

Импортирование контекста Spark --> не обязательно при работе непосредственно в Spark.

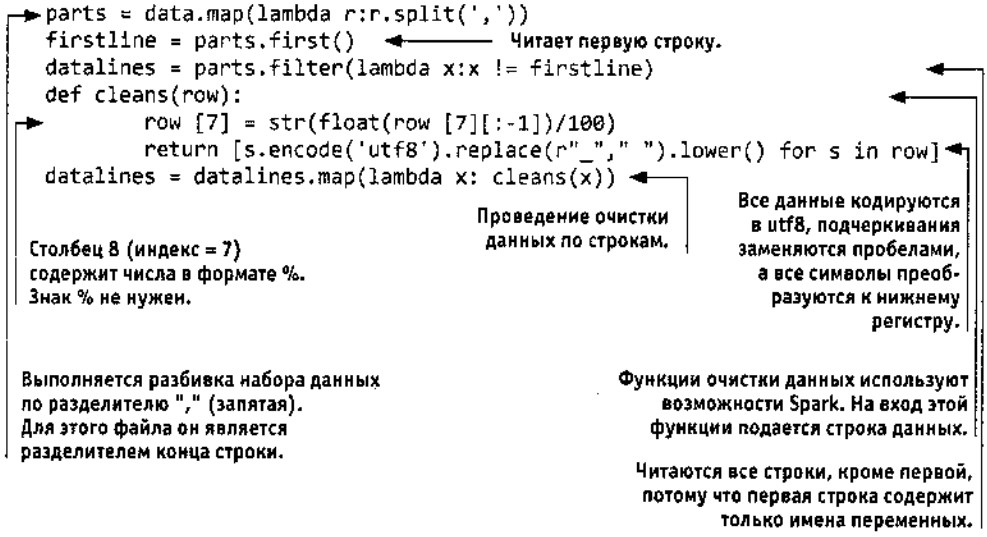
Импортирование контекста Hive.

В сеансе PySpark контекст Spark присутствует автоматически. В других случаях (веб-блокнот Zeppelin) придется его создать.

Создание контекста Spark.

Загрузка набора данных из каталога Hadoop.

```
from pyspark import SparkContext
from pyspark.sql import HiveContext
#sc = SparkContext()
sqlContext = HiveContext(sc)
data = sc.textFile("/chapter5/LoanStats3d.csv")
```



А теперь рассмотрим каждый этап более подробно.

Этап 1: Запуск Spark в интерактивном режиме и загрузка контекста

Импортировать контекст Spark в консоли PySpark не нужно, потому что контекст уже доступен в переменной `sc`. Возможно, вы также заметили соответствующее примечание при открытии PySpark; если не заметили — взгляните на рис. 5.9. Затем загружается контекст Hive, который позволяет интерактивно работать с Hive. Если вы работаете со Spark в интерактивном режиме, то контексты Spark и Hive загружаются автоматически, но если вы собираетесь использовать Spark в пакетном режиме, контекст придется загружать вручную. Для передачи кода в пакетном режиме используется команда spark-submit *имя_файла*.py в командной строке Horton Sandbox:

```
from pyspark import SparkContext
from pyspark.sql import HiveContext
sc = SparkContext()
sqlContext = HiveContext(sc)
```

После подготовки среды можно переходить к разбору файла .CSV.

Этап 2: Чтение и разбор файла .CSV

Затем мы читаем файл из файловой системы Hadoop и разбиваем его по запятым. В нашем коде первая строка читает файл .CSV из файловой системы Hadoop. Вто-

рая строка разбивает строки по запятым. Наша система разбора .CSV реализована наивно, потому что мы занимаемся изучением Spark; чтобы разбор строк выполнялся более корректно, используйте пакет .CSV:

```
data = sc.textFile("/chapter5/LoanStats3d.csv")
parts = data.map(lambda r:r.split(','))
```

Обратите внимание на сходство этой конструкции с синтаксисом функционального программирования. Это одна из наших любимых особенностей Spark.

Этап 3: Отделение строки заголовка от данных

Чтобы отделить заголовок от данных, мы читаем первую строку и оставляем все строки, отличные от строки заголовка:

```
firstline = parts.first()
datalines = parts.filter(lambda x:x != firstline)
```

Если при подготовке данных применялись рекомендации по работе с «большими данными», то этот шаг стал бы излишним, потому что первая строка уже хранилась бы в отдельном файле. На практике файлы .CSV часто содержат строку заголовка и вам приходится выполнять аналогичную операцию перед началом очистки данных.

Этап 4: Очистка данных

На этом шаге выполняются базовые действия по повышению качества данных. Это поможет повысить качество отчета.

После второго шага данные состоят из массивов. Мы будем рассматривать каждое входное значение лямбда-функции как массив и возвращать массив. Для упрощения этой задачи будет построена вспомогательная функция для выполнения очистки. Очистка состоит из переформатирования входных данных (например, «10,4%» в 0.104), перекодирования каждой строки в UTF-8, а также замены символов подчеркивания пробелами и перевода всех символов в нижний регистр. Вторая строка кода вызывает вспомогательную функцию для каждой строки массива:

```
def cleans(row):
        row [7] = str(float(row [7][:-1])/100)
        return [s.encode('utf8').replace(r"_"," ").lower() for s in row]
datalines = datalines.map(lambda x: cleans(x))
```

Наши данные подготовлены для отчета, теперь нужно открыть доступ к ним для средств построения отчетов. Hive хорошо подходит для этой цели, потому что

многие инструменты построения отчетов могут связываться с этой системой. Посмотрим, как это делается.

Сохранение данных в Hive

Для сохранения данных в Hive необходимо выполнить два действия:

1. Создание и регистрация метаданных.
2. Выполнение команд SQL для сохранения данных в Hive.

В этом разделе следующий фрагмент кода снова будет выполняться в оболочке PySpark, как показано в листинге 5.4.

Листинг 5.4. Сохранение данных в Hive

```
from pyspark.sql.types import *
fields = [StructField(field_name,StringType(),True) for field_name in
    firstline]
schema = StructType(fields)
schemaLoans = sqlContext.createDataFrame(datalines, schema)
schemaLoans.registerTempTable("loans")

sqlContext.sql("drop table if exists LoansByTitle")
sql = '''create table LoansByTitle stored as parquet as select title,
    count(1) as number from loans group by title order by number desc'''
sqlContext.sql(sql)

sqlContext.sql('drop table if exists raw')
sql = '''create table raw stored as parquet as select title,
    emp_title,grade,home_ownership,int_rate,recoveries,
    collection_recovery_fee,loan_amnt,term from loans'''
```

Создание метаданных: функция Spark SQL StructField представляет поле в StructType. Объект StructField состоит из трех полей: name (строка), dataType (DataType) и "nullable" (boolean). Поле name содержит имя StructField. Поле dataType определяет тип данных StructField. Поле nullable указывает, может ли поле StructField содержать значения None.

Импортирование типов данных SQL.

Функция StructType создает схему данных. Объект StructType получает на входе список StructField.

Регистрация как таблицы с именем loans.

Создание фрейма данных на основании данных (datalines) и схемы данных (schema).

Этот фрагмент удаляет таблицу (в том случае, если она уже существует) и сохраняет подмножество необработанных данных в Hive.

Этот фрагмент удаляет таблицу (в том случае, если она уже существует), обобщает данные и сохраняет в Hive. Таблица LoansByTitle представляет собой зависимость суммы кредитов от должностей.

А теперь рассмотрим каждый этап более подробно.

Этап 1: Создание и регистрация метаданных

Многие люди предпочитают использовать SQL при работе с данными. В Spark также существует такая возможность. Вы даже можете читать и хранить данные в Hive напрямую по мере изложения материала. Но чтобы это стало возможно, необходимо создать метаданные с именами и типами всех столбцов.

Первая строка кода содержит директивы импортирования. Вторая строка разбирает поля имени и типа и определяет, является ли поле обязательным. Структура `StructType` представляет строки как массив `StructField`. Затем данные помещаются во фрейм данных, зарегистрированный как (временная) таблица в Hive:

```
from pyspark.sql.types import *
fields = [StructField(field_name,StringType(),True) for field_name in
firstline]
schema = StructType(fields)
schemaLoans = sqlContext.createDataFrame(datalines, schema)
schemaLoans.registerTempTable("loans")
```

После того как метаданные будут подготовлены, данные можно вставить в Hive.

Этап 2: Выполнение запросов и сохранение таблицы в Hive

Теперь с данными можно работать на диалекте SQL. Сначала мы создадим обобщающую таблицу для подсчета количества кредитов. Затем подмножество очищенных данных будет сохранено в Hive для визуализации в Qlik.

Выполнение SQL-подобных команд сводится к передаче строки с командой функции `sqlContext.sql`. Обратите внимание: команда не содержит «чистый» синтаксис SQL, так как мы работаем с системой Hive, поддерживающей собственный диалект SQL с именем HjveQL. Например, в этой команде SQL мы немедленно приказываем сохранить данные в формате Parquet — популярном формате файлов больших данных:

```
sqlContext.sql("drop table if exists LoansByTitle")
sql = '''create table LoansByTitle stored as parquet as select title,
    count(1) as number from loans group by title order by number desc'''
sqlContext.sql(sql)

sqlContext.sql('drop table if exists raw')
sql = '''create table raw stored as parquet as select title,
    emp_title,grade,home_ownership,int_rate,recoveries,collection_recovery_f
    ee,loan_amnt,term from loans'''
sqlContext.sql(sql)
```

После того как данные будут сохранены в Hive, вы сможете подключиться к ним из программ визуализации.

5.2.4. Этап 4: Исследование данных и Этап 6: Построение отчета

Мы построим в Qlik Sense интерактивный отчет, который можно будет показать директору. Qlik Sense можно загрузить по адресу *http://www.qlik.com/try-or-buy/download-qlik-sense* после оформления подписки на веб-сайте. Когда загрузка начнется, вы будете перенаправлены на страницу с несколькими обучающими видеороликами по установке и работе с Qlik Sense. Мы рекомендуем просмотреть их перед работой.

Для чтения данных из Hive и организации доступа к ним из Qlik используется ODBC-соединитель Hive. Существует учебное руководство по установке ODBC-соединителей в Qlike; для основных операционных систем его можно найти по адресу *http:/ /hortonworks.com/hdp/addons/*.

ПРИМЕЧАНИЕ

В Windows это решение может не работать в стандартном варианте. После установки ODBC следует проверить менеджер ODBC системы Windows (Ctrl+F и найдите ODBC). В менеджере откройте раздел System-DSN и выберите вариант Sample Hive Hortonworks DSN. Убедитесь в том, что настройки заданы правильно (рис. 5.11); в противном случае Qlik не сможет подключиться к Hortonworks Sandbox.

Будем надеяться, что вы еще не забыли свой пароль Sandbox; как видно из рис. 5.11, он вам снова понадобится.

Теперь откройте Qlik Sense. При установке в системе Windows вам будет предоставлена возможность создать ярлык для файла .exe на Рабочем столе. Qlik не бесплатная программа; это коммерческий продукт с пробной версией, но нам пока хватит и этого. В последней главе мы создадим информационную панель с использованием бесплатных библиотек JavaScript.

Qlik может либо загружать данные прямо в память, либо каждый раз обращаться с вызовом к Hive. Мы выбрали первый способ, потому что он работает быстрее.

Эта операция состоит из трех шагов:

1. Загрузка данных в Qlik через подключение ODBC.
2. Создание отчета.
3. Исследование данных.

Начнем с первого шага — загрузки данных в Qlik.

Этап 1: Загрузка данных в Qlik

При запуске Qlik Sense на экране появляется заставка с существующими отчетами (которые называются *приложениями*) (рис. 5.12).

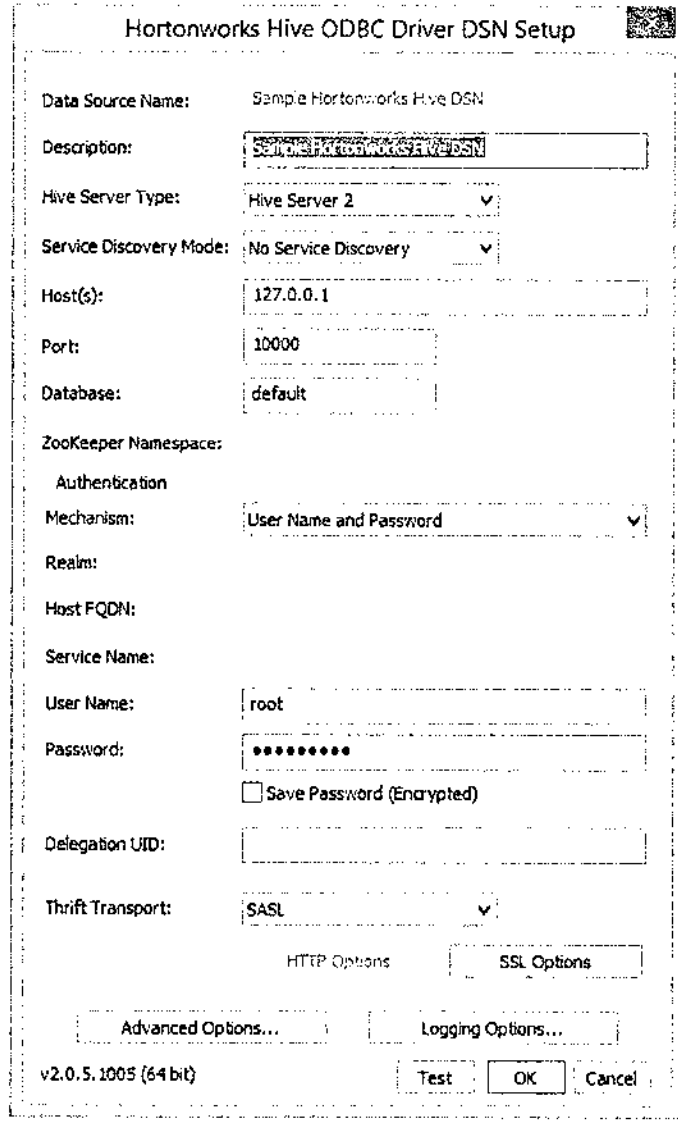


Рис. 5.11. Настройка Hortonworks ODBC в системе Windows

Рис. 5.12. Заставка Qlik Sense

Чтобы создать новое приложение, щелкните на кнопке Create new app в правой нижней части экрана (рис. 5.13.) Открывается новое диалоговое окно. Введите в поле имя «chapter 5»:

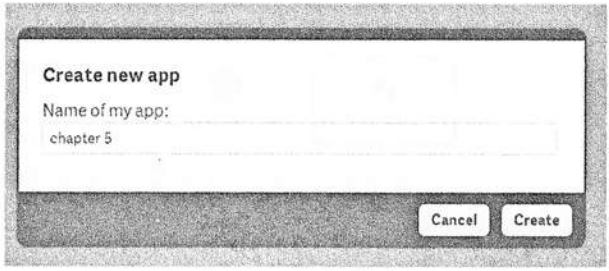


Рис. 5.13. Диалоговое окно Create new app

Если приложение было создано успешно, на экране появится окно с подтверждением (рис. 5.14).

Рис. 5.14. Подтверждение успешного создания приложения

Щелкните на кнопке Open app; открывается новое окно, которое предлагает вам начать добавлять данные в приложение (рис. 5.15).

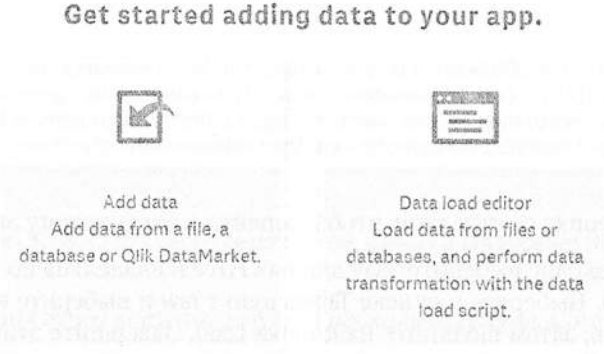


Рис. 5.15. При открытии нового приложения открывается окно с предложением начать добавление данных

Щелкните на кнопке Add data и выберите ODBC в качестве источника данных (рис. 5.16).

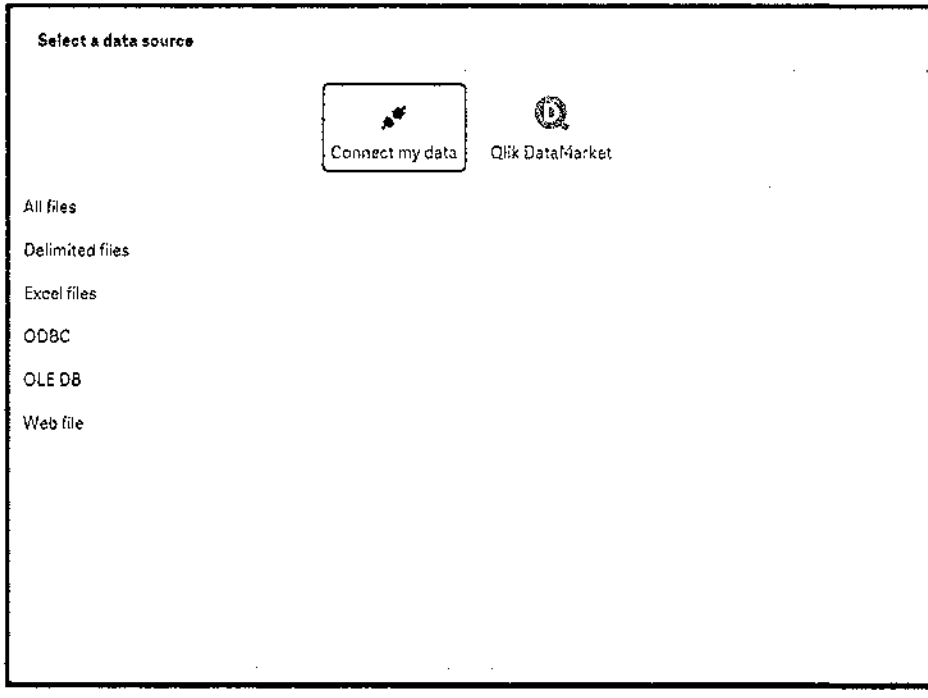


Рис. 5.16. Выбор ODBC в качестве источника данных на экране Select a data source

На следующем экране (рис. 5.17) выберите вкладку User DSN с Hortonworks, укажите имя пользователя root и пароль hadoop (или новый пароль, который вы ввели при первом входе в Sandbox).

ПРИМЕЧАНИЕ

Вариант Hortonworks не отображается по умолчанию. Чтобы он появился, необходимо установить ODBC-соединитель HDP 2.3 (как упоминалось ранее). Если вам еще не удалось его успешно установить, подробные инструкции можно найти по адресу https://blogs.perficient.com/multi-shoring/blog/2015/09/29/how-to-connect-hortonworks-hive-from-qlikview-with-odbc-driver/.

Щелкните на кнопке со стрелкой, чтобы перейти к следующему экрану.

На следующем экране выберите базу данных Hive и владельца по умолчанию (default) (рис. 5.18). Выберите в списке Tables пункт raw и выберите все столбцы для импортирования; затем щелкните на кнопке Load. Завершите этот шаг нажатием кнопки Finish.

Рис. 5.17. Выбор Hortonworks в разделе User DSN с указанием имени пользователя и пароля

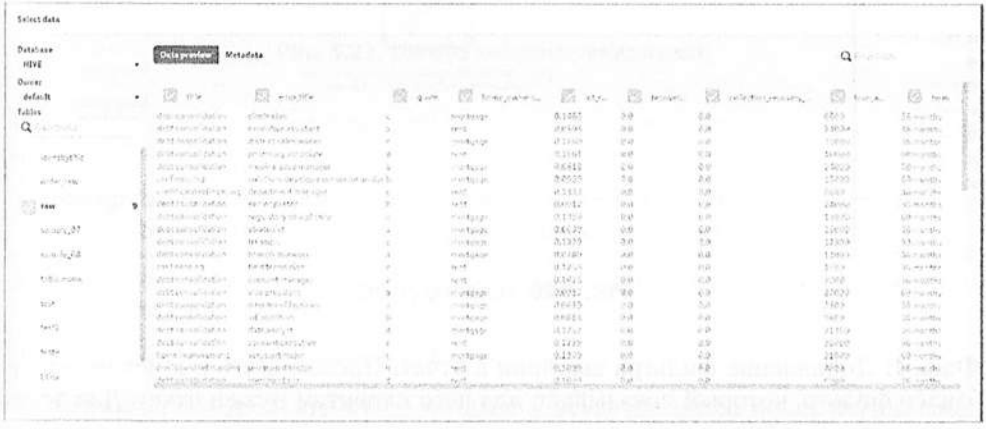


Рис. 5.18. Столбцовое представление данных в интерфейсе Hive

После выполнения этого шага потребуется несколько секунд для загрузки данных в Qlik (рис. 5.19).

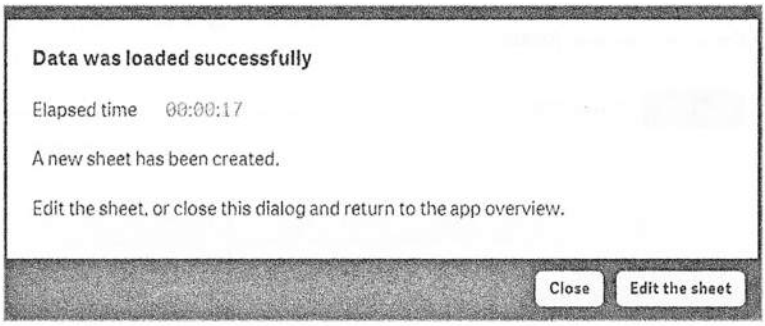


Рис. 5.19. Подтверждение загрузки данных в Qlik

Этап 2: Создание отчета

Щелкните на кнопке Edit the sheet, чтобы приступить к построению отчета. На экране появляется редактор отчета (рис. 5.20).

Рис. 5.20. Редактор отчета

Фаза 1: Добавление фильтра выборки в отчет. Прежде всего в отчет будет добавлен фильтр, который показывает, для чего клиентам нужен отчет. Для этого перетащите поле title с левой панели на панель отчета; выберите размер и позицию области по своему усмотрению (рис. 5.21). Чтобы иметь возможность перетаскивать поля, щелкните на панели Fields.

Фаза 2: Добавление ключевых показателей в отчет. На панели ключевых показателей выводятся такие показатели, как средняя процентная ставка и общее количество клиентов (рис. 5.22).

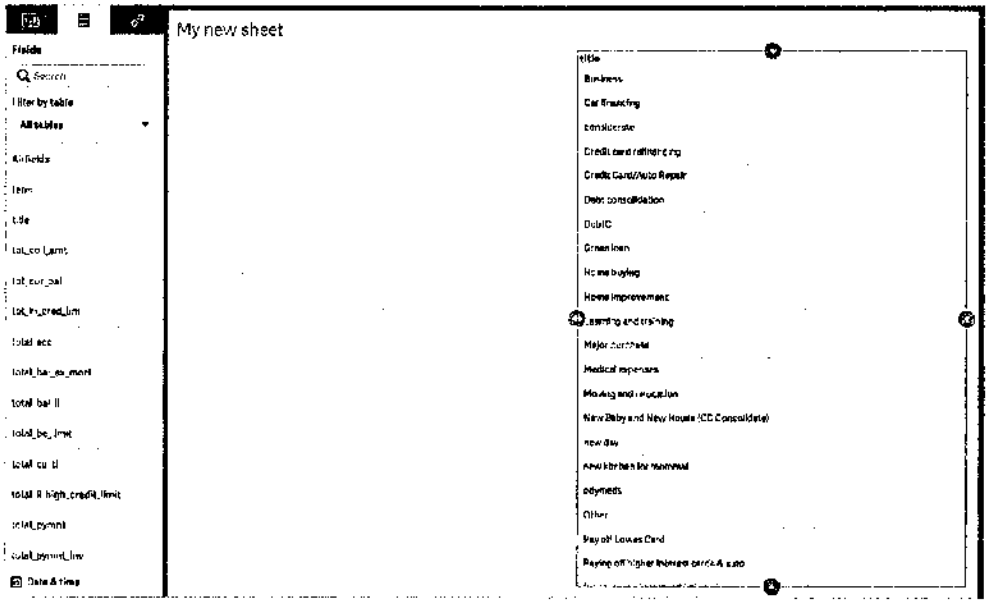


Рис. 5.21. Перетащите поле title с левой панели Fields на панель отчета

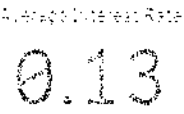


Рис. 5.22. Один из ключевых показателей

Добавление показателей в отчет состоит из следующих четырех этапов, представленных на рис. 5.23.

1. *Выбор диаграммы* — выберите тип диаграммы KPI и перенесите на экран отчета; выберите размер и позицию области по своему усмотрению.
2. *Выбор метрики* — щелкните на кнопке Add measure внутри диаграммы и выберите int_rate.
3. *Выбор метода агрегирования* — Avg(int_rate).
4. *Форматирование диаграммы* — на правой панели введите в поле Label название показателя Average Interest Rate.

Всего в отчет будут добавлены четыре ключевых показателя, поэтому эти действия должны быть выполнены для следующих показателей:

- Средняя процентная ставка (Average interest rate).
- Общая сумма кредита (Total loan amount).

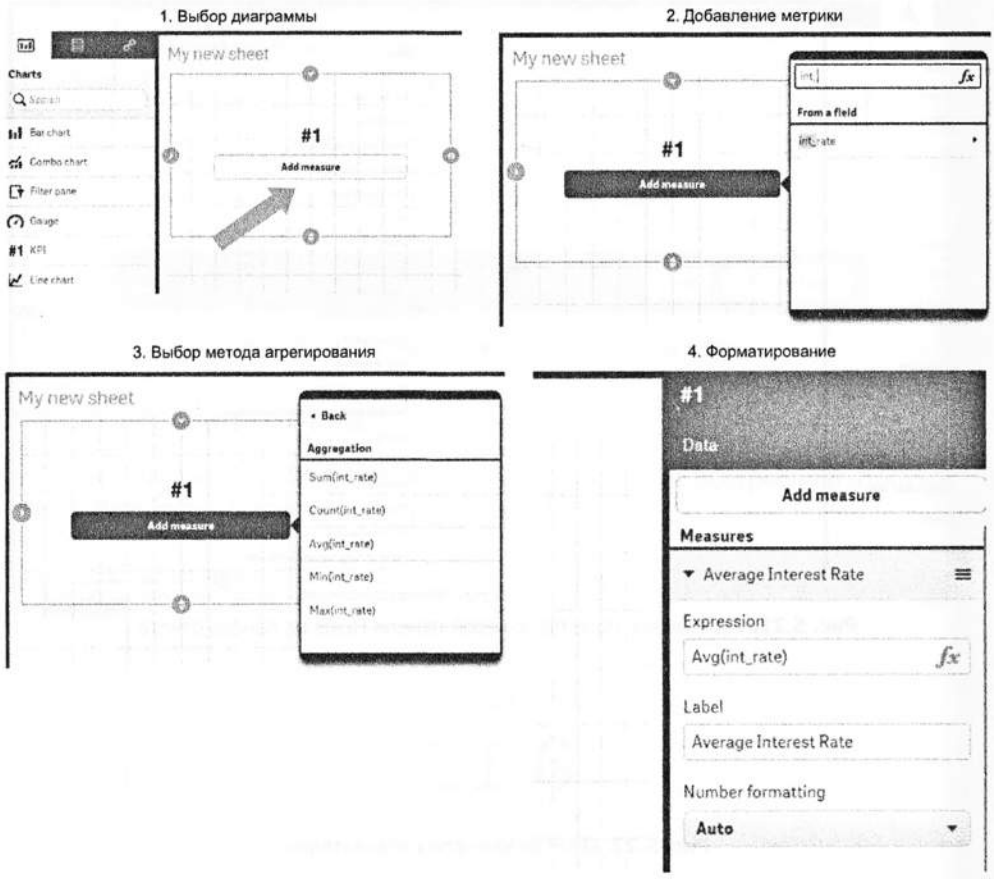


Рис. 5.23. Четыре этапа добавления диаграммы KPI в отчет Qlik

- Средняя сумма кредита (Average loan amount).
- Общее возмещение (Total recoveries).

Фаза 3: Добавление столбцовых диаграмм в отчет. Затем мы добавим четыре столбцовые диаграммы в отчет. На них будут отображаться разные числа для каждой группы риска. Одна столбцовая диаграмма будет показывать распределение средней процентной ставки по группам риска, а на другой будет отображаться распределение общей суммы кредита по группам риска (рис. 5.24).

Процесс добавления столбцовой диаграммы в отчет состоит из пяти шагов, представленных на рис. 5.25.

1. *Выбор диаграммы* — выберите столбцовую диаграмму (Bar chart) на левой панели и разместите ее на экране отчета; выберите размер и позицию области по своему усмотрению.

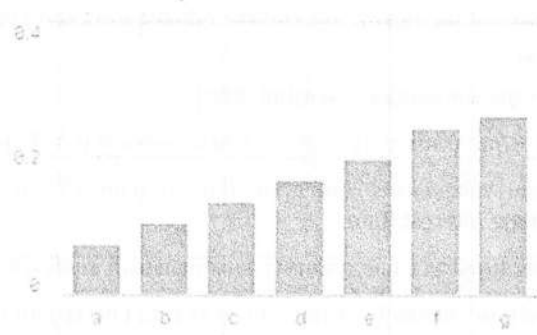


Рис. 5.24. Пример столбцовой диаграммы

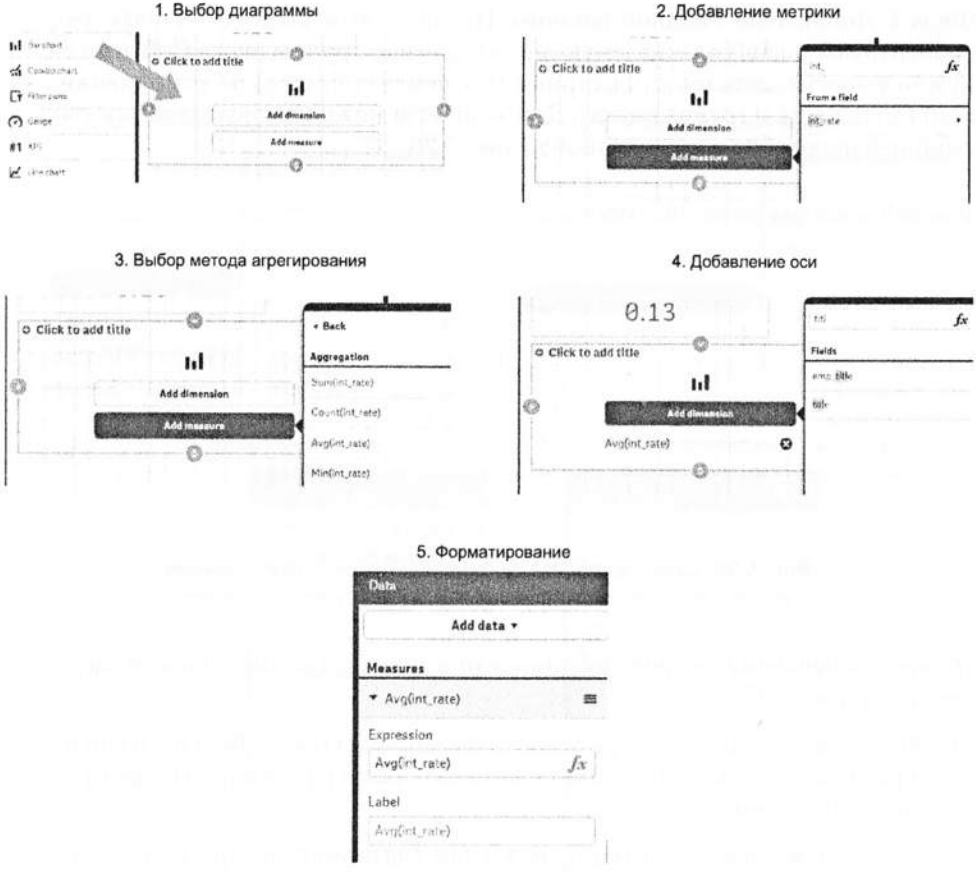


Рис. 5.25. Добавление столбцовой диаграммы состоит из пяти шагов

2. *Добавьте метрику* — щелкните на кнопке Add measure внутри диаграммы и выберите `int_rate`.
3. *Выбор метода агрегирования* — Avg(int_rate).
4. *Добавление оси* — щелкните на кнопке Add dimension и выберите grade.
5. *Форматирование диаграммы* — на правой панели введите в поле Label название показателя Average Interest Rate.

Повторите эту процедуру для следующих комбинаций осей и метрик:

- Средняя процентная ставка (Average interest rate) по группам риска (grade).
- Общая сумма кредита (Total loan amount) по группам риска (grade).
- Средняя сумма кредита (Average loan amount) по группам риска (grade).
- Общее возмещение (Total recoveries) по группам риска (grade).

Фаза 4: Добавление сводной таблицы. Предположим, вы хотите узнать среднюю процентную ставку, уплачиваемую директорами из группы риска C. В этом случае нужно вывести зависимость метрики (процентная ставка) от комбинации двух осей (должность и группа риска). Для этой цели можно воспользоваться сводной таблицей наподобие изображенной на рис. 5.26.

average interest rate per job title / risk grade

	a	b	c	d
electrician	0.072455172	0.099825	0.13674545	0.16789333
executive assistant	0.069928571	0.1023193	0.13515811	0.16489091
district sales leader	0.0692	0.1049	0.1269	-
pharmacy associate	-	-	-	0.1561
medical case manager	-	0.0818	-	-
solutions development senior analyst	-	0.0999	-	-
department manager	0.075866667	0.10396667	0.132172	0.17185

Рис. 5.26. Сводная таблица со средними процентными ставками, выплачиваемыми по комбинациям «должность/группа риска»

Процесс добавления сводной таблицы в отчет состоит из шести шагов, представленных на рис. 5.27.

1. *Выбор диаграммы* — выберите сводную таблицу (Pivot table) на левой панели и разместите ее на экране отчета; выберите размер и позицию области по своему усмотрению.
2. *Добавьте метрику* — щелкните на кнопке Add measure внутри диаграммы и выберите `int_rate`.
3. *Выбор метода агрегирования* — Avg(int_rate).

4. *Добавление горизонтальной оси* — щелкните на кнопке Add dimension и выберите emp_title.
5. *Добавление вертикальной оси* — щелкните на кнопке Add dimension, выберите Column и затем grade.
6. *Форматирование диаграммы* — на правой панели введите в поле Label название показателя Average Interest Rate.

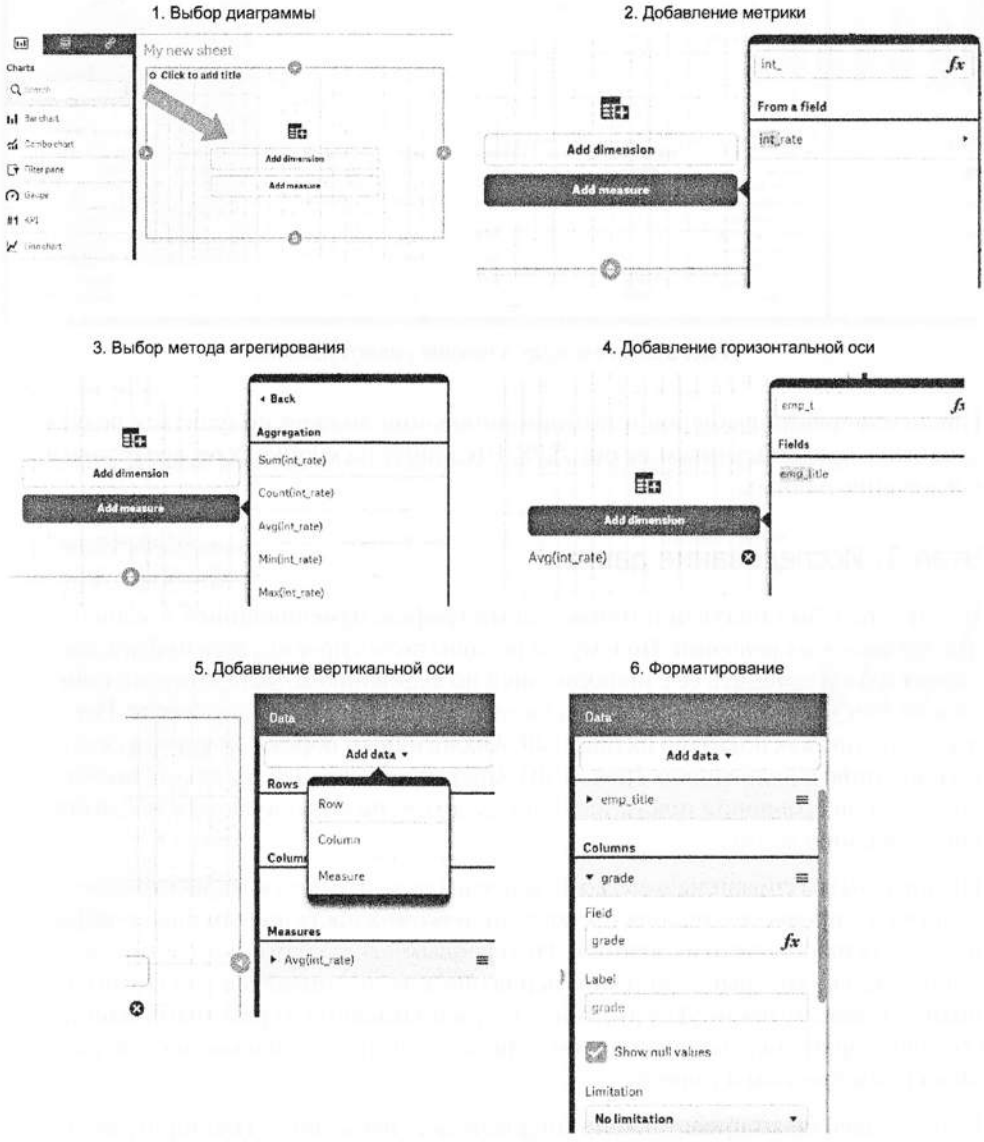


Рис. 5.27. Добавление сводной таблицы состоит из шести шагов

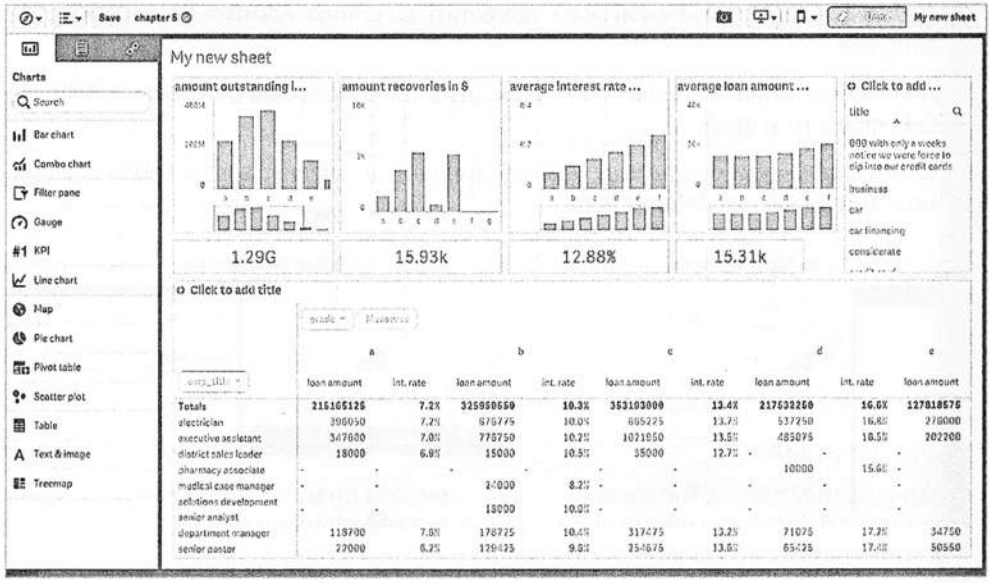


Рис. 5.28. Результат в режиме редактирования

После изменения размеров и позиционирования должен получиться результат, сходный с представленным на рис. 5.28. Щелкните на кнопке Done; все готово к исследованию данных.

Этап 3: Исследование данных

В результате был получен интерактивный график, изменяющийся в зависимости от принимаемых решений. Почему бы не попытаться просмотреть информацию по директорам и сравнить ее с информацией по художникам? Для этого щелкните на кнопке emp_title в сводной таблице и введите в поле поиска строку director. Результат выглядит так, как показано на рис. 5.29. Аналогичным образом можно просмотреть информацию о художниках (рис. 5.30). Другой интересный результат может быть получен при сравнении показателей по кредитам на покупку дома с кредитами на консолидацию долгов.

Наконец-то мы справились со своей задачей: создали отчет, который требует руководство, а в процессе создания предоставили возможность другим людям создавать их отчеты на основе этих данных. Интересным следующим шагом мог бы стать поиск людей, которые с большой вероятностью не справятся со своими долговыми обязательствами. Для этого можно воспользоваться средствами машинного обучения Spark под управлением онлайновых алгоритмов наподобие тех, которые были представлены в главе 4.

В этой главе было представлено практическое введение в Hadoop и Spark. Мы рассмотрели достаточно большой объем материала, но, откровенно говоря, Python

делает работу с технологиями больших данных попросту элементарной. В следующей главе мы поближе познакомимся с миром баз данных NoSQL, а попутно будут представлены другие технологии больших данных.

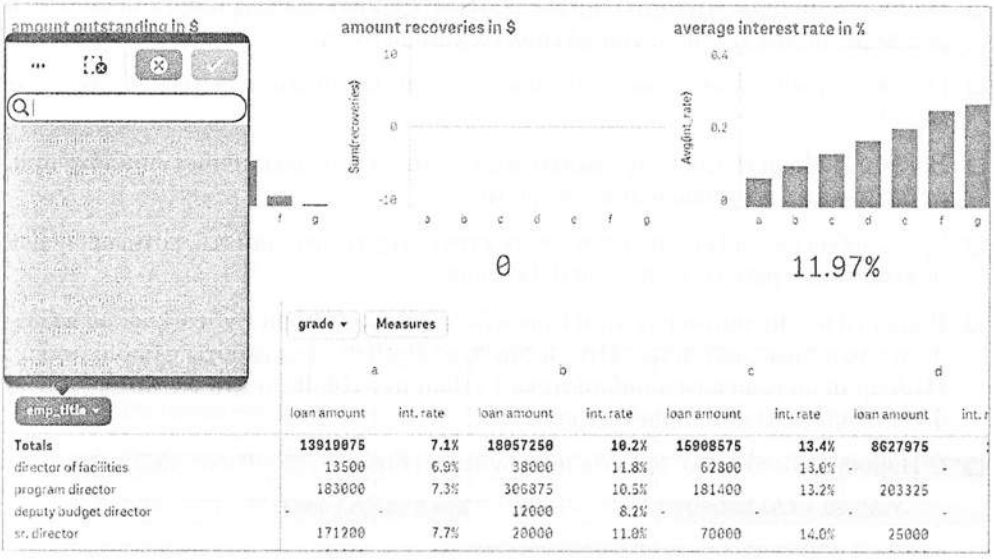


Рис. 5.29. Выбирая директоров, мы видим, что они платят среднюю процентную ставку 11,97%

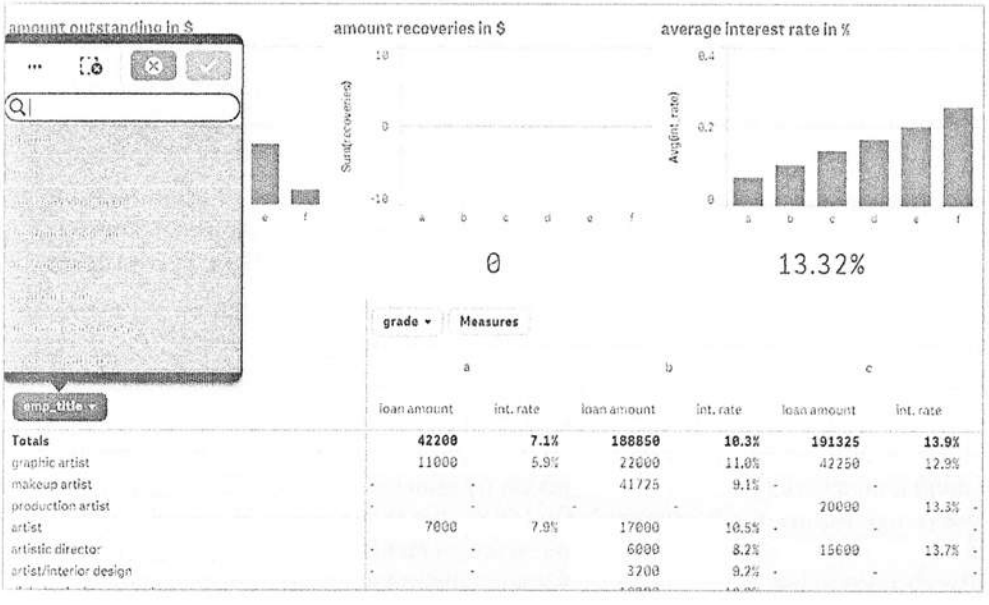


Рис. 5.30. А при выборе художников видно, что их средняя процентная ставка составляет 13,32%

5.3. Итоги

Основные положения этой главы:

- Hadoop – инфраструктура, позволяющая хранить файлы и организовывать распределенные вычисления на многих компьютерах.
- Hadoop скрывает все сложности работы в компьютерном кластере от программиста.
- Вокруг Hadoop и Spark сформировалась экосистема различных приложений, от баз данных до управления доступом.
- Spark добавляет в Hadoop Framework структуру общей памяти, которая лучше подходит для работы в области data science.
- В практических примерах этой главы библиотека Python PySpark использовалась для взаимодействия с Hive и Spark из Python. Для работы с библиотекой Hadoop использовалась библиотека Python pywebhdfs, но то же самое можно было сделать из командной строки ОС.
- С Hadoop можно легко связать инструменты бизнес-аналитики, такие как Qlik.

Присоединяйтесь к движению NoSQL

В этой главе:

- ✓ Базы данных NoSQL и причины для их использования в наши дни.
- ✓ Различия между базами данных NoSQL и реляционными базами данных.
- ✓ Определение принципа ACID и его связь с принципом BASE технологий NoSQL.
- ✓ Важность теоремы CAP для многоузловой конфигурации базы данных.
- ✓ Применение процесса data science к проекту в базе данных NoSQL Elasticsearch.

Эта глава делится на две части: теоретическую и практическую.

- В первой части главы рассматриваются базы данных NoSQL вообще и даются ответы на основные вопросы: для чего они нужны? почему появились только недавно? на какие типы делятся и почему представляют интерес для вас?
- Во второй части рассматривается реальная задача — диагностика и профилирование болезней. Для ее решения будут использоваться свободно распространяемые данные, Python и база данных NoSQL.

Несомненно, вы слышали о базах данных NoSQL и их популярности во многих передовых компаниях. Но что собой представляют базы данных NoSQL и чем они так отличаются от реляционных баз данных или привычных нам баз данных SQL?

NoSQL — сокращение от «Not Only SQL», т. е. *«не только* SQL». И хотя базы данных NoSQL предоставляют возможность обращаться с запросами на языке SQL, название не так уж важно. По поводу названия (и вообще того, может ли эта группа новых баз данных иметь объединяющее название) шли ожесточенные споры. Но нужно смотреть скорее не на название, а на то, какие возможности предоставляют эти базы данных по сравнению с традиционными *реляционными системами управления базами данных* (РСУБД). Традиционные базы данных существуют на одном компьютере или сервере. И такое решение отлично работает, если данные не выходят за границы

сервера, но уже давно и во многих компаниях ситуация изменилась. С ростом Интернета такие компании, как Google и Amazon, решили, что ограничения одноузловых баз данных им не подходят, и взялись за поиск альтернатив.

Многие компании используют одноузловые базы данных NoSQL (такие, как MongoDB), потому что им нужна гибкая схема или способность к иерархическому агрегированию данных. Несколько ранних примеров:

- Первое решение в области NoSQL от компании Google — Google BigTable — ознаменовало начало эпохи *столбцовых баз данных*[1].
- Компания Amazon представила Dynamo — *хранилище «ключ — значение»*[2].
- Стремление к реализации секционирования (partitioning) породило еще два типа баз данных: *хранилища документов* и *графовые базы данных*.

Все четыре типа будут более подробно рассмотрены далее в этой главе.

Важное замечание: хотя размер данных играет важную роль, базы данных NoSQL появились не только из-за необходимости работать с большими наборами данных. Важно каждое из «четырех V» больших данных (объем, разнообразие, скорость, достоверность). Например, графовые базы данных хорошо работают с сетевыми данными. Энтузиасты графовых баз данных даже утверждают, что все сущности можно рассматривать как сеть. Например, как приготовить обед? Из ингредиентов. Эти ингредиенты собираются вместе, составляя блюдо, и могут использоваться с другими ингредиентами для формирования других блюд. Если смотреть с этой точки зрения, ингредиенты и рецепты являются частью сети. Однако рецепты и ингредиенты также могут храниться в реляционной базе данных или в хранилище документов; все зависит от того, как рассматривать задачу. В этом проявляется сила NoSQL: возможность взглянуть на задачу под другим углом и сформировать структуру данных под сценарий использования. Ваша задача как специалиста data science — найти лучшее решение для любой задачи. Хотя иногда результат проще достигается с использованием РСУБД, часто одна конкретная база данных NoSQL обеспечивает лучшее решение.

Обречены ли реляционные базы данных на вымирание в компаниях с большими данными из-за необходимости в секционировании? Нет, платформы NewSQL (не путайте с NoSQL) предоставляют ответ РСУБД на потребность в кластерных конфигурациях. Базы данных NewSQL ориентированы на реляционную модель, но они могут обеспечивать параллелизацию в распределенных кластерах, как и базы данных NoSQL. Конечно, это еще не конец реляционных баз данных и, безусловно, не конец SQL, потому что такие платформы, как Hive, преобразуют SQL в задания MapReduce для Hadoop. Кроме того, не каждой компании нужны большие данные; многие спокойно обходятся небольшими объемами информации, и традиционные реляционные базы данных идеально подходят для этой цели.

---

[1] См. http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf.

[2] См. http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf.

На схеме (рис. 6.1) представлены четыре типа баз данных NoSQL.

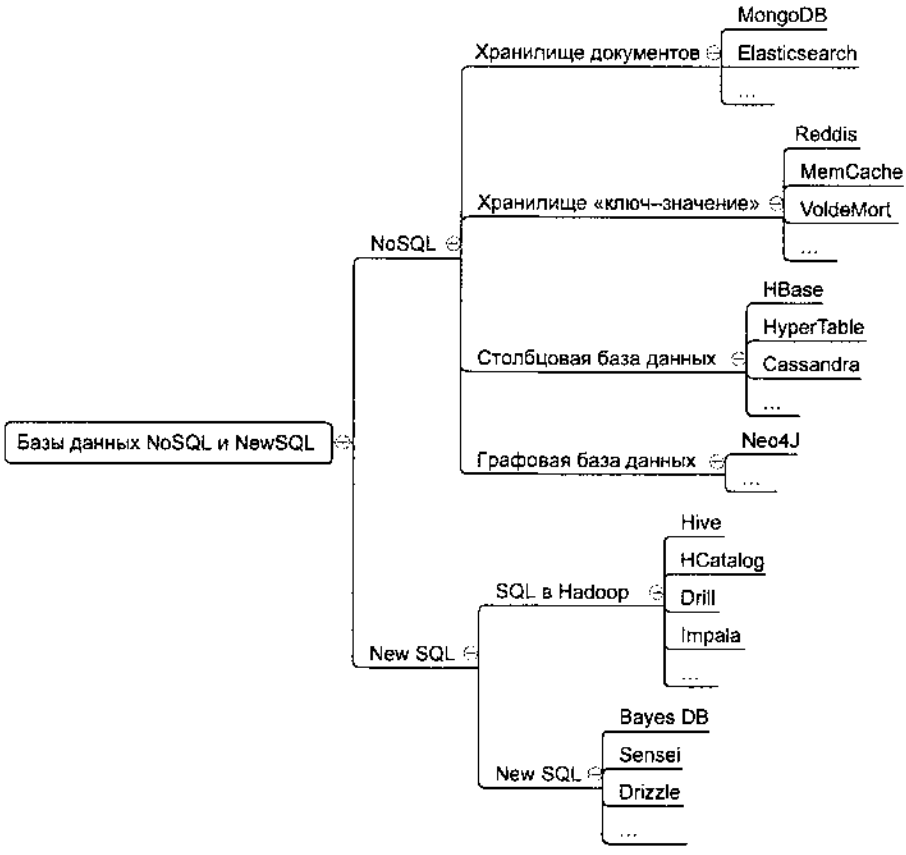


Рис. 6.1. Базы данных NoSQL и NewSQL

Эти четыре типа баз данных — хранилище документов, хранилище «ключ—значение», графовая база данных и столбцовая база данных. На схеме также представлены новые секционируемые реляционные базы данных NewSQL. В будущем четкая грань между NoSQL и NewSQL будет размываться, потому что каждый тип базы данных будет концентрироваться на каких-то отдельных аспектах, объединяя элементы из NoSQL и NewSQL. Эти грани постепенно размываются уже сейчас, по мере того как типы РСУБД наделяются возможностями NoSQL (например, столбцово-ориентированным индексированием, встречающимся в столбцовых базах данных). А пока этот термин удобно использовать для обозначения того, что старые реляционные базы данных выходят за рамки одноузловых конфигураций, а в категории NoSQL появляются другие типы баз данных.

Посмотрим, что же нам дает NoSQL.

6.1. Введение в NoSQL

Как упоминалось ранее, цель баз данных NoSQL заключается в том, чтобы предоставить не только успешный механизм секционирования баз данных по нескольким узлам, но и принципиально новые способы моделирования имеющихся данных и адаптации их структуры к сценарию использования (а не к требованиям реляционной базы данных).

Чтобы помочь вам понять NoSQL, мы сначала рассмотрим базовые принципы односерверных реляционных баз данных (ACID) и покажем, как базы данных NoSQL преобразуют их в форму (BASE), гораздо лучше подходящую для распределенной обработки. Также будет рассмотрена теорема CAP, описывающая главную проблему использования распределенных баз данных на нескольких узлах, и подход к ее решению базами данных ACID и BASE.

6.1.1. ACID: базовые принципы реляционных баз данных

Основные аспекты традиционных реляционных баз данных нередко обозначаются сокращением ACID:

- *Атомарность* (Atomicity) — принцип «все или ничего». Если блок данных включается в базу, то он либо включается полностью, либо не включается вообще. Например, если в середине операции записи произойдет сбой питания, в базу данных не будет занесена половина данных; они не будут записаны.
- *Согласованность* (Consistency) — этот важный принцип обеспечивает целостность данных. Запись, вставленная в базу данных, ни при каких условиях не должна конфликтовать с заранее установленными правилами (например, в ней не может отсутствовать обязательное поле или поле не может содержать числовую информацию вместо текстовой).
- *Изолированность* (Isolation) — когда в базе данных что-то изменяется, ничего не может происходить точно с одними и теми же данными точно в один момент. Вместо этого действия выполняются последовательно с другими изменениями. Изолированность — характеристика, измеряемая по шкале от низкой до высокой. На этой шкале традиционные базы данных располагаются в конце «высокой изолированности». Примером низкой изолированности служит система Google Docs: несколько людей могут вести запись в документ одновременно, и каждый из них моментально видит изменения, внесенные другими участниками. С другой стороны, традиционный документ Word находится в конце «высокой изолированности»; когда первый пользователь открывает его, документ блокируется для редактирования. Второй пользователь, открывший документ, сможет просмотреть последнюю сохраненную версию, но не увидит несохраненные изменения и не сможет редактировать документ,

если предварительно не сохранит его копию. После того как документ будет открыт, его последняя версия полностью изолируется от всех, кроме пользователя, заблокировавшего документ.

- *Долгосрочность* (Durability) — если данные внесены в базу данных, то они должны находиться в базе данных постоянно. Данные могут уничтожаться при физическом повреждении жестких дисков, но не при отключении электропитания и программных сбоях.

Принципы ACID распространяются на все реляционные базы данных и некоторые базы данных NoSQL (например, графовую базу данных Neo4j). Графовые базы данных будут рассматриваться позднее в этой главе и в главе 7. Большинство других баз данных NoSQL подчиняется другому набору принципов: BASE. Но чтобы понять, что такое принципы BASE и почему они относятся к большинству баз данных NoSQL, необходимо рассмотреть теорему CAP.

6.1.2. Теорема CAP: проблема баз данных, распределенных по многим узлам

После того как база данных будет распределена по разным серверам, соблюдение принципов ACID существенно затрудняется из-за гарантий согласованности; теорема CAP указывает, почему возникают эти проблемы. Теорема CAP утверждает, что база данных может обладать любыми двумя из следующих свойств, но никогда всеми тремя:

- *Устойчивость к разделению* (Partition tolerance) — база данных может справиться с сетевой сегментацией или сетевым сбоем.
- *Доступность* (Availability) — при условии, что узел, к которому вы подключаетесь, находится в работоспособном состоянии и с ним можно связаться, узел ответит даже при нарушении связи с другими узлами базы данных.
- *Согласованность* (Consistency) — с каким бы узлом ни была установлена связь, вы всегда получите одни и те же данные.

Очевидно, что база данных с одним узлом всегда обладает свойствами доступности и согласованности:

- *Доступность* — если узел работает, он доступен. Требование доступности CAP гарантирует только это.
- *Согласованность* — второго узла нет, поэтому рассогласование невозможно.

Но если база данных секционируется, ситуация становится более интересной. В этом случае приходится выбирать между доступностью и согласованностью (рис. 6.2).

Для примера рассмотрим интернет-магазин с серверами в Европе и в США с единым центром распределения. Немец по имени Фриц и американец по имени Фредди совершают покупку одновременно в одном магазине. Они видят товар,

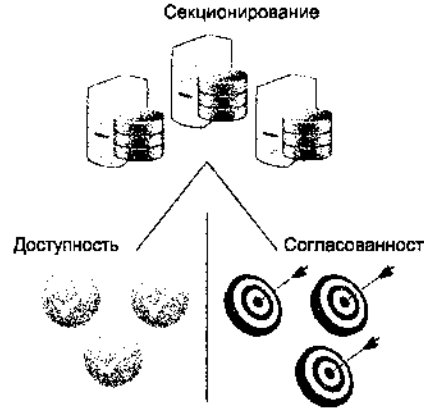


Рис. 6.2. Теорема CAP: при секционировании базы данных приходится выбирать между доступностью и согласованностью

который остался в магазине всего в одном экземпляре: бронзовый кофейный столик в виде осьминога. Происходит сбой, и связь между двумя локальными серверами временно нарушается. У владельца магазина есть два варианта.

- *Доступность* — он разрешает серверам продолжить обслуживание клиентов, чтобы потом разобраться со всеми возможными проблемами.
- *Согласованность* — все продажи приостанавливаются до восстановления связи.

В первом случае и Фриц и Фредди смогут купить столик, потому что последний известный товарный запас для обоих узлов был равен 1 и обоим узлам разрешена продажа товара (рис. 6.3).

Если кофейный столик является редкостью, вам придется сообщить либо Фрицу, либо Фредди, что он не получит свою покупку к обещанной дате доставки - или, того хуже, не получит ее никогда. Сознательный продавец может компенсировать неудобства купоном на скидку при последующих покупках, и, возможно, все кончится благополучно.

Во втором варианте (рис. 6.4) обработка всех входящих запросов временно приостанавливается.

Если через пять минут интернет-магазин снова откроется, все будет справедливо по отношению и к Фрицу и к Фредди, но может оказаться, что вы потеряли обе продажи (а возможно, много других). Интернет-магазины обычно выбирают доступность в ущерб согласованности, но этот выбор не является оптимальным во всех случаях. Возьмем какой-нибудь популярный фестиваль. Обычно по соображениям безопасности у фестивалей имеется максимально допустимая посещаемость. Если вы продадите больше билетов, чем разрешено, потому что ваши серверы продолжали продавать билеты во время сбоя сетевой связи, к моменту восстановления связи будет продано вдвое больше билетов, чем возможно. В таком случае разум-

нее выбрать согласованность и временно отключить узлы. Билеты на популярные фестивали распродаются за пару часов, так что небольшой простой не принесет столько вреда, как необходимость возвращать тысячи входных билетов.

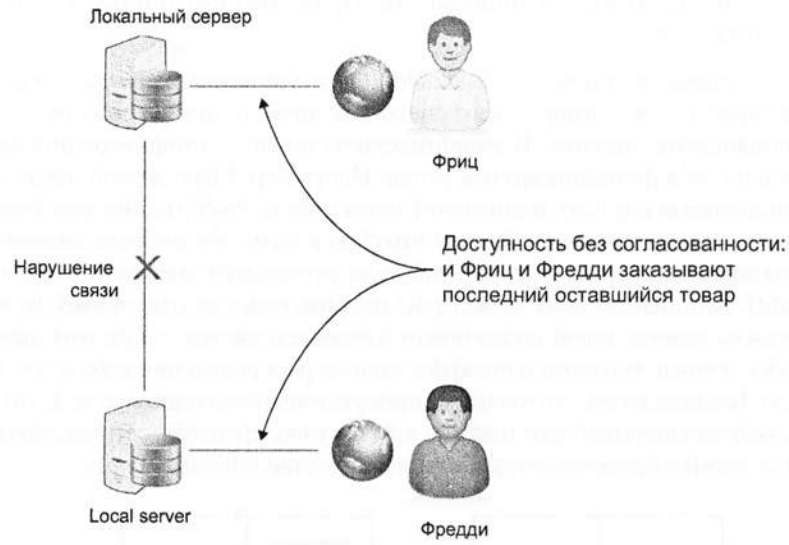


Рис. 6.3. Теорема CAP: если между узлами будет нарушена связь, можно выбрать сохранение доступности, но тогда возможно рассогласование данных

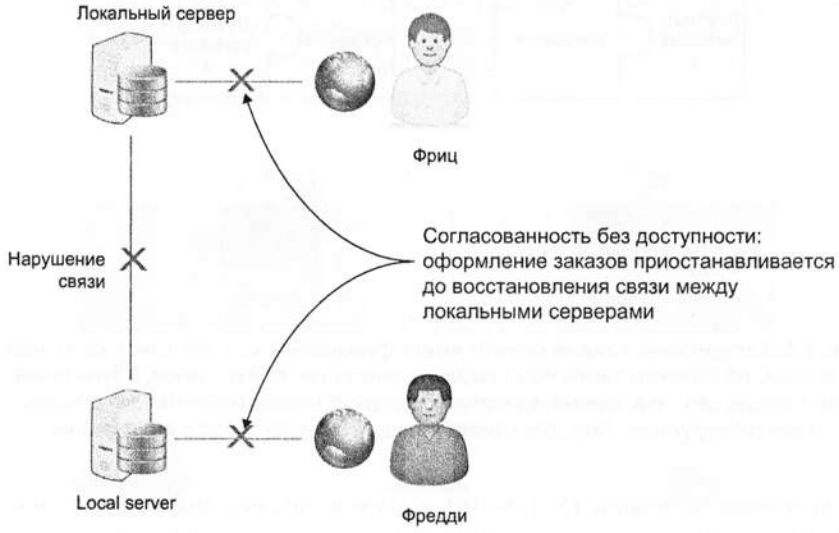


Рис. 6.4. Теорема CAP: если между узлами будет нарушена связь, можно выбрать обеспечение согласованности, приостановив доступ к базе данных до восстановления связи

6.1.3. Принципы BASE баз данных NoSQL

РСУБД обеспечивают соблюдение принципов ACID; базы данных NoSQL, не подчиняющиеся архитектуре ACID (такие, как хранилища документов и хранилища «ключ—значение»), соблюдают принципы BASE. Набор гарантий BASE отличается меньшей жесткостью:

- *Базовая доступность* (Basically Available) — доступность гарантируется в смысле CAP. Возвращаясь к примеру с интернет-магазином, если узел работает, клиенты могут продолжать покупки. В зависимости от текущей конфигурации узлы могут брать на себя функции других узлов. Например, Elasticsearch представляет собой поисковую систему полнотекстового поиска NoSQL, которая разделяет и реплицирует данные таким образом, что сбой узла не обязательно означает отказ в обслуживании; для достижения этой цели используется процесс *сегментации* (sharding). Каждый сегмент может рассматриваться как отдельный экземпляр сервера базы данных, но он также может взаимодействовать с другими экземплярами, обеспечивая максимально эффективное распределение рабочей нагрузки (рис. 6.5). Несколько сегментов могут присутствовать на одном узле. Если у каждого сегмента существует дубликат на другом узле, проблема сетевого сбоя легко решается перераспределением работы между оставшимися узлами.

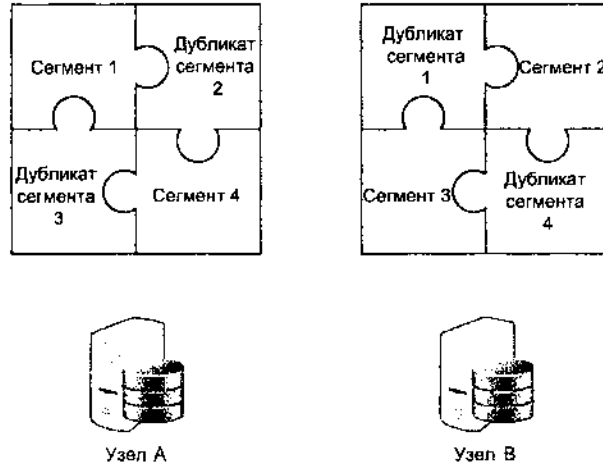


Рис. 6.5. Сегментация: каждый сегмент может функционировать как самодостаточная база данных, но сегменты также могут работать вместе как единое целое. В этом примере представлены два узла, каждый из которых содержит четыре сегмента: два основных и два дублирующих. При сбое одного узла другой берет на себя его функции

- *Неустойчивое состояние* (Soft state) — состояние системы может изменяться со временем. Это соответствует принципу *согласованности в конечном счете*: возможно, системе придется измениться, чтобы данные снова стали согласованными. На одном узле данные содержат «A», а на другом узле они содержат «B»,

потому что они были адаптированы. Позднее, при разрешении конфликта, когда сеть снова работает нормально, данные «А» первого узла могут быть заменены «В». И хотя для замены «А» на «В» явно ничего не предпринималось, это значение заменяется для обеспечения согласованности с другим узлом.

- *Согласованность в конечном счете* (Eventual consistency) — база данных восстанавливает согласованность с течением времени. В примере с интернет-магазином столик продавался дважды, что привело к рассогласованию данных. При восстановлении связи между отдельными узлами они взаимодействуют друг с другом и решают, как поступить. Например, конфликт может быть решен на основе «первым пришел, первым обслужен», или же можно отдать предпочтение клиенту с более низкой стоимостью доставки. В базах данных существует поведение по умолчанию, но с учетом того, что в данном случае должно приниматься конкретное бизнес-решение, это поведение может быть переопределено. Даже если сеть работает, задержки передачи могут привести к рассогласованию узлов. Часто товары хранятся в покупательской корзине, но помещение товара в корзину не блокирует его для других покупателей. Если Фриц нажмет кнопку оформления заказа раньше Фредди, то, когда Фредди попытается оформить заказ, возникнет проблема. Это можно легко объяснить покупателю: он опоздал. Но что, если оба покупателя нажмут кнопку строго одновременно и будут оформлены обе продажи?

ACID и BASE

Иногда проводится параллель между принципами BASE (*англ.* base — щелочь) и ACID (*англ.* acid — кислота) из области химии: кислота представляет собой жидкость с низким значением pH. Напротив, для щелочи характерно высокое значение pH. Мы не будем углубляться в химические подробности, но на рис. 6.6 изображено мнемоническое правило для читателей, знакомых с химическими эквивалентами кислоты и щелочи.

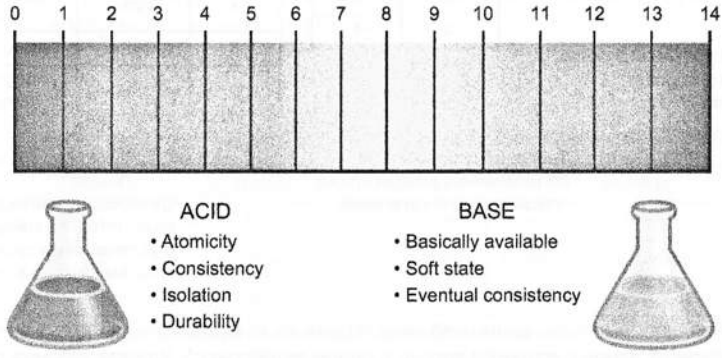


Рис. 6.6. ACID и BASE: традиционные реляционные базы данных и большинство баз данных NoSQL. Значения pH менее 7 характерны для кислот, значения выше 7 — для щелочей. На этой шкале обычная вода занимает область значений от 6,5 до 8,5

6.1.4. Типы баз данных NoSQL

Как упоминалось ранее, существует четыре основных типа баз NoSQL: хранилища «ключ—значение», хранилища документов, столбцово-ориентированные базы данных и графовые базы данных. Каждый тип решает проблему, которая не может быть нормально решена с использованием реляционных баз данных. Практические реализации часто представляют собой их комбинации. Например, OrientDB является *мультимодельной базой данных*, объединяющей разные типы NoSQL. OrientDB представляет собой графовую базу данных, в которой каждый узел является документом.

Прежде чем рассматривать разные базы данных NoSQL, обратимся к реляционным базам данных, чтобы у вас была основа для сравнения. При моделировании данных применяются разные методы. Реляционные базы данных обычно стремятся к нормализации (т. е. к тому, чтобы каждый фрагмент данных хранился только в одном экземпляре). Нормализация определяет их структурную конфигурацию. Если, например, вы захотите сохранить данные о человеке и его увлечениях, для этого можно воспользоваться двумя таблицами: в одной хранятся люди, а в другой увлечения. Как видно из рис. 6.7, для связывания людей с увлечениями необходима дополнительная таблица, так как эта связь относится к типу «многие ко многим»: у одного человека может быть несколько увлечений, а у каждого увлечения может быть много поклонников.

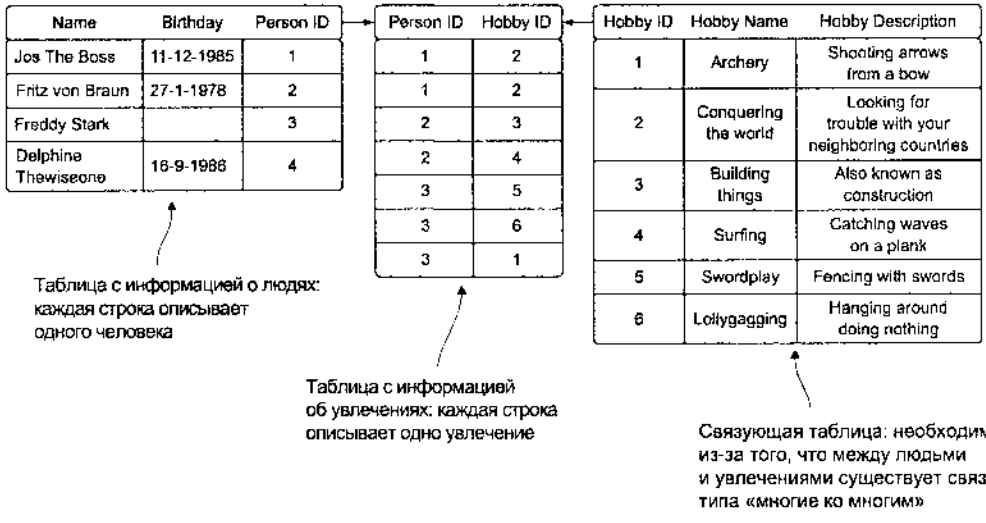


Рис. 6.7. Реляционные базы данных обычно стремятся к нормализации (т. е. к тому, чтобы каждый фрагмент данных хранился только в одном экземпляре). Каждая таблица содержит уникальные идентификаторы (первичные ключи), которые используются для моделирования отношений (relations) между сущностями (таблицами), отсюда и термин «реляционный»

Полномасштабная реляционная база данных может состоять из многих сущностей и связанных таблиц. А теперь, когда вам есть с чем сравнивать NoSQL, рассмотрим разные типы баз данных.

Столбцово-ориентированные базы данных

Традиционные реляционные базы данных являются строково-ориентированными: каждая строка имеет идентификатор, а все поля строки хранятся в таблице вместе. К примеру, допустим, что дополнительная таблица об увлечениях отсутствует, и у вас есть только одна таблица с описаниями людей (рис. 6.8). Обратите внимание: в этом случае возникает небольшая денормализация, потому что увлечения могут повторяться. Если информация об увлечениях не критична для вашего сценария использования, добавление списка увлечений в отдельный столбец Hobbies может оказаться приемлемым решением. Но если информация недостаточно важна для отдельной таблицы, то стоит ли ее хранить вообще?

Row ID	Name	Birthday	Hobbies
1	Jos The Boss	11-12-1985	Archery, conquering the world
2	Fritz von Braun	27-1-1978	Building things, surfing
3	Freddy Stark		Swordplay, lollygagging, archery
4	Delphine Thewiseone	16-9-1986	

Рис. 6.8. Строково-ориентированная структура базы данных. Каждая сущность (человек в данном случае) представляется одной строкой данных, состоящей из нескольких столбцов

Каждый раз, когда вы что-то ищете в строково-ориентированной базе данных, сканируются все строки — независимо от того, какие столбцы вам нужны. Допустим, нужно получить список дней рождения, приходящихся на сентябрь. База данных сканирует таблицу сверху вниз и слева направо, как показано на рис. 6.9, и в конечном итоге возвращает список дней рождения.

Row ID	Name	Birthday	Hobbies
1	Jos The Boss	11-12-1985	Archery, conquering the world
2	Fritz von Braun	27-1-1978	Building things, surfing
3	Freddy Stark		Swordplay, lollygagging, archery
4	Delphine Thewiseone	16-9-1986	

Рис. 6.9. Строково-ориентированный поиск: записи перебираются от начала к концу, и для каждой записи все столбцы читаются в память

Индексирование данных по некоторым столбцам может существенно повысить скорость поиска, но индексирование каждого столбца приводит к дополнительным затратам ресурсов, а база данных по-прежнему сканирует все столбцы.

В столбцовых базах данных все столбцы хранятся по отдельности, что приводит к ускорению сканирования при небольшом количестве задействованных столбцов (рис. 6.10).

Name	Row ID
Jos The Boss	1
Fritz von Braun	2
Freddy Stark	3
Delphine Thewiseone	4

Birthday	Row ID
11-12-1985	1
27-1-1978	2
16-9-1986	4

Hobbies	Row ID
Archery	1, 3
Conquering the world	1
Building things	2
Surfing	2
Swordplay	3
Lollygagging	3

Рис. 6.10. В столбцово-ориентированных базах данных все столбцы хранятся по отдельности с указанием соответствующих номеров строк. Каждая сущность (человек в данном случае) делится на несколько таблиц

Эта структура очень похожа на строково-ориентированную базу данных с индексированием по каждому столбцу. Индекс базы данных представляет собой структуру данных, обеспечивающую быстрый поиск за счет дополнительных затрат памяти и дополнительных операций записи (обновление индекса). Индекс связывает номер строки с данными, тогда как столбцовая база данных связывает данные с номерами строк; такое решение существенно упрощает подсчет (например, вы можете легко определить, сколько людей увлекается стрельбой из лука). Раздельное хранение столбцов также позволяет оптимизировать сжатие, потому что в каждой таблице хранится только один тип данных.

В каких случаях следует применять строково-ориентированную базу данных, а в каких выбирается столбцово-ориентированная база данных? В столбцово-ориентированной базе данных добавление новых столбцов не создает проблем, потому что они не влияют на существующие столбцы. С другой стороны, добавление целой записи требует внесения изменений во все таблицы. По этой причине строково-ориентированные базы данных предпочтительнее столбцово-ориентированных для оперативной обработки транзакций (OLTP, Online Transaction Processing), потому что она подразумевает частое добавление или изменение записей. Столбцово-ориентированные базы данных отлично проявляют себя в области аналитики и построения отчетов: суммировании и подсчете значений. Строково-ориентированные базы данных часто выбираются для обработки транзакций (например, продаж). Пакетная обработка данных по ночам приводит столбцово-ориентированную базу данных в актуальное состояние, с возможностью молниеносного поиска и агре-

гирования с использованием алгоритмов MapReduce для построения отчетов. В семейство столбцовых баз данных входят Apache HBase, Facebook Cassandra, Hypertable и Google BigTable.

Хранилища «ключ—значение»

Хранилища «ключ—значение» относятся к самой простой разновидности баз данных NoSQL. Как следует из названия, они представляют собой совокупности пар «ключ—значение» (рис. 6.11); благодаря своей простоте они обладают лучшей масштабируемостью среди всех разновидностей баз данных NoSQL, что позволяет хранить в них огромные объемы информации.

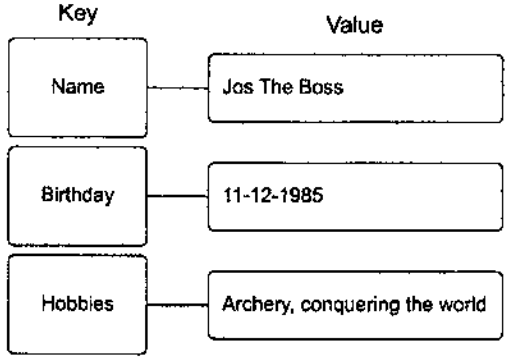


Рис. 6.11. В хранилищах «ключ—значение» вся информация состоит из пар «ключ—значение»

Значением в хранилище «ключ—значение» может быть все, что угодно: строка, число и даже другой набор пар «ключ—значение», инкапсулированный в объекте. На рис. 6.12 показана чуть более сложная структура «ключ—значение». Примеры хранилищ «ключ—значение»: Redis, Voldemort, Riak и Amazon Dynamo.

```
{"internal data":[{"entities":[
    {"customer":[
        {"id":1,"name":"Freddy"},
        {"id":2,"name":"Fritz"}
    ]},
    {"legal entities":[
        {"id":1,"company":"Maiton"}
    ]}]
},{"Products":[
    {"furniture":[
        {"id":1,"name":"Octopus Table","stock":1}
    ]
}]}]}
```

Рис. 6.12. Вложенная структура «ключ—значение»

Хранилища документов

Хранилища документов чуть превосходят по сложности хранилища «ключ—значение»: хранилище документов предполагает определенную структуру документа, которая задается схемой. Из всех типов баз данных NoSQL хранилища документов выглядят наиболее естественно, потому что они предназначены для хранения повседневных документов в том виде, в каком они существуют, и позволяют выполнять сложные запросы и вычисления с этой — нередко уже агрегированной — формой данных. Способ хранения информации в реляционной базе данных выглядит разумно с точки зрения нормализации: все данные должны храниться только в одном экземпляре и должны связываться через внешние ключи. Хранилища документов не заботятся о нормализации, если структура данных имеет смысл. Реляционная модель данных не всегда хорошо подходит для некоторых бизнес-моделей. Например, газеты и журналы содержат статьи. Чтобы сохранить статью в реляционной базе данных, необходимо сначала ее «нарезать»: текст помещается в одну таблицу, автор и вся информация о нем — в другую, комментарии к статье, опубликованной на сайте, — в третью и т. д. Как видно из рис. 6.13, газетная статья также может храниться как единая сущность; такой подход снижает когнитивную нагрузку при работе с данными у людей, постоянно работающих со статьями. Примеры хранилищ документов — MongoDB и CouchDB.

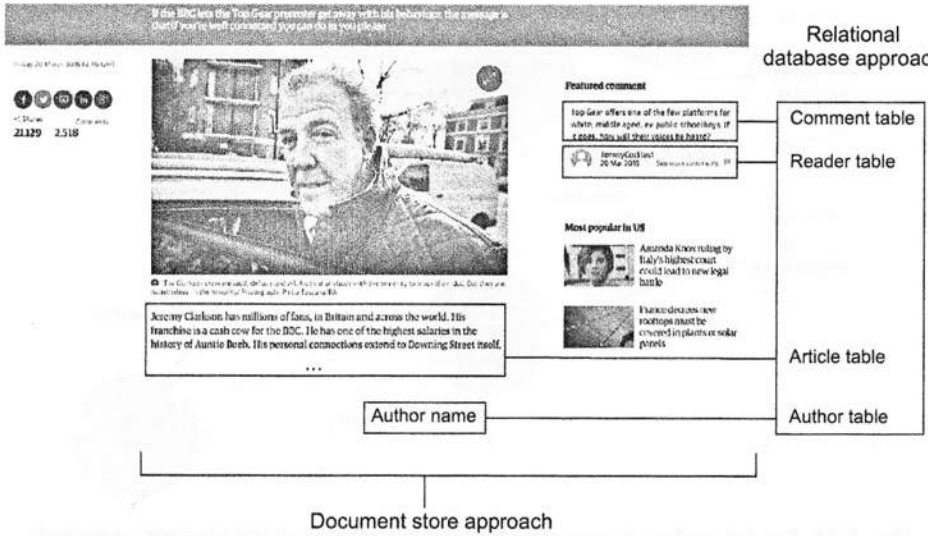
Графовые базы данных

Последняя крупная разновидность баз данных NoSQL одновременно является и самой сложной, ориентированной на эффективное представление отношений между сущностями. Для данных с высокой связностью (социальных сетей, библиографических ссылок в научных статьях и т. д.) графовые базы данных становятся идеальным решением. Графовые (или сетевые) данные состоят из компонентов двух основных типов:

- *Узел* — собственно сущность. В социальных сетях это может быть человек.
- *Ребро* — отношение между двумя сущностями. Отношение представляется соединительной линией и обладает собственными свойствами. Например, ребро может иметь направление: скажем, стрелка указывает, кто является начальником, а кто подчиненным.

При достаточно большом количестве сущностей и ребер графы могут стать невероятно сложными. На рис. 6.14 эта сложность уже проявляется при ограниченном количестве сущностей. Графовые базы данных, такие как Neo4j, также претендуют на соблюдение принципов ACID, тогда как хранилища документов и хранилища «ключ—значение» ограничиваются принципами BASE.

Возможности бесконечны, а поскольку плотность связей в окружающем мире непрерывно растет, графовые базы данных с большой вероятностью потеснят другие типы, включая все еще преобладающие реляционные базы данных. Рейтинг самых популярных баз данных и данные о динамике их развития можно найти по адресу *http://db-engines.com/en/ranking*.




```
{
    "articles": [
        {
            "title": "title of the article",
            "articleID": 1,
            "body": "body of the article",
            "author": "Isaac Asimov",
            "comments": [
                {
                    "username": "Fritz"
                    "join date": "1/4/2014"
                    "commentid": 1,
                    "body": "this is a great article",
                    "replies": [
                        {
                            "username": "Freddy",
                            "join date": "11/12/2013",
                            "commentid": 2,
                            "body": "seriously? it's rubbish"
                        }
                    ]
                },
                {
                    "username": "Stark",
                    "join date": "19/06/2011",
                    "commentid": 3,
                    "body": "I don't agree with the conclusion"
                }
            ]
        }
    ]
}
```

Рис. 6.13. В хранилищах документов документы хранятся как единое целое, тогда как в РСУБД статья «нарезается» и сохраняется в нескольких разных таблицах. Этот пример взят с веб-сайта Guardian

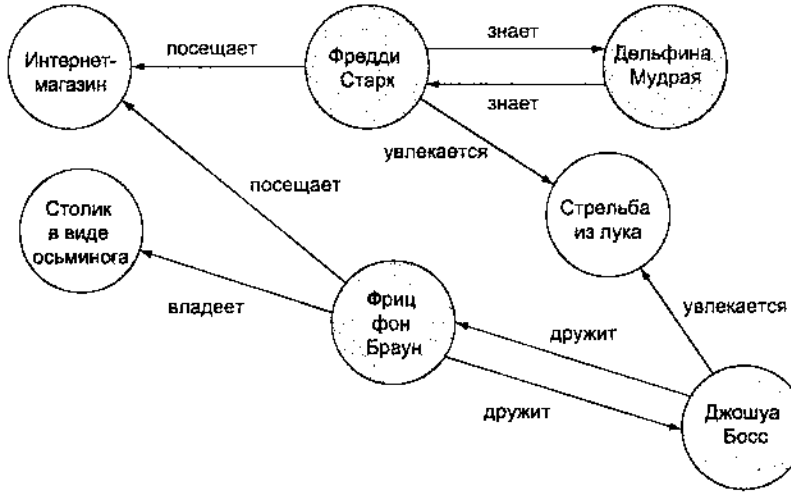


Рис. 6.14. Пример графовых данных с четырьмя типами сущностей (человек, увлечение, компания, мебель) и их отношениями без дополнительной информации об узлах или ребрах

257 systems in ranking, March 2015

Rank Mar 2015	Rank Feb 2015	Rank Mar 2014	DBMS	Database Model	Score Mar 2015	Score Feb 2015	Score Mar 2014
1.	1.	1.	Oracle	Relational DBMS	1469.09	+29.37	-22.71
2.	2.	2.	MySQL	Relational DBMS	1261.09	-11.36	-29.12
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1164.80	-12.68	-40.48
4.	4.	↑ 5.	MongoDB	Document store	275.01	+7.77	+75.03
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	264.44	+2.10	+29.38
6.	6.	6.	DB2	Relational DBMS	198.85	-3.57	+11.52
7.	7.	7.	Microsoft Access	Relational DBMS	141.69	+1.15	-4.79
8.	8.	↑ 10.	Cassandra	Wide column store	107.31	+0.23	+29.22
9.	9.	↓ 8.	SQLite	Relational DBMS	101.71	+2.14	+8.73
10.	10.	↑ 13.	Redis	Key-value store	97.05	-2.16	+43.59
11.	11.	↓ 9.	SAP Adaptive Server	Relational DBMS	85.37	-0.97	+3.81
12.	12.	12.	Solr	Search engine	81.88	+0.40	+20.74
13.	13.	↓ 11.	Teradata	Relational DBMS	72.78	+3.33	+10.15
14.	14.	↑ 16.	HBase	Wide column store	60.73	+3.59	+25.59
15.	↑ 16.	↑ 19.	Elasticsearch	Search engine	58.92	+6.09	+32.75

Рис. 6.15. 15 самых популярных баз данных (по данным DB-Engines.com за март 2015 г.)

Из рис. 6.15 видно, что на момент написания книги реляционные базы данных все еще занимали 9 позиций из 15, и даже с появлением баз данных NewSQL их еще рано списывать со счетов. Neo4j, самая популярная графовая база данных, занимает 23-ю позицию рейтинга, а Titan находится в позиции 53.

После теоретического знакомства со всеми разновидностями баз данных NoSQL пора опробовать одну из них в деле.

6.2. Учебный пример: Диагностика болезней

Такое случалось со многими из нас: внезапно вы чувствуете недомогание, и первое, что вы делаете, — ищете в Google, на какую болезнь это похоже. А потом вы решаете, что стоит обратиться к врачу. Поиск в Интернете работает неплохо, но специализированная база данных подошла бы лучше. Такие базы данных существуют и обладают достаточно впечатляющей функциональностью; по сути, это почти виртуальная версия доктора Хауса, гениального диагноста из телесериала. Однако такие базы данных строятся на защищенной информации и далеко не всегда доступны всем желающим. Кроме того, хотя эти виртуальные врачи доступны для больших фармацевтических компаний и современных больниц, многие врачи общей практики вынуждены ограничиваться книгами. Мало того что такая асимметрия информации и ресурсов прискорбна и опасна — без нее можно было бы легко обойтись. Если бы существовала простая поисковая система диагностики болезней, которая могла бы использоваться любым врачом в мире, это позволило бы избежать многих медицинских ошибок.

В этом учебном примере вы узнаете, как построить такую поисковую систему, пусть и для малой части медицинских данных, находящихся в свободном доступе. Для хранения данных будет использоваться современная база данных NoSQL, которая называется Elasticsearch, а процесс data science преобразует эти данные в ресурс с поддержкой быстрого и удобного поиска. Вот как работает этот процесс:

1. *Назначение цели исследования.*
2. *Сбор данных* — данные будут взяты из Википедии. Существуют и другие источники, но для демонстрационных целей сойдет и этот.
3. *Подготовка данных* — может оказаться, что данные Википедии не идеальны в своем текущем формате. Мы применим некоторые приемы для изменения ситуации.
4. *Исследование данных* — у нашего учебного примера есть одна особенность: этап 4 процесса data science также является предполагаемым конечным результатом; данные должны быть удобными для исследования.
5. *Моделирование данных* — в этой главе реальное моделирование данных не применяется. Матрицы DTM («документ—термин»), используемые при поиске, часто становятся отправной точкой для расширенного моделирования. Здесь эта тема подробно не рассматривается.
6. *Отображение результатов* — чтобы пользователь мог проводить поиск по данным, нам понадобится пользовательский интерфейс, например сайт, на котором пользователь может запрашивать и читать информацию о болезнях. В этой

главе мы не будем заходить настолько далеко, чтобы заниматься построением реального интерфейса. Вторичная цель — профилирование категорий болезней по ключевым словам; мы достигнем этой цели процесса data science, потому что данные будут представлены в формате облака ключевых слов наподобие изображенного на рис. 6.16.

Рис. 6.16. Облако ключевых слов по теме «диабет»

Чтобы воспроизвести этот код на своем компьютере, вам понадобится следующее:

- Сеанс Python с установленными библиотеками elasticsearch-py и Wikipedia (pip install elasticsearch и pip install wikipedia).
- Локальный экземпляр Elasticsearch; за инструкциями по установке обращайтесь к приложению А.
- Библиотека IPython.

ПРИМЕЧАНИЕ

Код этой главы можно загрузить с веб-сайта Manning по адресу https://manning.com/books/introducing-data-science в формате IPython.

ELASTICSEARCH: ПОИСКОВАЯ СИСТЕМА/БАЗА ДАННЫХ NOSQL С ОТКРЫТЫМ КОДОМ

Для решения поставленной задачи — диагностики болезней — мы воспользуемся базой данных NoSQL, которая называется Elasticsearch. Как и MongoDB, Elasticsearch представляет собой хранилище документов. Но в отличие от MongoDB, Elasticsearch также является поисковой системой. Если MongoDB прекрасно подходит для выполнения сложных вычислений и заданий MapReduce, главной целью Elasticsearch является полнотекстовый поиск. Elasticsearch выполняет базовые вычисления с индексируемыми числовыми данными, такие как суммирование, подсчет, вычисление ме-

дианы, среднего арифметического, стандартного отклонения и т. д., но по своей сути все равно остается поисковой системой.

Система Elasticsearch построена на базе Apache Lucene — поисковой системы Apache, созданной в 1999 году. Система Lucene печально известна своей сложностью в обращении и больше напоминает строительный элемент для более удобных приложений, чем самостоятельное решение. С другой стороны, поисковая система Lucene обладает чрезвычайно мощными возможностями, и в 2004 году появилась система Apache Solr, открытая для свободного использования в 2006 году. Solr (поисковая платформа корпоративного уровня, распространяемая с открытым кодом) строится на базе Apache Lucene; на сегодняшний день это самая гибкая и популярная поисковая система с открытым кодом. Solr — отличная платформа, которая определенно заслуживает внимания, если ваш проект требует применения поисковой системы. В 2010 году была выпущена система Elasticsearch, быстро набирающая популярность. И если при установке и настройке Solr могут возникнуть трудности (даже в малых проектах), работать с Elasticsearch предельно просто. К преимуществам Solr все еще относится количество возможных плагинов, расширяющих базовую функциональность системы, но Elasticsearch быстро сокращает разрыв, и сегодня эти системы обладают функциональностью сравнимого уровня.

6.2.1. Этап 1: Назначение цели исследования

Сможете ли вы к концу этой главы диагностировать болезнь, располагая только домашним компьютером, бесплатным ПО и имеющимися данными? Знать, что вы хотите сделать и как это сделать, — в этом заключается суть первого этапа процесса data science (рис. 6.17).

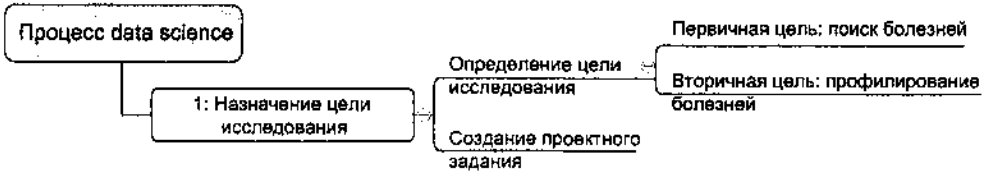


Рис. 6.17. Этап 1 процесса data science: назначение цели исследования

- Первичная цель: создание поисковой системы, которая помогала бы врачам в диагностике болезней.
- Вторичная цель: профилирование болезни — какие ключевые слова отличают ее от других болезней?

Вторичная цель полезна для образовательных задач или для формирования входных данных для нетривиальных применений (например, выявления распространяющихся эпидемий по информации из социальных сетей). А теперь, когда мы разобрались с целью исследования и определили план действий, можно переходить к сбору данных.

6.2.2. Этапы 2 и 3: Сбор и подготовка данных

Сбор данных и подготовка данных — два разных этапа в процессе data science. И хотя в нашем учебном примере такое положение дел сохраняется, оба этапа будут рассмотрены в одном разделе. Это избавит вас от необходимости создавать локальное промежуточное хранилище и позволит заняться подготовкой данных сразу же после их получения. Посмотрим, как выглядит текущая стадия процесса data science (рис. 6.18).

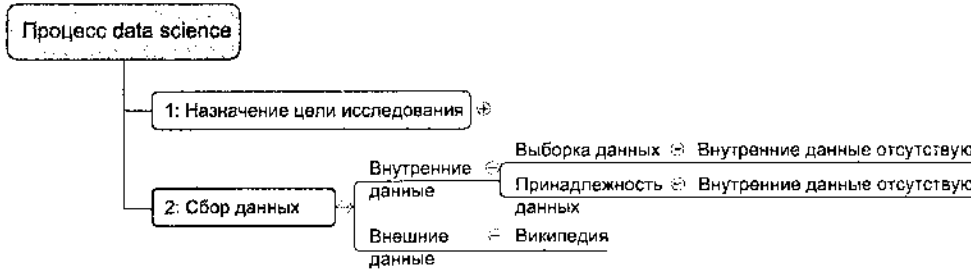


Рис. 6.18. Этап 2 процесса data science: сбор данных. В данном случае внутренние данные отсутствуют, вся информация будет читаться из Википедии

Как видно из рис. 6.18, есть всего два источника информации: внутренние и внешние данные.

- *Внутренние данные* — под рукой нет никакой информации о болезнях. Если вы сейчас работаете в фармацевтической компании или в больнице, возможно, вам повезло больше.
- *Внешние данные* — в нашей ситуации остаются только внешние данные. Есть несколько возможностей, но мы воспользуемся Википедией.

Данные, прочитанные из Википедии, следует сохранить в локальном индексе Elasticsearch, но сначала данные необходимо подготовить. После того как данные попадут в индекс Elasticsearch, изменить их уже не удастся, вы сможете только обращаться к ним с запросами.

Краткое описание подготовки данных представлено на рис. 6.19.

Как видно из рис. 6.19, мы должны рассмотреть три разные категории подготовки данных:

- *Преобразование данных* — пока особой надобности в преобразовании данных нет; требуется вести поиск данных в их исходном виде. Однако в тексте необходимо выделить название страницы, название болезни и тело страницы. Эти элементы практически обязательны для интерпретации результатов поиска.
- *Объединение данных* — все данные в этом случае взяты из одного источника, поэтому реально необходимости в объединении нет. Одно из возможных расширений этого упражнения — получение данных из другого источника и сопоставление информации о болезнях. Решить эту задачу будет непросто, так как уникальные идентификаторы болезней отсутствуют, а их названия часто различаются в зависимости от источника.

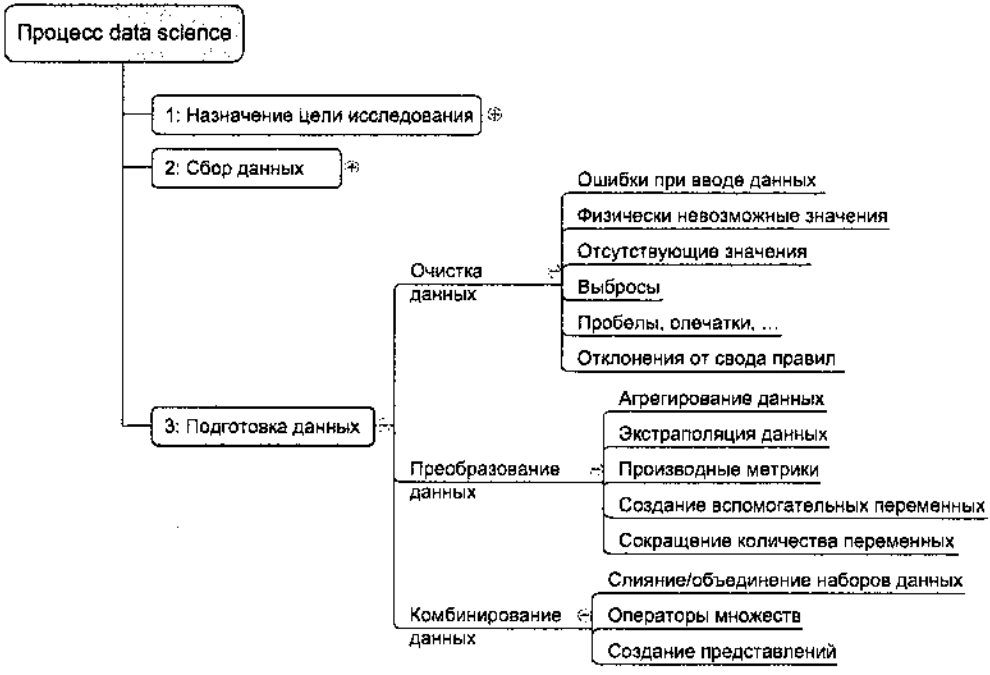


Рис. 6.19. Этап 3 процесса data science: подготовка данных

Очистка данных может производиться только на двух стадиях: при использовании программы Python, связывающей Elasticsearch с Википедией, и во время работы внутренней системы индексирования Elasticsearch:

- *Python* — здесь вы определяете, какие данные разрешено будет хранить в хранилище документов. Тем не менее ни очистка, ни преобразование данных на этой стадии производиться не будет, потому что Elasticsearch лучше справится с этой задачей при меньшем объеме работы.

- *Elasticsearch* — Elasticsearch в своей внутренней работе осуществляет обработку данных (при создании индекса.) Вы можете управлять этим процессом, и позднее в этой главе мы этим займемся.

Теперь, когда вы примерно представляете, что будет дальше, примемся за работу. Если вы выполнили инструкции из приложения, то у вас сейчас имеется работоспособный локальный экземпляр Elasticsearch. Сначала выполнятся сбор данных, т. е. информации о различных болезнях. Такие данные можно получить разными способами. Например, вы можете обратиться к компаниям за их данными, получить информацию из Freebase или других открытых и бесплатных источников. Сбор данных часто оказывается непростым делом, но в этом примере данные будут взяты из Википедии. (По иронии судьбы, поиск на сайте Википедии осуществляется с использованием Elasticsearch. Раньше Википедия использовала собственную систему, построенную на базе Apache Lucene, но со временем ее сопровождение стало создавать слишком много трудностей, и в январе 2014 года Википедия перешла на Elasticsearch.)

В Википедии имеется страница со списком болезней (рис. 6.20). С этой страницы можно взять данные из алфавитных списков.

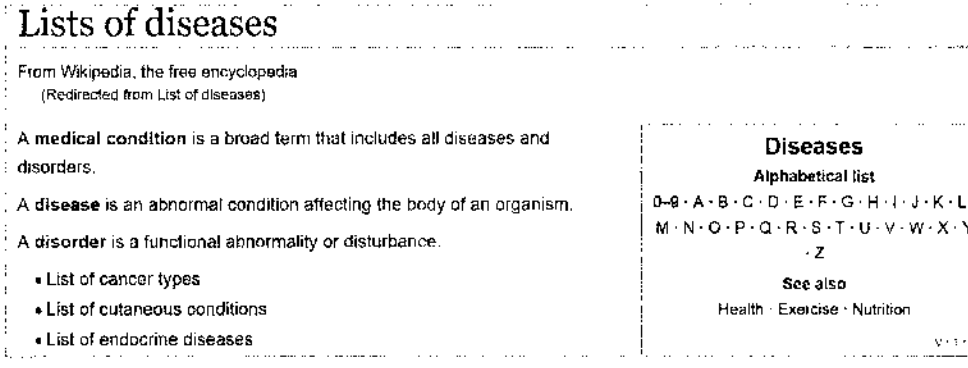


Рис. 6.20. Страница со списком болезней в Википедии станет отправной точкой для получения данных

Вы знаете, какие данные вам нужны; остается их взять. Можно загрузить весь дамп данных Википедии по адресу *http://meta.wikimedia.org/wiki/Data_dump_torrents#enwiki*.

Конечно, если вы решите проиндексировать всю Википедию, индекс займет около 40 Гбайт. При желании вы можете использовать это решение, но ради экономии дискового пространства и пропускной способности канала в этой книге мы ограничимся только теми данными, которые будут реально использоваться. Другой вариант — выборка нужных страниц. Вы можете написать программу, которая делает примерно то же, что и поисковые боты Google: она перемещается по страницам и читает всю сгенерированную разметку HTML. Вы получите необходимую

информацию, но вместе с ней будет получена разметка HTML, поэтому данные перед индексированием нужно будет очистить. И еще нужно учитывать, что сайты не особенно любят ботов, которые читают их страницы: это порождает излишний уровень трафика, и при большом количестве ботов работа сервера HTTP может быть парализована. Кроме того, одновременная отправка миллиардов запросов является одним из способов проведения атак отказа в обслуживании (DoS, Denial of Service). Если вам понадобится загрузить содержимое сайта, запрограммируйте задержку между запросами страницы. В этом случае программа сбора данных будет более точно имитировать поведение обычного посетителя и вы не выведете из строя сервер.

К счастью, создатели Википедии достаточно умны, чтобы понимать, что именно это произойдет, когда вся эта информация открыта для всех желающих. Они предоставили специальный API для безопасного получения информации. Подробнее о нем можно прочитать по адресу *http://www.mediawiki.org/wiki/API:Main_page*.

Мы будем получать данные через API. И разумеется, в Python уже есть готовая библиотека для решения этой задачи. Вообще говоря, таких библиотек несколько, но для наших целей хватит самой простой: Wikipedia.

Активизируйте виртуальную среду Python и установите все библиотеки, которые вам понадобятся в оставшейся части книги:

```
pip install wikipedia
pip install Elasticsearch
```

Библиотека Wikipedia будет использоваться для подключения к Википедии. Elasticsearch — основная библиотека для работы с Elasticsearch в языке Python; с ее помощью вы будете взаимодействовать с базой данных.

Откройте свой любимый интерпретатор Python и импортируйте необходимые библиотеки:

```
from elasticsearch import Elasticsearch
import wikipedia
```

Вы будете читать данные через API Википедии и в то же время проводить индексирование в локальном экземпляре Elasticsearch, поэтому сначала необходимо подготовиться к получению данных.

```
client = Elasticsearch()//
indexName = "medical"
client.indices.create(index=indexName)
```

Клиент Elasticsearch используется для взаимодействия с базой данных.

Имя индекса.

Создание индекса.

Прежде всего необходимо создать клиента (client). При вызове `Elasticsearch()` для инициализации можно указать конкретный адрес, но по умолчанию используется адрес `localhost:9200`. Таким образом, вызовы `Elasticsearch()` и `Elasticsearch('localhost:9200')` эквивалентны: клиент связывается с локальным узлом Elasticsearch. Затем создается индекс с именем `"medical"`. Если все пройдет хорошо, выводится ответ `"acknowledged:true"`, как показано на рис. 6.21.

```
In [7]: client = Elasticsearch() #elasticsearch client used to communicate with the database
        indexName = "medical" #the index name
        #client.indices.delete(index=indexName) #delete an index
        client.indices.create(index=indexName) #create an index

Out[7]: {u'acknowledged': True}
```

Рис. 6.21. Создание индекса Elasticsearch с использованием Python-Elasticsearch

Утверждается, что система Elasticsearch является *бессхемной* (schema-less); это означает, что Elasticsearch можно использовать без определения схемы базы данных и без указания информации о том, с какими данными она будет работать. И хотя в простых случаях это действительно так, в долгосрочной перспективе без схемы не обойтись. Поэтому мы создадим эту схему так, как показано в листинге 6.1.

Листинг 6.1. Добавление отображения в тип документа

```
diseaseMapping = {
        'properties': {
            'name': {'type': 'string'},
            'title': {'type': 'string'},
            'fulltext': {'type': 'string'}
        }
    }
client.indices.put_mapping(index=indexName,
doc_type='diseases',body=diseaseMapping )
```

Определение отображения и связь его с типом документа.

Тип документа "diseases" обновляется с включением нового отображения.

Этот фрагмент сообщает Elasticsearch, что ваш индекс будет содержать тип документа с именем «disease»; вы указываете тип каждого из полей. Документ содержит три поля: `name`, `title` и `fulltext`. Все три поля имеют тип `string`. Если бы отображение не было указано, то система Elasticsearch постаралась бы угадать их типы по первому полученному элементу. Если поле не распознается как относящееся к типу `boolean`, `double`, `float`, `long`, `integer` или `date`, оно интерпретируется как `string`. В данном случае вручную задавать отображение не обязательно.

Перейдем к работе с Википедией. Первое, что необходимо сделать, — это получить страницу со списком болезней, потому что она станет отправной точкой для дальнейших исследований:

```
dl = wikipedia.page("Lists_of_diseases")
```

Первая страница получена, но нас сейчас больше интересуют ссылки на болезни:

```
dl.links
```

Страница со списком болезней содержит больше ссылок, чем нам нужно. На рис. 6.22 показаны алфавитные списки, начиная с шестнадцатой ссылки:

```
dl = wikipedia.page("Lists_of_diseases")
dl.links
```

```
In [9]: dl = wikipedia.page("Lists_of_diseases")
        dl.links

Out[9]: [u'Airborne disease',
         u'Contagious disease',
         u'Cryptogenic disease',
         u'Disease',
         u'Disseminated disease',
         u'Endocrine disease',
         u'Environmental disease',
         u'Eye disease',
         u'Lifestyle disease',
         u'List of abbreviations for diseases and disorders',
         u'List of autism-related topics',
         u'List of basic exercise topics',
         u'List of cancer types',
         u'List of communication disorders',
         u'List of cutaneous conditions',
         u'List of diseases (0\u20139)',
         u'List of diseases (A)',
         u'List of diseases (B)',
```

Рис. 6.22. Ссылки на странице Википедии со списком болезней. Страница содержит больше ссылок, чем нам понадобится

Эта страница содержит большой массив ссылок, но нас интересуют только алфавитные списки болезней:

```
diseaseListArray = []
for link in dl.links[15:42]:
    try:
        diseaseListArray.append(wikipedia.page(link))
    except Exception,e:
        print str(e)
```

Вероятно, вы заметили, что подмножество жестко закодировано, потому что мы знаем, что ссылки располагаются с 16-й по 43-ю позицию в массиве. Если Википедия вдруг добавит хотя бы одну ссылку перед интересующими вас, результаты окажутся смещенными. Правильнее было бы воспользоваться регулярным выражением для решения этой задачи. Для исследования жестко фиксированные

номера элементов допустимы, но если вы освоили регулярные выражения во всех тонкостях или если вы собираетесь преобразовать этот код в пакетное задание, лучше применить регулярные выражения. Дополнительную информацию о них можно найти по адресу *https://docs.python.org/2/howto/regex.html*.

Одна из возможных версий с регулярным выражением могла бы выглядеть так:

```
diseaseListArray = []
check = re.compile("List of diseases*")
for link in dl.links:
    if check.match(link):
        try:
            diseaseListArray.append(wikipedia.page(link))
        except Exception,e:
            print str(e)
```

На рис. 6.23 показаны первые элементы того, что вас интересует: собственно болезни:

```
diseaseListArray[0].links
```

```
In [16]: diseaseListArray

Out[16]: [<WikipediaPage 'List of diseases (0-9)'>,
          <WikipediaPage 'List of diseases (A)'>,
          <WikipediaPage 'List of diseases (B)'>,
          <WikipediaPage 'List of diseases (C)'>,
          <WikipediaPage 'List of diseases (D)'>,
          <WikipediaPage 'List of diseases (E)'>,
          <WikipediaPage 'List of diseases (F)'>,
          <WikipediaPage 'List of diseases (G)'>,
          <WikipediaPage 'List of diseases (H)'>,
```

Рис. 6.23. Первый список болезней в Википедии

Теперь болезни нужно проиндексировать. После индексирования шаги сбора и подготовки данных фактически завершены, как видно из листинга 6.2.

Так как каждая из страниц со списком будет содержать ссылки, не интересующие вас, сначала нужно проверить, является ли элемент заболеванием. Мы проверяем символ, с которого начинается название болезни, а также дополнительно исключаем ссылки, начинающиеся с «list», для страницы со списком болезней на «L». Это весьма наивная проверка, но потери от нескольких лишних элементов незначительны, так как поисковые алгоритмы исключат нерелевантные результаты после выдачи запросов. Для каждой болезни индексируется ее название и полный текст страницы. Название также индексируется как идентификатор болезни; это полезно для некоторых расширенных возможностей Elasticsearch, а также для ускорения

Листинг 6.2. Индексирование болезней из Википедии

```
checkList = [["0","1","2","3","4","5","6","7","8","9"],
["A"],["B"],["C"],["D"],["E"],["F"],["G"],["H"],
["I"],["J"],["K"],["L"],["M"],["N"],["O"],["P"],
["Q"],["R"],["S"],["T"],["U"],["V"],["W"],["X"],["Y"],["Z"]]
docType = 'diseases'
for diseaselistNumber, diseaselist in enumerate(diseaseListArray):
    for disease in diseaselist.links:
        try:
            if disease[0] in checkList[diseaselistNumber]
and disease[0:3] !="List":
                currentPage = wikipedia.page(disease)
                client.index(index=indexName,
doc_type=docType,id = disease, body={"name": disease,
"title":currentPage.title ,
"fulltext":currentPage.content})
        except Exception,e:
            print str(e)
```

В массиве checkList хранятся допустимые первые символы. Если болезнь не отвечает критерию, она пропускается.

Индексируемый тип документа.

Сначала проверяем, является ли болезнью, а затем индексируем.

Перебор списков болезней.

Перебор списков ссылок для каждого списка болезней.

поиска в браузере. Например, попробуйте ввести в своем браузере следующий адрес:*http://localhost:9200/medical/diseases/11%20beta%20hydroxylase%20deficiency*. Название индексируется отдельно; в большинстве случаев имя ссылки и заголовок страницы будут идентичны, а иногда заголовок будет содержать альтернативное имя страницы.

После того как система проиндексирует хотя бы несколько болезней, вы сможете использовать URI Elasticsearch для простого поиска. На рис. 6.24 представлен пример полнотекстового поиска по слову «headache» (головная боль). Поиск можно проводить одновременно с индексированием; Elasticsearch может обновлять индекс и возвращать результаты запросов одновременно.

Если вы не обращаетесь с запросами к индексу, вы все равно можете получить результаты, ничего не зная об индексе. Поиск по адресу *http://localhost:9200/medical/diseases/_search* вернет первые пять результатов. Чтобы получить более структурированное представление данных, запросите отображение этого типа документа по адресу *http://localhost:9200/medical/diseases/_mapping?pretty*. С аргументом *pretty*

разметка JSON возвращается в более удобном формате (рис. 6.25.) Отображение выглядит так, как вы его определили: все поля имеют тип `string`.

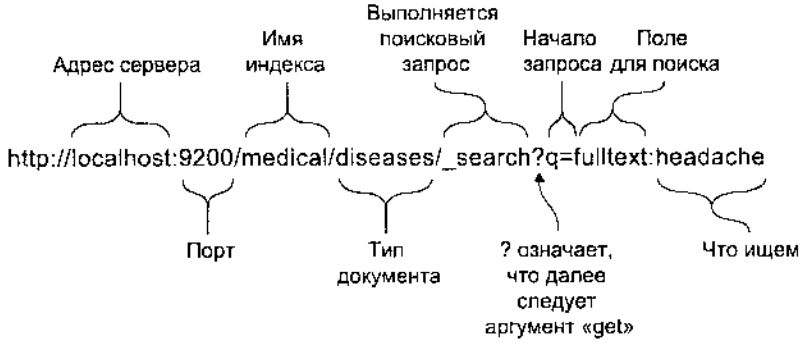
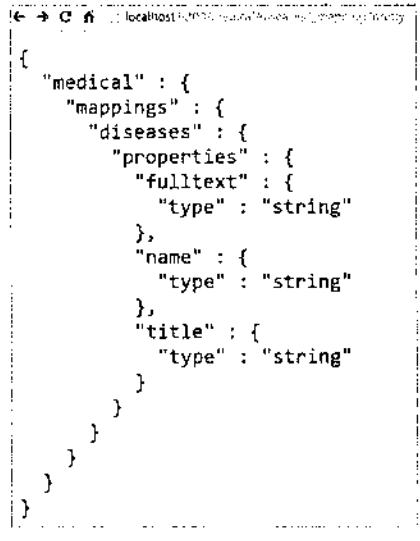


Рис. 6.24. Структура URL-адреса Elasticsearch



```
{
  "medical" : {
    "mappings" : {
      "diseases" : {
        "properties" : {
          "fulltext" : {
            "type" : "string"
          },
          "name" : {
            "type" : "string"
          },
          "title" : {
            "type" : "string"
          }
        }
      }
    }
  }
}
```

Рис. 6.25. Просмотр отображения типа документа diseases по URL-адресу Elasticsearch

Безусловно, URL-адреса Elasticsearch полезны, однако для ваших потребностей их недостаточно. Наша задача - диагностировать болезни, а для этого необходимо отправлять Elasticsearch запросы POST через библиотеку Python.

После завершения сбора и подготовки данных можно переходить к их исследованию.

6.2.3. Этап 4: Исследование данных

Это не волчанка. К черту волчанку.

Доктор Хаус
(сериал «Доктор Хаус»)

Исследование данных является важнейшим аспектом этого примера, потому что главная цель проекта (диагностика болезней) представляет собой конкретный способ исследования данных по симптомам. На рис. 6.26 представлены некоторые методы исследования данных, но в данном случае речь идет о неграфических методах: интерпретации результатов поиска текста.

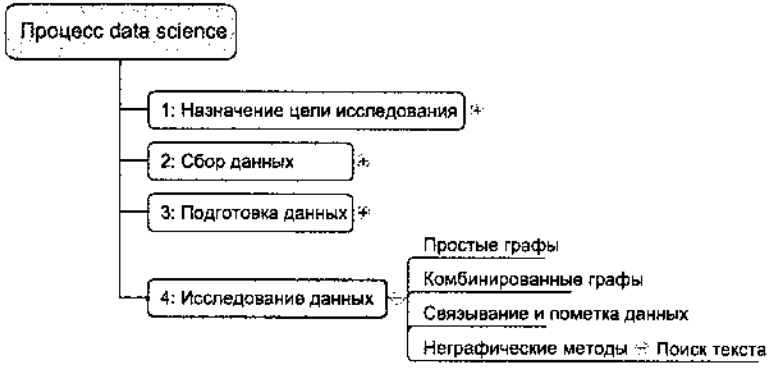


Рис. 6.26. Этап 4 процесса data science: Исследование данных

И тут наступает момент истины: удастся ли вам найти некоторые болезни, передав поисковой системе их симптомы? Для начала убедимся в том, что базовая структура подготовлена и работает. Импортируйте библиотеку Elasticsearch и определите глобальные параметры поиска:

```
from elasticsearch import Elasticsearch
client = Elasticsearch()
indexName = "medical"
docType="diseases"
searchFrom = 0
searchSize= 3
```

При поиске возвращаются только первые три результата; по умолчанию возвращаются пять.

В Elasticsearch поддерживается тщательно проработанный язык запросов JSON; при каждом поиске серверу отправляется запрос POST, на который сервер выдает

ответ в формате JSON. В общем и целом язык состоит из трех частей: запросы, фильтры и обобщения. *Запрос* получает ключевые слова поиска и обрабатывает их одним или несколькими анализаторами, прежде чем проводить поиск этих слов в индексе. Анализаторы будут более подробно рассмотрены позднее в этой главе. *Фильтр* получает ключевые слова, как и запрос, но не пытается анализировать полученную информацию; он фильтрует информацию по заданным условиям. Таким образом, фильтры обладают меньшей сложностью, но во много раз большей эффективностью, потому что они также временно сохраняются в Elasticsearch на случай повторного применения того же фильтра. *Обобщения* можно сравнить с группами SQL; они создают *гнезда* (buckets) слов, и для каждого гнезда может вычисляться актуальная статистика. У каждой из трех частей существует множество параметров и функциональных возможностей, поэтому привести сколько-нибудь подробное описание всего языка невозможно. К счастью, нет необходимости погружаться во все сложности запросов Elasticsearch. Мы воспользуемся «языком строки запроса» — способа описания запрашиваемых данных, близкого к языку поисковых запросов Google. Например, если какое-то условие в запросе должно быть обязательным, перед ним ставится плюс (+); для исключения условия используется минус (-). Вообще говоря, этот способ считается нежелательным, потому что он снижает быстродействие; поисковая система сначала должна преобразовать строку запроса в «родной» язык запросов JSON. Однако для наших целей он отлично сработает; для нескольких тысяч записей, хранящихся в вашем индексе, быстродействие никакой роли не играет. Итак, переходим к запросу данных болезней.

Первичная цель проекта: диагностика болезни по симптомам

Фраза «К черту волчанку» знакома всем поклонникам популярного телесериала «Доктор Хаус». Волчанка — аутоиммунное заболевание, при котором иммунная система организма атакует здоровые части тела. Посмотрим, какие симптомы понадобятся поисковой системе для определения волчанки.

Начнем с трех симптомов: слабость (fatigue), повышенная температура (fever) и суставные боли (joint pain). У вашего воображаемого пациента присутствуют все три симптома (и не только), поэтому мы объявим их обязательными при поиске, поставив плюс перед каждым симптомом (листинг 6.3).

В переменной `searchBody`, содержащей структуру данных JSON, указываются поля, которые вы хотите получить в возвращаемых результатах; в данном случае достаточно названия болезни. Синтаксис строки запроса используется для поиска по всем индексируемым полям: fulltext, title и name. Добавляя знак ^, можно назначить каждому полю вес. Если симптом встречается в заголовке, он в пять раз важнее вхождения в открытом тексте; если он встречается в самом названии, он считается в 10 раз более важным. Обратите внимание на то, как симптом суставной боли (joint pain) заключается в кавычки. Без кавычек слова *joint* и *pain* считались бы двумя разными ключевыми словами, а не одним выражением. Результаты показаны на рис. 6.27.

Листинг 6.3. Простая строка запроса Elasticsearch с тремя обязательными ключевыми словами

```
searchBody={
"fields":["name"],
"query":{
    "simple_query_string" : {
        "query": '+fatigue+fever+"joint pain"',
        "fields": ["fulltext","title^5","name^10"]
        }
    }
}
client.search(index=indexName,doc_type=docType, body=searchBody, from_ =
     searchFrom, size=searchSize)
```

Словарь searchBody содержит отправленную информацию поискового запроса.

В результатах должно присутствовать поле name.

Описание запроса. Здесь также возможны другие конструкции – например, обобщения (далее об этом рассказано более подробно).

Простая строка запроса – разновидность запросов, которая получает входные данные примерно в таком же формате, как у домашней страницы Google.

Поля, по которым проводится поиск. Не путайте их с полями, которые должны возвращаться в результатах поиска (вторая строка кода).

Как и в запросах Google, знак "+" обозначает обязательное условие. Заключение двух и более слов в кавычки указывает на то, что при поиске должна использоваться точно указанная форма слов.

Выполнение поиска. Переменные indexName, docType, searchFrom и searchSize были объявлены ранее: indexName = "medical", docType="diseases", searchFrom = 0, searchSize = 3.

```
{u'_shards': {u'failed': 0, u'successful': 5, u'total': 5},
 u'hits': {u'hits': [{u'_id': u'Macrophagic myofasciitis',
    u'_index': u'medical',
    u'_score': 0.014184786,
    u'_type': u'diseases',
    u'fields': {u'name': [u'Macrophagic myofasciitis']}},
   {u'_id': u'Human granulocytic ehrlichiosis',
    u'_index': u'medical',
    u'_score': 0.0072817733,
    u'_type': u'diseases',
    u'fields': {u'name': [u'Human granulocytic ehrlichiosis']}},
   {u'_id': u'Panniculitis',
    u'_index': u'medical',
    u'_score': 0.0058474476,
    u'_type': u'diseases',
    u'fields': {u'name': [u'Panniculitis']}}],
  u'max_score': 0.014184786,
  u'total': 34},
 u'timed_out': False,
 u'took': 106}
```

Волчанка не входит в число первых 3 возвращаемых болезней.

Всего найдено 34 болезни.

Рис. 6.27. Первый поиск с 34 результатами

На рис. 6.27 показаны первые 3 результата из 34 подходящих болезней. Результаты сортируются по рейтингу соответствия (переменная _score). Объяснить смысл рейтинга соответствия в двух словах непросто; в нем учитывается, насколько хорошо болезнь соответствует запросу, сколько раз было найдено ключевое слово, назначенные веса и т. д. В настоящее время волчанка (lupus) даже не входит в первую тройку результатов. К счастью, у волчанки есть еще один характерный симптом: сыпь. Она не всегда появляется на лице, но такое тоже бывает, и именно поэтому волчанка получила свое название: из-за сыпи лицо человека отдаленно напоминает волчью морду. У вашего пациента имеется сыпь, но не на лице, поэтому мы включаем сыпь (rash) в набор симптомов без упоминания лица:

```
"query": '+fatigue+fever+"joint pain"+rash',
```

Результаты нового поиска показаны на рис. 6.28.

```
{u'_shards': {u'failed': 0, u'successful': 5, u'total': 5},
 u'hits': {u'hits': [{u'_id': u'Human granulocytic ehrlichiosis',
    u'_index': u'medical',
    u'_score': 0.009982062,
    u'_type': u'diseases',
    u'fields': {u'name': [u'Human granulocytic ehrlichiosis']}},
   {u'_id': u'Lupus erythematosus',
    u'_index': u'medical',
    u'_score': 0.009000875,
    u'_type': u'diseases',
    u'fields': {u'name': [u'Lupus erythematosus']}},
   {u'_id': u'Panniculitis',
    u'_index': u'medical',
    u'_score': 0.007950994,
    u'_type': u'diseases',
    u'fields': {u'name': [u'Panniculitis']}}],
  u'max_score': 0.009982062,
  u'total': 6},
 u'timed_out': False,
 u'took': 15}
```

Рис. 6.28. Вторая попытка поиска с шестью результатами: волчанка входит в первую тройку результатов

Теперь количество результатов сократилось до 6, и волчанка (lupus) входит в первую тройку. На этой стадии поисковая система считает, что гранулоцитарный эрлихиоз человека (HGE, Human Granulocytic Ehrlichiosis) более вероятен. Эта болезнь распространяется клещами, как и печально известный клещевой боррелиоз. К настоящему моменту толковый доктор уже определит, какой болезнью страдает пациент, потому что в диагностике задействовано множество факторов – больше, чем можно передать нашей скромной поисковой системе. Например, сыпь встречается только у 10% пациентов HGE и у 50% пациентов с волчанкой. Волчанка развивается очень медленно, а для заражения HGE достаточно одного укуса клеща. Современные базы данных с машинным обучением, получающие всю эту информацию более структурированным способом, могут поставить диагноз с гораздо большей уверенностью.

С учетом того, что вам приходится иметь дело со страницами Википедии, вам понадобится еще один симптом для подтверждения диагноза. Пациент испытывает боль в груди (chest pain); добавим этот симптом в список.

```
"query": '+fatigue+fever+"joint pain"+rash+"chest pain"',
```

Результат показан на рис. 6.29.

```
{u'_shards': {u'failed': 0, u'successful': 5, u'total': 5},
 u'hits': {u'hits': [{u'_id': u'Lupus erythematosus',
    u'_index': u'medical',
    u'_score': 0.010452312,
    u'_type': u'diseases',
    u'fields': {u'name': [u'Lupus erythematosus']}}],
  u'max_score': 0.010452312,
  u'total': 1},
 u'timed_out': False,
 u'took': 11}
```

Рис. 6.29. Третья попытка поиска: симптомы достаточны для того, чтобы диагностировать волчанку

Похоже, это волчанка. Мы не сразу пришли к этому выводу, но все же это произошло. Конечно, наши возможности передачи симптомов Elasticsearch были ограничены: мы могли использовать либо одиночные слова (`fatigue`), либо четко определенные выражения (`"joint pain"`). В нашем примере это сработало, но Elasticsearch обладает большей гибкостью. Система поддерживает регулярные выражения и нечеткий поиск (fuzzy search), но эта тема выходит за рамки книги, хотя некоторые примеры были включены в архив кода примеров.

Обработка ошибок: расстояние Дамерау—Левенштейна

Допустим, пользователь случайно ввел «lupsu» вместо «lupus». Подобные опечатки встречаются сплошь и рядом в самых разных документах, созданных человеком. Для решения этой проблемы специалисты data science часто используют расстояние Дамерау—Левенштейна, которое определяется как количество операций, необходимых для преобразования одной строки в другую. При вычислении расстояния используются четыре операции:

- *Удаление* — исключение символа из строки.
- *Вставка* — добавление символа в строку.
- *Замена* — замена одного символа другим. Если замена не будет считаться одной операцией, для ее моделирования потребуются две операции: одно удаление и одна вставка.
- *Перестановка двух соседних символов* — два соседних символа меняются местами.

Последняя операция (перестановка) отличает традиционное расстояние Левенштейна от расстояния Дамерау—Левенштейна. Именно с ней ошибки в написании слов остаются в пределах допустимого. Расстояние Дамерау—Левенштейна снисходительно относится к подобным ошибкам, благодаря чему эта метрика отлично подходит для поисковых систем, но она также может использоваться для других вещей, например вычисления расстояния между цепочками ДНК.

На рис. 6.30 показано, как переход от «lupsu» к «lupus» выполняется всего за одну перестановку.

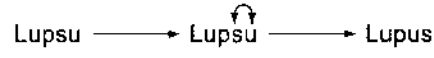


Рис. 6.30. Перестановка соседних символов — одна из операций при вычислении расстояния Дамерау—Левенштейна. Три другие операции — вставка, удаление и замена

В общем, мы достигли первой цели: *диагностики* болезни. Но не будем забывать о вторичной цели проекта: *профилировании*.

Вторичная цель проекта: профилирование болезни

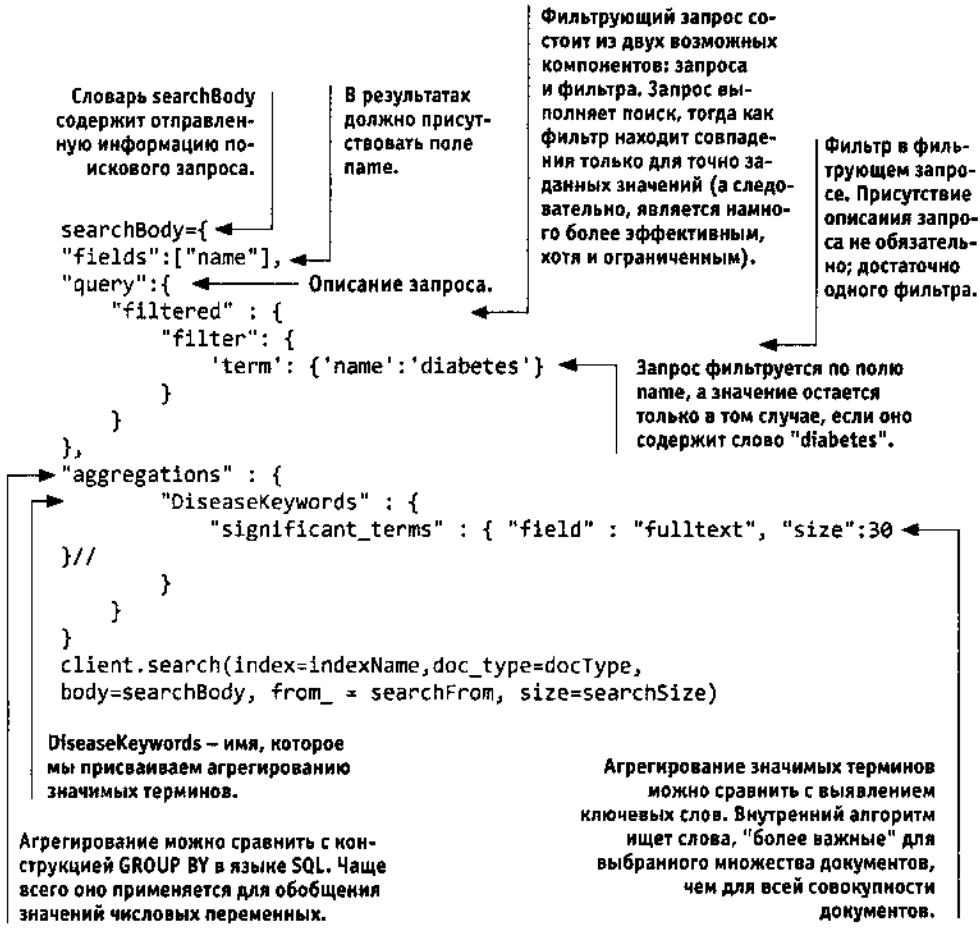
По сути, мы хотим получить список ключевых слов, соответствующих выбранному заболеванию. Для этого будет использоваться *агрегирование значимых терминов*. Метрика для определения того, какие слова должны считаться значимыми, снова представляет собой комбинацию факторов, но она приблизительно сводится к сравнению количества вхождений термина в итоговом наборе со всеми остальными документами. Таким образом Elasticsearch профилирует итоговый набор, предоставляя ключевые слова, отличающие его от других данных. Проделаем это для диабета – распространенной болезни, существующей во многих формах (листинг 6.4).

Обратите внимание на новый код. Мы избавились от строки запроса и воспользовались вместо нее фильтром. Фильтр инкапсулируется в части запроса, потому что поисковые запросы могут объединяться с фильтрами. К нашему примеру это не относится, но когда такое происходит, Elasticsearch перед попыткой применения поиска сначала применяет намного более эффективный фильтр. Если вы знаете, что поиск должен проводиться по подмножеству данных, всегда стоит добавить фильтр для предварительного создания этого подмножества. Для демонстрации рассмотрим два фрагмента кода, которые выдают одинаковые результаты, но при этом не эквивалентны.

Простая строка запроса для поиска слова «diabetes» в названии болезни:

```
"query":{
    "simple_query_string" : {
        "query": 'diabetes',
        "fields": ["name"]
        }
    }
```

Листинг 6.4. Запрос значимых терминов Elasticsearch для названия "diabetes"



Фильтр, который отбирает все болезни, в названии которых присутствует слово «diabetes»:

```
"query":{
    "filtered" : {
        "filter": {
            'term': {'name':'diabetes'}
        }
    }
}
```

И хотя при малом объеме данных это незаметно, фильтр работает намного быстрее поиска. Поисковый запрос вычисляет рейтинг поиска для каждой болезни и ранжирует их соответствующим образом, тогда как фильтр просто отфильтро-

вывает неподходящие записи. Таким образом, фильтр намного проще реального поиска; это просто выбор ответа «да» или «нет», совершенно очевидный по результатам работы. Рейтинг для всех результатов равен 1; в границах итогового набора все записи равноправны. Вывод сейчас состоит из двух частей из-за агрегирования значимых терминов. Раньше были только совпадения; теперь в результаты включаются совпадения и данные агрегирования. Для начала взгляните на совпадения на рис. 6.31.

```
u'hits': {u'hits': [{u'_id': u'Diabetes mellitus',
   u'_index': u'medical',
   u'_score': 1.0,
   u'_type': u'diseases',
   u'fields': {u'name': [u'Diabetes mellitus']}},
  {u'_id': u'Diabetes insipidus, nephrogenic type 3',
   u'_index': u'medical',
   u'_score': 1.0,
   u'_type': u'diseases',
   u'fields': {u'name': [u'Diabetes insipidus, nephrogenic type 3']}},
  {u'_id': u'Ectodermal dysplasia arthrogryposis diabetes mellitus',
   u'_index': u'medical',
   u'_score': 1.0,
   u'_type': u'diseases',
   u'fields': {u'name': [u'Ectodermal dysplasia arthrogryposis diabetes mellitus']}}],
 u'max_score': 1.0,
 u'total': 27},
u'timed_out': False,
u'took': 44}
```

Рис. 6.31. Вывод совпадений по фильтрующему запросу с фильтром «diabetes» в названии болезни

Результаты выглядят знакомо, если не считать одного важного исключения: всем результатам назначается рейтинг 1. Помимо простоты выполнения, фильтр на некоторое время кэшируется Elasticsearch. Последующие запросы с тем же фильтром будут выполняться еще быстрее, что приведет к огромному выигрышу по скорости по сравнению с поисковыми запросами.

Когда следует использовать фильтры, а когда поисковые запросы? Правило простое: выбирайте фильтры всегда, когда это возможно, и используйте поисковые запросы для полнотекстового поиска в том случае, если требуется ранжирование результатов, чтобы самые интересные результаты оказались в начале.

Взгляните на значимые термины на рис. 6.32. Чтобы понять, откуда они взялись, достаточно просмотреть описание диабета в статье Википедии.

Понятно, откуда взялись такие слова, как «diabetes» или «diabaineín»; как нетрудно догадаться, самые релевантные ключевые слова для заболевания связаны с его определением или происхождением названия. Впрочем, есть и другие интересные слова, например *ndi*, сокращенное название самой распространенной версии диабета («Nephrogenic Diabetes Insipidus», или «несахарный почечный диабет»).

```
{u'_shards': {u'failed': 0, u'successful': 5, u'total': 5},
 u'aggregations': {u'DiseaseKeywords': {u'buckets': [{u'bg_count': 18,
     u'doc_count': 9,
     u'key': u'siphon',
     u'score': 62.84567901234568},
    {u'bg_count': 18,
     u'doc_count': 9,
     u'key': u'diabainein',
     u'score': 62.84567901234568},
    {u'bg_count': 18,
     u'doc_count': 9,
     u'key': u'bainein',
     u'score': 62.84567901234568},
    {u'bg_count': 20,
     u'doc_count': 9,
     u'key': u'passer',
     u'score': 56.52777777777778},
    {u'bg_count': 14,
     u'doc_count': 7,
     u'key': u'ndi',
     u'score': 48.87997256515774},
```

Рис. 6.32. Агрегирование значимых терминов (первые пять ключевых слов)

Ключевые слова возвращаются в нижнем регистре потому, что они были сохранены в таком виде в индексе, когда мы обработали их стандартным анализатором в процессе индексирования. При индексировании мы ничего специально не указали, поэтому по умолчанию использовался стандартный анализатор. Среди других ключевых слов в «верхней тридцатке» встречается *avp* (ген, относящийся к диабету), *thirst* (жажда), симптом диабета, и *Amiloride*, лекарство от диабета. Похоже, эти ключевые слова описывают диабет, но мы упускаем составные ключевые слова; в индексе сохраняются только отдельные компоненты, потому что такое поведение используется по умолчанию. Некоторые слова не встречаются среди ключевых, потому что сами по себе они используются относительно редко, но при этом являются значимыми при использовании в сочетании с другими словами. В настоящее время связи между этими словами теряются. Для примера возьмем *ndi*; если бы это сокращение всегда записывалось в своей полной форме «Nephrogenic Diabetes Insipidus», то оно не было бы включено в состав ключевых слов. Хранение *n-грамм* (комбинаций из *n* слов) повышает затраты пространства, а применение их для запросов или агрегирования повышает нагрузку на сервер. Правильный выбор компромиссного решения требует практического опыта, зависит от специфики данных и предполагаемого использования.

Как правило, *диграммы* (комбинации двух слов) полезны, потому что осмысленные диграммы часто встречаются в естественном языке – в отличие, скажем, от «десятиграмм». Концепция ключей-диграмм будет полезна для профилирования болезней, но для этого они должны быть сохранены в виде диграмм в индексе. Как часто бывает в data science, для внесения изменений необходимо сделать несколько шагов назад. Вернемся к фазе подготовки данных.

6.2.4. Этап 3 (снова): Подготовка данных для профилирования болезни

Не приходится удивляться тому, что мы вернулись к подготовке данных (рис. 6.33). В конце концов, процесс data science имеет интерактивную природу. Во время индексирования данных мы практически не занимались очисткой или преобразованием данных. Можно добавить очистку данных сейчас, например провести фильтрацию *игнорируемых слов*. Так называются слова, встречающиеся настолько часто, что они обычно отбрасываются, потому что их присутствие может загрязнять результаты. Здесь мы не будем подробно рассматривать фильтрацию игнорируемых слов (или другую очистку данных), но при желании вы можете заняться ею самостоятельно.

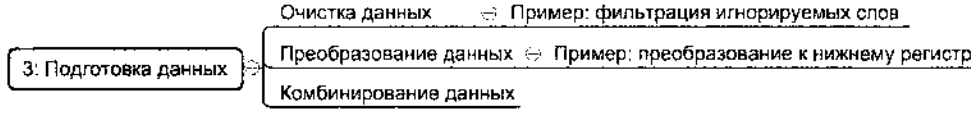


Рис. 6.33. Этап 3 процесса data science: Подготовка данных. Очистка данных для текста может включать фильтрацию игнорируемых слов, а преобразование — перевод символов в нижний регистр

Чтобы индексировать диграммы, вам необходимо создать собственный *фильтр лексем* и анализатор текста. Фильтр лексем позволяет применять преобразования к лексемам. Ваш фильтр лексем должен объединять лексемы для создания *n-грамм*. Фильтр лексем, используемый в Elasticsearch по умолчанию, называется стандартным фильтром лексем; он ищет границы слов (например, пробелы между словами), по которым текст будет разделяться на лексемы. Взгляните на новые настройки индекса болезней (листинг 6.5).

Листинг 6.5. Обновление настроек индексирования Elasticsearch

```
settings={
    "analysis": {
            "filter": {
                "my_shingle_filter": {
                    "type":             "shingle",
                    "min_shingle_size": 2,
                    "max_shingle_size": 2,
                    "output_unigrams":  False
                }
            },
            "analyzer": {
                "my_shingle_analyzer": {
                    "type":             "custom",
                    "tokenizer":        "standard",
                    "filter": [
```

```
                    "lowercase",
                    "my_shingle_filter"
                ]
            }
        }
    }
}
client.indices.close(index=indexName)
client.indices.put_settings(index=indexName , body = settings)
client.indices.open(index=indexName)
```

Чтобы вы могли изменить настройки, индекс необходимо закрыть. После изменения настроек индекс можно открыть заново.

Вы создаете два новых элемента: фильтр лексем с именем «my shingle filter» и новый анализатор с именем «my_shingle_analyzer». Так как n-граммы встречаются настолько часто, в поставку Elasticsearch включается встроенный фильтр лексем для n-грамм. От вас требуется лишь сообщить, что вам нужны диграммы: `"min_shingle_size" : 2, "max_shingle_size" : 2`, как показано на рис. 6.34. В принципе можно заняться триграммами и т. д., но для демонстрационных целей хватит и этого.

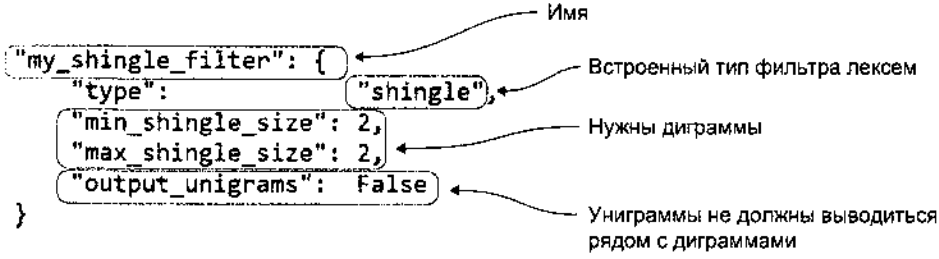


Рис. 6.34. Фильтр лексем для диграмм

Анализатор на рис. 6.35 представляет собой комбинацию всех операций, необходимых для перехода от входного текста к индексу. Он включает фильтр n-грамм, но это далеко не все. Генератор лексем разбивает текст на лексемы; после этого можно использовать фильтр нижнего регистра, чтобы при поиске слово «Diabetes» не отличалось от «diabetes». Наконец, мы применяем фильтр n-грамм, чтобы создать диграммы.

Обратите внимание: перед обновлением настроек индекс необходимо закрыть. После этого индекс можно снова открыть, зная, что настройки уже обновились. Не все конфигурационные изменения требуют закрытия индекса, но именно в этом случае оно необходимо. Сводка изменений, требующих закрытия индекса, приведена по адресу *http://www.elastic.co/guide/en/elastic-search/reference/current/indices-update-settings.html*.

Индекс готов к использованию нового анализатора. Для этого мы создадим новый тип документа `diseases2` с новым отображением (листинг 6.6).

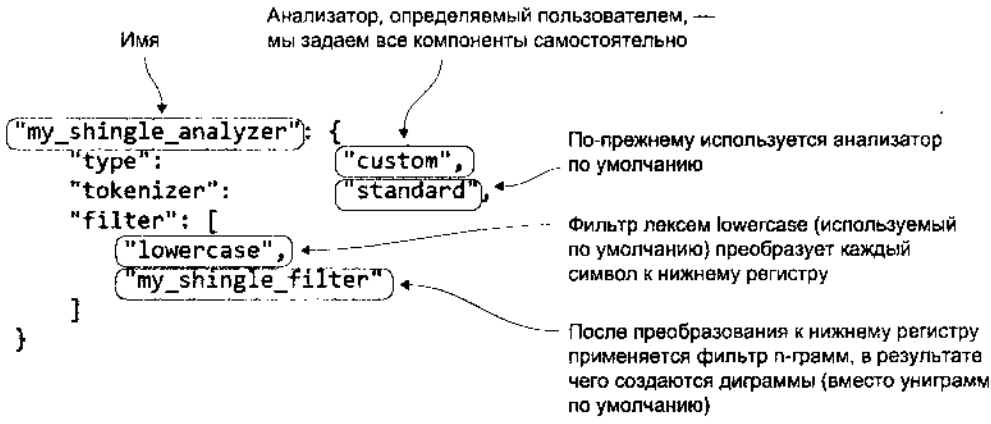


Рис. 6.35. Пользовательский анализатор со стандартным генератором лексем и фильтром лексем для создания диграмм

Листинг 6.6. Создание более сложного отображения для типа документа Elasticsearch

```
docType = 'diseases2'
diseaseMapping = {
        'properties': {
            'name': {'type': 'string'},
            'title': {'type': 'string'},
            'fulltext': {
                "type": "string",
                "fields": {
                    "shingles": {
                        "type":     "string",
                        "analyzer": "my_shingle_analyzer"
                    }
                }
            }
        }
    }
client.indices.put_mapping(index=indexName,
doc_type=docType,body=diseaseMapping )
```

Новое отображение отличается от старого добавлением поля fulltext.shingles, содержащего диграммы.

У свойства `fulltext` появляется дополнительный параметр `fields`. Здесь вы можете указать все аспекты обработки `fulltext`. Здесь такой аспект всего один; он называется `shingles` и анализирует `fulltext` с применением нового анализатора `my_shingle_analyzer`. У вас по-прежнему остается доступ к исходному содержимому `fulltext`, причем для него не был назначен анализатор, поэтому, как и прежде, будет использована стандартная версия. Чтобы обратиться к новой версии, укажите имя свойства, за которым следует имя поля: `fulltext.shingles`. Теперь остается лишь выполнить все предыдущие шаги заново и проиндексировать данные с использованием API Википедии (листинг 6.7).

Листинг 6.7. Повторное индексирование описаний болезней из Википедии с новым отображением

```
dl = wikipedia.page("Lists_of_diseases")
diseaseListArray = []
for link in dl.links[15:42]:
    try:
        diseaseListArray.append(wikipedia.page(link))
    except Exception,e:
        print str(e)

checkList = [["0","1","2","3","4","5","6","7","8","9"],
["A"],["B"],["C"],["D"],["E"],["F"],["G"],
["H"],["I"],["J"],["K"],["L"],["M"],["N"],
["O"],["P"],["Q"],["R"],["S"],["T"],["U"],
["V"],["W"],["X"],["Y"],["Z"]]
```

В массиве checkList хранятся допустимые первые символы. Если болезнь не отвечает критерию, она пропускается.

```
for diseaselistNumber, diseaselist in enumerate(diseaseListArray):
    for disease in diseaselist.links: #Перебор списка ссылок
     для каждого списка болезней
        try:
            if disease[0] in checkList[diseaselistNumber]
and disease[0:3] !="List":
                currentPage = wikipedia.page(disease)
                client.index(index=indexName,
doc_type=docType,id = disease, body={"name": disease,
"title":currentPage.title ,
"fulltext":currentPage.content})
        except Exception,e:
            print str(e)
```

Сначала проверяем, является ли болезнью, а затем индексируем.

Перебор списков болезней.

Ничего нового, просто на этот раз вместо `diseases` индексируется тип документа `diseases2`. Когда это будет сделано, можно снова вернуться к шагу 4 (исследование данных) и проверить результаты.

6.2.5. Этап 4 (повторно): Исследование данных для профилирования болезни

Мы снова вернулись к исследованию данных. Теперь мы можем изменить запрос и использовать новое поле для того, чтобы получить диграммные ключевые концепции, связанные с диабетом (листинг 6.8).

Листинг 6.8. Агрегирование значимых терминов с диграммами

```
searchBody={
"fields":["name"],
"query":{
    "filtered" : {
        "filter": {
```

```
            'term': {'name':'diabetes'}
        }
    }
},
"aggregations" : {
        "DiseaseKeywords" : {
            "significant_terms" : { "field" : "fulltext", "size" : 30 }
        },
        "DiseaseBigrams": {
            "significant_terms" : { "field" : "fulltext.shingles",
"size" : 30 }
        }
    }
}
client.search(index=indexName,doc_type=docType,
body=searchBody, from_ = 0, size=3)
```

Новое агрегирование с именем `DiseaseBigrams` использует поле `fulltext.shingles` для получения новой информации о диабете в форме диграмм.

В процессе поиска вы наверняка найдете много разных интересных биграмм, униграмм и даже триграмм. В совокупности они могут использоваться для анализа текста или набора текстов перед их чтением. Следует заметить, что желаемые результаты были достигнуты без перехода к стадии моделирования. Иногда при исследовании данных удается найти ценной информации по крайней мере не меньше, чем в процессе моделирования. Итак, вторичная цель полностью достигнута и мы можем переходить к этапу 6 процесса data science: отображению и автоматизации.

6.2.6. Этап 6: Отображение и автоматизация

Основная цель проекта — диагностика болезней — превратилась в инструмент самостоятельной диагностики. Врач может обращаться к поисковой системе за информацией, передавая запрос, допустим, через веб-приложение. Мы не будем строить веб-сайт, но если вы собираетесь этим заняться, прочитайте врезку «Elasticsearch для веб-приложений».

Вторая цель — профилирование болезней — также может быть выведена на уровень пользовательского интерфейса. По результатам поиска можно построить облако ключевых слов, наглядно обобщающее результаты поиска. Мы не будем заниматься этим в книге, но если вас интересует создание подобных конструкций на языке Python, воспользуйтесь библиотекой word_cloud (pip install word_cloud). А если вы предпочитаете JavaScript, к вашим услугам библиотека D3.js. Пример реализации можно найти по адресу *http://www.jasondavies.com/wordcloud/#%2F%2Fwww.jasondavies.com%2Fwordcloud%2Fabout%2F*.

При добавлении собственных ключевых слов на сайт на базе D3.js будет получено облако униграмм наподобие изображенного на рис. 6.36, которое может быть встроено в презентацию результатов вашего проекта. В этом примере ключевым словам веса не назначаются, но и такой результат получается достаточно наглядным.

ELASTICSEARCH ДЛЯ ВЕБ-ПРИЛОЖЕНИЙ

Как и для любой другой базы данных, предоставлять доступ Elasticsearch REST API прямо из интерфейсной части веб-приложений не рекомендуется. Если сайт сможет обращаться с запросами POST прямо к базе данных, любой желающий сможет с такой же легкостью удалять ваши данные; всегда существует необходимость в промежуточном уровне. Таким промежуточным уровнем может стать код Python, если вас это устраивает. Два популярных решения на базе Python — Django и REST-инфраструктура Django в сочетании с независимой интерфейсной частью. Django обычно используется для построения приложений, в которых сервер строит интерфейсную часть динамически на основе информации из базы данных и системы шаблонов. REST-инфраструктура Django реализована в виде плагина для Django, превращающего Django в REST-службу, что позволяет использовать Django как часть одностраничных приложений. Одностраничное приложение представляет собой веб-приложение, использующее всего одну веб-страницу, но способное динамически изменять контент за счет получения статических файлов у сервера HTTP и данных от REST-совместимых API. Годятся оба решения — лишь бы сервер Elasticsearch не был открыт для всех желающих, так как он не имеет встроенных средств безопасности. Для добавления безопасности к Elasticsearch можно воспользоваться «Shield», платной услугой Elasticsearch.

Рис. 6.36. Облако униграмм для ключевых слов без назначения весов, полученных от Elasticsearch

В приложение можно внести много улучшений, особенно в области подготовки данных. Впрочем, углубленное изучение всех возможностей завело бы нас слишком далеко; соответственно, глава подходит к концу. Следующая глава будет посвящена потоковым данным.

6.3. Итоги

Основные положения этой главы:

- Сокращение NoSQL означает «не только SQL» (Not Only SQL»). Концепция NoSQL возникла из необходимости обработки экспоненциально растущих объемов и разнообразия данных, а также растущей потребности в более разнообразных и гибких схемах (таких, как сетевые и иерархические структуры).
- Обработка всех этих данных требовала секционирования базы данных, потому что одна машина не способна справиться с этой работой. При секционировании базы данных действует теорема CAP: можно обеспечить либо доступность, либо согласованность — но не оба свойства сразу.
- Реляционные и графовые базы данных соответствуют набору принципов ACID: атомарность, согласованность, изолированность и долгосрочность. Базы данных NoSQL обычно обеспечивают соблюдение принципов BASE: базовая доступность, неустойчивое состояние и согласованность в конечном счете.
- Четыре основных типа баз данных NoSQL:
  - Хранилища «ключ—значение» — фактически набор пар «ключ—значение», хранящихся в базе данных. Такие базы данных могут быть просто огромными, они отличаются большой гибкостью, но сложность таких данных относительно невелика. Известный пример — Redis.
  - *Столбцовые базы данных* — эти базы данных немного сложнее хранилищ «ключ—значение»: они тоже используют столбцы, но более эффективно, чем обычная РСУБД. Столбцы фактически изолированы друг от друга, что позволяет быстро получить данные из одного столбца. Известный пример — Cassandra.
  - *Хранилища документов* — эти базы данных устроены немного сложнее, а данные в них хранятся в виде документов. В настоящее время самым популярным представителем этой категории является MongoDB, но в нашем примере используется система Elasticsearch, которая сочетает функциональность хранилища документов с функциональностью поисковой системы.
  - *Графовые базы данных* — эти базы данных способны хранить самые сложные структуры данных, так как они уделяют равное внимание сущностям и отношениям между сущностями. За эту сложность приходится расплачиваться скоростью поиска. В этой категории популярна система Neo4j, но в последнее время ее обгоняет GraphX (графовая база данных, связанная с Apache Spark).
- Elasticsearch — хранилище документов и полнотекстовая поисковая система на базе Apache Lucene, поисковой системы с открытым кодом. Elasticsearch может использоваться для выделения лексем, выполнения агрегатных запросов, выполнения профилирующих запросов и многих других целей.

Графовые базы данных

В этой главе:

- ✓ Концепция связанных данных, ее отношение к графам и графовым базам данных.
- ✓ Различия между графовыми и реляционными базами данных.
- ✓ Знакомство с графовой базой данных Neo4j.
- ✓ Применение процесса data science в проекте рекомендательной системы с использованием графовой базы данных Neo4j.

С одной стороны, производство данных в огромных масштабах заставляет такие компании, как Google, Amazon или Facebook, изобретать интеллектуальные способы работы с ними; с другой стороны, данные, с которыми мы имеем дело, становятся все более взаимосвязанными. Графы и сети все глубже проникают в нашу жизнь. Представив несколько интересных примеров, мы надеемся научить читателя узнавать задачи на применение графов там, где он столкнется с ними. В этой главе мы рассмотрим возможности использования этих связей в графовых базах данных и продемонстрируем, как использовать Neo4j, популярную графовую базу данных.

7.1. Связанные данные и графовые базы данных

Начнем с концепции связанных данных и их представления в виде графовых данных.

- *Связанные данные* — как следует из названия, связанные данные характеризуются участием в отношениях, образующих некоторые связи.

- *Графы* — часто упоминаются наряду со связанными данными. Графы хорошо подходят для содержательного представления связности данных.
- *Графовые базы данных* — были кратко представлены в главе 6. Эта тема заслуживает особого внимания, потому что современные данные не только увеличиваются в размерах, но и становятся все более взаимосвязанными. За примерами связанных данных далеко ходить не нужно, они у всех на виду.

Яркий пример данных, принимающих сетевую форму, — данные социальных сетей. Социальные сети позволяют обмениваться информацией в сетях, генерируя тем самым большой объем связанных данных. Это можно продемонстрировать на простом примере. Допустим, в данных хранится информация о двух людях, Пользователь1 и Пользователь2. Кроме того, мы знаем имя и фамилию Пользователь1 (имя: Пол, фамилия: Бьен) и Пользователь2 (имя: Джелме, фамилия: Рагнар). Один из естественных способов представления этой информации — изображение ее на доске (рис. 7.1).

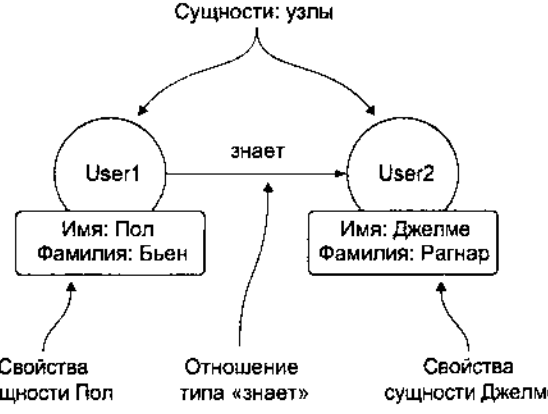


Рис. 7.1. Простой пример связанных данных: две сущности, или узла (Пользователь1, Пользователь2), обладающих свойствами (имя, фамилия) и связанных отношением (знает)

Ниже описана терминология рис. 7.1:

- *Сущности* — две сущности представляют людей (Пользователь1 и Пользователь2). Эти сущности обладают свойствами «имя» и «фамилия».
- *Свойства* — свойства определяются парами «ключ—значение». Из этого графа также можно узнать, что пользователь Пользователь1 со свойством «имя» Пол знает пользователя Пользователь2, у которого свойство «имя» содержит Джелме.
- *Отношения* — между сущностями Пол и Джелме существует отношение. Обратите внимание на то, что это отношение является направленным: Пол «знает»

Джелме, но не наоборот. Сущности Пользователь1 и Пользователь2 также могут включаться в группы.

- *Метки* — в графовой базе данных узлы могут группироваться по меткам. Так, в данном примере сущности Пользователь1 и Пользователь2 можно объединить в группу «Пользователи».

Связанные данные часто содержат много других сущностей и связей. На рис. 7.2 изображен более крупный граф с двумя сущностями: Страна1 с названием Камбоджа и Страна2 с названием Швеция. Также на графе существует два отношения: «Бывал в» и «Родился в». На предыдущем графе свойствами обладали только сущности, теперь отношения тоже содержат свойства. Такие графы называются *графами свойств*. Отношение, связывающее узлы Пользователь1 и Страна1, относится к типу «Бывал в» и обладает свойством «Дата», которое представляет значение данных. Аналогичным образом Пользователь2 связывается со Страна2, но через другой тип отношения «Родился в». Обратите внимание: типы отношений предоставляют контекст для отношений между узлами. Каждый узел также может участвовать в нескольких отношениях.

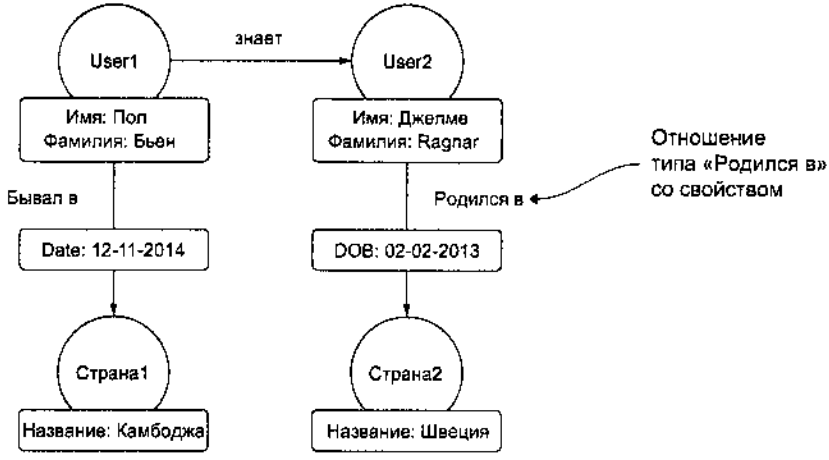


Рис. 7.2. Более сложный пример связанных данных с включением еще двух сущностей (Страна1 и Страна2) и двух отношений («Бывал в» и «Родился в»)

Такое представление данных формирует интуитивно понятный подход к хранению связанных данных. Чтобы исследовать данные, необходимо перемещаться по графу по заранее определенным путям для выявления искомых закономерностей. Что, если пользователь захочет узнать, в каких странах бывал Пол? Если преобразовать этот запрос в терминологию графовых баз данных, нам потребуется найти закономерность «Пол бывал в». Чтобы ответить на этот вопрос, мы начнем с узла с именем «Пол» и перейдем в Камбоджу по отношению «Бывал в». Следовательно, обход графа, представляющий запрос к базе данных, будет выглядеть так.

- *Начальный узел* — в данном случае это узел пользователя, у которого свойство «Имя» содержит «Пол».
- *Путь обхода* — в данном случае это путь, начинающийся с узла «Пол» и переходящий в узел «Камбоджа».
- *Конечный узел* — узел страны, у которого свойство «Имя» содержит «Камбоджа».

Чтобы лучше понять, как графовые базы данных работают со связанными данными, стоит рассказать подробнее о графах в целом. Графы подробно изучаются в дисциплинах информатики и математики — в области, называемой *теорией графов*. Теория графов изучает *графы* — математические структуры, используемые для моделирования парных отношений между объектами (рис. 7.3). Графы особенно удобны тем, что их структура хорошо подходит для визуализации связанных данных. Граф состоит из *вершин* (также называемых *узлами* в теории графовых баз данных) и *ребер* (также называемых *отношениями*). Эти концепции образуют фундамент, на котором строятся графовые структуры данных.

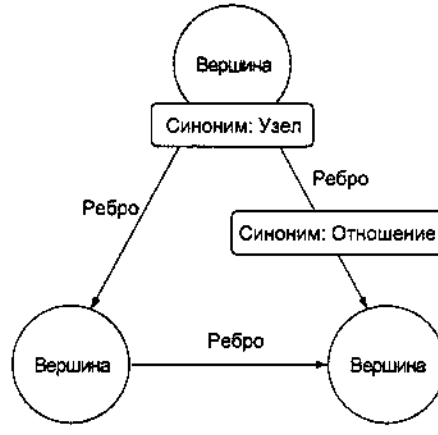


Рис. 7.3. Как известно из математического определения, граф состоит из узлов (также называемых вершинами) и ребер (соединяющих вершины). Эти совокупности объектов представляют собой граф

Отличительной особенностью связанных данных в сравнении с другими структурами данных является их нелинейная природа: любая сущность может быть соединена с любыми другими сущностями отношениями разных типов, с промежуточными типами и путями. Графы дополнительно делятся на направленные и ненаправленные. Ребра направленных графов имеют направление (кто бы сомневался?). И хотя каждая задача может быть тем или иным образом представлена в виде графа, важно понимать, когда такое решение идеально, а когда нет.

7.1.1. Когда и почему используются графовые базы данных?

Принять решение о том, нужно ли применять графовую базу данных в конкретном случае, может быть весьма непросто. Одним из важных аспектов процесса принятия решения становится выбор правильного представления данных. С начала 1970-х годов самую распространенную категорию баз данных составляли *реляционные* базы данных. Позднее появились и другие типы, например иерархические базы данных (такие, как IMS), и ближайший родственник графовых баз данных — сетевые базы данных (такие, как IDMS). Однако за последние десятилетия панорама стала куда более разнообразной, а у пользователей появилось больше свободы выбора в зависимости от их конкретных потребностей. Что касается недавних тенденций с появлением доступных данных, стоит особо выделить две характеристики: размер и сложность данных (рис. 7.4).

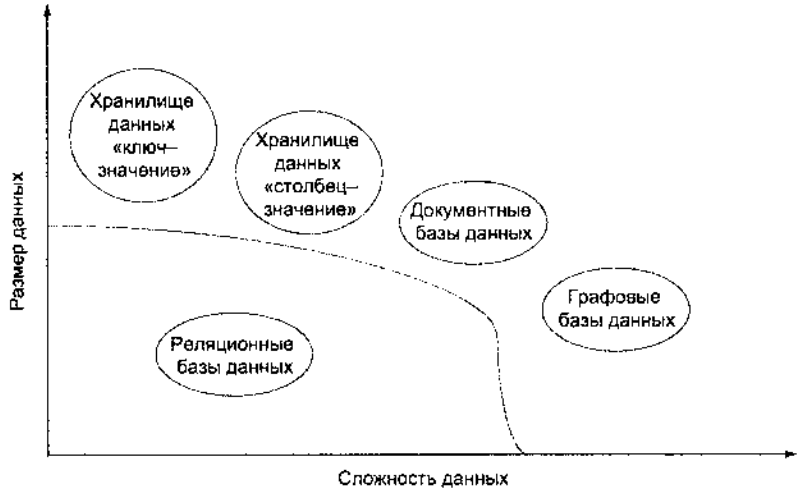


Рис. 7.4. На диаграмме показано место графовых баз данных на плоскости, одно измерение которой представляет собой размер данных, а другое — сложность (в контексте связности). Если реляционные базы данных не справляются со сложностью набора данных из-за его высокой связности, но не размера, возможно, лучшим вариантом окажется графовая база данных

Как видно из рис. 7.4, графовые базы данных следует использовать только в том случае, если данные сложны, но при этом имеют небольшой размер. Конечно, «небольшой» — понятие относительное, речь идет о сотнях миллионов узлов. Эффективная работа со сложными данными — главное преимущество графовой базы данных и главный ответ на вопрос «почему?» (следует использовать базу данных). Чтобы понять, какая *сложность* имеется в виду, сначала задумайтесь над тем, как работает традиционная реляционная база данных.

Вопреки названию, отношений (relations) в реляционных базах данных не так уж много, разве что внешние и первичные ключи, используемые для установления связей между таблицами. Напротив, в графовых базах данных отношения играют главную роль, и поэтому они хорошо вписываются в механизмы моделирования и выборки связанных данных. Реляционная база данных стремится к минимизации избыточности данных. Этот процесс называется *нормализацией* базы данных: таблица разбивается на меньшие таблицы (с меньшей избыточностью) без потери информации. В нормализованной базе данных любой атрибут достаточно изменить только в одной таблице. Собственно, целью процесса нормализации является ограничение изменений в данных одной таблицей. Реляционные системы управления базами данных (РСУБД) хорошо подходят для данных, укладывающихся в табличный формат. Отношения в таких данных могут выражаться соединением таблиц. Впрочем, их эффективность начинает снижаться с усложнением соединений, особенно когда они превращаются в соединения типа «многие ко многим». С ростом размера данных также увеличивается время обработки запросов, а сопровождение базы данных усложняется. Эти факторы влияют на эффективность базы данных. С другой стороны, в графовых базах данных информация изначально хранится в виде узлов и отношений. И хотя графовые базы данных по классификации относятся к типу баз данных NoSQL, в последнее время их нередко выделяют в самостоятельную категорию. Обычно это обосновывается тем, что другие типы баз данных NoSQL ориентированы на агрегирование информации, а графовые базы данных — нет.

Например, в реляционной базе данных может присутствовать таблица people, представляющая людей и их свойства. Любой человек связан с другими людьми всевозможными отношениями: дружескими, родственными и т. д.; каждая запись может представлять одного человека, но попытки связать их с другими записями в таблице могут невероятно осложниться. Добавить переменную для хранения уникального идентификатора первого ребенка? Потом другую для идентификатора второго ребенка? И где остановиться — на десятом?

Альтернативное решение — использовать промежуточную таблицу для отношений «потомок—родитель», но тогда вам понадобится отдельная таблица для другого типа отношений (например, дружбы). В этом случае размножаются уже не столбцы, а таблицы; каждый тип отношений требует новой таблицы. Даже если вам как-то удастся смоделировать данные так, чтобы в них присутствовали все семейные отношения, вам потребуются сложные запросы для получения ответов даже на простые вопросы типа «Получить имена всех внуков Джона Макбейна». Сначала необходимо найти детей Джона Макбейна. Потом нужно найти всех их детей. К тому моменту, когда вы найдете всех внуков, вы обратитесь к таблице people три раза:

- Поиск Макбейна и получение его детей.
- Поиск детей по полученным идентификаторам и получение идентификаторов их детей.
- Поиск внуков Макбейна.

На рис. 7.5 показаны рекурсивные операции с базой данных, необходимые для перехода от Джона Макбейна к его внукам, если все данные хранятся в одной таблице.

Рис. 7.5. Рекурсивный поиск, версия 1: все данные в одной таблице

На рис. 7.6 изображен другой вариант моделирования данных: с хранением отношений «родитель—потомок» в отдельной таблице.

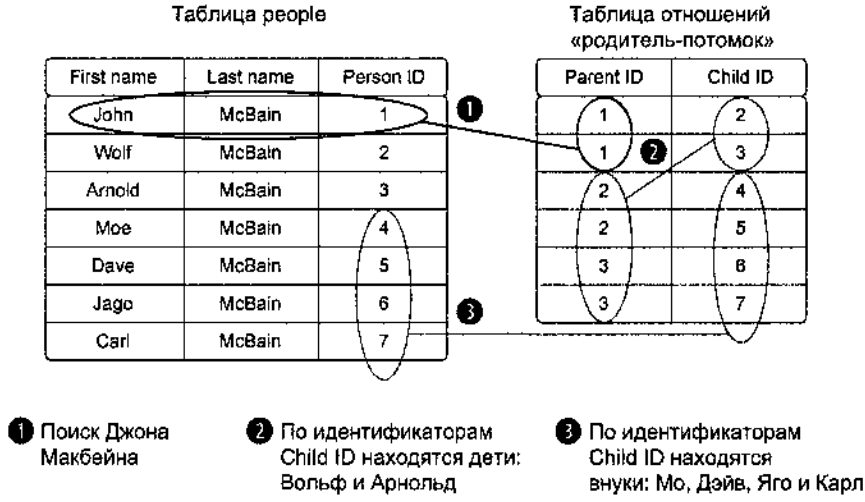


Рис. 7.6. Рекурсивный поиск, версия 2: использование отношений «родитель—потомок»

Подобные рекурсивные операции поиска по меньшей мере неэффективны.

Графовые базы данных особенно эффективны при появлении сложности такого типа. Рассмотрим самые популярные разновидности.

7.2. Neo4j: графовая база данных

Связанные данные обычно хранятся в графовых базах данных. Эти базы данных специально спроектированы с расчетом на структуру связанных данных. В наши дни выбор графовых баз данных достаточно широк. Три самые известные (по убыванию популярности): Neo4j, OrientDb и Titan. Для нашего примера будет выбрана самая популярная база данных на момент написания книги (см. *http://db-engines.com/en/ranking/graph+dbms*, сентябрь 2015).

Neo4j — графовая база данных, которая хранит информацию в виде графа с узлами и отношениями (и те и другие могут содержать свойства). Такой тип графовой базы данных, называемый *графом свойств*, хорошо подходит для хранения связанных данных. Его гибкая схема дает возможность оперативно изменять структуру данных и добавлять новые данные и новые отношения при необходимости. Это проект с открытым кодом, технология прошла проверку временем, программа проста в установке, удобна в работе и хорошо документирована. Neo4j также поддерживает графический интерфейс, упрощающий построение графов для целей визуализации. Если вы хотите повторить наш пример на своем компьютере, сейчас самое время установить Neo4j. Neo4j можно загрузить по адресу *http://neo4j.com/download/*. Краткое описание процесса установки приведено в приложении В.

Рассмотрим четыре базовые структуры Neo4j:

- *Узлы* — представляют всевозможные сущности: документы, пользователи, рецепты и т. д. Узлам могут назначаться некоторые свойства.
- *Отношения* — существуют между узлами. К отношениям можно обращаться либо напрямую, либо через узлы, к которым они присоединены. Отношения также могут содержать свойства, отсюда и название модели «граф свойств». У каждого отношения есть имя и направление; в совокупности они предоставляют семантический контекст для узлов, связанных данным отношением.
- *Свойства* — как узлы, так и отношения могут иметь свойства. Свойства определяются парами «ключ—значение».
- *Метки* могут использоваться для группировки похожих узлов. Пометка обеспечивает возможность ускоренного обхода графа.

Прежде чем проводить анализ, тщательно спроектируйте свою базу данных с учетом запросов, которые будут выполняться в ходе анализа. У графовых баз данных есть одна приятная особенность: их удобно рисовать на доске. Если вы попытаетесь изобразить условия своей задачи на доске, ваш рисунок будет очень похож на схему базы данных определяемой задачи. Таким образом, рисунок на доске станет хорошей отправной точкой для проектирования базы данных.

Как получить данные? Для этого следует обойти граф по заранее определенным путям, чтобы найти искомые закономерности. Интерфейс Neo4j идеально подходит для создания связанных данных и экспериментов с ними, пока вы не найдете оптимальное представление для своих запросов (рис. 7.7). Гибкая схема графовой базы данных отлично подходит для этого. В браузере можно просматривать данные в виде строк или в виде графа. Neo4j поддерживает собственный язык запросов, упрощающий создание графов и выполнение запросов.

Cypher — выразительный язык, который имеет достаточно общего с SQL, чтобы упростить процесс изучения. В следующем разделе мы создадим собственные данные при помощи Cypher и вставим их в Neo4j, после чего немного поэкспериментируем с данными.

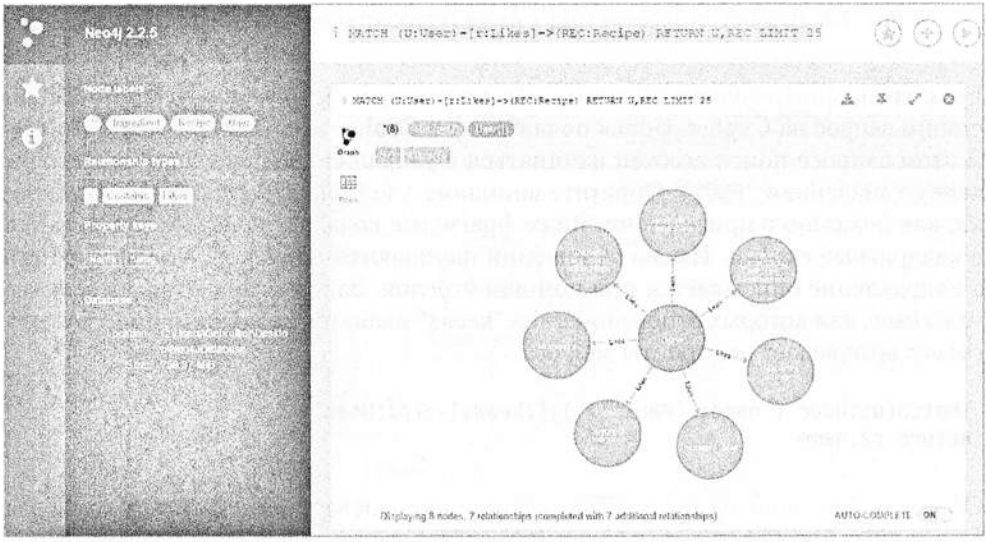


Рис. 7.7. Интерфейс Neo4j с запросом из примера этой главы

7.2.1. Cypher: язык запросов к графам

В этом разделе представлен язык Cypher и его базовый синтаксис для операций с графами. Предполагается, что приведенной информации о Cypher будет достаточно для того, чтобы вы начали работать в интерфейсе Neo4j. К концу раздела вы сможете создавать собственные связанные данные, используя Cypher в интерфейсе Neo4j, и выполнять базовые запросы для получения результатов. За более подробным вводным курсом Cypher обращайтесь по адресу *http://neo4j.com/docs/stable/cypher-query-lang.html*. Начнем с рисования простого социального графа с базовым запросом для получения данных по заранее определенному шаблону. На следующей стадии будет нарисован более сложный граф для использования более сложных запросов в Cypher. Это позволит вам познакомиться с Cypher, а также приблизит

учебный сценарий к реальности. Кроме того, мы покажем, как создать ваши собственные связанные данные при помощи Cypher.

На рис. 7.8 изображен простой социальный граф из двух узлов, соединенных отношением «знает» (knows). Оба узла имеют свойства «имя» (name) и «фамилия» (lastname).

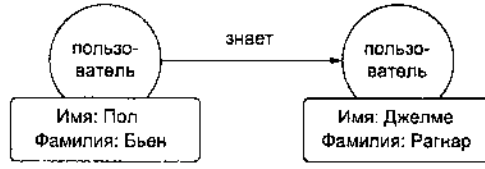


Рис. 7.8. Простой социальный граф с двумя пользователями и одним отношением

Если теперь потребуется получить ответ на вопрос: «Кого знает Пол?», мы составим запрос на Cypher. Поиск по шаблону в Cypher начинается с секции `Match`. В этом запросе поиск должен начинаться с узла `User`, обладающего свойством `name` со значением `"Paul"`. Обратите внимание: узел заключается в круглые скобки, как показано в приведенном ниже фрагменте кода, а отношение заключается в квадратные скобки. Имена отношений начинаются с префикса : (двоеточие), а направление описывается при помощи стрелок. Заполнитель `p2` содержит все узлы `User`, для которых отношение типа `"knows"` является входящим. Конструкция `return` возвращает результаты запроса:

```
Match(p1:User { name: 'Paul' } )-[:knows]->(p2:User)
Return p2.name
```

Обратите внимание на тесную связь формулировки запроса и того, как графовая база данных преобразует его в обход. В Neo4j эта выразительность стала возможной благодаря специальному языку запросов к графам, Cypher.

Чтобы пример был более интересным, предположим, что данные представлены графом на рис. 7.9.

Для вставки связанных данных на рис. 7.9 в Neo4j используется Cypher. Команды Cypher вводятся либо непосредственно в интерфейсе Neo4j, либо через драйвер Python (за дополнительной информацией обращайтесь по адресу *http://neo4j.com/developer/python/*). Это отличный способ получить практическое представление о работе со связанными данными и графовыми базами данных.

Чтобы написать команду создания на Cypher, сначала необходимо четко понять, какие данные будут храниться в узлах, а какие в отношениях, какими свойствами они будут обладать и пригодятся ли вам при этом метки. Прежде всего следует

решить, какие данные станут рассматриваться как узлы, а какие будут рассматриваться как отношения, предоставляющие смысловой контекст для этих узлов. На рис. 7.9 мы выбрали пользователей и страны, в которых они были, в качестве узлов. Данные, предоставляющие информацию о конкретном узле (например, имя человека, связанное с узлом), могут быть представлены в виде свойств. Все данные, предоставляющие контекст для двух и более узлов, будут считаться отношениями. Узлы с общими признаками (например, и Камбоджа и Швеция являются странами) будут группироваться с использованием меток. На рис. 7.9 это уже сделано.

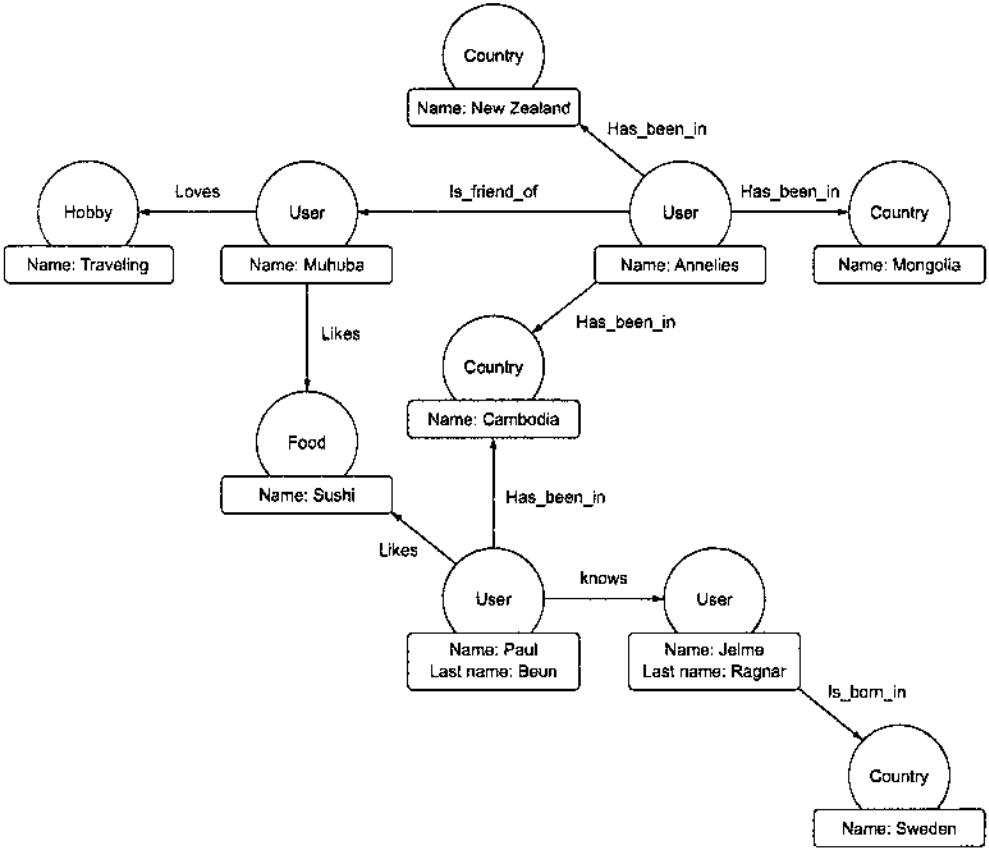


Рис. 7.9. Более сложный пример связанных данных с несколькими взаимосвязанными узлами разных типов

В листинге 7.1 показано, как разные объекты кодируются на Cypher в одной большой команде `create`. *Учтите, что язык Cypher чувствителен к регистру символов.*

Листинг 7.1. Команда создания данных на языке Cypher

```
CREATE (user1:User {name :'Annelies'}),
 (user2:User {name :'Paul' , LastName: 'Beun'}),
 (user3:User {name :'Muhuba'}),
 (user4:User {name : 'Jelme' , LastName: 'Ragnar'}),
 (country1:Country { name:'Mongolia'}),
 (country2:Country { name:'Cambodia'}),
 (country3:Country { name:'New Zealand'}),
 (country4:Country { name:'Sweden'}),
 (food1:Food { name:'Sushi' }),
 (hobby1:Hobby { name:'Travelling'}),
 (user1)-[:Has_been_in]->(country1),
 (user1)-[: Has_been_in]->(country2),
 (user1)-[: Has_been_in]->(country3),
 (user2)-[: Has_been_in]->(country2),
 (user1)-[: Is_mother_of]->(user4),
 (user2)-[: knows]->(user4),
 (user1)-[: Is_friend_of]->(user3),
 (user2)-[: Likes]->( food1),
 (user3)-[: Likes]->( food1),
 (user4)-[: Is_born_in]->(country4)
```

У выполнения этой команды за один шаг есть важное преимущество: оно гарантирует, что графовая база данных была создана успешно. Если произойдет ошибка, то граф создан не будет.

В реальной ситуации для данных также обычно определяются индексы и ограничения, чтобы выборка осуществлялась быстро и не требовала поиска по всей базе данных. В нашем примере это не делается, потому что моделируемый набор данных невелик. Впрочем, сделать это на Cypher несложно. За дополнительной информацией об индексах и ограничениях обращайтесь к документации Cypher (*http://neo4j.com/docs/stable/cypherdoc-labels-constraints-and-indexes.html*).

Данные были успешно созданы, теперь к ним можно обратиться с запросом. Следующий запрос возвращает все узлы и отношения в базе данных:

На рис. 7.10 изображена только что созданная база данных. Сравните этот граф с тем, который был нарисован на доске в ходе проектирования. На доске узлы, представляющие людей, были сгруппированы при помощи метки `User`, а узлы, представляющие страны, группируются при помощи метки `Country`. Хотя узлы на иллюстрации не представлены их метками, сами метки хранятся в базе данных.

Кроме того, в результатах отсутствует узел (Hobby) и отношение типа "Loves". Эти данные можно легко добавить командой merge, создающей узел и отношение, которые еще не существуют в данных:

```
Merge (user3)-[: Loves]->( hobby1)
```

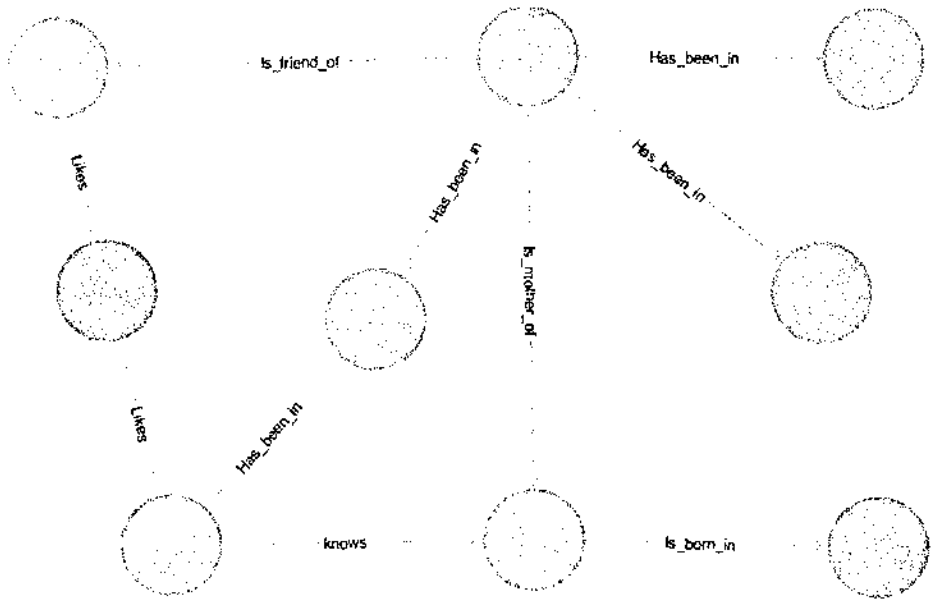


Рис. 7.10. Граф, изображенный на рис. 7.9, воссоздан в интерфейсе Neo4j. Узлы представлены не метками, а именами. Из графа видно, что в данных отсутствует метка Hobby с именем Travelling. Это объясняется тем, что мы забыли включить этот узел и соответствующее отношение в команду create

По этому графу можно получить ответы на самые разнообразные вопросы. Например:

- Вопрос 1: В каких странах побывала Аннелиз? Код Cypher для получения ответа (рис. 7.11) выглядит так:

```
Match(u:User{name:'Annelies'}) - [:Has_been_in]-> (c:Country)
Return u.name, c.name
```

- Вопрос 2: Кто в какой стране бывал? Код Cypher для получения ответа (рис. 7.12) выглядит так:

```
Match ()-[r: Has_been_in]->()
Return r LIMIT 25
```

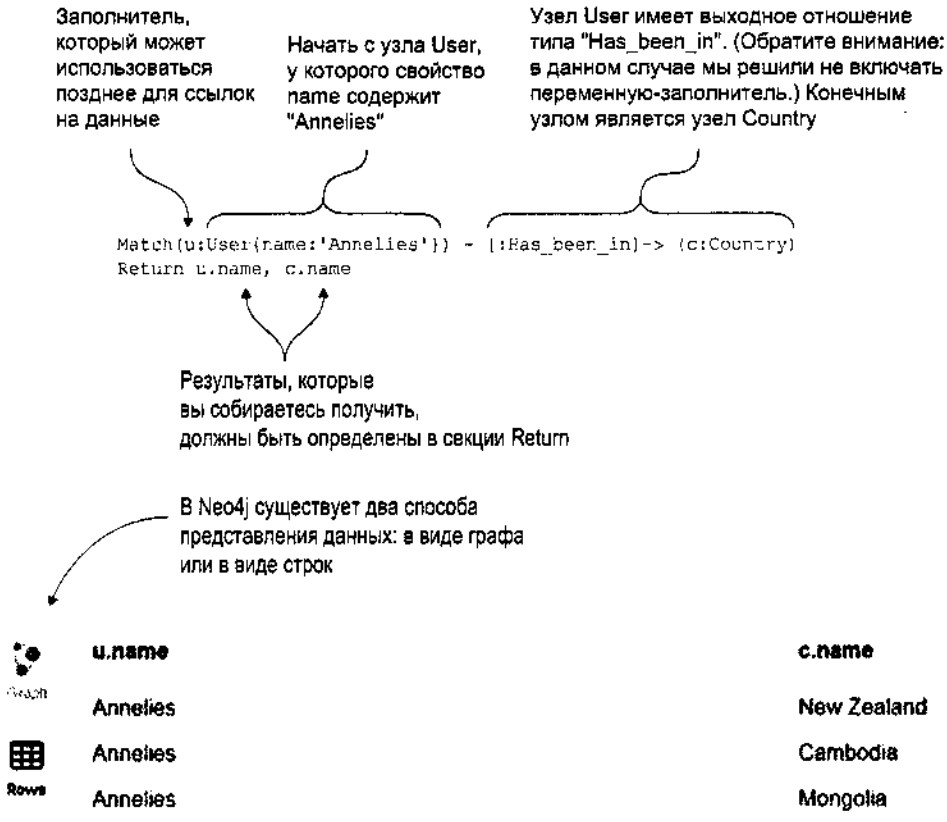


Рис. 7.11. Результаты вопроса 1: какие страны посетила Аннелиз? В строковом представлении данных Neo4j видно, что она побывала в трех странах. Обход занял всего 97 миллисекунд

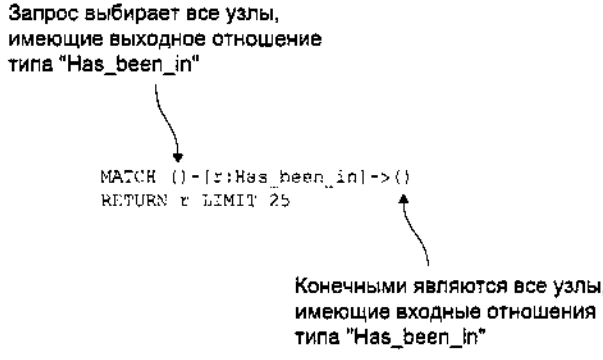


Рис. 7.12. Кто в каких странах бывал? Объяснение структуры запроса

При выполнении запроса будет получен ответ, показанный на рис. 7.13.

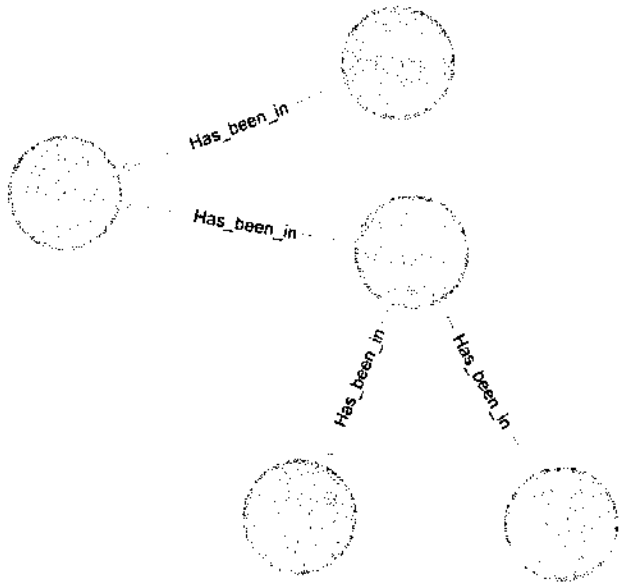


Рис. 7.13. Результаты запроса 2: Кто в каких странах бывал? На этот раз результаты запроса изображены в графовом представлении данных Neo4j. Мы видим, что кроме Аннелиз в Камбодже также побывал Пол

В вопросе 2 начальный узел не указан, поэтому Cypher ищет в базе данных все узлы с выходным отношением типа "Has_been_in". Обычно начальный узел следует указывать там, где это возможно, потому что в зависимости от размера базы данных обработка таких запросов может занять много времени.

Если вы экспериментируете с данными для получения правильной графовой базы данных, вам придется часто удалять данные. В Cypher существует команда Delete для удаления небольших объемов данных. Следующий запрос показывает, как удалить все узлы и отношения в базе данных:

```
MATCH(n)
Optional MATCH (n)-[r]-()
Delete n,r
```

Итак, вы познакомились со связанными данными и примерно представляете, как они хранятся в графовых базах данных. Теперь можно сделать следующий шаг и перейти к рассмотрению реальных, практических применений связанных данных. Например, социальный граф может использоваться для нахождения кластеров тесно связанных узлов в сообществах внутри графа. Если участники кластера не знакомы друг с другом, их можно познакомить. Концепция поиска тесно связанных узлов (т. е. узлов, обладающих значительным количеством общих аспектов) находит широкое практическое применение. В следующем разделе мы воспользуемся этой концепцией для поиска кластеров в сети кулинарных ингредиентов.

7.3. Пример использования связанных данных: рекомендательная система

Одной из самых популярных областей применения графовых баз данных является разработка рекомендательных систем. Рекомендательные системы широко применяются, так как способны создавать релевантный контент. Изобилие данных в современной жизни оказывается непомерным для многих пользователей. Предприятия увидели очевидную необходимость в творческом подходе к привлечению клиентов посредством персонализированного контента и стали пользоваться преимуществами рекомендательных систем.

В нашем примере система будет рекомендовать рецепты блюд на основании предпочтений пользователей и сети ингредиентов. В ходе приготовления блюда мы воспользуемся системой Elasticsearch, чтобы ускорить процесс и в большей степени сосредоточиться на работе с графовой базой данных. Нашей главной целью будет замена списка ингредиентов «сырых» загруженных данных ингредиентами из нашего собственного «чистого» списка.

Если вы сразу перешли к этой главе, пропустив начало книги, возможно, стоит как минимум прочитать приложение А с описанием установки Elasticsearch. Вы всегда можете загрузить индекс, который будет использоваться в этой главе, со страницы загрузки издательства Manning и вставить его в локальный каталог данных Elasticsearch, если вы предпочитаете обойтись без хлопот с разбором примера из главы 6.

На сайте Manning можно загрузить следующую информацию для этой главы:

Три файла с программным кодом .py и их аналоги .ipynb:

- Data Preparation Part 1 – отправляет данные на Elasticsearch (также можно вставить индекс в локальную папку данных Elasticsearch).
- Data Preparation Part 2 — переменает данные с Elasticsearch в Neo4j.
- Исследование и рекомендательная система.

Три файла данных:

- Ingredients (.txt) — самостоятельно скомпилированный файл ингредиентов.
- Recipes (.json) – содержит все ингредиенты.
- Elasticsearch index (.zip) — содержит «гастрономический» индекс Elasticsearch, который можно использовать, чтобы пропустить фазу подготовки данных, часть 1.

Теперь, когда у вас есть все необходимое, рассмотрим цель исследования и шаги, которые необходимо предпринять для ее достижения.

7.3.1. Этап 1: Определение цели исследования

А теперь посмотрим, как проходит процесс data science (рис. 7.14).

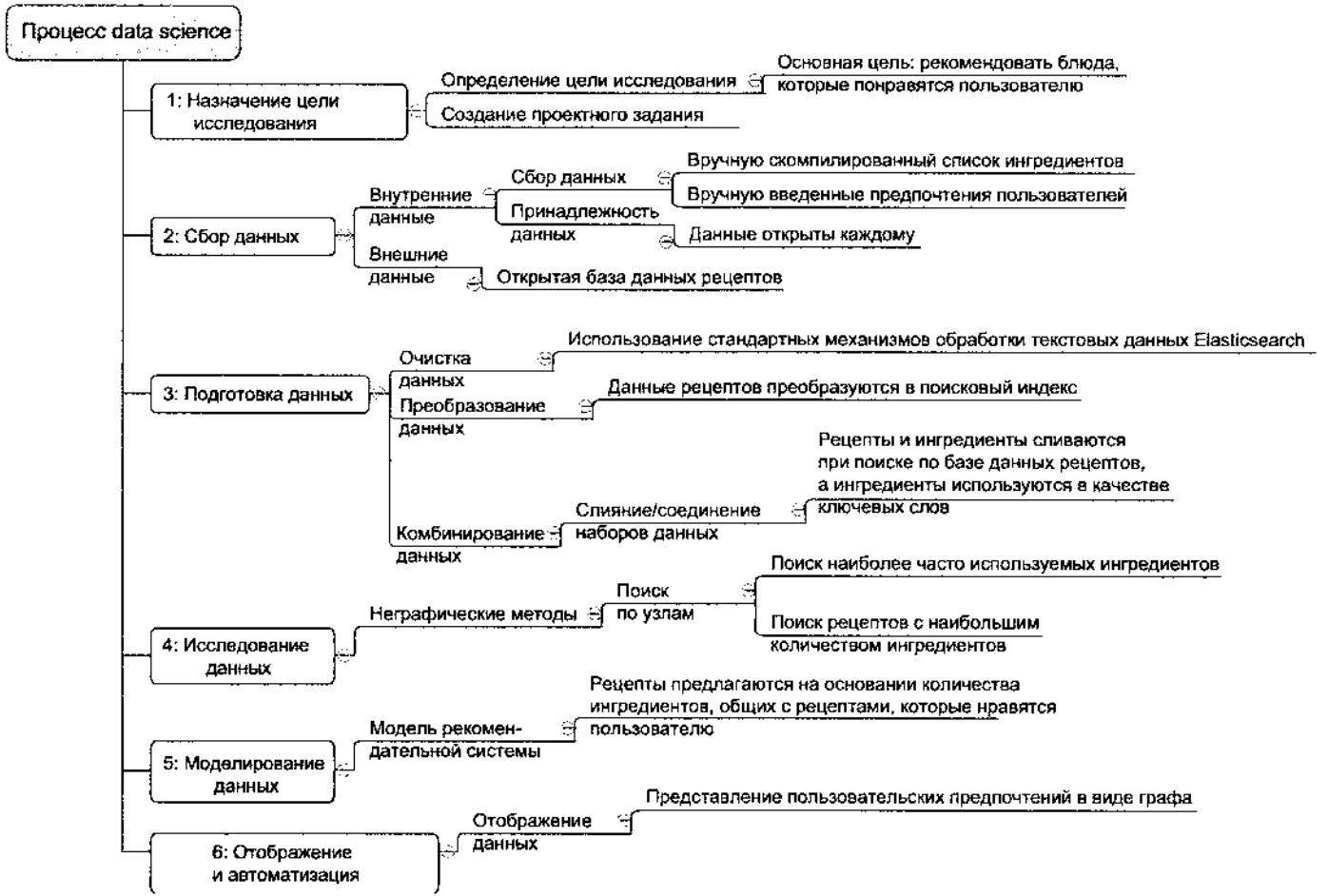


Рис. 7.14. Процесс data science, примененный к рекомендательной системе

Наша главная цель — создание рекомендательной системы, которая помогает пользователям кулинарных сайтов найти подходящие рецепты. Пользователь запрашивает несколько рецептов, и наши рекомендации блюд будут выбираться с учетом перекрытия ингредиентов в сети рецептов. Этот подход прост и интуитивно понятен, однако и он дает довольно точные результаты. Рассмотрим три элемента данных, которые для этого потребуются.

7.3.2. Этап 2: Сбор данных

В этом упражнении потребуются данные трех типов:

- Рецепты и их ингредиенты.
- Список разных ингредиентов, включаемых в модель.
- По крайней мере один пользователь и его предпочтения в выборе блюд.

Как всегда, данные можно разделить на созданные/доступные на собственных ресурсах и полученные извне.

- *Внутренние данные* — ни пользовательских предпочтений, ни ингредиентов под рукой нет, но это самая малая часть данных, которая создается относительно легко. Нескольких вручную введенных предпочтений должно быть достаточно для построения рекомендации. Чем больше информации сообщает пользователь, тем более интересные и точные результаты он получает. Пользовательские предпочтения будут вводиться позднее в ходе исследования. Список ингредиентов можно скомпилировать вручную, и он останется актуальным на годы, поэтому ничто не мешает вам использовать список в загружаемых материалах для любых целей (коммерческих или иных).
- *Внешние данные* — с рецептами все иначе. Существуют тысячи ингредиентов, которые могут объединяться в миллионы блюд. Впрочем, нам повезло: достаточно обширный список распространяется бесплатно по адресу *https://github.com/fictivekin/openrecipes*. Большое спасибо Fictive Kin за этот замечательный набор данных, содержащий более ста тысяч рецептов. Конечно, в списке встречаются дубликаты, но от них особого вреда не будет.

У нас в распоряжении два файла данных: список 800+ ингредиентов (ingredients.txt) и свыше ста тысяч рецептов в файле recipes.json. Короткий фрагмент списка ингредиентов приведен в листинге 7.2.

Листинг 7.2. Фрагмент текстового файла со списком ингредиентов

```
Ditalini
Egg Noodles
Farfalle
Fettuccine
Fusilli
Lasagna
Linguine
Macaroni
Orzo
```

Файл openrecipes в формате JSON содержит свыше ста тысяч рецептов с многочисленными свойствами: датой публикации, источником, временем приготовления, описанием и т. д. Нас интересуют только название и список ингредиентов. В листинге 7.3 приведен пример рецепта.

Листинг 7.3. Пример рецепта в формате JSON

```
{ "_id" : { "$oid" : "5160756b96cc62079cc2db15" },
    "name" : "Drop Biscuits and Sausage Gravy",
    "ingredients" : "Biscuits\n3 cups All-purpose Flour\n2 Tablespoons Baking
    Powder\n1/2 teaspoon Salt\n1-1/2 stick (3/4 Cup) Cold Butter, Cut Into
    Pieces\n1-1/4 cup Buttermilk\n SAUSAGE GRAVY\n1 pound Breakfast Sausage,
    Hot Or Mild\n1/3 cup All-purpose Flour\n4 cups Whole Milk\n1/2 teaspoon
    Seasoned Salt\n2 teaspoons Black Pepper, More To Taste",
    "url" : "http://thepioneerwoman.com/cooking/2013/03/drop-biscuits-and-
    sausage-gravy/",
    "image" : "http://static.thepioneerwoman.com/cooking/files/2013/03/
     bisgrav.jpg",
    "ts" : { "$date" : 1365276011104 },
    "cookTime" : "PT30M",
    "source" : "thepioneerwoman",
    "recipeYield" : "12",
    "datePublished" : "2013-03-11",
    "prepTime" : "PT10M",
    "description" : "Late Saturday afternoon, after Marlboro Man had returned
    home with the soccer-playing girls, and I had returned home with the..."
}
```

Так как мы работаем с текстовыми данными, задача состоит из двух частей: сначала нужно подготовить текстовые данные так, как описано в главе 8. Затем после тщательной очистки данные могут использоваться для получения рекомендаций, выработанных на основе сети ингредиентов. В этой главе тема подготовки текстовых данных не рассматривается, потому что она подробно изложена в другом месте.

7.3.3. Этап 3: Подготовка данных

Итак, в нашем распоряжении имеется два файла данных, которые необходимо объединить в одну графовую базу данных. «Грязные» данные рецептов создают проблему, которая может быть решена при помощи чистого списка ингредиентов и поискового ядра/базы данных NoSQL Elasticsearch. Мы уже воспользовались помощью Elasticsearch в предыдущей главе, а сейчас Elasticsearch будет использоваться для неявной чистки данных рецептов при создании индекса. Затем можно провести поиск по данным и связать каждый ингредиент с каждым рецептом, в котором он встречается. Вообще говоря, текстовые данные можно почистить в коде Python, как это делается в главе 8, но этот подход показывает, как полезно знать сильные стороны каждой базы данных NoSQL; не привязывайтесь к одной конкретной технологии, а используйте их вместе на пользу проекта.

Начнем с ввода данных рецептов в Elasticsearch. Если вы не понимаете, что здесь происходит, вернитесь к главе 6 — ситуация должна проясниться. Перед выполнением фрагмента кода из листинга 7.4 не забудьте включить локальный экземпляр Elasticsearch и активизировать среду Python с установленным модулем Elasticsearch. Мы не рекомендуем выполнять этот код «как есть» в Ipython (или Jupyter), потому что он выводит на экран все ключи рецептов, а возможности вашего браузера по выводу информации ограниченны. Либо отключите команды `print`, либо выполните код в другой интегрированной среде Python. Код этого фрагмента содержится в файле Data Preparation Part 1.py.

Листинг 7.4. Импортирование данных рецептов в Elasticsearch

```
from elasticsearch import Elasticsearch
import json

client = Elasticsearch ()
indexName = "gastronomical"
docType = 'recipes'

client.indices.create(index=indexName)

file_name = 'C:/Users/Gebruiker/Downloads/recipes.json'

recipeMapping = {
        'properties': {
            'name': {'type': 'string'},
            'ingredients': {'type': 'string'}
        }
    }

client.indices.put_mapping(index=indexName,doc_
type=docType,body=recipeMapping )

with open(file_name, encoding="utf8") as data_file:
    recipeData = json.load(data_file)

for recipe in recipeData:
    print recipe.keys()
    print recipe['_id'].keys()
    client.index(index=indexName,
        doc_type=docType,id = recipe['_id']['$oid'],
        body={"name": recipe['name'], "ingredients":recipe['ingredients']})
```

Импортирование модулей.

Клиент Elasticsearch используется для взаимодействия с базой данных.

Создание индекса.

Местонахождение файла рецептов JSON: приведите в соответствие со своей конфигурацией!

Отображение на тип Elasticsearch "recipe".

Загрузка файла рецептов JSON в память. Другой способ выглядит так:

```
recipeData = []
with open(file_name) as f:
 for line in f:
  recipeData.append(json.loads(line))
```

Индексирование рецептов. В данном случае важны только названия и ингредиенты. При возникновении проблемы тайм-аута можно увеличить величину задержки, передав, например, аргумент 30.

Если все было сделано правильно, в результате вы должны получить индекс Elasticsearch с именем «gastronomical», заполненный тысячами рецептов. Обратите внимание: мы разрешаем присутствие дубликатов рецептов, не назначая название рецепта в качестве ключа документа. Если, например, рецепт называется «лазанья», то это может быть лазанья с лососем, лазанья с говядиной, лазанья с курицей или лазанья любого другого типа. Не существует единственного рецепта, который является «прототипом» лазаньи; все рецепты загружаются на Elasticsearch под одним именем «лазанья». Это наш выбор, никто не мешает вам поступить иначе. Как вы вскоре увидите, принятое решение имеет значительные последствия: оно открывает возможность систематических подгрузок в локальную графовую базу данных. Проследите за тем, чтобы локальный экземпляр графовой базы данных был включен при применении следующего кода (листинг 7.5). В нашей базе данных используется имя пользователя по умолчанию *Neo4j* с паролем *Neo4ja*; приведите их в соответствие со своей локальной конфигурацией. Для этого вам также потребуется библиотека Python для Neo4j с именем `py2neo`. Если она еще не была установлена ранее, сейчас самое время установить ее в вашей виртуальной среде командой `pip install py2neo` или `conda install py2neo` (при использовании Anaconda). Еще раз напомним, что этот код приведет к сбою вашего браузера, если запустить его непосредственно в Ipython или Jupiter. Код этого фрагмента содержится в файле Data Preparation Part 2.py.

Листинг 7.5. Использование индекса Elasticsearch для заполнения графовой базы данных

```
from elasticsearch import Elasticsearch
from py2neo import Graph, authenticate, Node, Relationship

client = Elasticsearch ()
indexName = "gastronomical"
docType = 'recipes'

authenticate("localhost:7474", "user", "password")
graph_db = Graph("http://localhost:7474/db/data/")

filename = 'C:/Users/Gebruiker/Downloads/ingredients.txt'
ingredients =[]
with open(filename) as f:
    for line in f:
        ingredients.append(line.strip())        //

print ingredients

ingredientnumber = 0//
grandtotal = 0
for ingredient in ingredients:
```

Импортирование модулей.

Клиент Elasticsearch используется для взаимодействия с базой данных.

Аутентификация с вашим именем пользователя и паролем.

Сущность графовой базы данных.

Текстовый файл со списком ингредиентов загружается в память.

Вызов strip() необходим из-за символа \n, полученного при чтении из .txt.

Перебор ингредиентов и выборка результата Elasticsearch.

```
    try:
        IngredientNode = graph_db.merge_one("Ingredient","Name",ingredient)
    except:
        continue

    ingredientnumber +=1
    searchbody = {
        "size" : 99999999,
        "query": {
            "match_phrase":
                {
                    "ingredients":{
                       "query":ingredient,
                    }
                }
        }
    }
    result = client.search(index=indexName,doc_type=docType,body=searchbody)

    print ingredient
    print ingredientnumber
    print "total: " +  str(result['hits']['total'])

    grandtotal = grandtotal + result['hits']['total']
    print "grand total: " +  str(grandtotal)

    for recipe in result['hits']['hits']:

        try:
            RecipeNode =
     graph_db.merge_one("Recipe","Name",recipe['_source']['name'])
            NodesRelationship = Relationship(RecipeNode, "Contains",
IngredientNode)
            graph_db.create_unique(NodesRelationship)
            print "added: " + recipe['_source']['name'] + " contains " +
ingredient

        except:
            continue

    print "************************************"
```

Создание узла в графовой базе данных для текущего ингредиента.

Используется режим поиска совпадения для фразы, так как названия некоторых ингредиентов состоят из нескольких слов.

Перебор рецептов, найденных для этого конкретного ингредиента.

Создание узла для каждого рецепта, отсутствующего в базе данных.

Создание отношения между этим рецептом и ингредиентом.

Замечательно, у нас появилась графовая база данных, заполненная рецептами! Пришло время заняться исследованием связанных данных.

7.3.4. Этап 4: Исследование данных

Теперь, когда наши данные заняли положенное место, можно вручную исследовать их через интерфейс Neo4j по адресу *http://localhost:7474/browser/*.

Ничто не мешает вам выполнять свой код Cypher в этой среде, но его также можно выполнять с помощью библиотеки py2neo. Например, можно получить ответ на один интересный вопрос: какие ингредиенты встречаются чаще всего в разных рецептах? Какие из них с наибольшей вероятностью попадут в наш пищеварительный тракт, если случайно выбирать и есть блюда из этой базы данных?

```
from py2neo import Graph, authenticate, Node, Relationship
authenticate("localhost:7474", "user", "password")
graph_db = Graph("http://localhost:7474/db/data/")graph_db.cypher.execute("
   MATCH (REC:Recipe)-[r:Contains]->(ING:Ingredient) WITH ING, count(r) AS num
   RETURN ING.Name as Name, num ORDER BY num DESC LIMIT 10;")
```

Этот запрос, написанный на Cypher, означает: для всех рецептов и их ингредиентов подсчитать количество отношений на ингредиент, вернуть десять ингредиентов с наибольшим количеством отношений, а также соответствующие количества отношений. Результаты показаны на рис. 7.15.

	Name	num
1	Salt	53885
2	Oil	42585
3	Sugar	38519
4	Pepper	38118
5	Butter	35610
6	Garlic	29879
7	Flour	28175
8	Olive Oil	25979
9	Onion	24888
10	Cloves	22832

Рис. 7.15. 10 ингредиентов, которые чаще всего встречаются в рецептах

Большая часть десятки на рис. 7.15 вряд ли вас удивит. Список гордо возглавляет соль... стоит ли удивляться тому, что сердечно-сосудистые заболевания стали главной причиной смертности в большинстве западных стран? Но на ум приходит другой вопрос, требующий взглянуть на данные под другим углом: какие рецепты требуют больше всего ингредиентов?

```
from py2neo import Graph, Node, Relationship
graph_db = Graph("http://neo4j:neo4ja@localhost:7474/db/data/")
graph_db.cypher.execute("
    MATCH (REC:Recipe)-[r:Contains]->(ING:Ingredient) WITH REC, count(r) AS num
    RETURN REC.Name as Name, num ORDER BY num DESC LIMIT 10;")
```

Запрос практически не изменился, но вместо ингредиентов он возвращает рецепты. Результат показан на рис. 7.16.

	Name	num
1	Spaghetti Bolognese	59
2	Chicken Tortilla Soup	56
3	Kedgeree	55
4	Butternut Squash Soup	54
5	Hearty Beef Stew	53
6	Chicken Tikka Masala	52
7	Fish Tacos	52
8	Cooking For Others: 25 Years of Jor, 1 of BGSK	51
9	hibernation fare	50
10	Gazpacho	50

Рис. 7.16. 10 блюд с наибольшим количеством ингредиентов

Выглядит немного странно. Неужели такое блюдо, как «спагетти болоньезе», состоит из 59 ингредиентов? Давайте поближе присмотримся к ингредиентам «спагетти болоньезе».

```
from py2neo import Graph, Node, Relationship
graph_db = Graph("http://neo4j:neo4ja@localhost:7474/db/data/")
graph_db.cypher.execute("MATCH (REC1:Recipe{Name:'Spaghetti Bolognese'})-
[r:Contains]->(ING:Ingredient) RETURN REC1.Name, ING.Name;")
```

Запрос Cypher просто перечисляет ингредиенты, связанные с рецептом «спагетти болоньез». На рис. 7.17 показан результат в веб-интерфейсе Neo4j.

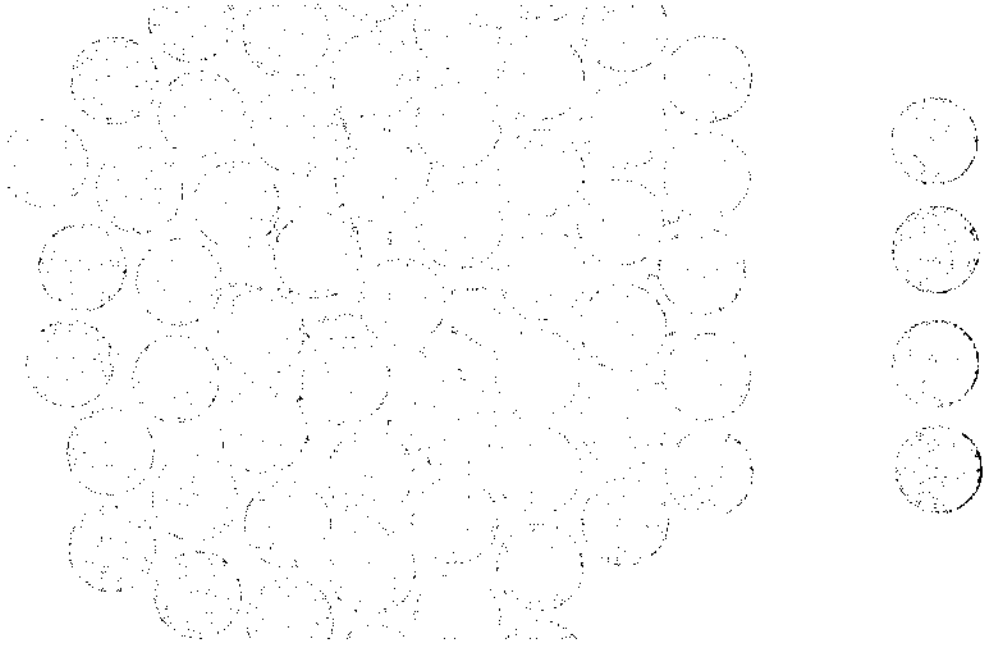


Рис. 7.17. Возможные ингредиенты «спагетти болоньезе»

Вспомните, что было сказано при индексировании данных в Elasticsearch. Быстрый поиск Elasticsearch показывает, что рецепт «спагетти болоньезе» встречается в базе данных несколько раз, и с каждым экземпляром связываются некоторые ингредиенты. «Спагетти болоньезе» следует рассматривать не как отдельный рецепт, а как коллекцию рецептов, по которым люди готовят собственные варианты этого блюда.

Отсюда следует интересная точка зрения на эти данные. Люди могут создавать свои версии блюда с кетчупом, красным вином и курицей, они даже могут добавлять в него бульон. Блюдо «спагетти болоньезе» можно изменять как угодно; неудивительно, что у него столько поклонников.

История со «спагетти болоньезе» была интересной, но мы здесь не за этим. Пришло время порекомендовать блюда нашим гурманам.

7.3.5. Этап 5: Моделирование данных

Слегка расширив свое понимание данных, мы добрались до цели упражнения: рекомендаций.

Для этого мы введем в модель пользователя с именем Рагнар (Ragnar), которому нравятся некоторые блюда. Новую информацию необходимо включить в графовую базу данных до того, как мы сможем ему что-то предложить. Создадим узел пользователя с несколькими предпочтениями.

В листинге 7.6 гурман Рагнар добавляется в базу данных вместе с его предпочтениями по нескольким блюдам. Если выбрать Рагнара в интерфейсе Neo4j, вы получите рис. 7.18. Соответствующий запрос Cypher выглядит так:

```
MATCH (U:User)-[r:Likes]->(REC:Recipe) RETURN U,REC LIMIT 25
```

Рисунок 7.18 обходится без сюрпризов; «спагетти болоньезе» нравится многим, и скандинавский гурман Рагнар не является исключением.

В простой рекомендательной системе, которую мы собираемся построить, остается лишь запросить у базы данных ближайшие по составу ингредиентов блюда. Еще раз подчеркнем, что это упрощенный подход к построению рекомендательных систем, так как в нем не учитываются многие факторы:

- Отвращение к некоторым ингредиентам или блюдам.
- Степень любви или отвращения. Оценка по шкале от 1 до 10 вместо бинарного признака «нравится/не нравится» может сыграть важную роль.
- Количество ингредиента, присутствующего в блюде.
- Порог, на котором проявляется влияние ингредиента на вкус блюда. Некоторые ингредиенты (например, перец) сильнее влияют на вкус даже в меньшем количестве.

- Пищевые аллергии. Хотя этот фактор будет неявно смоделирован в отношении пользователя к блюдам, содержащим определенные ингредиенты, пищевая аллергия может быть настолько важна, что всего одна ошибка может иметь фатальные последствия. Исключение аллергенов радикально изменяет всю систему рекомендаций.
- Еще много всего, о чем стоит подумать.

Листинг 7.6. Создание узла пользователя, которому нравятся некоторые рецепты, в базе данных Neo4j

```
from py2neo import Graph, Node, Relationship

graph_db = Graph("http://neo4j:neo4ja@localhost:7474/db/data/")

UserRef = graph_db.merge_one("User","Name","Ragnar")

RecipeRef = graph_db.find_one("Recipe",property_key="Name",
     property_value="Spaghetti Bolognese")
NodesRelationship = Relationship(UserRef, "Likes", RecipeRef)
graph_db.create_unique(NodesRelationship) #Commit his like to database

graph_db.create_unique(Relationship(UserRef, "Likes",
    graph_db.find_one("Recipe",property_key="Name",
    property_value="Roasted Tomato Soup with Tiny Meatballs
    and Rice")))
graph_db.create_unique(Relationship(UserRef, "Likes",
    graph_db.find_one("Recipe",property_key="Name",
    property_value="Moussaka")))
graph_db.create_unique(Relationship(UserRef, "Likes",
    graph_db.find_one("Recipe",property_key="Name",
    property_value="Chipolata &amp;amp; spring onion frittata")))
graph_db.create_unique(Relationship(UserRef, "Likes",
    graph_db.find_one("Recipe",property_key="Name",
    property_value="Meatballs In Tomato Sauce")))
graph_db.create_unique(Relationship(UserRef, "Likes",
    graph_db.find_one("Recipe",property_key="Name",
    property_value="Macaroni cheese")))
graph_db.create_unique(Relationship(UserRef, "Likes",
    graph_db.find_one("Recipe",property_key="Name",
    property_value="Peppered Steak")))
```

Создание нового пользователя с именем "Ragnar"

Импортирование модулей.

Создание объекта подключения к графовой базе данных.

Рагнару нравится блюдо "спагетти болоньезе".

Процесс повторяется для нескольких других блюд.

Создание отношения "like" между Рагнаром и спагетти.

Поиск рецепта с названием "Spaghetti Bolognese".

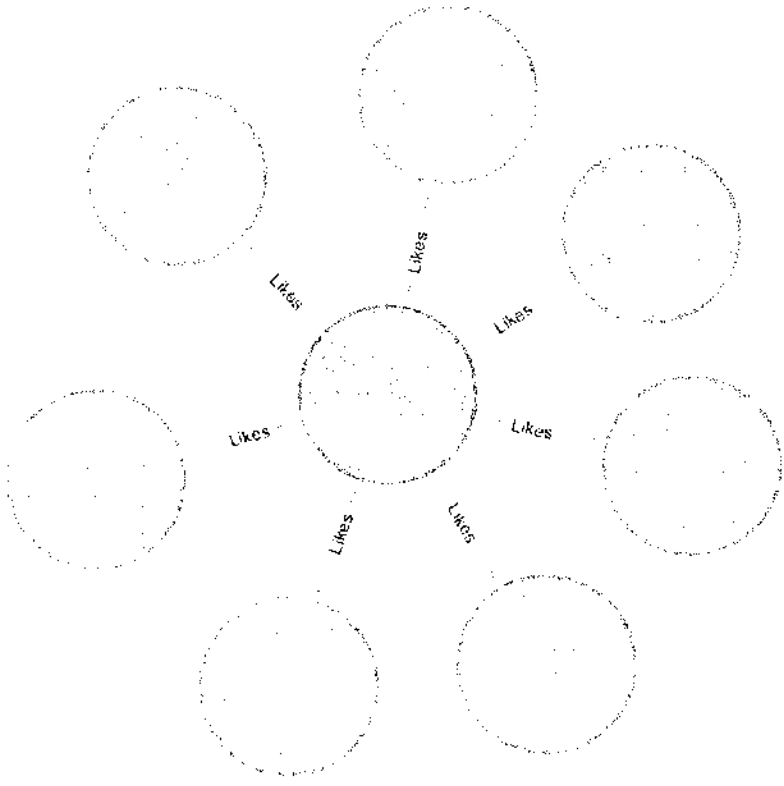


Рис. 7.18. Блюда, которые нравятся Рагнару

Как ни странно, для получения нужной информации достаточно всего одной команды Cypher:

```
from py2neo import Graph, Node, Relationship
graph_db = Graph("http://neo4j:neo4ja@localhost:7474/db/data/")
graph_db.cypher.execute("
   MATCH (USR1:User{Name:'Ragnar'})-[l1:Likes]->(REC1:Recipe),
         (REC1)-[c1:Contains]->(ING1:Ingredient)
      WITH  ING1,REC1 MATCH (REC2:Recipe)-[c2:Contains]->(ING1:Ingredient)
      WHERE REC1 <> REC2
   RETURN REC2.Name,count(ING1) AS IngCount ORDER BY IngCount DESC LIMIT 20;")
```

Сначала команда получает все рецепты, которые нравятся Рагнару. Ингредиенты этих рецептов используются для поиска других блюд, в состав которых они входят. Далее ингредиенты подсчитываются для каждого блюда, и данные сортируются по количеству убывания общих ингредиентов. В список включаются только 20 блюд из начала списка; результат показан в таблице на рис. 7.19.

	REC2.Name	IngCount
1	Spaghetti and Meatballs	104
2	Hearty Beef Stew	91
3	Cassoulet	89
4	Lasagne	88
5	Spaghetti & Meatballs	86
6	Good old lasagne	84
7	Beef Wellington	84
8	Braised Short Ribs	83
9	Lasagna	83
10	Italian Wedding Soup	82
11	French Onion Soup	82
12	Coq au vin	82
13	Shepherd's pie	81
14	Great British pork: from head to toe	81
15	Three Meat Cannelloni Bake	81
16	Cioppino	81
17	hibernation fare	80
18	Spaghetti and Meatballs Recipe with Oven Roasted Tomato Sauce	80
19	Braised Lamb Shanks	80
20	Lamb and Eggplant Casserole (Moussaka)	80

Рис. 7.19. Результат работы рекомендательной системы; 20 блюд в начале списка могут понравиться пользователю

Из рис. 7.19 можно сделать вывод, что Рагнару стоит попробовать спагетти с фрикадельками (`Spaghetti and Meatballs`) — блюдо, которое обрело бессмертную славу после диснеевского мультфильма «Леди и Бродяга». Похоже, эта рекомендация прекрасно подойдет всем, кто любит блюда с макаронными изделиями и фрикадельками, но, как видно из счетчика ингредиентов, эту рекомендацию подкрепляет множество других ингредиентов. Чтобы дать представление о том, на чем основана эта рекомендация, можно вывести любимые блюда Рагнара, первоочередные рекомендации и некоторые перекрывающиеся ингредиенты на одном сводном графе.

7.3.6. Этап 6: Отображение

Веб-интерфейс Neo4j позволяет запустить модель и получить симпатичный граф, поясняющий часть логики, на которой базируются рекомендации. Граф показывает, каким образом рекомендуемые блюда связаны с любимыми блюдами через ингредиенты. Граф на рис. 7.20 завершает рассмотрение учебного примера.

Этот красивый граф завершает главу; похоже, Рагнару предстоит попробовать еще немало вкусных блюд. Не забудьте опробовать рекомендательную систему на собственных предпочтениях.

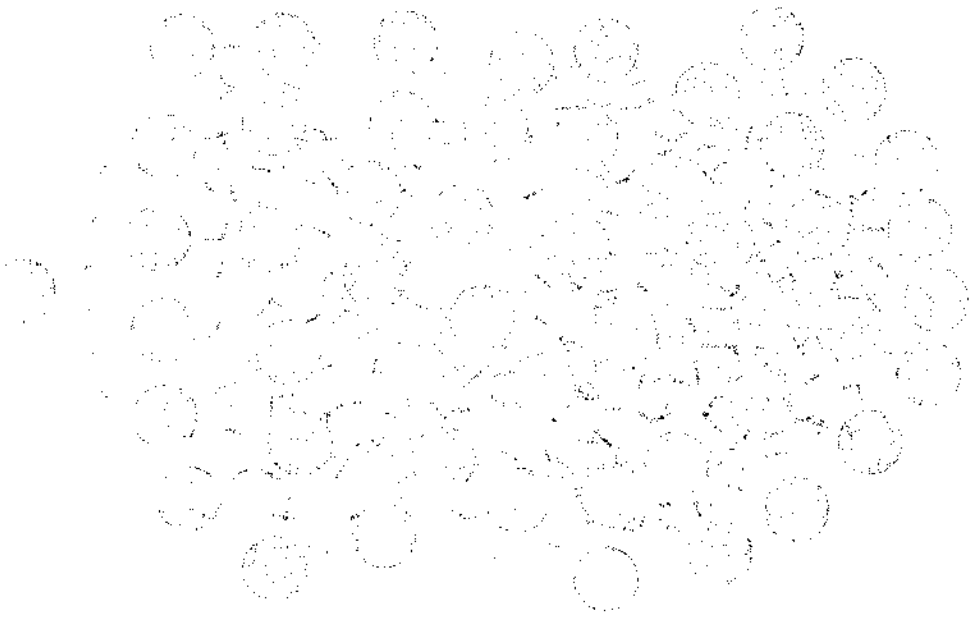


Рис. 7.20. Взаимосвязь любимых блюд пользователя и 10 рекомендуемых блюд через субвыборку перекрывающихся ингредиентов

7.4. Итоги

Основные положения этой главы:

- Графовые базы данных особенно полезны при работе с данными, в которых отношения между сущностями не менее важны, чем сами сущности. По сравнению с другими базами данных NoSQL они лучше всего проявляют себя при максимальной сложности в сочетании с небольшим объемом данных.
- Графовые структуры данных состоят из двух основных компонентов:
  - *Узлы* — сами сущности. В нашем примере это рецепты и ингредиенты.
  - *Ребра* — отношения между сущностями. Отношения, как и сущности, могут иметь разные типы («содержит», «любит», «бывал в» и т. д.) и обладать собственными свойствами: именами, весами и другими метриками.
- Мы рассмотрели Neo4j, самую популярную в настоящее время графовую базу данных. За инструкциями по установке обращайтесь к приложению Б. Мы рассмотрели, как добавить новые данные в Neo4j, как запросить данные в Cypher и как работать с веб-интерфейсом Neo4j.

- Cypher — язык запросов базы данных Neo4j. Мы рассмотрели несколько примеров его использования, а также применили в учебном примере — рекомендательной системе по выбору блюд.
- В примере этой главы мы воспользовались Elasticsearch для очистки огромной «свалки данных» с рецептами. Данные были преобразованы в базу данных Neo4j с рецептами и ингредиентами. Цель примера заключалась в том, чтобы порекомендовать пользователям рецепты на основании их интереса к другим блюдам, проявленного ранее. Для этого мы воспользовались связностью рецептов по ингредиентам. Для взаимодействия с сервером Neo4j из кода Python была использована библиотека py2neo.
- Как выясняется, графовые базы данных могут использоваться не только для реализации рекомендательных систем, но и для исследования данных. Одним из интересных фактов, обнаруженных в ходе анализа, является разнообразие (в отношении ингредиентов) существующих рецептов «спагетти болоньезе».
- Веб-интерфейс Neo4j был использован для создания визуального представления логики перехода от предпочтений к рекомендациям через узлы ингредиентов.

Глубокий анализ текста

В этой главе:

- ✓ Важность глубокого анализа текста.
- ✓ Важнейшие концепции глубокого анализа текста.
- ✓ Проект из области глубокого анализа текста.

Большая часть информации, сохраняемой людьми в мире, существует в виде текста. Мы учимся читать и писать еще в детстве, чтобы выражать свои мысли и узнавать, что знают, думают и чувствуют другие люди. Мы постоянно используем эти навыки во время чтения или написания электронного письма, сообщений в блогах, текстовых сообщений или этой книги, поэтому неудивительно, что письменный язык представляется большинству из нас естественным. Деловые круги убеждены, что из текстов, написанных людьми, можно извлечь большую пользу, — и совершенно справедливо, потому что эти тексты содержат информацию о том, что нравится или не нравится людям, что они знают или хотят узнать, чего они желают или опасаются, в каком состоянии здоровья или настроении они находятся и т. д. Эта информация очень часто представляет ценность для компаний или исследователей, но одному человеку ни за что не удастся прочитать и интерпретировать этот объем письменных материалов своими силами. И снова приходится прибегать к помощи компьютеров, которые выполняют эту работу за нас.

Как ни печально, компьютеру естественный язык вовсе не кажется естественным. Извлечение смысла и отделение главного от второстепенного — задачи, с которыми человек все еще справляется лучше любой машины. К счастью, специалисты data science могут воспользоваться методами *глубокого анализа текста* (text mining) для поиска актуальной информации в горах текста, на прочтение которых ушли бы века.

Глубокий анализ текста — дисциплина, объединяющая науку о языке и информатику со статистикой и методами машинного обучения. Глубокий анализ текста используется для анализа текстов и преобразования их в более структурированную форму. Затем на основании этой структурированной формы делаются какие-то

выводы. Например, при анализе преступлений из полицейских отчетов глубокий анализ поможет выделить из отчета имена людей, названия мест и типы преступлений. Затем эта новая структура используется для извлечения информации об эволюции преступлений (рис. 8.1).

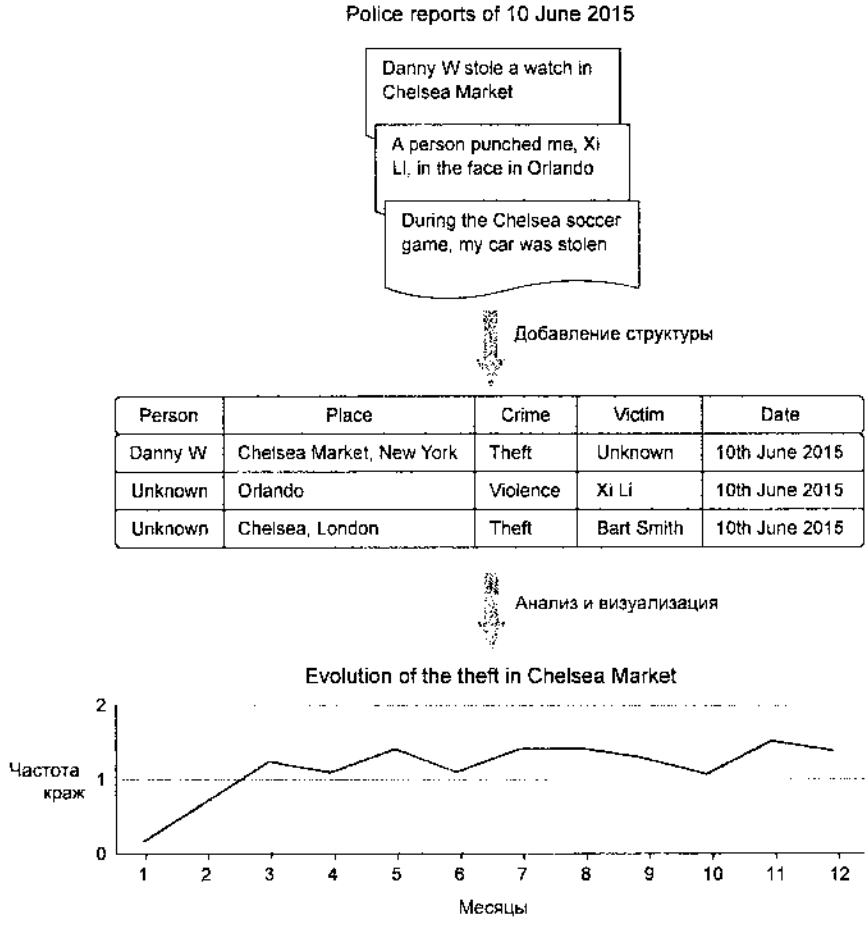


Рис. 8.1. В глубоком анализе текста первой проблемой (обычно) является структурирование входного текста; затем этот текст может быть тщательно проанализирован

Хотя глубокий анализ не ограничивается естественным языком, основной темой этой главы станет обработка текста на естественном языке (NLP, Natural Language Processing). Примеры текста на искусственном языке – журналы, математические формулы и код Морзе. Формально эсперанто, клингонский язык и язык драконов не относятся к области естественных языков, потому что они были изобретены сознательно и не развивались со временем; они не появились «естественным» образом. Тем не менее эти языки пригодны для естественных коммуникаций (речь,

письмо); в них, как и в естественных языках, существует свой словарный запас и грамматика, и к ним могут применяться те же методы глубокого анализа текста.

8.1. Глубокий анализ текста в реальном мире

Вы уже сталкивались с практическими применениями глубокого анализа текста и естественных языков в своей повседневной жизни. Средства автозаполнения и автоматической проверки правописания постоянно анализируют вводимый текст при отправке сообщений электронной почты или текста. Когда Facebook автоматически дополняет ваш статус именем друга, при этом используется метод, называемый *распознаванием сущностей* (entity recognition), хотя это лишь одна из его возможностей. Необходимо не только распознать, что вы вводите существительное, но и предположить, что вы желаете упомянуть человека, и предположить, кто бы это мог быть. Другой пример распознавания именованных сущностей представлен на рис. 8.2. Google знает, что «Челси» — футбольный клуб, но выдает другой результат, когда пользователь запрашивает информацию о человеке по имени Челси.

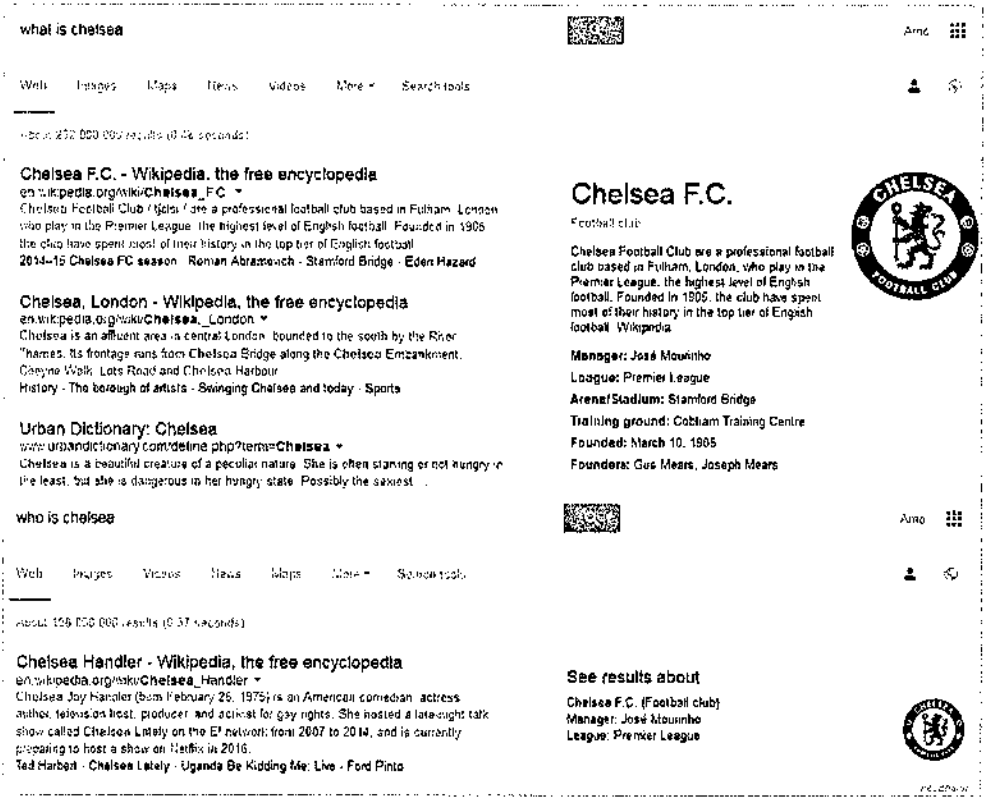


Рис. 8.2. Разные ответы на вопросы наводят на мысль, что Google применяет методы глубокого анализа текста при поиске ответа

Google использует много разных типов глубокого анализа текста для представления результатов запроса. Что вам приходит в голову, когда вы слышите «Челси»? Вариантов много: это может быть человек, футбольный клуб, район Нью-Йорка или Лондона. Google знает это и возвращает разные результаты в зависимости от формулировки запроса. Чтобы предоставить наиболее подходящий ответ, Google необходимо сделать (среди прочего) следующее:

- Провести предварительную обработку всех документов, собранных для именованных сущностей.
- Выполнить идентификацию языка.
- Определить, на какой тип сущности ссылается пользователь.
- Получить результат, соответствующий запросу.
- Определить тип возвращаемого содержимого (PDF, возрастные ограничения).

Этот пример показывает, что глубокий анализ текста не только связан с непосредственным смыслом текста, но и включает такие метаатрибуты, как язык и тип документа.

Google применяет глубокий анализ текста не только для ответов на вопросы. Кроме защиты пользователей Gmail от спама глубокий анализ текста также используется для разделения сообщений на категории (рис. 8.3).

Рис. 8.3. Сообщения электронной почты могут делиться на категории с учетом их содержания и источника

Объединив обработку текста с дополнительной логикой и математикой, вы сможете выйти далеко за рамки ответов на простые вопросы. В частности, появляется возможность создания *систем формирования рассуждений* (reasoning engines), управляемых запросами на естественном языке. Рисунок 8.4 показывает, как база знаний Wolfram Alpha использует глубокий анализ текста и автоматизированные рассуждения для получения ответа на вопрос о соотношении численности населения в США и в Китае.

Если вас это недостаточно впечатлило, то расскажем, что система IBM Watson произвела сенсацию в 2011 году, когда машина победила в Jeopardy двух игроков-людей (Jeopardy — американская телевизионная викторина, в которой игроки получают ответ на вопрос и пытаются подобрать правильный вопрос для этого ответа) — рис. 8.5.

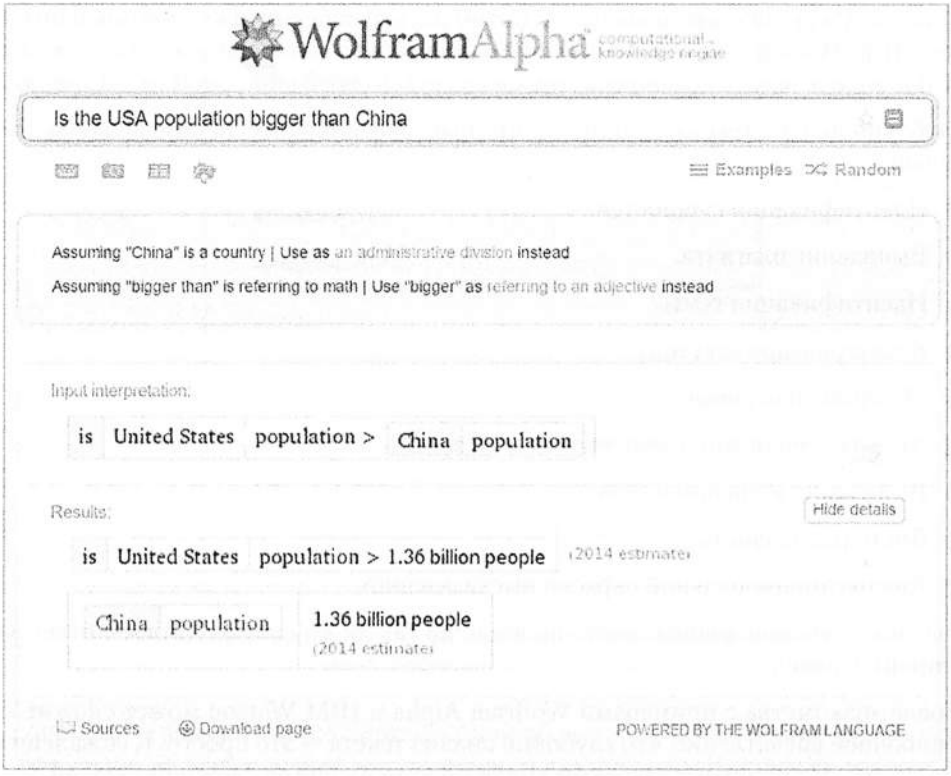


Рис. 8.4. Система Wolfram Alpha использует глубокий анализ текста и логические рассуждения для поиска ответов на вопросы

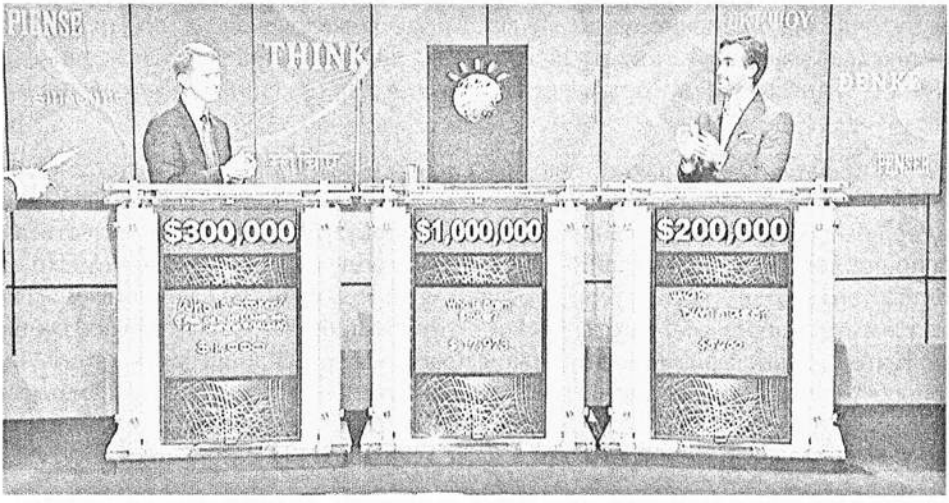


Рис. 8.5. IBM Watson побеждает людей в Jeopardy

Можно с уверенностью заявить, что в этом раунде победил искусственный интеллект. IBM Watson — когнитивная система, способная интерпретировать естественный язык и отвечать на вопросы, руководствуясь своей обширной базой знаний.

Глубокий анализ текста находит много практических применений, включая следующие:

- Идентификация сущностей.
- Выявление плагиата.
- Идентификация темы.
- Кластеризация текстов.
- Машинный перевод.
- Автоматизированное обобщение текста.
- Выявление мошенничества.
- Фильтрация спама.
- Анализ эмоциональной окраски высказываний.

Словом, глубокий анализ текста полезен, но так ли он сложен? Боюсь, ответ вас огорчит: сложен.

После знакомства с примерами Wolfram Alpha и IBM Watson может сложиться ошибочное впечатление, что глубокий анализ текста - - это просто. К сожалению, это не так. На практике глубокий анализ текста является сложной задачей, и он недостаточно хорошо справляется даже со многими простыми на первый взгляд операциями. Для примера возьмем задачу поиска адреса. Рисунок 8.6 наглядно показывает, как трудно получить точный результат и как Google Maps запрашивает уточняющую информацию. В данном случае человек без дополнительного контекста справился бы ничуть не лучше, но такая неоднозначность, лишь одна из многих проблем, с которыми сталкиваются приложения глубокого анализа текста.

Другая проблема — опечатки и разные (правильные) формы написания слова. Возьмем три обозначения: «NY», «Neww York» и «New York». Для человека очевидно, что все они относятся к городу Нью-Йорку. Человек вполне естественно воспринимает текст, содержащий ошибки; иногда мы их даже не замечаем. Но для компьютера это совершенно разные строки, если только мы не воспользуемся специальными алгоритмами, которые укажут, что эти строки относятся к одной сущности. К этой же категории относятся такие проблемы, как синонимы и местоимения. Попробуйте понять, к какому человеку относится слово «она» в следующем отрывке: «Когда Джон впервые встретился с родителями Мэри, он подарил им цветы. Она была обрадована этим жестом». Скажете, просто? Но не для компьютера.

Многие похожие задачи легко решаются человеком, но оказываются достаточно сложными для машины. Мы можем натренировать алгоритмы, хорошо работающие

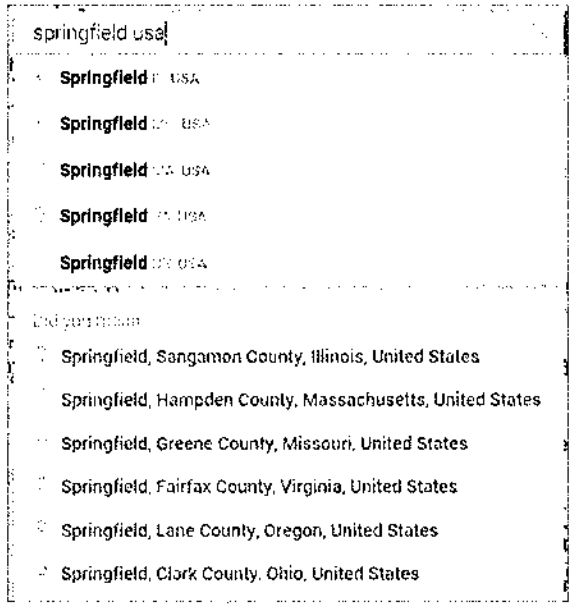


Рис. 8.6. Google Maps запрашивает дополнительный контекст из-за неоднозначности запроса

в конкретной задаче в четко определенной области, но более общие алгоритмы, работающие во всех случаях, — совершенно другое дело. Например, мы можем обучить компьютер распознавать и извлекать из текста номера банковских счетов в США, но это решение не будет хорошо обобщаться для номеров банковских счетов из других стран.

Языковые алгоритмы также чувствительны к контексту, в котором используется язык, даже если сам язык остается неизменным. Англоязычные модели не подойдут для арабского языка и наоборот, но даже если придерживаться английского, алгоритм, натренированный на данных Твиттера, вряд ли будет хорошо работать на юридических текстах. Помните об этом, когда мы перейдем к учебному примеру этой главы: в области глубокого анализа текста не существует идеального решения «на все случаи жизни».

8.2. Методы глубокого анализа текста

В предстоящем учебном примере мы займемся проблемой классификации текста: автоматической классификацией текста по категориям. Для перехода от «сырых» текстовых данных к конечному состоянию нам потребуются методы глубокого анализа текста, для эффективного использования которых нам понадобится дополнительная информация. Итак, первая важная концепция из области глубокого анализа текста — набор слов.

8.2.1. Набор слов

Для построения классификационной модели мы воспользуемся концепцией набора слов. *Набор слов* (bag of words) — простейший способ структурирования текстовых данных: каждый документ преобразуется в вектор слов. Если некоторое слово присутствует в векторе, оно снабжается флагом `True`; остальные слова помечаются флагом `False`. На рис. 8.7 изображен упрощенный пример всего с двумя документами: первый относится к сериалу «Игра престолов», а второй к data science. Два вектора совместно образуют *матрицу «документ–термин»*; эта матрица содержит столбец для каждого слова (термина) и строку для каждого документа. Значение элемента матрицы выбирается вами. В этой главе мы будем использовать простой флаг присутствия слова в тексте (`True` или `False`).

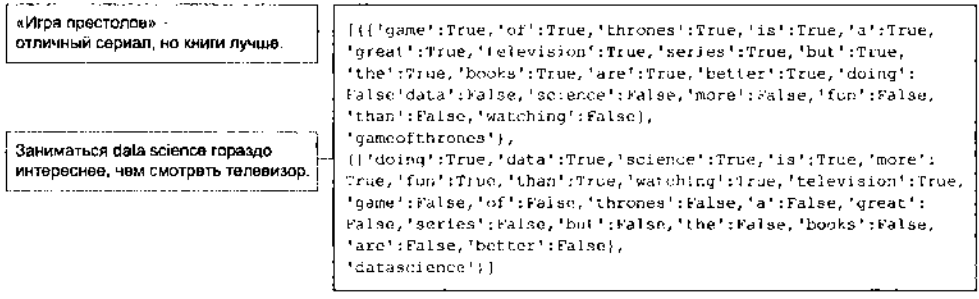


Рис. 8.7. Текст преобразуется в набор слов: каждое слово (термин) помечается флагом True, если слово встречается в документе, или флагом False, если не встречается

Пример на рис. 8.7 дает представление о структурированных данных, необходимых для начала анализа текста, но этот пример сильно упрощен: ни одно слово не отфильтровано, выделение основы (об этом позднее) не применялось. Большой корпус текстов может содержать тысячи уникальных слов. Если все они будут помечаться без фильтрации, мы получим большой объем данных. Двоичная кодировка набора слов на рис. 8.7 — всего лишь один способ структурирования данных; существуют и другие.

TF-IDF

Для заполнения матрицы «документ–термин» часто применяется формула TF-IDF (Term Frequency-Inverse Document Frequency, «частота слова — обратная частота документа»). В двоичном наборе слов элементам присваиваются значения True или False (термин присутствует или отсутствует), а простые частоты определяют количество вхождений термина. Метрика TF-IDF более сложна; она учитывает, сколько раз термин встречается в до-

кументе (TF). Значением TF может быть простой счетчик, двоичное значение (True/False) или счетчик с логарифмическим масштабированием — все зависит от того, какой вариант лучше подходит в вашем случае. Частота встречаемости термина

$$TF = f_{t,d},$$

где TF — частота встречаемости (f) термина (t) в документе (d).

С другой стороны, метрика TF-IDF также учитывает все остальные документы. IDF дает представление о том, насколько распространено слово во всем корпусе текстов. Чем выше IDF, тем чаще встречается слово, а слишком часто встречающиеся слова не несут информации. Например, английские слова «a» или «the» вряд ли сообщают что-нибудь о тексте. Самой распространенной формой IDF является формула IDF с логарифмическим масштабированием:

$$IDF = \log(N / |\{d \in D : t \in\}|),$$

где N — общее количество документов в корпусе текстов, а $|\{d \in D : t \in\}|$ — количество документов (d), в которых встречается термин (t).

Метрика TF-IDF отвечает на вопрос: насколько важно это слово для того, чтобы отличать этот документ от других документов в корпусе текстов? Формула TF-IDF выглядит так:

$$\frac{1F}{IDF} = f_{t,d} / \log(N | \{d \in D : t \in\}|),$$

Мы не будем использовать TF-IDF в примере, но когда вы начнете делать первые шаги в области глубокого анализа текстов, вы непременно столкнетесь с этой метрикой. TF-IDF также использовалась системой Elasticsearch (незаметно для вас) в главе 6. Если вы захотите применить TF-IDF для аналитики текста, поручите глубокий анализ специализированным программам (таким, как SOLR или Elasticsearch) и возьмите полученную матрицу «документ—термин» для дальнейшей обработки.

Прежде чем вы получите собственно набор слов, происходит много других операций обработки данных:

- *Генерирование лексем* — текст разбивается на фрагменты, называемые «лексемами», или «терминами». Эти лексемы — простейшая единица информации, используемая в вашей модели. Лексемы часто совпадают со словами, но это не обязательно. Для анализа могут использоваться целые предложения. Мы будем использовать *униграммы*: лексемы, состоящие из одного слова. Впрочем,

часто бывает полезно включать в анализ *диграммы* (два слова на лексему) или *триграммы* (три слова на лексему), чтобы извлечь дополнительное смысловое содержимое и повысить эффективность моделей. Впрочем, за это расширение приходится расплачиваться: включение диграмм и (или) триграмм приводит к увеличению размера векторов терминов.

- *Фильтрация игнорируемых слов* — в каждом языке существуют слова, которые используются настолько часто, что их можно не учитывать в ходе глубокого анализа текста. Если текст разбивается на лексемы по словам, часто есть смысл удалить малоинформативные игнорируемые слова из векторов.
- *Преобразование к нижнему регистру* — слова, стоящие в начале предложения, обычно начинаются с символа верхнего регистра. Также с символа верхнего регистра могут начинаться имена собственные. Различие в регистре символов только усложнит матрицу терминов, поэтому все термины следует преобразовать к нижнему регистру.

Еще один метод подготовки данных — *выделение основы*. Впрочем, он заслуживает более подробного рассмотрения.

8.2.2. Выделение основы и лемматизация

Выделением основы (stemming) называется приведение слова к корневой форме; оно снижает вариативность данных. Это особенно полезно, если слова имеют сходный смысл, но пишутся по-разному: например, если одно из них записано во множественном числе. Метод выделения основы пытается перейти к корневой форме отсечением части слова: например, и «planes», и «plane» преобразуются в «plane».

Другой метод, называемый *лемматизацией* (lemmatization), преследует ту же цель, но делает это более осмысленно с точки зрения грамматики. Например, хотя и выделение основы и лемматизация преобразуют «cars» в «car», лемматизация также может вернуть спрягаемую форму глагола к исходной, скажем, преобразовать «are» в «be». Выбор зависит от конкретной ситуации, и лемматизация интенсивно использует *разметку частей речи* (Part Of Speech Tagging, POS Tagging).

Разметкой частей речи называется процесс назначения грамматической метки каждой части предложения. Вероятно, вы уже занимались этим в школе. Для примера возьмем предложение «Game of Thrones is a television series». Применяя к ней разметку частей речи, мы получим

```
({"game":"NN"},{"of":"IN},{"thrones":"NNS},{"is":"VBZ},{"a":"DT},
              {"television":"NN}, {"series":"NN})
```

Здесь NN — существительное, IN — предлог, NNS — существительное во множественном числе, VBZ — глагол 3-го лица единственного числа, а DT — определяющее слово. Некоторые теги перечислены в табл. 8.1 .

Таблица 8.1. Список всех тегов разметки частей речи

Тег	Описание	Тег	Описание
CC	Соединительный союз	CD	Числительное
DT	Определяющее слово	EX	Связка существования
FW	Иноязычное слово	IN	Предлог или подчинительный союз
JJ	Прилагательное	JJR	Прилагательное (сравнительное)
JJS	Прилагательное (в превосходной степени)	LS	Маркер пункта списка
MD	Модальный глагол	NN	Существительное в единственном числе или неисчисляемое
NNS	Существительное во множественном числе	NNP	Имя собственное в единственном числе
NNPS	Имя собственное во множественном числе	PDT	Предетерминатив

Разметка частей речи ориентирована на уровень фраз, а не слов. После того как разметка частей речи будет завершена, можно переходить к лексемам уровня слов, но для разметки частей речи нужны целые фразы. Объединение разметки частей речи и лемматизации с большой вероятностью даст более чистые данные, чем при использовании только выделения основы. Для простоты в своем примере мы ограничимся выделением основы, но вы при желании можете воспользоваться этой возможностью для расширения этого упражнения.

Итак, мы выяснили, что самое важное — очистка данных и их обработка (глубокий анализ текста). Добавим в наш арсенал аналитики еще один прием — классификатор на базе дерева принятия решений.

8.2.3. Классификатор на базе дерева принятия решений

Фаза анализа данных в нашем учебном примере также будет по возможности простой. Мы протестируем наивный классификатор Байеса и классификатор на базе дерева принятия решений. Как было показано в главе 3, наивный классификатор Байеса называется так, потому что он предполагает, что каждая входная переменная независима от других, а это предположение наивно, особенно в области глубокого анализа текста. Очевидно, что при простой «нарезке» данных на униграммы будут потеряны смысловые связи. Эта проблема может быть решена созданием диграмм и триграмм.

С другой стороны, классификатор на базе дерева принятия решений не считает переменные независимыми друг от друга и активно создает переменные взаимодействия и гнезда. *Переменная взаимодействия* объединяет другие переменные.

Например, слова «анализ» и «данных» сами по себе могут быть хорошими прогностическими переменными, но, вероятно, их частое соседство в одном тексте может обладать самостоятельной ценностью. Гнездо (bucket) имеет отчасти противоположный смысл. Вместо того чтобы объединять другие переменные, переменная разбивается на несколько новых переменных. Такой подход имеет смысл для числовых переменных. На рис. 8.8 показано, как может выглядеть дерево принятия решений и где в нем применяются переменные взаимодействия и гнезда.

Рис. 8.8. Вымышленная модель дерева принятия решений. Дерево принятия решений автоматически создает гнезда и предполагает зависимости между входными переменными

Если наивный классификатор Байеса предполагает независимость всех входных переменных, дерево принятия решений строится на основе предположений о независимости. Но как строится эта структура? У дерева принятия решений есть несколько возможных критериев, которые могут использоваться для разбиения на ветви и принятия решений о том, какие переменные более важны (и располагаются ближе к корню дерева), чем другие. В нашем классификаторе на базе дерева принятия решений NLTK будет использоваться критерий «прироста информации». Чтобы понять его смысл, необходимо сначала обратиться к понятию энтропии. *Энтропия* является мерой непредсказуемости, или хаоса. Простой пример — пол младенца. Во время беременности пол плода заранее неизвестен. Попытка угадать его будет успешной на 50% (примерно, потому что распределение полов не является равномерным на 100%). Однако во время беременности можно провести УЗИ для определения пола плода. Результаты диагностики никогда не являются 100% точными, но их точность растет по мере развития плода. Этот *прирост точности* обусловлен падением неопределенности, или энтропии. Допустим, УЗИ на 12-й неделе определяет пол будущего младенца с точностью 90%. 10%-ная неопределенность все еще остается, но диагностика сократила ее с 50 до 10%. Это довольно неплохой признак классификации. Дерево принятия решений действует по тому же принципу (рис. 8.9).

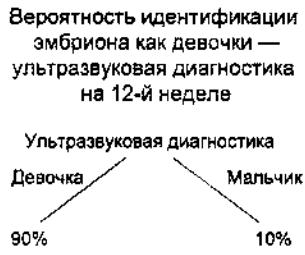


Рис. 8.9. Дерево принятия решений с одной переменной: решением врача, принятым после знакомства с данными ультразвуковой диагностики во время беременности. Какова вероятность того, что родится девочка?

Если существует другой тест, обладающий большей прогностической способностью, он может стать корнем дерева, а ультразвуковой тест переместится в ветви; процесс может продолжаться до тех пор, пока не кончатся переменные или наблюдения. Исчерпание наблюдений возможно, потому что при каждом ветвлении также происходит отсечение части входных данных. Это большая слабость дерева принятия решений, потому что на листовом уровне дерева нехватка наблюдений может привести к нарушению надежности модели; дерево старается обеспечить чрезмерно точную подгонку данных, в результате которой случайные отклонения могут быть приняты за настоящие корреляции. Чтобы этого не происходило, дерево принятия решений усекается: его ветви, не имеющие смысла, исключаются из итоговой модели.

После знакомства с важнейшими новыми методами можно переходить к рассмотрению примера.

8.3. Учебный пример: классификация сообщений Reddit

Хотя глубокий анализ текста находит много разнообразных применений, в учебном примере этой главы мы сосредоточимся на классификации документов. Как упоминалось ранее в этой главе, именно это делает Google, когда разбирает ваши сообщения по категориям или пытается отличить спам от обычных сообщений. Классификация также широко применяется контакт-центрами, обрабатывающими входящие запросы или жалобы клиентов: жалобы сначала проходят фильтр идентификации темы, чтобы их можно было закрепить за соответствующим специалистом для обработки. Классификация документов также относится к числу обязательных возможностей современных систем мониторинга социальных сетей. Отслеживаемым сообщениям в Твиттере, форумам или сообщениям в Facebook, газетным статьям и многим другим интернет-ресурсам назначаются метки темы, чтобы они могли повторно использоваться в отчетах. Одной из конкретных разновидностей классификации текста является анализ эмоциональной окраски: как можно охарактеризовать отношение автора к некоторому аспекту — как позитивное, негативное,

нейтральное? Для распознавания «некоторого аспекта» можно воспользоваться методом распознавания сущностей.

В этом примере мы воспользуемся сообщениями с Reddit (сайта, провозгласившего себя «первой полосой Интернета») и попробуем натренировать модель, способную отличить обсуждение «data science» от обсуждения «Игры престолов».

Конечным результатом может быть представление модели или полноценное интерактивное приложение. В главе 9 основное внимание будет направлено на построение приложения для конечного пользователя, поэтому сейчас ограничимся представлением модели классификации.

Для достижения этой цели нам понадобятся вся поддержка и все инструменты, которые только можно получить; и как обычно, Python снова готов предоставить все необходимое.

8.3.1. NLTK

Возможно, Python не ставит рекордов по эффективности выполнения, но он предоставит в ваше распоряжение полноценный пакет для глубокого анализа текста и обработки естественного языка: *NLTK* (Natural Language Toolkit). NLTK представляет собой набор алгоритмов и функций, которые помогут вам сделать первые шаги в области глубокого анализа текста и обработки естественного языка. У NLTK также имеется отличная документация на сайте *nltk.org*. Впрочем, в продуктах промышленного уровня NLTK применяется не так часто, как другие библиотеки (скажем, scikit-learn).

УСТАНОВКА NLTK

Установите NLTK при помощи своей любимой системы установки пакетов. Если вы используете Anaconda, пакет устанавливается в стандартной конфигурации. В противном случае можно воспользоваться «pip» или «easy_install». Когда это будет сделано, для достижения полноценной функциональности необходимо также установить модели и корпусы текстов. Для этого выполните следующий код Python:

```
import nltk
nltk.download()
```

В зависимости от вашей конфигурации установки эти команды могут открыть всплывающее окно или предоставить дополнительные параметры командной строки. На рис. 8.10 показано окно, появляющееся при выполнении команды nltk.download(). При желании вы можете загрузить все корпусы текстов, но в этой главе мы будем использовать только «punkt» и «stopwords». Ссылки на них явно присутствуют в коде, прилагаемом к книге.

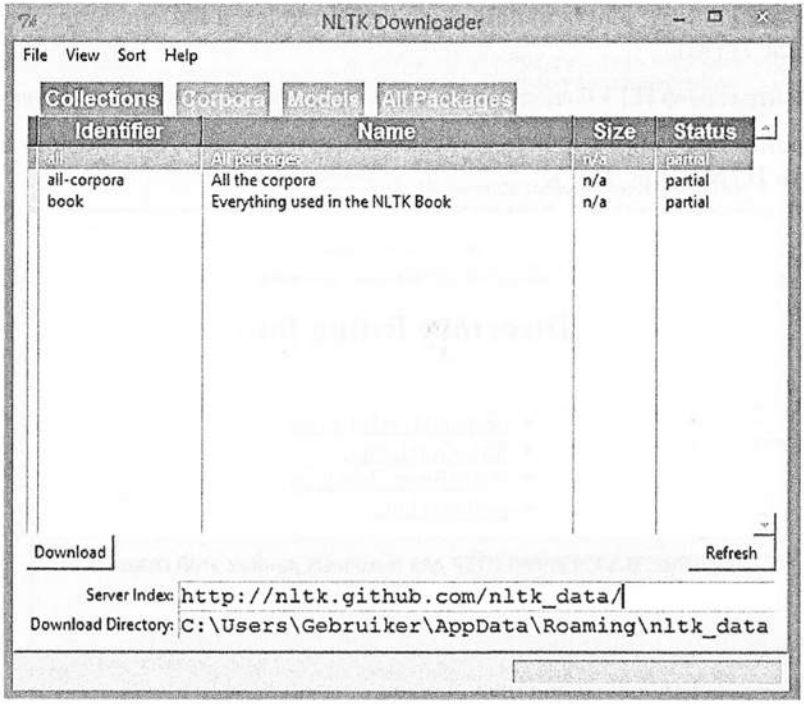


Рис. 8.10. Выберите вариант All packages, чтобы выполнить полную установку NLTK

К этой главе также прилагаются два файла IPython:

- Data collection — часть учебного примера, относящаяся к сбору данных.
- Data preparation and analysis — сохраненные данные проходят подготовку данных, после чего подвергаются анализу.

Весь код следующего примера находится в этих двух файлах и может выполняться в той же последовательности. Кроме того, для загрузки доступны две интерактивные диаграммы:

- forceGraph.html — 20 первых показателей модели наивного классификатора Байеса.
- Sunburst.html — верхние 4 ветви модели дерева принятия решений.

Для открытия этих двух страниц HTML потребуется сервер HTTP, который можно загрузить при помощи Python и окна командной строки:

- Откройте окно командной строки (Linux, Windows — какое пожелаете).
- Перейдите в папку, содержащую файлы HTML и их файлы данных JSON: decisionTreeData.json для диаграммы Sunburst и NaiveBayesData.json для forceGraph. Важно, чтобы файлы HTML находились в одной папке со своими

файлами данных; в противном случае вам придется изменять код JavaScript в файле HTML.

- Создайте сервер HTTP следующей командой: `python -m Simple-HTTPServer 8000`
- Откройте браузер и введите адрес *localhost:8000*; здесь вы сможете выбрать файлы HTML (рис. 8.11).

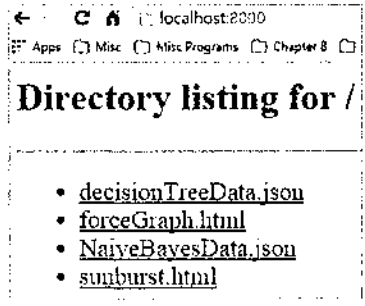


Рис. 8.11. Сервер HTTP для выходных данных этой главы

В этой главе будут использоваться следующие пакеты Python:

- NLTK — для глубокого анализа текста.
- PRAW — для загрузки сообщений с Reddit.
- SQLite3 — для сохранения данных в формате SQLite.
- Matplotlib — библиотека построения диаграмм для визуализации данных.

Прежде чем двигаться дальше, убедитесь в том, что вы установили все необходимые библиотеки и корпусы текстов. Впрочем, сначала стоит поближе рассмотреть те действия, которые нам предстоит выполнить для создания модели классификации тем.

8.3.2. Обзор процесса data science и этап 1: Назначение цели исследования

Для достижения цели этого упражнения мы снова воспользуемся процессом data science. На рис. 8.12 показано, как выглядит процесс data science применительно к нашему примеру с классификацией сообщений Reddit.

Некоторые элементы, представленные на рис. 8.12, сейчас вам покажутся непонятными. Оставшаяся часть главы будет посвящена тому, чтобы проработать весь процесс на практике, пока мы идем к своей конечной цели: построению модели классификации, способной отличать сообщения об «анализе данных» от сообщений об «Игре престолов». Итак, начнем со сбора данных.

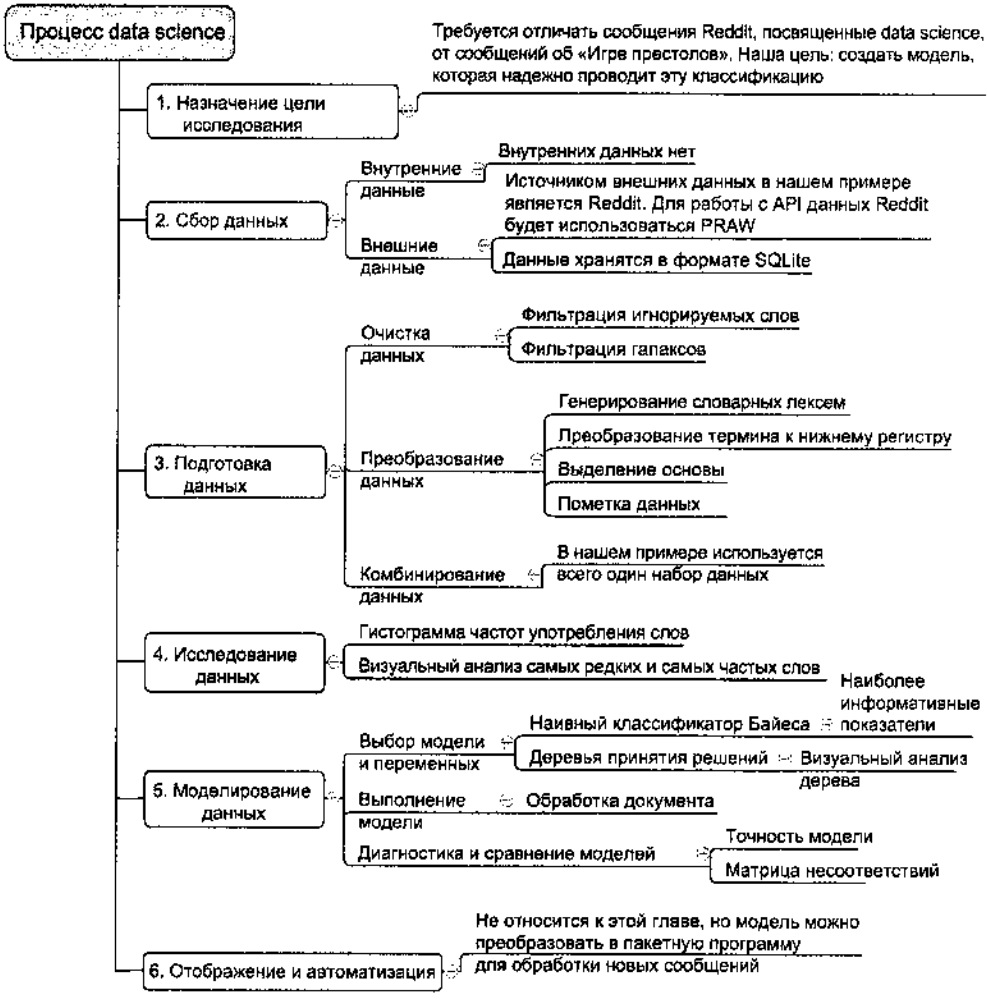


Рис. 8.12. Процесс data science применительно к примеру с классификацией сообщений Reddit

8.3.3. Этап 2: Сбор данных

В нашем примере будут использоваться данные Reddit. Если вы еще не знакомы с Reddit, не пожалейте времени и изучите основные концепции на сайте *www.reddit.com*.

Reddit называет себя «первой полосой Интернета», потому что пользователи могут публиковать здесь информацию, которая показалась им интересной. Но только та информация, которая показалась интересной многим другим людям, упоминается как «популярная» на домашней странице сайта. Можно сказать, что Reddit публикует обзор модных новинок Интернета. Любой пользователь может публиковать со-

общения в заранее определенной категории, называемой «подфорумом» (subreddit). Другие пользователи могут комментировать опубликованные сообщения: «плюсовать» понравившиеся или «минусовать» неудачные сообщения. Так как сообщение всегда является частью некоторого подфорума, эти метаданные будут доступны нам при связывании с Reddit API для получения данных. Фактически мы занимаемся выборкой помеченных данных, поскольку мы будем предполагать, что сообщение в подфоруме «gameofthrones» имеет некоторое отношение к «Игре престолов».

Для получения доступа к данным будет использоваться PRAW — официальная библиотека Python API сайта Reddit. После того как необходимые данные будут получены, мы сохраняем их в простейшем аналоге базы данных SQLite. Формат SQLite идеально подходит для хранения малых объемов данных, потому что он может использоваться без предварительной настройки и реагирует на запросы SQL точно так же, как любая другая реляционная база данных. Подойдет любой механизм хранения данных; если вы предпочитаете базы данных Oracle или Postgres, в Python существует отличная библиотека для работы с этими базами данных без написания кода SQL. Кстати, SQLAlchemy также работает и с файлами SQLite. На рис. 8.13 изображен этап сбора данных в процессе data science.

Рис. 8.13. Процесс data science: шаг сбора данных для примера с классификацией сообщений Reddit

Откройте свой любимый интерпретатор Python; пора браться за дело (листинг 8.1).

Сначала необходимо получить данные с сайта Reddit. Прежде чем выполнять приведенный ниже сценарий, выполните команду pip install praw или conda install praw (Anaconda), если это не было сделано ранее.

ПРИМЕЧАНИЕ

Код этапа 2 также содержится в файле IPython "Chapter 8 data collection." Файл входит в архив загружаемого кода книги.

Все начинается с импортирования необходимых библиотек.

После того как в нашем распоряжении окажется функциональность SQLite3 и PRAW, необходимо подготовить локальную базу данных к получению данных. Определяя подключение к файлу SQLite, мы автоматически создаем этот файл, если он еще не существует. Затем определяется курсор данных, который может выполнить любую команду SQL; мы используем его для предварительного определения структуры базы данных. База будет содержать две таблицы: таблица topics

Листинг 8.1. Настройка базы данных SQLite и клиента API Reddit

```
import praw
import sqlite3

conn = sqlite3.connect('reddit.db')
c = conn.cursor()

c.execute('''DROP TABLE IF EXISTS topics''')
c.execute('''DROP TABLE IF EXISTS comments''')
c.execute('''CREATE TABLE topics
             (topicTitle text, topicText text, topicID text,
topicCategory text)''')
c.execute('''CREATE TABLE comments
             (commentText text, commentID text ,
topicTitle text, topicText text, topicID text ,
 topicCategory text)''')

user_agent = "Introducing Data Science Book"
r = praw.Reddit(user_agent=user_agent)

subreddits = ['datascience','gameofthrones']

limit = 1000
```

Импортирование библиотек PRAW и SQLite3.

Создание подключения к базе данных SQLite.

Выполнение команд SQL для создания таблиц topics и comments.

Создание пользовательского агента PRAW для использования Reddit API.

Список подфорумов, которые будут перенесены в базу данных SQLite.

Максимальное количество сообщений, которые будут загружаться из Reddit в каждой категории.

содержит темы Reffit, а таблица comments содержит комментарии и связывается с таблицей topics через столбец topicID. Эти две таблицы связаны отношением «один ко многим». В данном примере мы ограничимся использованием таблицы topics, но в сборе данных будут задействованы обе таблицы, потому что это позволит вам при желании поэкспериментировать с новыми данными. Чтобы отточить свои навыки глубокого анализа текста, вы можете провести анализ эмоциональной окраски комментариев и определить, какие темы получают отрицательные или положительные комментарии. Затем результат можно связать с функциональными возможностями модели, которые будут реализованы к концу этой главы.

Для получения доступа к данным необходимо создать клиента PRAW. Каждый подфорум идентифицируется по имени, нас интересуют подфорумы «datascience» и «gameofthrones.» Переменная `limit` представляет максимальное количество тем (сообщений, не комментариев!), которые будут загружаться из Reddit. Кроме того, API разрешает получить не более 1000 сообщений по любому отдельному запросу, хотя мы можем запросить дополнительные сообщения позднее, когда появятся новые публикации. Собственно, запрос API можно выполнять периодически для динамического сбора данных. Хотя в любой конкретный момент вы ограничены тысячей сообщений, ничто не мешает вам наращивать базу данных на протяжении месяцев. Стоит заметить, что на выполнение этого сценария потребуется около часа. Если вам не хочется ждать, воспользуйтесь готовым файлом SQLite из загружаемого архива.

Также учтите, что при выполнении этого сценария сейчас вы вряд ли получите точно такой же результат, какой получили мы для создания данных этой главы.

В листинге 8.2 приведена функция сбора данных.

Листинг 8.2. Выборка и сохранение данных в SQLite

К списку присоединяются поля темы. В этом упражнении используются только поля title и text, но идентификатор темы может пригодиться для построения вашей собственной (более масштабной) базы данных тем.

Получение 1000 (в нашем случае) самых популярных тем из подразделов.

```
def prawGetData(limit,subredditName):
    topics = r.get_subreddit(subredditName).get_hot(limit=limit)
    commentInsert = []
    topicInsert = []
    topicNBR = 1
    for topic in topics:
        if (float(topicNBR)/limit)*100 in xrange(1,100):
            print '********** TOPIC:' + str(topic.id)
+ ' *********COMPLETE: ' + str((float(topicNBR)/limit)*100)
+ ' % ****'
        topicNBR += 1
        try:
            topicInsert.append((topic.title,topic.selftext,topic.id,
            subredditName))
        except:
            pass
        try:
            for comment in topic.comments:
                commentInsert.append((comment.body,comment.id,
topic.title,topic.selftext,topic.id,subredditName))
        except:
            pass
    print  '*********************************'
    print  'INSERTING DATA INTO SQLITE'
    c.executemany('INSERT INTO topics VALUES (?,?,?,?)', topicInsert)
    print  'INSERTED TOPICS'
    c.executemany('INSERT INTO comments VALUES (?,?,?,?,?,?)',
                  commentInsert)
    print  'INSERTED COMMENTS'
    conn.commit()

for subject in subreddits:
    prawGetData(limit=limit,subredditName=subject)
```

Служебная информация, несущественная для работы кода. Она всего лишь информирует вас о ходе загрузки.

К списку присоединяются комментарии. В упражнении они не используются, но вы можете поэкспериментировать с ними самостоятельно.

Все комментарии вставляются в базу данных SQLite.

Функция выполняется для всех подфорумов, указанных ранее.

Изменения (вставка данных) закрепляются в базе данных. Без закрепления данные не появятся в базе.

Все темы вставляются в базу данных SQLite.

Функция `prawGetData()` получает самые популярные темы из их подфорумов, присоединяет их к массиву, а затем получает все сопутствующие комментарии. Это продолжается до тех пор, пока не будут загружены все 1000 тем или не окажется, что тем больше не существует и все данные сохранены в базе SQLite. Команды `print` сообщают пользователю о прогрессе операции по сбору 1000 тем. Нам остается лишь выполнить функцию для каждого подфорума.

Если вы хотите, чтобы анализ включал более двух подфорумов, просто добавьте дополнительную категорию в массив `subreddits`.

Когда данные будут собраны, можно переходить к подготовке.

8.3.4. Этап 3: Подготовка данных

Как обычно, подготовка данных является самым важным шагом для получения правильных результатов. Для глубокого анализа текста его роль только возрастает, потому что вначале мы даже не располагаем структурированными данными. Приведенный ниже код также содержится в файле IPython «Chapter 8 data preparation and analysis». Начнем с импортирования необходимых библиотек и подготовки базы данных SQLite (листинг 8.3).

Листинг 8.3. Глубокий анализ текста, библиотеки, зависимости корпусов текстов и подключение к базе данных SQLite

```
import sqlite3
import nltk
import matplotlib.pyplot as plt
from collections import OrderedDict
import random
```

Импортирование всех необходимых библиотек.

```
nltk.download('punkt')
nltk.download('stopwords')
```

Загрузка используемых корпусов текстов.

```
conn = sqlite3.connect('reddit.db')
c = conn.cursor()
```

Создание подключения к базе данных SQLite, содержащей данные Reddit.

Если вы еще не загрузили полный корпус текстов NLTK, то сейчас мы загрузим ту часть, которая будет использоваться в нашем примере. Не беспокойтесь, если тексты уже были загружены, — сценарий проверит их актуальность.

Наши данные хранятся в файле SQLite Reddit, поэтому для работы с ними необходимо создать объект подключения.

Еще перед исследованием данных следует выполнить с ними как минимум две операции по очистке: фильтрацию игнорируемых слов и преобразование к нижнему регистру.

Обобщенная функция фильтрации слов (листинг 8.4) позволит исключить из данных неинформативные компоненты.

Листинг 8.4. Функции фильтрации слов и преобразования к нижнему регистру

```
def wordFilter(excluded,wordrow):
    filtered = [word for word in wordrow if word not in excluded]
    return filtered
stopwords = nltk.corpus.stopwords.words('english')
def lowerCaseArray(wordrow):
    lowercased = [word.lower() for word in wordrow]
    return lowercased
```

Функция wordFilter() исключает термин из массива терминов.

Переменная stopwords содержит набор игнорируемых слов для английского языка, определенный по умолчанию в NLTK.

Функция lowerCaseArray() преобразует заданный термин в его версию в нижнем регистре.

Игнорируемые слова английского языка первыми исключаются из данных. Для получения списка этих слов используется следующий фрагмент:

```
stopwords = nltk.corpus.stopwords.words('english')
print stopwords
```

Перечень игнорируемых слов английского языка в NLTK показан на рис. 8.14.

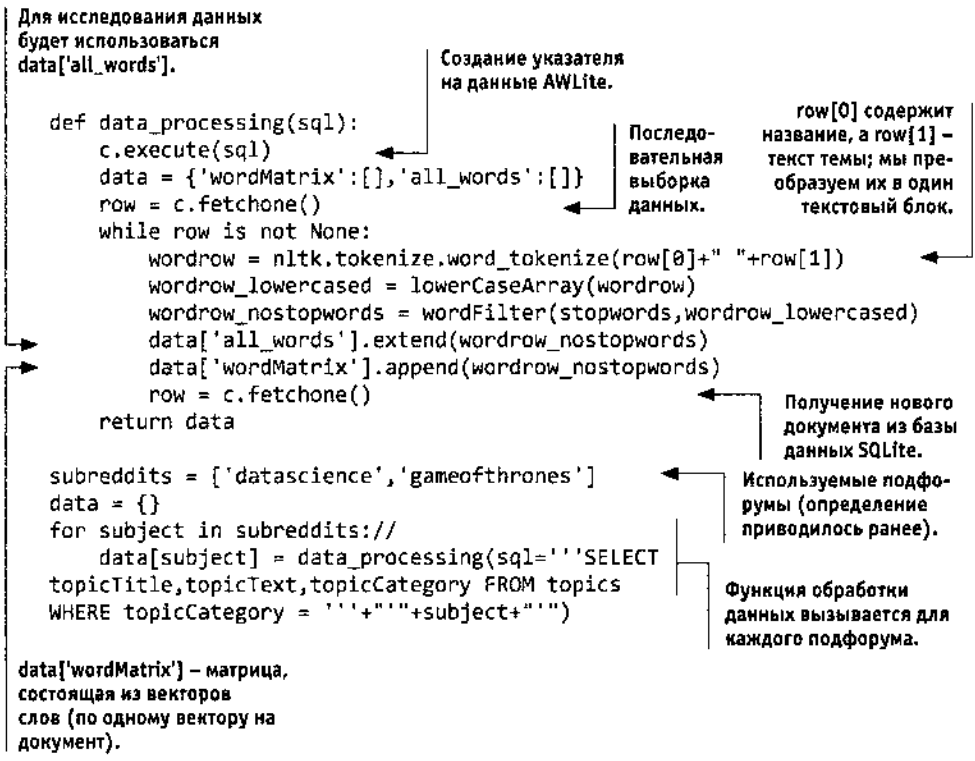
```
stopwords = nltk.corpus.stopwords.words('english')
print stopwords

[u'i', u'me', u'my', u'myself', u'we', u'our', u'ours', u'ourselves', u'you',
u'your', u'yours', u'yourself', u'yourselves', u'he', u'him', u'his', u'himsel
f', u'she', u'her', u'hers', u'herself', u'it', u'its', u'itself', u'they', u'th
em', u'their', u'theirs', u'themselves', u'what', u'which', u'who', u'whom', u't
his', u'that', u'these', u'those', u'am', u'is', u'are', u'was', u'were', u'be',
u'been', u'being', u'have', u'has', u'had', u'having', u'do', u'does', u'did',
u'doing', u'a', u'an', u'the', u'and', u'but', u'if', u'or', u'because', u'as',
u'until', u'while', u'of', u'at', u'by', u'for', u'with', u'about', u'against',
u'between', u'into', u'through', u'during', u'before', u'after', u'above', u'bel
ow', u'to', u'from', u'up', u'down', u'in', u'out', u'on', u'off', u'over', u'un
der', u'again', u'further', u'then', u'once', u'here', u'there', u'when', u'wher
e', u'why', u'how', u'all', u'any', u'both', u'each', u'few', u'more', u'most',
u'other', u'some', u'such', u'no', u'nor', u'not', u'only', u'own', u'same', u's
o', u'than', u'too', u'very', u's', u't', u'can', u'will', u'just', u'don', u'sh
ould', u'now']
```

Рис. 8.14. Список игнорируемых слов английского языка в NLTK

Итак, у нас имеются все необходимые компоненты, и мы можем рассмотреть первую функцию обработки данных (листинг 8.5).

Функция data_processing() получает команду SQL и матрицу «документ—термин». Для этого она перебирает данные по одной записи (теме Reddit), объединяя название и основной текст темы в один вектор слов посредством выделения словарных

Листинг 8.5. Первая функция подготовки данных и ее выполнение




```
def data_processing(sql):
    c.execute(sql)
    data = {'wordMatrix':[],'all_words':[]}
    row = c.fetchone()
    while row is not None:
        wordrow = nltk.tokenize.word_tokenize(row[0]+" "+row[1])
        wordrow_lowercased = lowerCaseArray(wordrow)
        wordrow_nostopwords = wordFilter(stopwords,wordrow_lowercased)
        data['all_words'].extend(wordrow_nostopwords)
        data['wordMatrix'].append(wordrow_nostopwords)
        row = c.fetchone()
    return data

subreddits = ['datascience','gameofthrones']
data = {}
for subject in subreddits://
    data[subject] = data_processing(sql='''SELECT
topicTitle,topicText,topicCategory FROM topics
WHERE topicCategory = '''+"'"+subject+"'")
```

лексем. *Генератором лексем* (tokenizer) называется сценарий обработки текста, который «нарезает» текст на части. Существует много разных способов деления текста на лексемы: на предложения или слова, по пробелам или знакам препинания, с учетом других символов и т. д. Здесь мы используем стандартный словарный генератор лексем NLTK. Принцип его работы очень прост: текст разбивается на лексемы по пробелам между словами. Затем вектор преобразуется к нижнему регистру, и из него отфильтровываются игнорируемые слова. Обратите внимание на важность порядка операций: игнорируемое слово в начале предложения не будет отфильтровано, если фильтрация будет выполняться перед преобразованием регистра. Например, в предложении «I like Game of Thrones» слово «I» не будет преобразовано к нижнему регистру, а поэтому не будет отфильтровано. Затем мы создаем матрицу слов (матрица «документ—термин») и список, содержащий все слова. Также обратите внимание на то, что список расширяется без фильтрации дубликатов; это позволит нам построить гистограмму по частотам вхождения слов в процессе исследования данных. А теперь выполним функцию для двух категорий тем.

На рис. 8.15 показан первый вектор слов из категории «datascience»:

```
print data['datascience']['wordMatrix'][0]
```

```
print data['datascience']['wordMatrix'][0]

[u'data', u'science', u'freelancing', u"'m", u'currently', u'master
s', u'program', u'studying', u'business', u'analytics', u"'m", u'try
ing', u'get', u'data', u'freelancing', u'.', u"'m", u'still', u'lear
ning', u'skill', u'set', u'typically', u'see', u'right', u"'m", u'fa
irly', u'proficient', u'sql', u'know', u'bit', u'r.', u'freelancer
s', u'find', u'jobs', u'?']
```

Рис. 8.15. Первый вектор слов из категории «datascience» после первой попытки обработки данных

Данные определенно не выглядят чистыми: знаки препинания сохранены как отдельные термины, а некоторые слова даже остались неразделенными. Дальнейшие исследования данных должны немного прояснить ситуацию.

8.3.5. Этап 4: Исследование данных

Итак, текст был разделен на термины, но из-за значительного размера данных трудно сразу понять, достаточно ли данные чисты для фактического использования. Впрочем, даже беглый взгляд выявляет несколько проблем: некоторые слова разбиты неправильно, вектор содержит много односимвольных терминов. Односимвольные термины могут быть хорошими отличительными признаками в некоторых случаях: например, экономический текст будет содержать больше символов $, £ и €, чем текст по медицине. Тем не менее в большинстве случаев односимвольные термины бесполезны. Для начала рассмотрим распределение частот наших терминов:

```
wordfreqs_cat1 = nltk.FreqDist(data['datascience']['all_words'])
plt.hist(wordfreqs_cat1.values(), bins = range(10))
plt.show()
wordfreqs_cat2 = nltk.FreqDist(data['gameofthrones']['all_words'])
plt.hist(wordfreqs_cat2.values(), bins = range(20))
plt.show()
```

После построения гистограммы частотного распределения (рис. 8.16) мы сразу видим, что большинство терминов встречается только в одном документе.

Термины, встречающиеся по одному разу, также называются *гапаксами* (hapaxes). С точки зрения модели они бесполезны, потому что одного вхождения никогда не бывает достаточно для построения надежной модели. Мы воспользуемся этим обстоятельством для своих целей; исключение гапаксов существенно сократит объем данных без вреда для итоговой модели. Рассмотрим некоторые из этих «одноразовых» терминов:

```
print wordfreqs_cat1.hapaxes()
print wordfreqs_cat2.hapaxes()
```

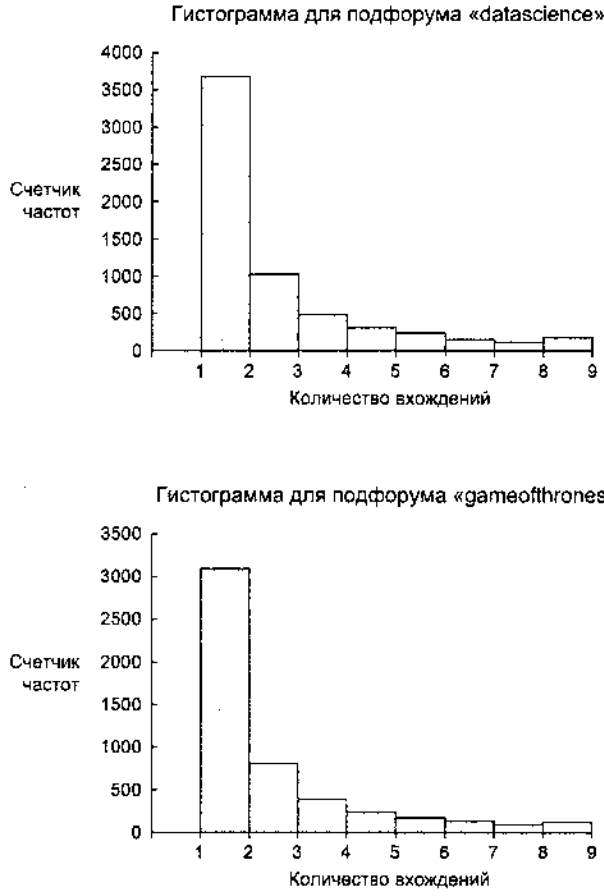


Рис. 8.16. Гистограмма частот вхождения терминов показывает, что в матрицах «datascience» и «gameofthrones» 3000 терминов встречаются только по одному разу

Термины на рис. 8.17 выглядят осмысленно. Вероятно, если бы у нас было больше данных, то они чаще встречались бы в тексте:

```
print wordfreqs_cat1.hapaxes()
print wordfreqs_cat2.hapaxes()
```

Многие из этих терминов представляют собой ошибочные варианты написания других, более содержательных: Jaimie вместо Jamie, Milisandre вместо Melisandre и т. д. Хороший справочник по «Игре престолов» помог бы нам найти и заменить ошибочные варианты написания алгоритмом нечеткого поиска. В общем, очистку данных в ходе глубокого анализа при желании можно продолжать бесконечно долго; важно соблюдать баланс между затраченными усилиями и результатом.

Термины, однократно встречающиеся в сообщениях подфорума «datascience»

```
print wordfreqs_cat1.hapaxes()

[u'post-grad', u'marching', u'cytoscape', u'wizardry', u"'pure", u'i
mmature', u'socrata', u'filenotfoundexception', u'side-by-side', u'b
ringing', u'non-experienced', u'zestimate', u'formatting*', u'sustai
```

Термины, однократно встречающиеся в сообщениях подфорума «gameofthrones»

```
print wordfreqs_cat2.hapaxes()

[u'hordes', u'woods', u'comically', u'pack', u'seventy-seven', u"'co
ntext", u'shaving', u'kennels', u'differently', u'screaming', u'her-
', u'complainers', u'sailed', u'contributed', u'payoff', u'hallucina
```

Рис. 8.17. Термины, встречающиеся в подфорумах «datascience» и «gameofthrones» по одному разу (гапаксы)

А теперь рассмотрим самые частые слова:

```
print wordfreqs_cat1.most_common(20)
print wordfreqs_cat2.most_common(20)
```

На рис. 8.18 представлен результат запроса 20 наиболее часто встречающихся слов в каждой категории.

20 наиболее часто встречающихся слов в сообщениях подфорума «datascience» после расширенной подготовки данных

```
print wordfreqs_cat1.most_common(20)

[(u'.', 2833), (u',', 2831), (u'data', 1882), (u'?', 1190), (u'scien
ce', 887), (u')', 812), (u'(', 739), (u"'m", 566), (u':', 548), (u'w
ould', 427), (u"'s", 323), (u'like', 321), (u"n't", 288), (u'get', 2
52), (u'know', 225), (u"'ve", 213), (u'scientist', 211), (u'!', 20
9), (u'work', 204), (u'job', 199)]
```

20 наиболее часто встречающихся слов в сообщениях подфорума «gameofthrones» после расширенной подготовки данных

```
print wordfreqs_cat2.most_common(20)

[(u'.', 2909), (u',', 2478), (u'[', 1422), (u']', 1420), (u'?', 113
9), (u"'s", 886), (u"n't", 494), (u')', 452), (u'(', 426), (u'55', 3
99), (u':', 380), (u'spoilers', 332), (u'show', 325), (u'would', 31
1), (u"''", 305), (u'``', 276), (u'think', 248), (u'season', 244),
(u'like', 243), (u'one', 238)]
```

Рис. 8.18. 20 слов, наиболее часто встречающихся в сообщениях подфорумов «datascience» и «gameofthrones»

А эти результаты обнадеживают: похоже, некоторые распространенные слова относятся к своим темам. Такие слова, как «data», «science» и «season», с большой вероятностью станут хорошими отличительными признаками. Также следует обратить внимание на изобилие односимвольных терминов, таких как «.» и «,»; мы избавимся от них.

Вооружившись всей этой дополнительной информацией, переработаем наш сценарий подготовки данных.

8.3.6. Этап 3 (повторно): Подготовка данных (адаптированная)

Даже наше краткое исследование данных уже выделило несколько очевидных возможностей усовершенствования нашего текста. Другое важное усовершенствование — выделение основы терминов.

В листинге 8.6 продемонстрирован простой алгоритм выделения основы, который называется выделением основы Snowball. Алгоритмы выделения основы Snowball могут быть привязаны к конкретному языку, поэтому мы используем английскую версию; тем не менее эти алгоритмы поддерживают разные языки.

Листинг 8.6. Обработка данных Reddit, усовершенствованная после исследования данных

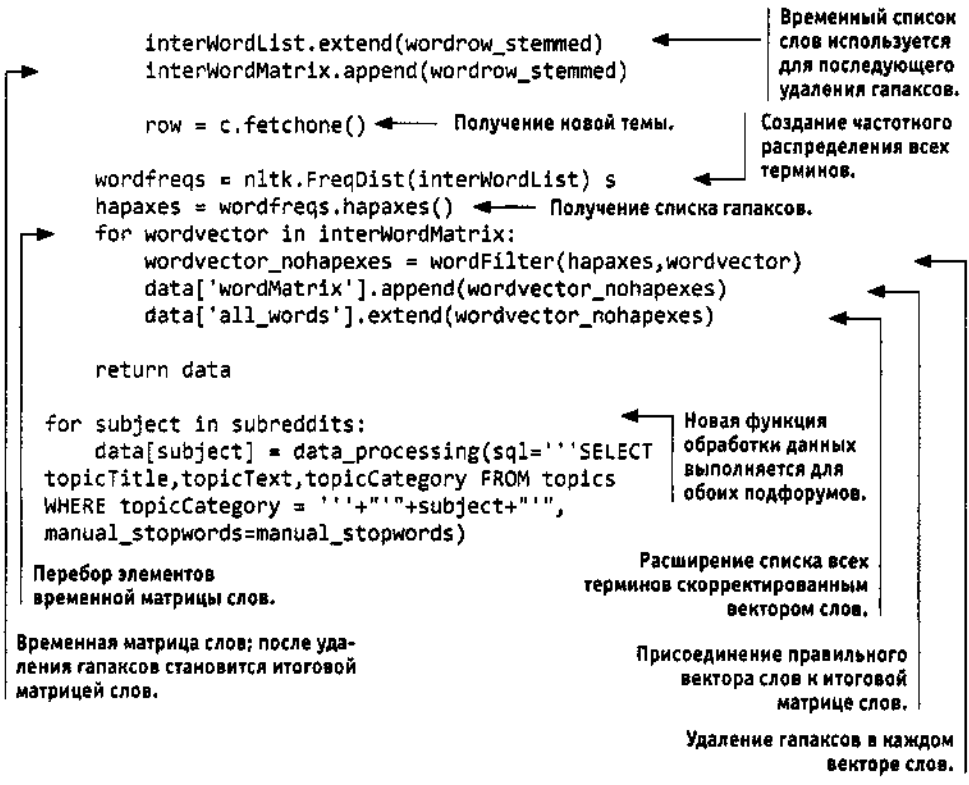
```
Удаление вручную добавленных
игнорируемых слов из текстового блока.

    row[0] содержит название, а row[1] – текст темы;
    мы преобразуем их в один текстовый блок.

    stemmer = nltk.SnowballStemmer("english")
    def wordStemmer(wordrow):
        stemmed = [stemmer.stem(word) for word in wordrow]
        return stemmed

    manual_stopwords = [',','.',')',',','(','m',"'m","n't",'e.g',"'ve",'s',
         '#','/','``',"'s","''",'!','r','}','=','[','s','&','%','*','...',
         '1','2','3','4','5','6','7','8','9','10','--',"''",';','-',':']

    def data_processing(sql,manual_stopwords):
        #create pointer to the sqlite data
        c.execute(sql)
        data = {'wordMatrix':[],'all_words':[]}
        interWordMatrix = []
        interWordList = []

        row = c.fetchone()
        while row is not None:
            tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+|[^\w\s]+')

            wordrow = tokenizer.tokenize(row[0]+" "+row[1])
            wordrow_lowercased = lowerCaseArray(wordrow)
            wordrow_nostopwords = wordFilter(stopwords,wordrow_lowercased)

            wordrow_nostopwords =
               wordFilter(manual_stopwords,wordrow_nostopwords)
            wordrow_stemmed = wordStemmer(wordrow_nostopwords)
```

Инициализация алгоритма выделения основы из библиотеки NLTK.

Массив определяет удаляемые/игнорируемые термины.

Определение пересмотренной подготовки данных.

Последовательная выборка данных (сообщений Reddit) из базы данных SQLite.



Обратите внимание на изменения по сравнению с предыдущей версией `data_processing()`. Новый генератор лексем использует регулярное выражение. В книге регулярные выражения не рассматриваются и часто считаются достаточно сложными, но это простое выражение просто «разрезает» текст на слова. Слова могут содержать произвольные комбинации алфавитно-цифровых символов w), поэтому специальные символы и знаки препинания теряются. Мы также применили словарный алгоритм выделения основы и удалили список дополнительных игнорируемых слов. Кроме того, все гапаксы удаляются в конце, потому что сначала необходимо выполнить выделение основы. Давайте повторим подготовку данных.

Если провести тот же исследовательский анализ, что и прежде, вы увидите, что данные стали более содержательными и в них не осталось гапаксов:

```
print wordfreqs_cat1.hapaxes()
print wordfreqs_cat2.hapaxes()
```

Снова просмотрим первые 20 слов каждой категории (рис. 8.19).

Из рис. 8.19 видно, что качество данных значительно повысилось. Также обратите внимание на то, что некоторые слова укоротились из-за примененного выделения

20 наиболее часто встречающихся слов в сообщениях подфорума «datascience» после расширенной подготовки данных.

```
wordfreqs_cat1 = nltk.FreqDist(data['datascience']['all_words'])
print wordfreqs_cat1.most_common(20)
```

```
[(u'data', 1971), (u'scienc', 955), (u'would', 418), (u'work', 368), (u'use', 34
7), (u'program', 343), (u'learn', 342), (u'like', 341), (u'get', 325), (u'scient
ist', 310), (u'job', 268), (u'cours', 265), (u'look', 257), (u'know', 239), (u's
tatist', 228), (u'want', 225), (u've', 223), (u'python', 205), (u'year', 204),
(u'time', 196)]
```

20 наиболее часто встречающихся слов в сообщениях подфорума «gameofthrones» после расширенной подготовки данных.

```
wordfreqs_cat2 = nltk.FreqDist(data['gameofthrones']['all_words'])
print wordfreqs_cat2.most_common(20)
```

```
[(u's5', 426), (u'spoiler', 374), (u'show', 362), (u'episod', 300), (u'think', 2
89), (u'would', 287), (u'season', 286), (u'like', 282), (u'book', 271), (u'one',
249), (u'get', 236), (u'sansa', 232), (u'scene', 216), (u'cersei', 213), (u'kno
w', 192), (u'go', 188), (u'king', 183), (u'throne', 181), (u'see', 177), (u'char
act', 177)]
```

Рис. 8.19. 20 слов, наиболее часто встречающихся в сообщениях подфорумов «datascience» и «gameofthrones» после подготовки данных

основы. Например, «science» и «sciences» превратились в «scienc»; слова «courses» и «course» превратились в «cours» и т. д. Полученные термины не являются полноценными словами, но понять их смысл нетрудно.

Если вы настаиваете на том, чтобы термины оставались реальными словами, возможно, вам стоит воспользоваться лемматизацией.

После «завершения» процесса очистки данных (примечание: очистка данных при глубоком анализе почти никогда не бывает полностью завершенной) остается выполнить несколько преобразований данных для перевода данных в формат набора слов.

Сначала пометим данные и создадим контрольную выборку из 100 наблюдений на категорию (листинг 8.7).

Контрольная выборка будет использоваться для итогового тестирования модели и создания матрицы несоответствий. Матрица несоответствий используется для проверки того, насколько хорошо модель справилась с ранее не встречавшимися данными. В матрице хранится информация о том, сколько наблюдений было классифицировано правильно или неправильно.

Прежде чем переходить к созданию, тренировке и тестированию, необходимо сделать еще один шаг: свести данные в формат набора слов, в котором каждому термину назначается метка «True» или «False» в зависимости от того, присутствует или отсутствует оно в конкретном сообщении. То же самое необходимо проделать и для непомеченной контрольной выборки.

Листинг 8.7. Итоговое преобразование данных и разбиение данных перед моделированием

Контрольная выборка состоит из непомеченных данных из двух подфорумов: по 100 наблюдений из каждого набора данных. Метки хранятся в отдельном наборе данных.

Контрольная выборка будет использоваться для выявления недостатков модели за счет построения матрицы несоответствий.

```
holdoutLength  = 100//

labeled_data1 = [(word,'datascience') for word in
data['datascience']['wordMatrix'][holdoutLength:]]
labeled_data2 = [(word,'gameofthrones') for word in
data['gameofthrones']['wordMatrix'][holdoutLength:]]
labeled_data = []
labeled_data.extend(labeled_data1)
labeled_data.extend(labeled_data2)
```

Создание объединенного набора данных, в котором каждый вектор слов снабжается меткой "datascience" или "gameofthrones." Часть данных резервируется для контрольной выборки.

```
holdout_data = data['datascience']['wordMatrix'][:holdoutLength]
holdout_data.extend(data['gameofthrones']['wordMatrix'][:holdoutLength])
holdout_data_labels = ([('datascience')
for _ in xrange(holdoutLength)] + [('gameofthrones') for _ in
    xrange(holdoutLength)])

data['datascience']['all_words_dedup'] =
list(OrderedDict.fromkeys(
data['datascience']['all_words']))
data['gameofthrones']['all_words_dedup'] =
list(OrderedDict.fromkeys(
data['gameofthrones']['all_words']))
all_words = []
all_words.extend(data['datascience']['all_words_dedup'])
all_words.extend(data['gameofthrones']['all_words_dedup'])
all_words_dedup = list(OrderedDict.fromkeys(all_words))
```

Создание списка всех уникальных терминов для построения набора слов, необходимого для тренировки или оценки модели.

```
prepared_data = [({word: (word in x[0]) for word
in all_words_dedup}, x[1]) for x in labeled_data]
prepared_holdout_data = [({word: (word in x[0])
for word in all_words_dedup})
for x in holdout_data]
```

Данные преобразуются в формат набора слов.

```
random.shuffle(prepared_data)
train_size = int(len(prepared_data) * 0.75)
train = prepared_data[:train_size]
test = prepared_data[train_size:]
```

Данные для тренировки модели и тестирования сначала проходят случайную перестановку.

Размер тренировочных данных составляет 75%, а остальные 25% используются для проверки эффективности модели.

Непомеченные данные теперь содержат каждый термин для каждого вектора (рис. 8.20.)

```
print prepared_data[0]
```

```
print prepared_data[0]

({u'sunspear': False, u'profici': False, u'pardon': False, u'selye
s': False, u'four': False, u'davo': False, u'sleev': False, u'slee
                              ⋮
u'daeron': False, u'portion': False, u'emerg': False, u'fifti': Fals
e, u'decemb': False, u'defend': False, u'sincer': False}, 'datascien
ce')
```

Рис. 8.20. Двоичный набор слов, готовый к моделированию: данные сильно разрежены

Мы создали очень большую, но разреженную матрицу; если бы матрица была слишком велика для вашего компьютера, можно было бы воспользоваться методами из главы 5. Впрочем, при такой маленькой таблице необходимости в этом нет, и мы можем сразу перейти к случайной перестановке и разбиению данных на тренировочный и тестовый наборы.

И хотя большая часть данных всегда должна использоваться для тренировки модели, оптимальная пропорция существует. В данном случае было выбрано разбиение 3 : 1, но ничто не мешает вам поэкспериментировать. Чем больше наблюдений, тем больше у вас свободы действий. При малом количестве наблюдений под тренировку модели выделяется относительно большая доля данных. А пока все готово к тому, чтобы перейти к самой интересной части: анализу данных.

8.3.7. Этап 5: Анализ данных

В своем анализе мы применим к данным два алгоритма классификации: наивный классификатор Байеса и дерево принятия решений. Наивный классификатор Байеса рассматривался в главе 3, а дерево принятия решений — ранее в этой главе.

Для начала протестируем эффективность наивного классификатора Байеса. NLTK включает собственный классификатор, но вы можете воспользоваться алгоритмами из других пакетов, таких как SciPy:

```
classifier  = nltk.NaiveBayesClassifier.train(train)
```

После тренировки классификатора тестовые данные могут использоваться для оценки общей точности:

```
nltk.classify.accuracy(classifier, test)
```

```
nltk.classify.accuracy(classifier, test)
```

```
0.9681528662420382
```

Рис. 8.21. Точность классификации — метрика, представляющая собой процент правильно классифицированных наблюдений в тестовых данных

Как видно из рис. 8.21, точность с тестовыми данными оценивается выше 90%. Точность классификации определяется как количество правильно классифицированных наблюдений — процент от общего количества наблюдений. Однако учтите, что на других данных в вашем случае значение может быть другим:

```
nltk.classify.accuracy(classifier, test)
```

Что ж, это хороший результат. Теперь можно расслабиться? Нет, не совсем. Протестируем данные снова на контрольной выборке из 200 наблюдений и на этот раз создадим матрицу несоответствий:

```
classified_data = classifier.classify_many(prepared_holdout_data)
cm = nltk.ConfusionMatrix(holdout_data_labels, classified_data)
print cm
```

Матрица несоответствий на рис. 8.22 показывает, что значение 97% слишком оптимистично, потому что на этот раз мы имеем 28 (23 + 5) неверно классифицированных случаев. И снова в вашем случае значения могут быть другими, если вы заполняли файл SQLite самостоятельно.

```
                    |     g |
                    |     a |
                    |   d m |
                    |   a e |
                    |   t o |
                    |   a f |
                    |   s t |
                    |   c h |
                    |   i r |
                    |   e o |
                    |   n n |
                    |   c e |
                    |   e s |
--------------------+-------+
  datascience       |<77>23 |
gameofthrones       |  5<95>|
--------------------+-------+
(row = reference; col = test)
```

Рис. 8.22. Матрица несоответствий для модели наивного классификатора Байеса показывает, что из 200 наблюдений 28 (23 + 5) были классифицированы неправильно

28 ошибочных классификаций означают, что на контрольной выборке точность составила 86%. Этот результат следует сравнить со случайным назначением со-

общения в группу «datascience» или «gameofthrones». При случайном назначении ожидаемая точность составит 50%; похоже, наша модель работает более эффективно. Чтобы узнать, какие факторы используются для определения категории, следует обратиться к самым информативным показателям модели:

```
print(classifier.show_most_informative_features(20))
```

На рис. 8.23 перечислены 20 первых терминов, по которым различаются две категории.

```
Most Informative Features
                    data = True              datasc : gameof =    365.1 : 1.0
                   scene = True              gameof : datasc =     63.8 : 1.0
                  season = True              gameof : datasc =     62.4 : 1.0
                    king = True              gameof : datasc =     47.6 : 1.0
                      tv = True              gameof : datasc =     45.1 : 1.0
                    kill = True              gameof : datasc =     31.5 : 1.0
                  compani = True             datasc : gameof =     28.5 : 1.0
                 analysi = True              datasc : gameof =     27.1 : 1.0
                 process = True              datasc : gameof =     25.5 : 1.0
                   appli = True              datasc : gameof =     25.5 : 1.0
                research = True              datasc : gameof =     23.2 : 1.0
                  episod = True              gameof : datasc =     22.2 : 1.0
                  market = True              datasc : gameof =     21.7 : 1.0
                   watch = True              gameof : datasc =     21.6 : 1.0
                     man = True              gameof : datasc =     21.0 : 1.0
                   north = True              gameof : datasc =     20.8 : 1.0
                      hi = True              datasc : gameof =     20.4 : 1.0
                   level = True              datasc : gameof =     19.1 : 1.0
                   learn = True              datasc : gameof =     16.9 : 1.0
                     job = True              datasc : gameof =     16.6 : 1.0
```

Рис. 8.23. Самые важные термины в модели наивного классификатора Байеса

Термину «data» («данные») назначен большой вес; похоже, это самый важный признак темы, относящейся к категории data science. Такие термины, как «scene», «season» и «king» («сцена», «сезон» и «король»), указывают на то, что тема относится к сериалу «Игра престолов», а не к data science. Все это абсолютно разумно, так что модель проходит проверку как на точность, так и на здравый смысл.

Наивный классификатор Байеса работает неплохо. Теперь обратимся к дереву принятия решений (листинг 8.8).

Как видно из рис. 8.24, предполагаемая точность составляет 93%.

```
nltk.classify.accuracy(classifier2, test)

0.9333333333333333
```

Рис. 8.24. Точность модели дерева принятия решений

Листинг 8.8. Тренировка и оценка модели дерева принятия решений

Создание матрицы несоответствий на основании результатов классификации и фактических меток.

Тренировка классификатора на базе дерева принятия решений.

```
classifier2 = nltk.DecisionTreeClassifier.train(train)
nltk.classify.accuracy(classifier2, test)
classified_data2 = classifier2.classify_many(prepared_holdout_data)
cm = nltk.ConfusionMatrix(holdout_data_labels, classified_data2)
print cm
```

Вывод матрицы несоответствий.

Попытка классификации контрольной выборки.

Проверка точности классификатора.

Теперь мы уже знаем, что полагаться исключительно на одну проверку не стоит, поэтому проверяем матрицу несоответствий для второго набора данных (рис. 8.25).

```
                    |     g |
                    |     a |
                    |  d  m |
                    |  a  e |
                    |  t  o |
                    |  a  f |
                    |  s  t |
                    |  c  h |
                    |  i  r |
                    |  e  o |
                    |  n  n |
                    |  c  e |
                    |  e  s |
--------------------+-------+
  datascience       |<26>74 |
gameofthrones       |  2<98>|
--------------------+-------+
(row = reference; col = test)
```

Рис. 8.25. Матрица несоответствий для модели дерева принятия решений

На рис. 8.25 предстает совершенно иная картина. Из 200 наблюдений контрольной выборки модель дерева принятия решений неплохо классифицирует сообщения, относящиеся к «Игре престолов», но терпит оглушительный провал с сообщениями, относящимися к data science. Похоже, модель неравнодушна к «Игре престолов»... и кто ее за это осудит? Рассмотрим фактическую модель:

```
print(classifier2.pseudocode(depth=4))
```

Как подсказывает название, дерево принятия решений имеет древовидную структуру (рис. 8.26).

```
if data == False:
  if learn == False:
    if python == False:
      if tool == False: return 'gameofthrones'
      if tool == True: return 'datascience'
    if python == True: return 'datascience'
  if learn == True:
    if go == False:
      if wrong == False: return 'datascience'
      if wrong == True: return 'gameofthrones'
    if go == True:
      if upload == False: return 'gameofthrones'
      if upload == True: return 'datascience'
if data == True: return 'datascience'
```

Рис. 8.26. Представление структуры дерева принятия решений

Наивный классификатор Байеса рассматривает все термины и назначает им веса, а модель дерева принятия решений перебирает их последовательно на пути от корня к внешним ветвям и листьям. На рис. 8.26 показаны только верхние четыре уровня, начиная со слова «data». Если слово «data» присутствует в сообщении, модель всегда считает, что сообщение относится к data science. Если слово «data» не найдено, проверяется термин «learn», и т. д. Возможная причина неэффективной работы дерева принятия решений — отсутствие *отсечения* (pruning). Построенное дерево принятия решений содержит много листьев, обычно *слишком* много. Затем дерево обрезается до определенного уровня для того, чтобы свести к минимуму чрезмерно близкую подгонку данных. Одно из больших преимуществом дерева принятия решений — неявное взаимодействие между словами, учитываемыми при построении ветвей. Если несколько терминов совместно создают более сильную классификацию, чем одиночные термины, дерево принятия решений обычно превосходит наивный классификатор Байеса по эффективности. Мы не будем углубляться в подробности, но это один из следующих шагов, которые можно предпринять для улучшения модели.

Итак, у нас имеется две модели классификации, которые дают представление о различиях между содержимым двух подфорумов. Остается сделать последний шаг и поделиться полученной информацией с другими людьми.

8.3.8. Этап 6: Отображение и автоматизация

Итак, теперь мы используем то, что узнали ранее, и либо преобразуем информацию в полезное приложение, либо представляем свои результаты другим. В последней главе этой книги рассматривается построение интерактивного приложения, которое само по себе является полноценным проектом. А пока будет достаточно найти какой-нибудь симпатичный способ передачи информации о находке. Красивый (а еще лучше — интерактивный) график привлекает внимание; это своего рода «вишенка на торте» отображения данных. И хотя для отображения числовых данных легко и соблазнительно ограничиться чем-нибудь вроде столбцовой диаграммы, лучше сделать еще один шаг.

Например, для представления модели наивного классификатора Байеса можно воспользоваться *силовым графом* (force graph) на рис. 8.27, в котором размеры кружков и связей показывают, насколько сильно связано слово с подфорумом «Игры престолов» или data science. Обратите внимание: слова на кружках часто усечены; напомним, что это происходит из-за применения выделения основы.

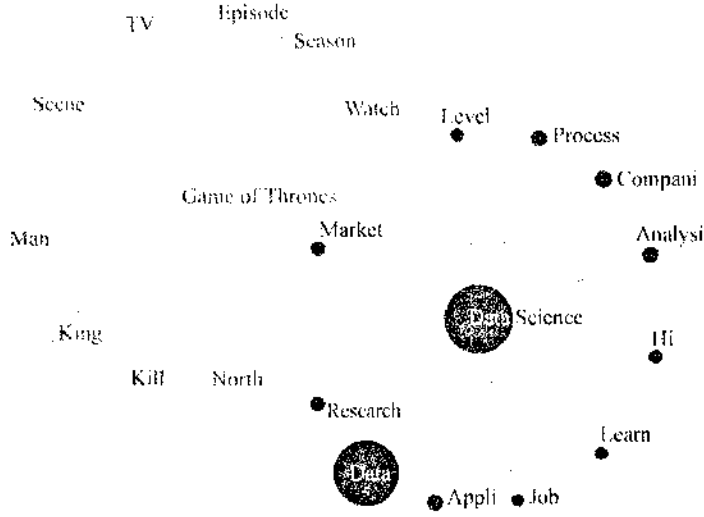


Рис. 8.27. Интерактивный силовой граф с 20 наиболее значимыми терминами наивного классификатора Байеса и их весами

Хотя сам граф на рис. 8.27 статичен, вы можете открыть HTML-файл forceGraph.html и оценить эффект силового графа d3.js. Тема библиотеки d3.js выходит за рамки книги, но для использования d3.js подробно знать ее не нужно. Многочисленные примеры из галереи по адресу *https://github.com/mbostock/d3/wiki/Gallery* могут использоваться с минимальными изменениями в коде. От вас потребуется лишь здравый смысл и небольшое знание JavaScript. Код примера с силовым графом находится по адресу *http://bl.ocks.org/mbostock/4062045*.

Дерево принятия решений также можно представить довольно оригинальным образом. Конечно, можно построить диаграмму с непосредственно используемым деревом, но следующая диаграмма «солнечные лучи» выглядит более оригинально и интересно.

На рис. 8.28 изображен верхний слой диаграммы. Пользователь может увеличить изображение, щелкая на сегментах круга; щелчок в центре круга снова уменьшает его. Код этого примера доступен по адресу *http://bl.ocks.org/metmajer/5480307*.

Оригинальный способ представления результатов может стать ключом к успеху проекта. Люди никогда не оценят усилия, затраченные вами для достижения результатов, если вы не сможете донести до них эти результаты и если они не

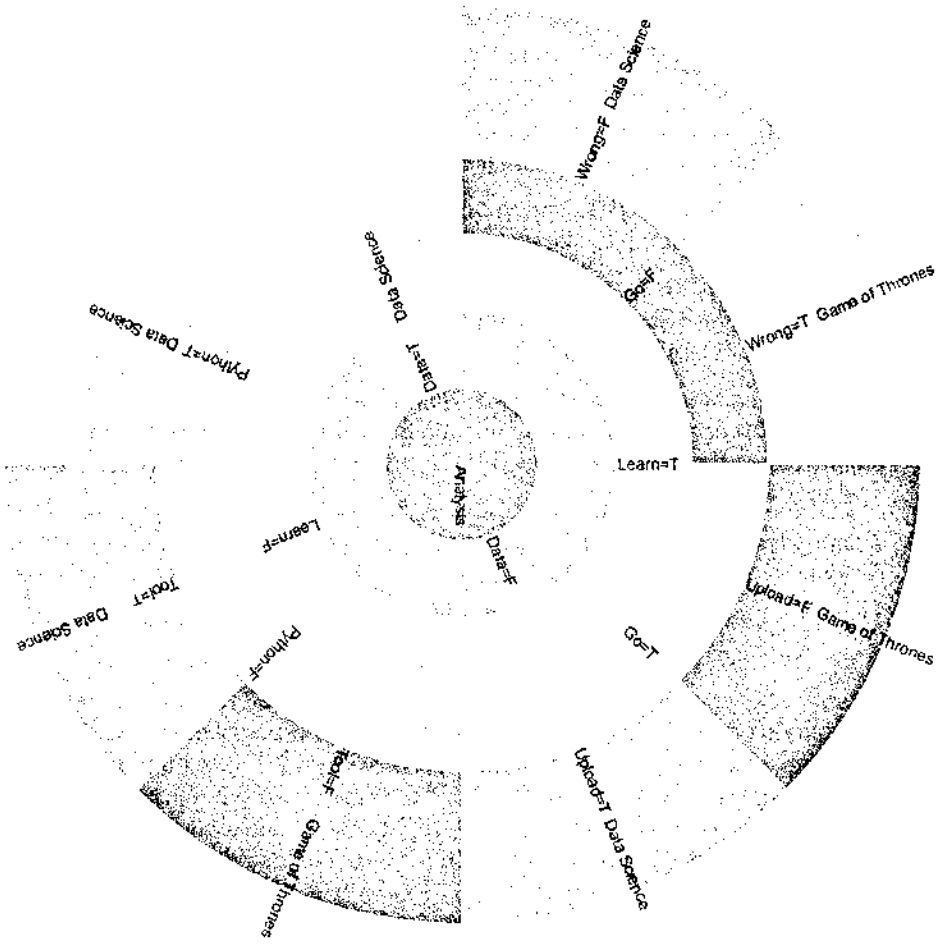


Рис. 8.28. Диаграмма, построенная для четырех верхних ветвей модели дерева принятия решений

покажутся им осмысленными. Оригинальная визуализация данных, примененная к месту, несомненно поможет вам в этом.

8.4. Итоги

- Глубокий анализ текста широко применяется для решения таких задач, как идентификация сущностей, выявление плагиата, идентификация темы, машинный перевод, выявление мошенничества, фильтрация спама и т. д.
- В Python существует полнофункциональный инструментарий для глубокого анализа текста NLTK (Natural Language Toolkit). NLTK хорошо подходит для

экспериментов и освоения азов; тем не менее для реальных приложений Scikit-learn обычно считается более «пригодным». Примеры использования Scikit-learn часто встречаются в предыдущих главах.

- Подготовка текстовых данных требует бо́льших усилий, чем подготовка числовых данных. В ней задействованы специальные приемы:
  - *Выделение основы* — осмысленное отсечение окончания слова, чтобы его можно было сопоставить со склоняемыми или множественными формами этого слова.
  - *Лемматизация* — как и выделение основы, предназначена для устранения дубликатов. В отличие от выделения основы, учитывает смысл слова.
  - *Фильтрация игнорируемых слов* — некоторые слова встречаются слишком часто, чтобы приносить пользу при анализе. Исключение таких слов может значительно улучшить модель. Игнорируемые слова часто зависят от конкретных корпусов текстов.
  - *Генерирование лексем* — разбиение текста на части. Лексемами могут быть отдельные слова, комбинации слов (n-граммы) и даже целые предложения.
  - *Пометка частей речи* — иногда бывает полезно знать функцию некоторого слова в предложении, чтобы лучше понять его смысл.
- В учебном примере мы попытались отличать сообщения Reddit по теме «Игры престолов» от сообщений на тему data science. В реализации примера были опробованы как наивный классификатор Байеса, так и классификатор на основе дерева принятия решений. Наивный классификатор Байеса предполагает, что все показатели независимы друг от друга; классификатор на основе дерева принятия решений предполагает зависимость и допускает применение разных моделей.
- В нашем примере наивный классификатор Байеса предоставляет более эффективную модель, но очень часто классификатор на основе дерева принятия решений лучше справляется с задачей, особенно при большем объеме доступных данных.
- Для оценки различий в эффективности используется матрица несоответствий, вычисленная после применения обеих моделей к новым (но помеченным) данным.
- Для представления результатов проведенных исследований желательно найти интересную визуализацию данных, позволяющую ярко и выразительно передать результаты.

Визуализация данных для конечного пользователя

В этой главе:

- ✓ Варианты визуализации данных для конечных пользователей.
- ✓ Подготовка базового MapReduce-приложения Crossfilter.
- ✓ Создание информационной панели с использованием dc.js.
- ✓ Работа со средствами создания информационных панелей.

Наверняка вы заметите, что эта глава отличается от глав 3–8 тем, что основное внимание здесь уделяется этапу 6 процесса data science. А если говорить более конкретно, здесь мы будем заниматься созданием небольшого приложения data science. По этой причине этапы процесса data science в этой главе подробно не рассматриваются. Данные, используемые в примере, реальны только отчасти, но они имитируют получение данных из фаз подготовки или моделирования данных.

Часто специалисту data science приходится заниматься представлением новых результатов конечному пользователю. Существует несколько способов передачи результатов:

- *Одноразовое представление результатов* — в зависимости от ответов на поставленные вопросы исследования будут приняты решения, определяющие стратегию организации на многие годы. Для примера возьмем инвестиционные решения: нужно ли распространять продукцию компании через два дистрибьюторских центра или хватит одного? Где следует расположить эти центры для достижения оптимальной эффективности? Когда решение будет принято, вполне возможно, что вам не придется заново отвечать на эти вопросы до ухода на пенсию. В таком случае результаты представляются в виде отчета, а созданное вами представление послужит «вишенкой на торте».
- *Новая точка зрения на данные* — наиболее очевидным примером служит сегментация клиентов. Конечно, сами сегменты могут быть описаны в отчетах и презентациях, но, по сути, они образуют инструменты, а не конечный резуль-

тат. Если вам удастся обнаружить четкую и актуальную систему сегментации клиентов, ее можно сохранить в базе данных как «новое измерение» для данных, на основе которых она была получена. В дальнейшем пользователи смогут создавать специализированные отчеты (например, с объемами продаж по каждому клиентскому сегменту).

- *Информационная панель для отображения данных в реальном времени* — в некоторых случаях ваша задача как специалиста data science не завершается с обнаружением новой информации. Информацию можно сохранить в базе данных и считать свою работу завершенной, но когда другие пользователи начнут строить отчеты на основе ваших открытий, они могут неправильно интерпретировать результаты и создать бессмысленные отчеты. Специалист data science, обнаруживший новую информацию, должен «задать направление»: он создает первый отчет с возможностью обновления информации, чтобы другие (в основном IT-сотрудники и аналитики) могли понять результаты и последовать за ним. Создание прототипа информационной панели также сократит время донесения результатов до конечного пользователя, который будет применять их в своей повседневной работе. В этом случае у пользователей хотя бы будет информация для работы, пока отдел учета и отчетности не создаст окончательный вариант отчета с использованием программного обеспечения, принятого в компании.

При этом приходится учитывать некоторые важные факторы:

- *Какого рода решения* обеспечивает ваша информация? Стратегические или оперативные? Для стратегических решений часто достаточно однократного анализа и представления результатов, тогда как оперативные решения требуют регулярного обновления отчетов.
- *Насколько велика ваша организация?* В малых организациях вы отвечаете за весь цикл: от сбора данных до представления результатов. В больших организациях может быть доступна группа специалистов по составлению отчетов, которая создаст информационную панель за вас. Впрочем, даже в последней ситуации может быть полезно создать прототип, потому что он формирует ориентиры и часто ускоряет получение конечного результата.

Хотя вся книга посвящена извлечению скрытой информации из данных, в этой последней главе мы сосредоточимся на построении работоспособной информационной панели. Создание презентаций для продвижения полученных результатов или представления стратегически важной информации выходит за рамки книги.

9.1. Способы визуализации данных

У вас есть несколько вариантов представления информационной панели для конечных пользователей. Здесь мы сосредоточимся на одном способе, а к концу главы вы сможете построить информационную панель самостоятельно.

В примере этой главы рассматривается больничная аптека, на складе которой хранятся несколько тысяч лекарств. Правительство приняло новую норму для всех аптек: все лекарства должны проверяться на реакцию на свет, а для их хранения должны использоваться новые специальные контейнеры. Однако при этом правительство не передало аптекам актуальный список светочувствительных лекарств. Для вас как для специалиста data science это не представляет проблемы, потому что к каждому лекарству прилагается инструкция по применению, где это указано. Для извлечения информации следует применить глубокий анализ текста и назначить каждому лекарству метку «чувствительно к свету» или «нечувствительно к свету». Полученная информация записывается в центральную базу данных. В частности, аптеке необходимо знать, сколько для этого потребуется контейнеров. По этой причине аптеки должны предоставить доступ к своим данным складских запасов. На рис. 9.1 показано, как выглядит набор данных, содержащий только необходимые переменные, открытый в Excel.

	A	B	C	D	E	F
1	MedName	LightSen	Date	StockOut	StockIn	Stock
2	Acupan 30 mg	No	1/01/2015	-8	150	142
3	Acupan 30 mg	No	2/01/2015	-6	5	141
4	Acupan 30 mg	No	3/01/2015	-2	0	139
5	Acupan 30 mg	No	4/01/2015	0	5	144
6	Acupan 30 mg	No	5/01/2015	-8	0	136
7	Acupan 30 mg	No	6/01/2015	-1	0	135
8	Acupan 30 mg	No	7/01/2015	-1	15	149
9	Acupan 30 mg	No	8/01/2015	-10	10	149
10	Acupan 30 mg	No	9/01/2015	-8	15	156

Рис. 9.1. Набор данных с запасами лекарств, открытый в Excel: первые 10 строк дополнены переменной чувствительности к свету LightSen

Как видите, информация представляет собой данные временного ряда за целый год перемещения складских запасов, поэтому каждое лекарство представлено в наборе данных 365 записями. Хотя учебный пример вполне реален, как и названия лекарств в наборе данных, значения других представленных переменных были сгенерированы случайно, так как исходные данные засекречены. Кроме того, набор данных ограничен 29 видами лекарств, что соответствует немногим более 10 000 строк данных. И хотя разработчики строят отчеты с использованием библиотек crossfilter.js (библиотека MapReduce для Javascript) и dc.js (библиотека для создания информационных панелей Javascript) более чем для миллионов строк данных, в этом примере используется малая часть этого количества. Кроме того, загружать всю базу данных в браузере пользователя не рекомендуется; браузер «подвисает» на время загрузки, а при слишком большом объеме данных даже произойдет аварийный сбой. Обычно данные заранее обрабатываются на сервере, после чего приложение запрашивает их по частям, например, через службу REST.

Есть много вариантов преобразования данных в информационную панель; краткая сводка этих инструментов приведена позднее в этой главе.

В частности, в этой книге мы решили выбрать dc.js — своего рода гибрид между CrossFilter (MapReduce-библиотекой для JavaScript) и библиотекой визуализации данных d3.js. Разработку Crossfilter ведет Square Register — компания, занимающаяся платежными операциями, своего рода аналог PayPal, но специализирующаяся на мобильных платежах. Компания Square разработала Crossfilter, чтобы предоставить своим клиентам исключительно быстрый инструмент для анализа истории платежей. Crossfilter не только единственная библиотека JavaScript с возможностью обработки MapReduce; она отлично справляется со своим делом, распространяется с открытым кодом, бесплатна для использования, а ее сопровождением занимается солидная компания (Square). Быть может, альтернативы для Crossfilter — Map.js, Meguro и Underscore.js. Возможно, JavaScript и не пользуется репутацией языка обработки данных, но эти библиотеки расширяют возможности браузеров на тот случай, если данные должны обрабатываться в браузере. Мы не будем углубляться в то, как использовать JavaScript для массовых вычислений в кооперативных распределенных инфраструктурах, но армия гномов может победить гиганта. Если эта тема интересует вас, обращайтесь за дополнительной информацией по адресам *https://www.igvita.com/2009/03/03/collaborative-map-reduce-in-the-browser/* и *http://dyn.com/blog/browsers-vs-servers-using-javascript-for-number-crunching-theories/*.

Библиотеку d3.js можно уверенно назвать наиболее универсальной библиотекой визуализации данных JavaScript на момент написания книги; она была разработана Майком Бостоком в качестве наследника его библиотеки Protovis. На базе d3.js построены многие библиотеки JavaScript.

NVD3, C3.js, xCharts и Dimple предоставляют примерно одно и то же: абстрактную прослойку на базе d3.js, упрощающую рисование простых диаграмм. Эти библиотеки различаются в основном типами поддерживаемых диаграмм и их оформлением по умолчанию. Заходите на их веб-сайты и убедитесь сами:

- NVD3 – *http://nvd3.org/*
- C3.js – *http://c3js.org/*
- xCharts – *http://tenxer.github.io/xcharts/*
- Dimple – *http://dimplejs.org/*

В общем, вариантов много. Тогда для чего использовать dc.js?

Основная причина: dc.js позволяет на удивление легко (по сравнению с эффектом) создать интерактивную информационную панель, у которой щелчок на одной диаграмме создает фильтрованные представления на связанных диаграммах. Это делается настолько легко, что к концу этой главы у нас появится работоспособный пример. Вы как специалист data science уже провели достаточно времени за анализом; простая в реализации информационная панель станет безусловно полезной.

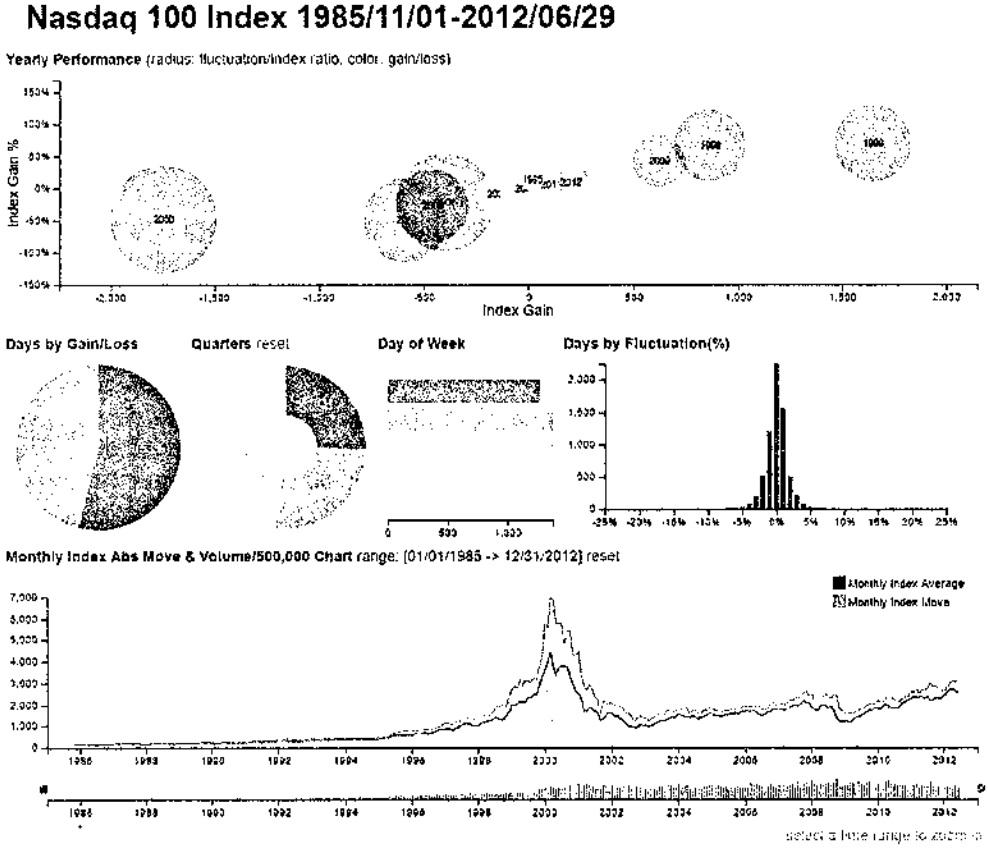


Рис. 9.2. Интерактивный пример dc.js на официальном веб-сайте

Чтобы получить представление о том, что нам предстоит создать, зайдите на веб-сайт *http://dc-js.github.io/dc.js/* и прокрутите страницу до примера NASDAQ (рис. 9.2).

Пощелкайте на информационной панели и посмотрите, как реагируют диаграммы на выделение и отмену выделения точек данных. Впрочем, не увлекайтесь экспериментами; пора создать такую панель самостоятельно.

Как упоминалось ранее, у dc.js есть два важных предусловия: d3.js и crossfilter.js. Изучение d3.js потребует основательных усилий; на эту тему написано несколько полезных книг, которые стоит прочитать, если вас интересует возможность полной настройки ваших визуализаций. Однако для того, чтобы использовать dc.js, хорошо знать эту библиотеку не обязательно, поэтому в книге эта тема подробно рассматриваться не будет. Crossfilter.js — другое дело; вы должны в некоторой степени знать эту библиотеку MapReduce, чтобы обеспечить работу dc.js с вашими данными. Но поскольку концепция MapReduce вам уже знакома, процесс пройдет достаточно гладко.

9.2. Crossfilter, библиотека MapReduce для JavaScript

JavaScript не самый лучший язык для обработки данных. Но это не мешает разным людям (например, работающим в Square) разрабатывать для него библиотеки MapReduce. Если вы работаете с данными, даже незначительный прирост скорости полезен. Впрочем, пересылать огромные массивы данных по Интернету и даже по внутренней сети нежелательно по следующим причинам:

- Отправка больших объемов данных повышает нагрузку на *сеть* до состояния, в котором она начинает создавать проблемы для других пользователей.
- Браузер находится на стороне получателя, и на время загрузки данных его работа приостанавливается. Для небольших объемов данных пауза незаметна, но когда вы начинаете проводить поиск по 100 000 строк, появляется ощутимая задержка. А если объем данных превышает 1 000 000 строк (в зависимости от ширины данных), все может кончиться аварийным завершением браузера.

Вывод: суть происходящего – поиск сбалансированного решения. Для отправляемых данных есть библиотека Crossfilter, которая обрабатывает данные после их поступления. В нашем примере аптека запросила с центрального сервера данные за 2015 год по 29 видам лекарств, представляющим особый интерес. Данные уже были представлены ранее, поэтому обратимся к самому приложению.

9.2.1. Подготовка необходимых компонентов

Пора переходить к построению приложения. Наше небольшое приложение dc.js включает следующие компоненты:

- JQuery — обеспечение интерактивности.
- Crossfilter.js — библиотека MapReduce, необходимая для dc.js.
- d3.js – популярная библиотека визуализации данных, необходимая для dc.js.
- dc.js – библиотека визуализации, используемая для создания интерактивной информационной панели.
- Bootstrap — широко используемая библиотека макетирования, с помощью которой можно улучшить внешний вид приложения.

Мы напишем всего три файла:

- index.html — страница HTML, содержащая приложение.
- application.js — для хранения всего кода JavaScript, написанного вами.
- application.css — для вашего кода CSS.

Кроме того, необходимо обеспечить выполнение кода на сервере HTTP. Вы можете потратить усилия на настройку сервера LAMP (Linux, Apache, MySQL, PHP), WAMP

(Windows, Apache, MySQL, PHP) или XAMPP (Cross Environment, Apache, MySQL, PHP, Perl), но для простоты мы не будем возиться с настройкой серверов, а ограничимся одной командой Python. Воспользуйтесь режимом командной строки (командным интерпретатором в Linux или cmd в Windows) и перейдите в папку с файлом index.html. Поддержка Python должна быть установлена для других глав книги, поэтому следующая команда должна запустить сервер HTTP Python на локальном хосте:

```
python -m SimpleHTTPServer
```

Для Python 3.4 команда выглядит так:

```
python -m http.server 8000
```

Как видно из рис. 9.3, сервер HTTP запускается на порте 8000 локального хоста. В вашем браузере это соответствует адресу «localhost:8000»; строка «0.0.0.0:8000» не подойдет.

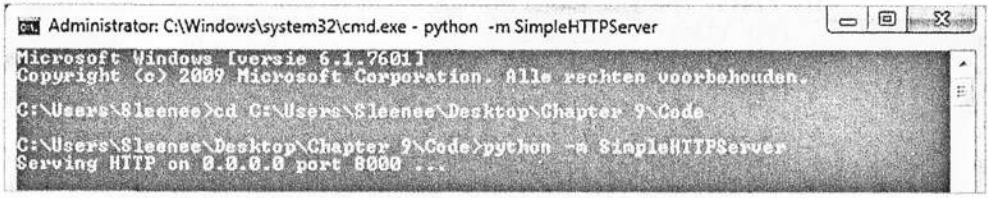


Рис. 9.3. Запуск простого сервера HTTP Python

Убедитесь в том, что все необходимые файлы находятся в одной папке с файлом index.html. Эти файлы можно загрузить с сайта Manning или с сайтов их создателей:

- dc.css и dc.min.js — *https://dc-js.github.io/dc.js/*
- d3.v3.min.js — *http://d3js.org/*
- crossfilter.min.js — *http://square.github.io/crossfilter/*

Теперь вы знаете, как будет запускаться код, который мы собираемся создать, и мы можем обратиться к странице index.html, которая представлена в листинге 9.1.

Никаких сюрпризов. Заголовок содержит все библиотеки CSS, которые будут использоваться в приложении, поэтому код JavaScript загружается в конце тела HTML. Благодаря обработчику JQuery `onload` ваше приложение будет загружено тогда, когда оставшаяся часть страницы будет готова. Сначала создаются два заполнителя для таблиц: одна таблица показывает, как выглядят исходные данные, `<div id="input-table"></div>`, а другая будет использоваться Crossfilter для вывода отфильтрованной таблицы, `<div id="filteredtable"></div>`. Также используются некоторые классы CSS из Bootstrap (такие, как «well», «container», сетка Bootstrap с «row» и «col-xx-xx» и т. д.). С ними результат выглядит лучше, но они не являются обязательными. За дополнительной информацией о CSS-классах Bootstrap обращайтесь на сайт *http://getbootstrap.com/css/*.

Листинг 9.1. Исходная версия index.html

```
<html>
<head>
    <title>Chapter 10.  Data Science Application</title>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
bootstrap /3.3.0/css/bootstrap.min.css">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
bootstrap/3.3.0/css/bootstrap-theme.min.css">

    <link rel="stylesheet" href="dc.css">
    <link rel="stylesheet" href="application.css">
</head>
<body>

    <main class='container'>
        <h1>Chapter 10:  Data Science Application</h1>
        <div class="row">
            <div class='col-lg-12'>
                <div id="inputtable" class="well well-sm"></div>
            </div>
        </div>
        <div class="row">
            <div class='col-lg-12'>
                <div id="filteredtable" class="well well-sm"></div>
            </div>
        </div>
    </main>

    <script src="https://code.jquery.com/jquery-1.9.1.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0/js
/bootstrap.min.js"></script>

    <script src="crossfilter.min.js"></script>
    <script src="d3.v3.min.js"></script>
    <script src="dc.min.js"></script>
    <script src="application.js"></script>
</body>
</html>
```

Вся разметка CSS находится здесь.

Используйте файл dc.css, загруженный со страницы издательства Manning или с сайта dc: https://dc-js.github.io/dc.js/. Файл должен находиться в одной папке с файлом index.html.

В главном контейнере собраны все элементы, видимые пользователю.

Используйте файлы crossfilter.min.js, d3.v3.min.js и dc.min.js, загруженные со своих сайтов или со страницы издательства Manning. Crossfilter: http://square.github.io/crossfilter/, d3.js: http://d3js.org/, dc.min.js: https://dc-js.github.io/dc.js/.

Здесь загружается весь код JavaScript.

Разметка HTML готова, теперь данные нужно вывести на экране. Для этого обратитесь к созданному вами файлу application.js. Сначала весь код заключается в обработчик события JQuery onload:

```
$(function() {
    // Весь будущий код будет заключен здесь
})
```

Теперь можно быть уверенным в том, что приложение будет загружаться только тогда, когда готовы все остальные элементы. Это важно, потому что мы будем использовать селекторы JQuery для манипуляций с HTML. Теперь нужно загрузить данные:

```
d3.csv('medicines.csv',function(data) {
    main(data)
});
```

У нас нет готовой REST-службы, которая могла бы предоставить данные, поэтому в этом примере данные будут загружены из файла .csv. Этот файл доступен для загрузки на сайте Manning. В d3.js для работы с такими данными существует удобная функция. После загрузки данных они передаются главной функции приложения в функции обратного вызова d3.csv.

Кроме главной функции приложения, также имеется функция `CreateTable`, которая... да, вы угадали — она создаст таблицы (листинг 9.2).

Листинг 9.2. Функция CreateTable

```
var tableTemplate = $([
        "<table class='table table-hover table-condensed table-striped'>",
        "  <caption></caption>",
        "  <thead><tr/></thead>",
        "  <tbody></tbody>",
        "</table>"
    ].join('\n'));
    CreateTable = function(data,variablesInTable,title){
        var table = tableTemplate.clone();
        var ths = variablesInTable.map(function(v) { return $("<th>").text(v)
});
        $('caption', table).text(title);
        $('thead tr', table).append(ths);
        data.forEach(function(row) {
            var tr = $("<tr>").appendTo($('tbody', table));
            variablesInTable.forEach(function(varName) {
                var val = row, keys = varName.split('.');
                keys.forEach(function(key) { val = val[key] });
                tr.append($("<td>").text(val));
            });
        });
        return table;
    }
```

Функция `CreateTable()` получает три аргумента:

- `data` — данные, которые необходимо поместить в таблицу.
- `variablesInTable` — переменные, которые должны отображаться.
- `Title` — заголовок таблицы (чтобы пользователь понимал, что именно он видит).

Функция `CreateTable()` использует заранее определенную переменную `tableTemplate` с описанием общей структуры таблицы. `CreateTable()` может добавлять строки данных к этому шаблону.

Теперь у нас есть все необходимое; перейдем к функции `main` приложения (листинг 9.3).

Листинг 9.3. Функция main программы JavaScript

```
Переменные, выводимые в таблице,
помещаются в массив, чтобы их можно было перебирать при создании кода таблицы.

main = function(inputdata){

    var medicineData = inputdata;                    <- Наши данные: обычно загружаются с сервера, но в этом случае читаются из локального файла .csv.

    var dateFormat = d3.time.format("%d/%m/%Y");     <- Дата преобразуется к правильному формату, чтобы переменная распознавалась Crossfilter.
    medicineData.forEach(function (d) {
        d.Day = dateFormat.parse(d.Date);
    })
    var variablesInTable =
    ['MedName','StockIn','StockOut','Stock','Date','LightSen']
            var sample = medicineData.slice(0,5);    <- Выводится только небольшое подмножество данных.
            var inputTable = $("#inputtable");       <- Создание таблицы.
        inputTable
            .empty()
            .append(CreateTable(sample,variablesInTable,"The input table"));
}
```

Для начала данные нужно просто вывести на экран, но желательно ограничиться небольшой их частью; в нашем случае хватит первых пяти записей (рис. 9.4). В данных присутствует переменная даты, и мы хотим, чтобы библиотека Crossfilter позднее распознавала ее как дату, поэтому сначала мы разбираем ее и создаем новую переменную с именем `Day`. Пока в таблице выводится оригинал (`Date`), но во всех последующих вычислениях будет использоваться `Day`.

The input table

MedName	StockIn	StockOut	Stock	Date	LightSen
Acupan 30 mg	150	-7	143	1/01/2015	No
Acupan 30 mg	6	-6	142	2/01/2015	No
Acupan 30 mg	15	-9	148	3/01/2015	No
Acupan 30 mg	0	-11	137	4/01/2015	No
Acupan 30 mg	10	-8	139	5/01/2015	No

Рис. 9.4. Таблица с входными данными в браузере: выводятся первые пять строк

В итоге получилась таблица, которая мало чем отличается от приведенной выше таблицы Excel. А теперь, когда вы поняли, как работает этот механизм, мы воспользуемся возможностями Crossfilter.

9.2.2. Использование Crossfilter для фильтрации набора данных

Для применения фильтрации и MapReduce будет использоваться библиотека Crossfilter. Весь последующий код можно разместить после кода из раздела 9.2.1 внутри функции `main()`. Первое, что необходимо сделать, – это создать экземпляр Crossfilter и инициализировать его вашими данными:

```
CrossfilterInstance = crossfilter(medicineData);
```

После этого можно браться за работу. Созданный экземпляр используется для регистрации *измерений* (dimensions), которые являются столбцами вашей таблицы. В настоящее время Crossfilter ограничивается 32 измерениями. Если вы работаете с данными, ширина которых превышает 32 измерения, придется сократить их ширину перед отправкой браузеру. Создадим первое измерение с названием лекарства:

```
var medNameDim = CrossfilterInstance.dimension(function(d) {return
d.MedName;});
```

Первое измерение (название лекарства) уже может использоваться для фильтрации набора данных и вывода отфильтрованных данных нашей функцией `CreateTable()`:

```
        var dataFiltered= medNameDim.filter('Grazax 75 000 SQ-T')
        var filteredTable = $('#filteredtable');
        filteredTable
      .empty().append(CreateTable(dataFiltered.top(5),variablesInTable,'Our
First Filtered Table'));
```

Выводятся только первые пять наблюдений (рис. 9.5); всего их 365, потому что используются данные одного лекарства за целый год.

Our First Filtered Table

MedName	StockIn	StockOut	Stock	Date	LightSen
Grazax 75 000 SQ-T	15	0	205	31/08/2015	Yes
Grazax 75 000 SQ-T	0	-4	62	30/12/2015	Yes
Grazax 75 000 SQ-T	10	-15	66	29/12/2015	Yes
Grazax 75 000 SQ-T	15	0	71	28/12/2015	Yes
Grazax 75 000 SQ-T	10	-4	56	27/12/2015	Yes

Рис. 9.5. Данные, отфильтрованные по названию лекарства (Grazax 75 000 SQ-T)

Таблица отсортирована, хотя это сразу и не очевидно. Функция `top()` отсортировала ее по названию лекарства. Так как выбрано всего одно лекарство, это ни на что не влияет. Данные можно легко отсортировать по дате при помощи новой переменной `Day`. Зарегистрируем еще одно измерение для даты:

```
        var DateDim = CrossfilterInstance.dimension(
function(d) {return d.Day;});
```

Теперь можно переключиться на сортировку по дате вместо сортировки по названию:

```
        filteredTable
                .empty()
                .append(CreateTable(DateDim.bottom(5),variablesInTable,'Our
First Filtered Table'));
```

Результат, показанный на рис. 9.6, выглядит гораздо лучше.

Our First Filtered Table

MedName	StockIn	StockOut	Stock	Date	LightSen
Grazax 75 000 SQ-T	55	-12	53	1/01/2015	Yes
Grazax 75 000 SQ-T	15	-11	57	2/01/2015	Yes
Grazax 75 000 SQ-T	5	-9	53	3/01/2015	Yes
Grazax 75 000 SQ-T	5	-4	54	4/01/2015	Yes
Grazax 75 000 SQ-T	0	-14	40	5/01/2015	Yes

Рис. 9.6. Данные, отфильтрованные по названию лекарства (Grazax 75 000 SQ-T) и отсортированные по дате

Таблица создает окно для просмотра данных, но пока не обобщает их. И здесь на помощь приходит функциональность MapReduce в Crossfilter. Допустим, вы хотите узнать, сколько наблюдений существует для одного лекарства. Логика подсказывает, что для всех лекарств должно получиться одно и то же число: 365, или по одному наблюдению на каждый день 2015 года.

```
        var countPerMed = medNameDim.group().reduceCount();
        variablesInTable = ["key","value"]
        filteredTable
                .empty()
      .append(CreateTable(countPerMed.top(Infinity),
variablesInTable,'Reduced Table'));
```

Crossfilter включает две функции MapReduce: `reduceCount()` и `reduceSum()`. Если ваши потребности не ограничиваются подсчетом и суммированием, вам придется написать нужную функцию свертки. Переменная `countPerMed` теперь содержит данные, сгруппированные по измерению названия, и счетчик строк для каждого лекарства в форме «ключ—значение». Чтобы создать таблицу, следует указать переменную `key` вместо `medName` и `value` в столбце счетчика (рис. 9.7).

Reduced Table

key	value
Adoport 1 mg	365
Atenolol EG 100 mg	365
Ceftriaxone Actavis 1 g	365
Cefuroxim Mylan 500 mg	365
Certican 0.25 mg	365

Рис. 9.7. Таблица с обобщенными данными: название лекарства обеспечивает группировку, а в столбце value выводится количество строк данных

Используя конструкцию `.top(Infinity)`, вы приказываете вывести на экран все 29 лекарств, но для экономии места на рис. 9.7 показаны только первые пять результатов. Что же, можно расслабиться: данные содержат 365 строк для каждого лекарства. Обратите внимание: Crossfilter игнорирует фильтр «Grazax». Если измерение используется для группировки, то фильтр к нему не применяется. Только фильтры по другим измерениям могут сужать результаты.

Как насчет более интересных вычислений, которые не входят в комплект Crossfilter, например вычисление среднего арифметического? Такая возможность существует, но вам придется написать три функции и передать их методу `.reduce()`. Допустим, вы хотите вычислить средний запас каждого лекарства. Как упоминалось ранее, почти вся логика MapReduce должна быть написана вами. Среднее арифметическое представляет собой не что иное, как результат деления суммы на количество. Кроме функций `reduceCount()` и `reduceSum()` в Crossfilter существует более общая функция `reduce()`. Эта функция получает три аргумента:

- Функция `reduceAdd()` — функция, которая описывает, что должно происходить при добавлении нового наблюдения.
- Функция `reduceRemove()` — функция, которая описывает, что должно происходить при исключении наблюдения (например, из-за применения фильтра).
- Функция `reduceInit()` — функция задает начальные значения для всех вычисляемых показателей. Для суммы и количества наиболее логичной начальной точкой будет 0.

Рассмотрим отдельные функции свертки, прежде чем пытаться вызывать метод Crossfilter `.reduce()`, в аргументах которого передаются эти три компонента. Для работы обобщенной функции свертки необходимы три компонента: инициализация, функция добавления и функция удаления. Функция инициализации задает начальные значения объекта p:

```
var reduceInitAvg = function(p,v){
    return {count: 0, stockSum : 0, stockAvg:0};
}
```

Как видно из этого фрагмента, сами функции свертки получают два аргумента. Значения аргументов автоматически передаются им методом Crossfilter `.reduce()`:

- p — объект, содержащий описание ситуации на текущий момент, сохраняет свое значение между наблюдениями. Эта переменная отслеживает сумму и количество, а следовательно, представляет цель, т. е. конечный результат.
- v представляет запись входных данных, а все переменные этой записи доступны для вас. В отличие от p, эта переменная не сохраняется, а заменяется новой строкой данных при каждом вызове функции. Функция `reduceInit()` вызывается только один раз, `reduceAdd()` вызывается при каждом добавлении новой записи, а `reduceRemove()` — при каждом удалении строки данных.
- Функция `reduceInit()` (здесь она называется `reduceInitAvg()`, потому что мы собираемся вычислять среднее значение), по сути, инициализирует объект p, определяя его компоненты (`count`, `sum` и `average`) и задавая их исходные значения. Взгляните на код `reduceAddAvg()`:

```
var reduceAddAvg = function(p,v){
    p.count += 1;
    p.stockSum  = p.stockSum  + Number(v.Stock);
    p.stockAvg = Math.round(p.stockSum  / p.count);
    return p;
}
```

Функция `reduceAddAvg()` получает те же аргументы p и v, но на этот раз значение v реально используется; для задания исходных значений данные не нужны. Значение `Stock` суммируется для всех добавляемых записей, после чего среднее значение вычисляется на основании вычисленной суммы и количества записей:

```
var reduceRemoveAvg = function(p,v){
    p.count -= 1;
    p.stockSum  = p.stockSum  -  Number(v.Stock);
    p.stockAvg = Math.round(p.stockSum  / p.count);
    return p;
}
```

Функция `reduceRemoveAvg()` выглядит аналогично, но решает противоположную задачу: когда запись удаляется, счетчик и сумма уменьшаются. Среднее значение всегда вычисляется одинаково, поэтому изменять формулу не нужно.

А теперь наступает момент истины: самодельная функция MapReduce применяется к набору данных:

```
        dataFiltered = medNameDim.group().reduce(reduceAddAvg
reduceRemoveAvg,reduceInitAvg)
        variablesInTable = ["key","value.stockAvg"]
        filteredTable
                .empty()
      .append(CreateTable(dataFiltered.top(Infinity),
variablesInTable,'Reduced Table'));
```

Все как обычно: вывод таблицы результатов.

Функции reduce() во входных аргументах передаются 3 функции (reduceInitAvg(), reduceAddAvg() и reduceRemoveAvg()).

Обратите внимание на изменение выходной переменной с `value` на `value.stockAvg`. Так как вы сами определяли функции свертки, при желании вы можете вывести сразу несколько переменных. Так, `value` преобразуется в объект, содержащий все вычисленные переменные; `stockSum` и `count` здесь тоже присутствуют.

Результаты говорят сами за себя (рис. 9.8). Вероятно, отрицательный средний запас в строке `Cimalgex` обусловлен тем, что лекарство передавалось из других больниц.

Собственно, это все, что необходимо знать о Crossfilter для работы с dc.js. Давайте возьмемся за дело и создадим интерактивные диаграммы.

Reduce() Table

key	value.stockAvg
Adoport 1 mg	36
Atenolol EG 100 mg	49
Ceftriaxone Actavis 1 g	207
Cefuroxim Mylan 500 mg	118
Certican 0.25 mg	158
Cimalgex 8 mg	-24

Рис. 9.8. Таблица со средними запасами лекарств на складе

9.3. Создание информационной панели с использованием dc.js

После знакомства с азами Crossfilter наступило время сделать последний шаг: построить информационную панель. Начнем с выделения места для диаграмм на странице index.html.

Новая разметка тела страницы приведена в листинге 9.4. В целом она похожа на исходную разметку, если не считать появления новых тегов `<div>` и тега кнопки сброса `<button>`.

В разметке применяется форматирование Bootstrap, но самыми важными элементами здесь являются три тега `<div>` с идентификаторами и кнопка. Мы хотим построить представление общих запасов по времени `<div id="StockOverTime"></div>`, диаграмму с возможностью фильтрации по лекарствам `<div id="StockPerMedicine"></div>` и диаграмму чувствительности лекарств к свету `<div id="LightSensitiveStock"></div>`. Также в разметку включается кнопка для сброса всех фильтров, `<button class="btn btn-success">Reset Filters</button>`. Кнопка сброса не обязательна, но удобна.

А теперь обратимся к application.js. Весь код можно добавить в функцию `main()`, как и прежде. Однако у правила есть одно исключение: `dc.renderAll()`, команда dc для рисования диаграмм. Эту команду достаточно включить всего один раз, в конце функции `main()`. Первая диаграмма представляет динамику общих запасов

Листинг 9.4. Обновленная версия index.html с местом для диаграмм, сгенерированных в dc.js

```
Структура:
| исходная таблица                                                    | (строка 1)
| отфильтрованная таблица                                             | (строка 2)
[кнопка сброса]
| диаграмма динамики запасов по времени | диаграмма динамики запасов по типу лекарства | (строка 3)
| диаграмма чувствительности к свету |                                | (строка 4)
  (столбец 1 )      (столбец 1 )

    <body>
        <main class='container'>

            <h1>Chapter 10:  Data Science Application</h1>
            <div class="row">
                <div class='col-lg-12'>
                    <div id="inputtable" class="well well-sm">
    </div>

                    </div>
                </div>
                <div class="row">
                    <div class='col-lg-12'>
                        <div id="filteredtable" class="well well-sm">
    </div>//

          </div>
      </div>

      <button class="btn btn-success">Reset Filters</button>
       <div class="row">
           <div class="col-lg-6">
               <div id="StockOverTime" class="well well-sm"></div>

               <div id="LightSensitiveStock" class="well well-sm"></div>

             </div>
             <div class="col-lg-6">
                 <div id="StockPerMedicine" class="well well-sm"></div>
             </div>
         </div>
     </main>
```

Заполнитель для исходной таблицы, которая будет вставлена позднее.

Заполнитель для отфильтрованной таблицы, которая будет вставлена позднее.

Новый элемент: кнопка сброса.

Новый элемент: заполнитель для круговой диаграммы чувствительности к свету.

Новый элемент: заполнитель для диаграммы запасов по типу лекарства.

Новый элемент: заполнитель для диаграммы динамики запасов по времени.

```
    <script src="https://code.jquery.com/jquery-1.9.1.min.js"></script>

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap
/3.3.0/js/bootstrap.min.js"></script>

    <script src="crossfilter.min.js"></script>
    <script src="d3.v3.min.js"></script>
    <script src="dc.min.js"></script>

    <script src="application.js"></script>
</body>
```

Библиотеки Crossfilter, d3 и dc можно загрузить с их веб-сайтов.

Код JavaScript нашего приложения.

Стандартный прием – библиотеки JS загружаются последними, чтобы ускорить загрузку страницы.

- **jQuery: необходимое взаимодействие HTML-JavaScript**
- **Bootstrap: упрощенные средства работы с CSS и макетом**
- **Crossfilter: выбранная нами MapReduce-библиотека для JavaScript**
- **d3: сценарий d3, необходимый для выполнения dc.js**
- **DC: наша библиотека визуализации**
- **application: вся логика нашего приложения**
- ***.min.js: минимизированный код JavaScript для сторонних библиотек**

по времени (листинг 9.5). Временное измерение уже объявлено, так что остается просуммировать запасы по времени.

Листинг 9.5. Код для построения диаграммы динамики запасов по времени

```
    var SummatedStockPerDay =
DateDim.group().reduceSum(function(d){return d.Stock;})
    var minDate = DateDim.bottom(1)[0].Day;
    var maxDate = DateDim.top(1)[0].Day;
    var StockOverTimeLineChart = dc.lineChart("#StockOverTime");

    StockOverTimeLineChart
            .width(null) // null - по размерам контейнера
            .height(400)
            .dimension(DateDim)
            .group(SummatedStockPerDay)
            .x(d3.time.scale().domain([minDate,maxDate]))
            .xAxisLabel("Year 2015")
            .yAxisLabel("Stock")
            .margins({left: 60, right: 50, top: 50, bottom: 50})

dc.renderAll();
```

Диаграмма динамики запасов по времени.

Линейный график.

Диаграмма состояния запасов по дням.

Отображение всех диаграмм.

Разберемся, что же здесь происходит. Сначала необходимо вычислить диапазон по оси x, чтобы библиотека dc.js знала, где должен начинаться и заканчиваться график. Затем выполняется инициализация и настройка графика. Пожалуй, наименее очевидными здесь являются методы `.group()` и `.dimension()`. Метод `.group()` получает временное измерение и представляет ось x. Парный метод `.dimension()` представляет ось y и получает на входе просуммированные данные. График на рис. 9.9 выглядит довольно банально, но внешность бывает обманчивой.

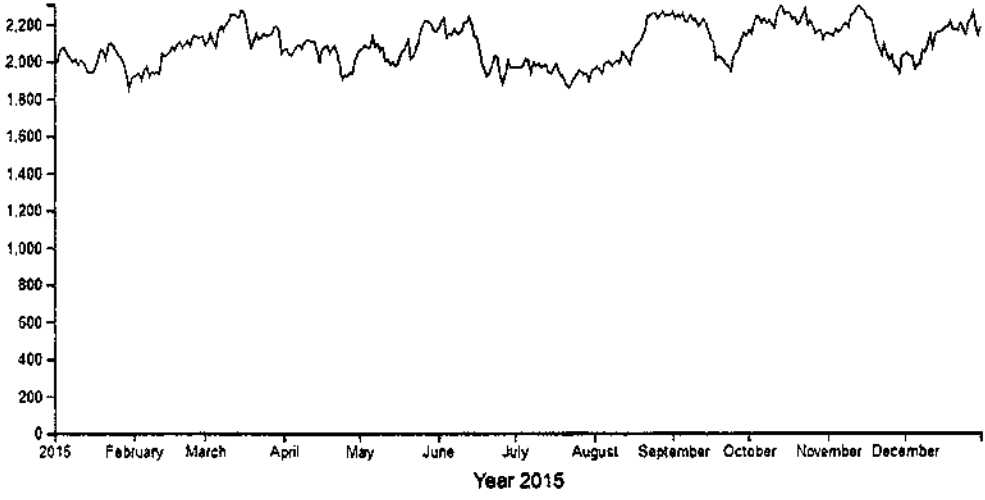


Рис. 9.9. График, построенный с использованием dc.js: сумма запасов лекарств за 2015 год

Ситуация кардинально меняется с добавлением второго элемента, поэтому мы создадим строковую диаграмму, представляющую средние запасы лекарств (листинг 9.6).

Листинг 9.6. Код для построения строковой диаграммы средних запасов в зависимости от типа лекарства

```
var AverageStockPerMedicineRowChart = dc.rowChart("#StockPerMedicine");
var AvgStockMedicine =  medNameDim.group().reduce(reduceAddAvg,
reduceRemoveAvg,reduceInitAvg);

AverageStockPerMedicineRowChart
                .width(null)
                .height(1200)
                .dimension(medNameDim)
                .group(AvgStockMedicine)
                .margins({top: 20, left: 10, right: 10, bottom: 20})
                .valueAccessor(function (p) {return p.value.stockAvg;});
```

Диаграмма средних запасов по типам лекарств.

Null означает "по размерам контейнера".

Выглядит знакомо: по сути это графическое представление таблицы, созданной ранее. Один важный момент: так как на этот раз функция `reduce()` определяется пользователем, dc.js не знает, какие данные следует отображать. При помощи метода `.valueAccessor()` можно задать `p.value.stockAvg` отображаемым значением. По умолчанию меткам строковой диаграммы dc.js назначается серый цвет; это несколько затрудняет восприятие диаграммы. Ситуацию можно исправить переопределением CSS в файле application.css:

```
.dc-chart g.row text {fill: black;}
```

Всего одна простая строка способна существенно улучшить внешний вид диаграммы (рис. 9.10).

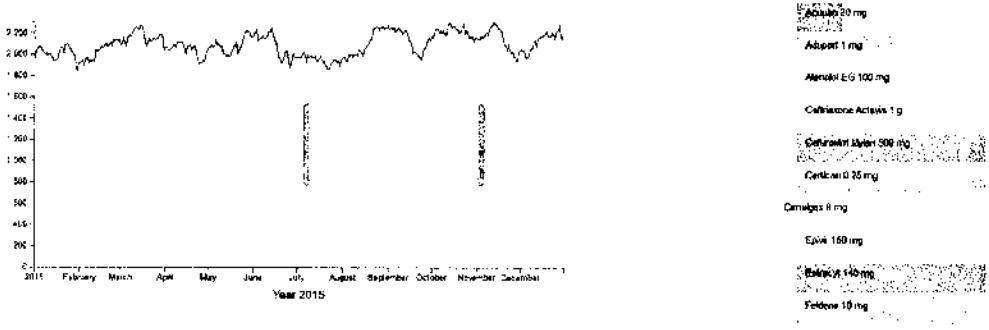


Рис. 9.10. Взаимодействие графика dc.js и строковой диаграммы

Если теперь выделить область на графике, строковая диаграмма автоматически изменяется, и на ней выводятся данные за правильный период времени. И наоборот, при выделении одного или нескольких лекарств на строковой диаграмме график изменяется соответствующим образом. Наконец, добавим измерение чувствительности к свету, чтобы фармацевт мог различать лекарства по этому признаку (листинг 9.7).

Листинг 9.7. Добавление измерения чувствительности к свету

```
        var lightSenDim = CrossfilterInstance.dimension(
function(d){return d.LightSen;});
        var SummatedStockLight =  lightSenDim.group().reduceSum(
function(d) {return d.Stock;});

        var LightSensitiveStockPieChart = dc.pieChart("#LightSensitiveStock");
        LightSensitiveStockPieChart
                .width(null) // null - по размерам контейнера
                .height(300)
                .dimension(lightSenDim)
                .radius(90)
                .group(SummatedStockLight)
```

Измерение чувствительности к свету еще не было добавлено; его необходимо сначала зарегистрировать в экземпляре Crossfilter. Также добавим кнопку сброса всех фильтров (листинг 9.8).

Листинг 9.8. Кнопка сброса фильтров

```
resetFilters = function(){//
    StockOverTimeLineChart.filterAll();
    LightSensitiveStockPieChart.filterAll();
    AverageStockPerMedicineRowChart.filterAll();
    dc.redrawAll();
}
$('.btn-success').click(resetFilters);
```

Функция resetFilters() сбрасывает данные dc.js и перерисовывает диаграммы.

При щелчке на кнопке с классом btn-success (кнопке сброса) вызывается функция resetFilters().

Метод `.filterAll()` удаляет все фильтры для заданного измерения; тогда `dc.redrawAll()` инициирует перерисовку всех диаграмм dc.

В результате получается информационная панель (рис. 9.11), с помощью которой фармацевт сможет получить представление о динамике складских запасов.

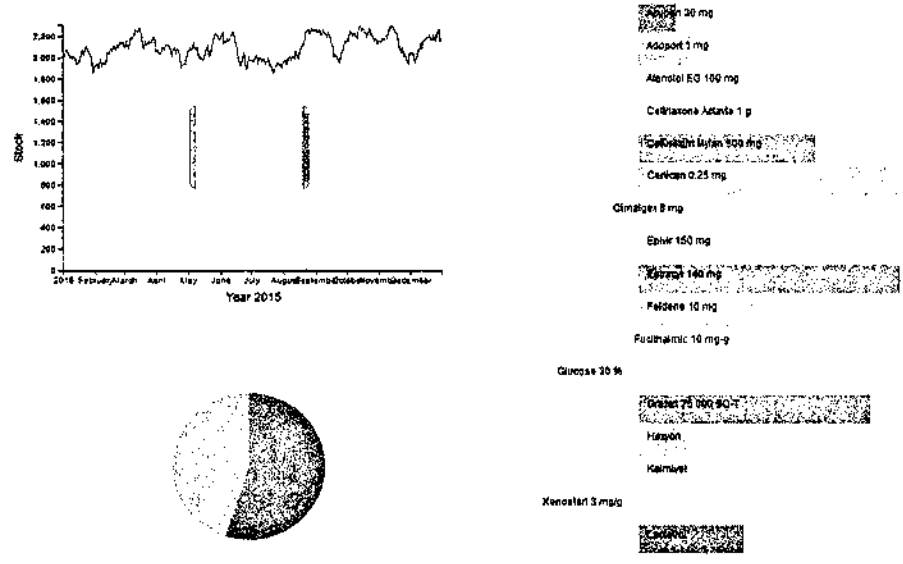


Рис. 9.11. Полностью интерактивная информационная панель, построенная с использованием dc.js, и информация о состоянии запасов лекарств

9.4. Средства разработки

Наша замечательная информационная панель готова, но мы хотим завершить эту главу коротким (и далеко не полным) обзором альтернативных программных средств, используемых для эстетичного представления числовых данных.

Вы можете воспользоваться проверенными полноценными пакетами от известных разработчиков, таких как Tableau, MicroStrategy, Qlik, SAP, IBM, SAS, Microsoft, Spotfire и т. д. Все эти компании предлагают инструменты для создания информационных панелей, заслуживающие вашего внимания. Если вы работаете в большой компании, скорее всего, по меньшей мере один из этих платных инструментов будет вам доступен. Разработчики также предлагают бесплатные пробные версии с ограниченной функциональностью. Обязательно присмотритесь к Tableau (*http://www.tableausoftware.com/public/download*), если вы еще не сделали этого ранее.

Другие компании предоставляют пробную версию своих продуктов. В конечном итоге вам придется заплатить за полную версию любого из этих пакетов, но вполне возможно, что эти затраты окупятся, особенно если крупная компания может себе это позволить.

Однако в этой книге основное внимание уделяется бесплатным инструментам. В начале изучения бесплатных инструментов визуализации данных вы быстро окажетесь в мире HTML, в котором хватает бесплатных библиотек JavaScript для представления любых данных. Выбор огромен:

- *HighCharts* — одна из самых зрелых библиотек построения диаграмм в браузере. Бесплатная лицензия относится только к некоммерческому использованию. Если вы захотите использовать эту библиотеку в коммерческом контексте, вам это обойдется в сумму от $90 до $4000. См. *http://shop.highsoft.com/highcharts.html*.
- *Chartkick* — библиотека построения JavaScript-диаграмм для любителей Ruby on Rails. См. *http://ank-ane.github.io/chartkick/*.
- *Google Charts* — бесплатная библиотека построения диаграмм от компании Google. Как и многие продукты Google, эта библиотека бесплатна даже для коммерческого использования, и она поддерживает множество разных видов диаграмм. См. *https://developers.google.com/chart/*.
- *d3.js* — этот вариант отличается от других, потому что это не библиотека построения диаграмм, а библиотека визуализации данных. На первый взгляд может показаться, что отличие не столь существенно, но оно ведет к серьезным последствиям. Если такие библиотеки, как HighCharts и Google Charts, предназначены для построения диаграмм заранее определенных типов, в d3.js таких ограничений нет. В настоящее время d3.js является наиболее гибкой библиотекой визуализации данных JavaScript. Чтобы понять, чем она отличается от традиционных библиотек построения диаграмм, достаточно вкрат-

це ознакомиться с интерактивными примерами на официальном веб-сайте: *http://d3js.org/*.

Конечно, есть и другие библиотеки, которые здесь не упомянуты.

Вы также можете воспользоваться библиотеками визуализации с пробным периодом, у которых нет бесплатной версии, такими как Wijmo, Kendo и FusionCharts. К ним тоже стоит присмотреться, потому что эти библиотеки предоставляют поддержку и гарантируют регулярные обновления.

Словом, вариантов достаточно. Но почему и когда следует хотя бы рассматривать возможность построения собственного интерфейса HTML5 вместо применения таких альтернатив, как SAP BusinessObjects, SAS JMP, Tableau, Clickview или одной из многих других? Причин может быть несколько:

- *Отсутствие бюджета* — если вы работаете в начинающей фирме или небольшой компании, лицензионные затраты на приобретение таких программных продуктов могут быть достаточно высокими.
- *Высокая доступность* — результаты работы приложений data science должны быть доступны любым пользователям, и особенно людям, в распоряжении которых может быть только браузер. Визуализация данных средствами HTML5 плавно работает на мобильных устройствах.
- *Квалифицированные кадры* — специалистов по Tableau не так много, а навыки веб-разработки есть у множества людей. В ходе планирования проекта важно учесть, сможете ли вы подобрать для него персонал.
- *Скорость получения результатов* — полный IT-цикл в вашей компании может занимать слишком много времени, а вы хотите как можно быстрее донести результаты своего анализа до пользователей. После того как ваш интерфейс заработает и начнет реально использоваться, IT-отдел может сколь угодно долго заниматься выводом продукта на промышленный уровень.
- *Построение прототипа* — чем лучше вы сможете объяснить представителям IT-отдела, для чего нужен продукт и какими возможностями он должен обладать, тем проще им будет построить или купить приложение, которое делает то, что вам нужно.
- *Широкие возможности настройки* — программные продукты «с именем» отлично справляются со своим делом, но такие приложения никогда не удастся настроить в такой степени, как приложения, написанные вами.

А какие причины могут препятствовать такому решению?

- *Политика компании* — самая серьезная причина: это запрещено. В больших компаниях существуют правила, которые разрешают использовать только определенные инструменты, чтобы IT-отдел мог обеспечить поддержку.
- *В вашей организации есть опытная группа составителей отчетов* – вы отнимаете у них работу, они вряд ли этому обрадуются.

- *Ваш инструмент обладает достаточными возможностями настройки* — некоторые серьезные платформы представляют собой интерфейс для браузера, работающий на базе JavaScript. Tableau, BusinessObjects Webi, SAS Visual Analytics — все эти продукты предоставляют интерфейсы HTML; со временем широта возможностей их настройки может возрасти.

Сердца пользователей завоевывает «фасад» приложения — то, что непосредственно видит пользователь. Вся напряженная работа по подготовке данных и хитроумные аналитические методы чего-то стоят только в том случае, если вам удастся донести их смысл до пользователей. И теперь вы стоите на правильном пути к достижению этой цели! Пожалуй, на этой оптимистичной ноте мы завершим эту главу.

9.5. Итоги

- Эта глава посвящена последней части процесса data science. Мы постарались построить приложение data science, предоставляющее в распоряжение пользователя информационную панель с интерактивными возможностями. После прохождения всех этапов процесса data science остается вывести чистые данные, часто специально сжатые или обладающие высокой информационной плотностью. Это позволяет получать нужную информацию с запросом меньших объемов данных.
- В нашем примере предполагалось, что данные запасов лекарств прошли тщательную очистку и подготовку. И так должно быть всегда к тому моменту, когда информация достигает конечного пользователя.
- Информационные панели на базе JavaScript идеально подходят для быстрого предоставления доступа к результатам data science, потому что от пользователя требуется только наличие браузера. Также существуют другие альтернативы, такие как Qlik (глава 5).
- Crossfilter — лишь одна из многих библиотек MapReduceдля JavaScript, но эта библиотека доказала свою надежность, а ее разработку ведет Square — компания, специализирующаяся на платежных операциях. Применение MapReduce эффективно даже на одном узле и в браузере; оно повышает скорость вычислений.
- dc.js — библиотека построения диаграмм на базе d3.js and Crossfilter, позволяющая быстро создавать информационные панели в браузере.
- Мы исследовали набор данных больничной аптеки и построили интерактивную панель для фармацевта. Главное достоинство информационной панели — возможность «самообслуживания»: чтобы получить желанную информацию, пользователю не потребуется прибегать к услугам специалиста по составлению отчетов или data science.
- Также существуют другие средства визуализации данных. Не жалейте времени на знакомство с ними и найдите тот вариант, который лучше всего соответствует вашим потребностям.

- Самостоятельное построение отчетов вместо использования (часто более дорогих) «фирменных» инструментов обычно объясняется несколькими причинами:
  - *Отсутствие бюджета* — начинающие фирмы не могут позволить себе покупку любых существующих продуктов.
  - *Высокая доступность* — браузер есть у любого пользователя.
  - *Квалифицированные кадры* — (относительно) многочисленность разработчиков JavaScript.
  - *Скорость получения результатов* — цикл разработки полноценного продукта может занимать много времени.
  - *Построение прототипа* — приложение-прототип может предоставить необходимую информацию, после чего у IT будет время заняться построением полной версии.
  - *Широкие возможности настройки* — иногда приложение должно работать точно так, как вам представляется.

Конечно, существуют доводы и против разработки собственных приложений:

- *Политика компании* — избыток приложений создает проблемы с сопровождением, поэтому компания может ограничить локальную разработку.
- *Опытная группа составителей отчетов* — если в компании уже есть специалисты соответствующего профиля, зачем вам заниматься этим?
- *Достаточные возможности настройки* — не всем нужны визуальные «навороты»; порой хватает базовых возможностей.

Поздравляем! Книга подошла к концу, а ваша карьера специалиста data science только начинается. Хочется надеяться, что во время чтения и проработки примеров вам не было скучно. А теперь, когда вы получили общее представление о мире data science, пришло время выбирать собственный путь. История продолжается, и мы желаем вам стать самым великим и знаменитым экспертом в области data science. Возможно, когда-нибудь мы еще встретимся. ;)

Приложение А
Настройка Elasticsearch

В этом приложении рассматривается процедура установки и настройки базы данных Elasticsearch, использованной в главах 6 и 7, с инструкциями как для Linux, так и для Windows. Если у вас возникнут проблемы или вы захотите получить больше информации об Elasticsearch, по адресу *https://www.elastic.co/guide/en/elasticsearch/reference/1.4/setup.html* доступна качественная документация.

ПРИМЕЧАНИЕ

Elasticsearch зависит от Java, поэтому в приложении будет рассмотрена тема установки Java.

A.1. Установка в Linux

Сначала проверьте, установлена ли на вашей машине поддержка Java.

1. Версию Java можно проверить в консольном окне командой java –version. Если поддержка Java установлена, то на экране появится сообщение наподобие показанного на рис. А.1. Для запуска версии Elasticsearch, использованной в книге (1.4), потребуется версия не ниже Java 7. Примечание: на момент публикации книги система Elasticsearch перешла на версию 2, и хотя код может слегка измениться, базовые принципы остаются неизменными.

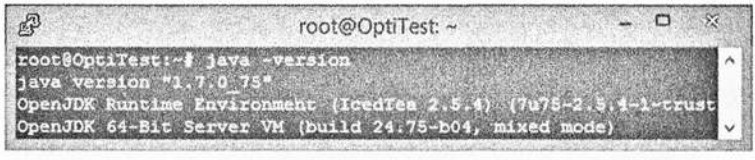


Рис. А.1. Проверка версии Java в Linux. Elasticsearch требует Java 7 и выше

2. Если поддержка Java на вашем компьютере не установлена (или установлена слишком старая версия), Elasticsearch рекомендует версию Java от Oracle. Для установки используется следующая последовательность консольных команд:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get install oracle-java7-installer
```

Теперь можно установить Elasticsearch:

1. Добавьте репозиторий Elasticsearch 1.4 (последний на момент написания книги) в список репозиториев, после чего установите его следующими командами:

```
sudo add-apt-repository "deb http://packages.Elasticsearch.org/
➥                        Elasticsearch/1.4/debian stable main"
sudo apt-get update && sudo apt-get install Elasticsearch
```

2. Чтобы система Elasticsearch запускалась при перезагрузке системы, выполните следующую команду:

```
sudo update-rc.d Elasticsearch defaults 95 10
```

3. Запустите Elasticsearch (рис. А.2).

```
sudo /etc/init.d/Elasticsearch start
```

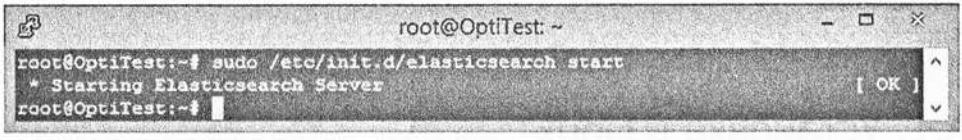


Рис. А.2. Запуск Elasticsearch в Linux

Если ваш локальный компьютер работает под управлением Linux, откройте браузер и введите адрес *localhost:9200* (9200 — порт по умолчанию для Elasticsearch API), как показано на рис. А.3.

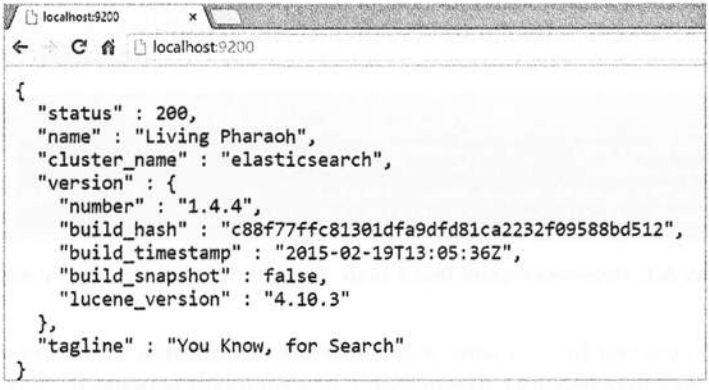


Рис. А.3. Начальный экран Elasticsearch на локальном хосте

Вас приветствует заставка Elasticsearch. Обратите внимание: у базы данных даже есть имя. Оно выбирается из набора персонажей Marvel и изменяется при каждой перезагрузке базы данных. В условиях реальной эксплуатации непостоянное и неуникальное имя может создать проблемы. Запущенный вами экземпляр становится одним узлом того, что может стать частью огромного распределенного кластера. Если все такие узлы будут менять имена при перезагрузке, отслеживать их в журнале при возникновении проблем будет почти невозможно. Elasticsearch гордится тем, что эта система является распределенной по своей природе, и для начала работы достаточно минимальных настроек. И хотя все это правда, такие аспекты, как случайное имя, доказывают, что при развертывании реальной многоузловой конфигурации необходимо дважды подумать о некоторых настройках по умолчанию. К счастью, в Elasticsearch есть хорошая документация почти по всем сторонам, включая развертывание (*http://www.Elasticsearch.org/guide/en/Elasticsearch/guide/current/deploy.html*). Тема развертывания Elasticsearch в многоузловой конфигурации здесь не рассматривается, но о ней следует помнить.

А.2. Установка в Windows

В системе Windows для работы Elasticsearch также требуется Java не ниже версии 7 (JRE и JDK), а переменная `JAVA_HOME` должна содержать ссылку на папку Java.

1. Загрузите установочные пакеты Java для Windows по адресу *http://www.oracle.com/tech-network/java/javase/downloads/index.html* и запустите их.
2. После установки убедитесь в том, что переменная окружения Windows `JAVA_HOME` указывает на каталог установки Java Development Kit. Чтобы просмотреть список переменных среды, откройте раздел System Control Panel Advanced System Settings (рис. А.4).

Если вы попытаетесь установить Elasticsearch без соответствующей версии Java, произойдет ошибка (рис. А.5).

УСТАНОВКА НА PC С ОГРАНИЧЕННЫМИ ПРИВИЛЕГИЯМИ

Иногда вы хотите опробовать программный продукт, но у вас нет прав для установки собственных программ. В таком случае не огорчайтесь: в вашем распоряжении переносимые версии JDK. Если вы найдете одну из таких версий, вы можете временно присвоить переменной JAVA_HOME путь к переносимой версии JDK и запустить Elasticsearch. Вам даже не обязательно устанавливать Elasticsearch, если вы только проверяете работу системы (рис. А.6).

Рис. А.4. Переменной JAVA_HOME присваивается имя папки установки Java

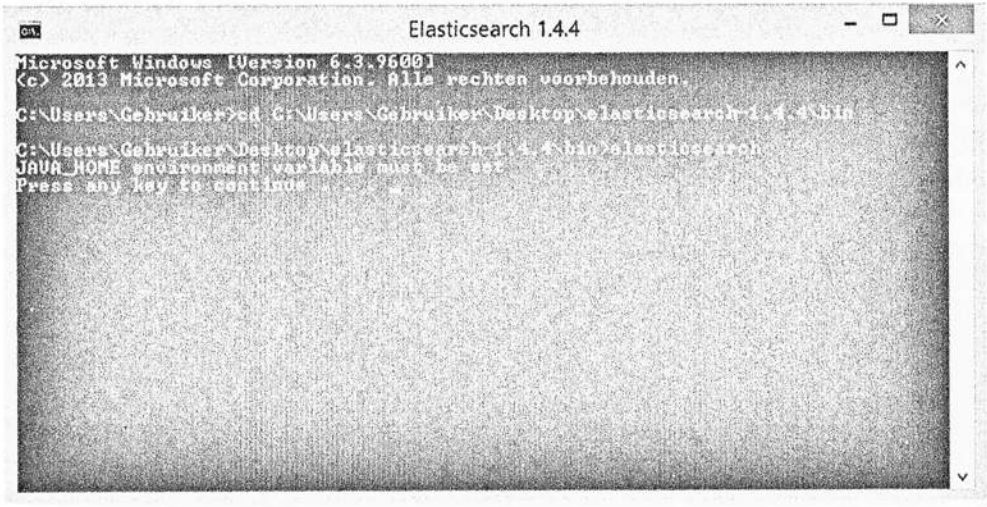


Рис. А.5. Если переменной JAVA_HOME присвоено неправильное значение, то попытка установки Elasticsearch завершается ошибкой

Рис. А.6. Запуск Elasticsearch без установки; рекомендуется только для тестирования на компьютерах, на которых вы имеете ограниченные привилегии

Когда поддержка Java будет установлена, можно переходить к установке Elasticsearch.

1. Вручную загрузите пакет Elasticsearch по адресу *http://www.Elasticsearch.org/download/*. Распакуйте его в любую папку на своем компьютере. Эта папка станет вашей автономной базой данных. Если у вас имеется SSD-диск, возможно, базу данных стоит разместить на нем, потому что это значительно повысит скорость Elasticsearch.
2. Если окно командной строки Windows уже открыто, не используйте его для установки; откройте новое окно. Переменные среды в открытом окне уже не актуальны. Перейдите в папку /bin в папке Elasticsearch и установите Elasticsearch командой service install (рис. А.7).

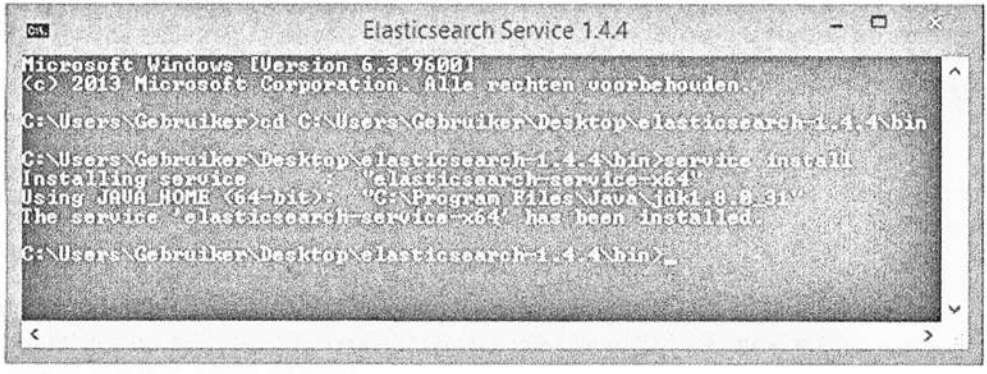


Рис. А.7. Установка Elasticsearch в 64-разрядной версии Windows

3. База данных готова к запуску. Используйте команду service start (рис. А.8).

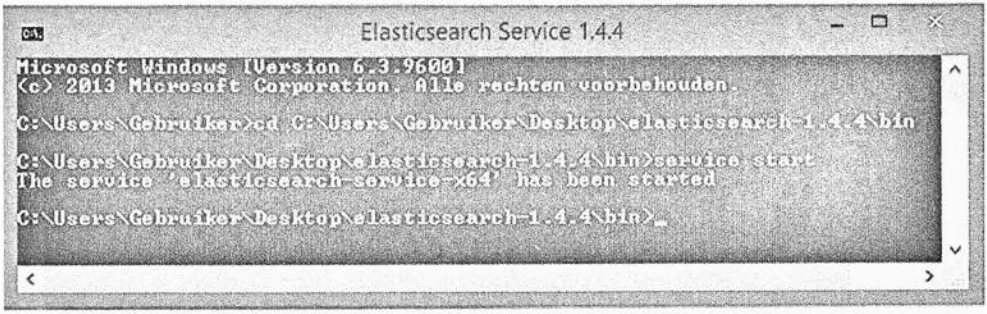


Рис. А.8. Elasticsearch запускает узел в системе Windows

Если вы захотите остановить сервер, введите команду service stop. Откройте браузер и введите адрес *localhost:9200* в адресной строке. Если появится начальный экран Elasticsearch (рис. А.9), значит, установка Elasticsearch прошла успешно.

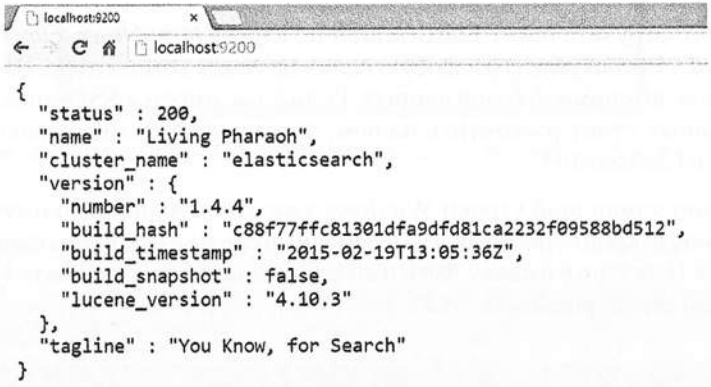


Рис. А.9. Начальный экран Elasticsearch на локальном хосте

Приложение Б
Установка Neo4j

В этом приложении рассматривается установка и настройка версии Community Edition базы данных Neo4j, использованной в главе 7. В приложении приводятся инструкции по установке как для Linux, так и для Windows.

Б.1. Установка в Linux

Чтобы установить Neo4j для Linux в версии Community Edition, воспользуйтесь режимом командной строки так, как описано здесь: *http://debian.neo4j.org/?_ga=1.84149595.332593114.1442594242*.

Neo Technology предоставляет репозиторий Debian, упрощающий установку Neo4j. Он включает три репозитория:

- *Stable* – все версии Neo4j, кроме перечисленных ниже. Выбирайте этот репозиторий по умолчанию.
- *Testing* – предварительные версии (контрольные точки и кандидаты на выпуск.)
- *Oldstable* – этот репозиторий, в настоящее время активно не используемый, содержит обновления для старых версий. Если вы не сможете найти нужную версию в Stable, поищите здесь.

Чтобы использовать новые пакеты Stable, выполните следующие команды с правами привилегированного пользователя (обратите внимание на использование sudo):

```
sudo -s
wget -O - https://debian.neo4j.org/neotechnology.gpg.key| apt-key add - #
Импортирование ключа подписи
echo 'deb http://debian.neo4j.org/repo stable/' > /etc/apt/sources.list.d/
neo4j.list # Создание файла Apt sources.list
aptitude update -y # Получение информации о файлах в репозитории
aptitude install neo4j -y # Установка Neo4j в версии Community Edition
```

Вместо папки Stable можно использовать папку Testing, если вы предпочитаете более новую (но официально не поддерживаемую) сборку Neo4j. Если же вместо Community Edition вы предпочитаете другую версию, выполните команду:

```
apt-get install neo4j-advanced
```

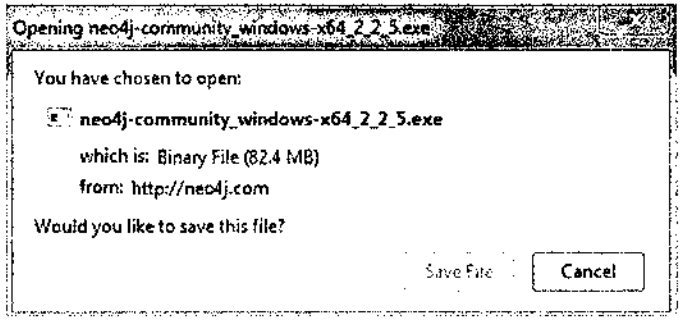
или

```
apt-get install neo4j-enterprise
```

Б.2. Установка в Windows

Чтобы установить Neo4j в версии Community Edition в системе Windows, выполните следующие действия:

1. Откройте страницу *http://neo4j.com/download/* и загрузите версию Community Edition. На экране появляется следующее диалоговое окно:

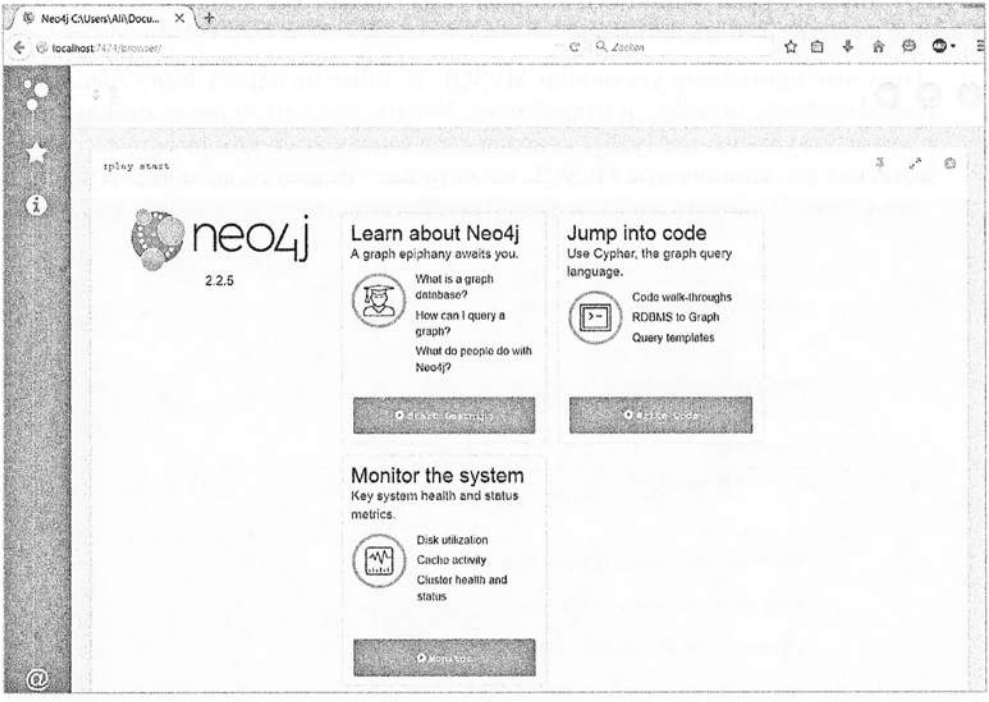


2. Сохраните файл и запустите его.
3. После завершения установки открывается другое окно, в котором вы можете выбрать расположение базы данных по умолчанию или ввести другую папку.
4. После того как решение будет принято, нажмите кнопку Start.

   Через несколько секунд база данных будет готова к работе. Чтобы остановить сервер, нажмите кнопку Stop.

5. Откройте браузер и введите адрес *localhost:7474* в адресной строке.
6. Когда вам будет предложено ввести учетные данные, введите имя пользователя и пароль «neo4j», после чего нажмите **Connect**.

В следующем окне вы можете назначить собственный пароль.



Теперь вы можете вводить свои запросы Cypher, а также получать информацию об узлах, отношениях и результатах.

Приложение B
Установка сервера MySQL

В этом приложении рассматривается процедура установки и настройки базы данных MySQL. В приложении приводятся инструкции по установке как для Linux, так и для Windows.

B.1. Установка в Windows

Самый удобный (и рекомендуемый) способ — загрузка программы установки MySQL Installer (для Windows), которая сама создаст все компоненты MySQL в вашей системе. Это делается так:

1. Загрузите программу установки MySQL Installer по адресу *http://dev.mysql.com/downloads/installer/* и откройте ее. Учтите, что в отличие от стандартной программы установки MySQL сокращенная версия «web-group» автоматически включает все компоненты MySQL, но загружает только те, которые вы решили установить. Выберите любой вариант установки на свое усмотрение (рис. B.1).

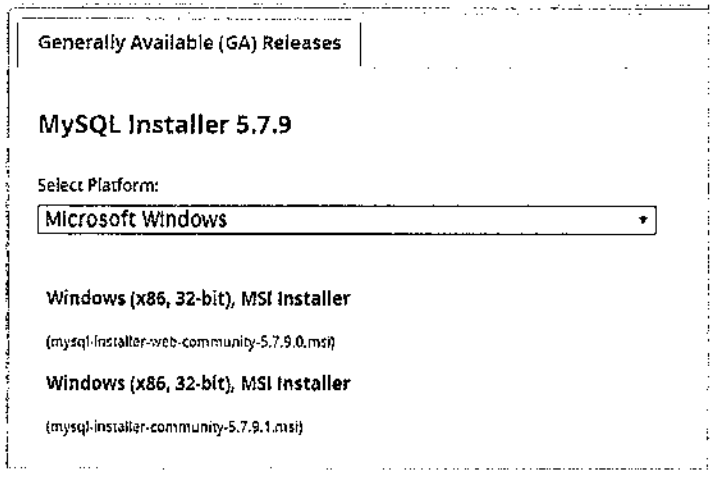


Рис. B.1. Варианты программ установки MySQL для Windows

2. Выберите тип установки (Setup Type). Вариант Developer Default устанавливает сервер MySQL и другие компоненты MySQL, а также такие вспомогательные функции, как MySQL Workbench. Также можно выбрать пользовательскую установку (Custom Setup), если вы предпочитаете выбрать компоненты MySQL, устанавливаемые в вашей системе. При желании в одной системе можно установить несколько разных версий MySQL. Диспетчер MySQL Notifier помогает наблюдать за работающими экземплярами, останавливать и перезапускать их. Его также можно добавить позднее в программе установки MySQL.

3. Выполните инструкции мастера установки MySQL, чтобы завершить процесс установки. В основном вам достаточно подтверждать предлагаемые решения. В качестве типа конфигурации сервера выберите машину разработки. Назначьте административный пароль MySQL и не забудьте его, потому что он вам понадобится позднее. MySQL может запускаться как служба Windows; в этом случае вам не придется запускать систему вручную.

4. Установка завершается. Если вы выбрали полную установку, по умолчанию сервер MySQL, программы MySQL Workbench и MySQL Notifier будут автоматически запускаться при запуске компьютера. Программа установки MySQL может использоваться для обновления или изменения настроек установленных компонентов.

5. Экземпляр сервера работает в системе, а вы можете подключиться к нему из MySQL Workbench (рис. B.2).

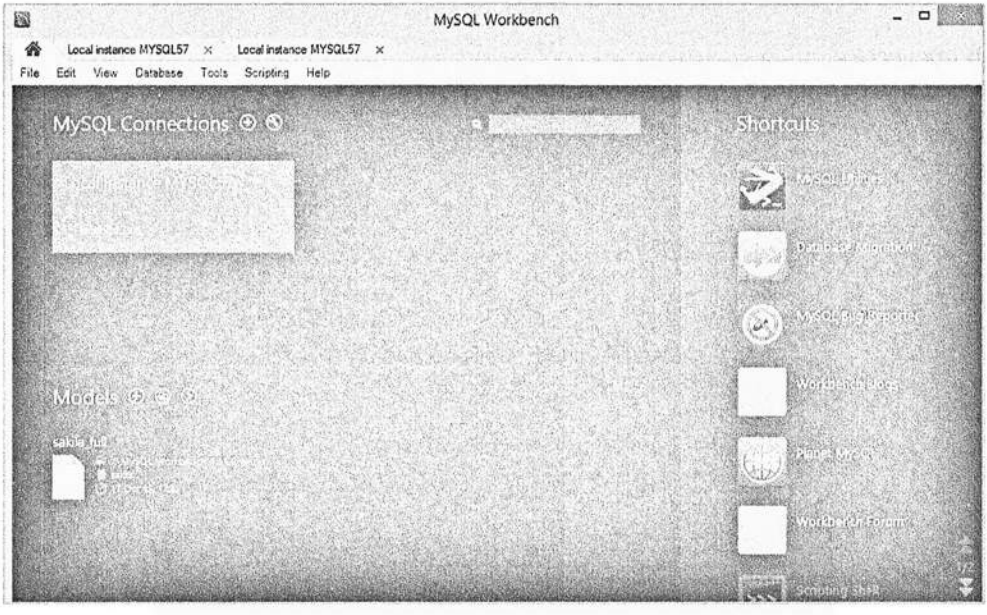


Рис. B.2. Интерфейс MySQL Workbench

B.2. Установка в Linux

Официальные инструкции по установке MySQL в системе Linux доступны по адресу *https://dev.mysql.com/doc/refman/5.7/en/linux-installation.html*.

Однако для некоторых дистрибутивов Linux написаны специальные руководства. Например, инструкции по установке MySQL в Ubuntu 14.04 можно найти по адресу *https://www.linode.com/docs/databases/mysql/how-to-install-mysql-on-ubuntu-14-04*. Следующие инструкции базируются на официальных инструкциях.

1. Проверьте имя хоста:

```
hostname
hostname -f
```

2. Первая команда выводит короткое имя хоста, а вторая должна вывести полное доменное имя (FQDN, Fully Qualified Domain Name).
3. Обновите свою систему:

```
sudo apt-get update
sudo apt-get upgrade
```

4. Установите MySQL:

```
Sudo apt-get install msql-server
```

В процессе установки выводится сообщение о необходимости выбрать пароль для административного пользователя MySQL (рис. B.3).

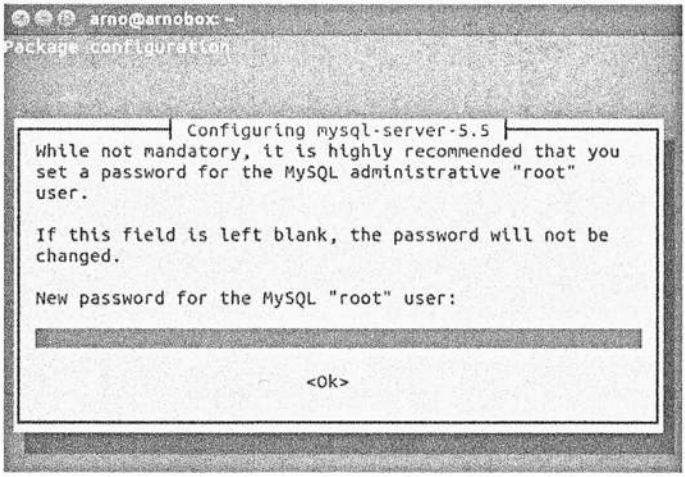


Рис. B.3. Выбор пароля для административного пользователя MySQL

По умолчанию MySQL связывается с локальным хостом (127.0.0.1).

5. Откройте сеанс работы с MySQL:

```
mysql -u root -p
```

6. Введите выбранный вами пароль. На экране должна появиться консоль MySQL, изображенная на рис. B.4.

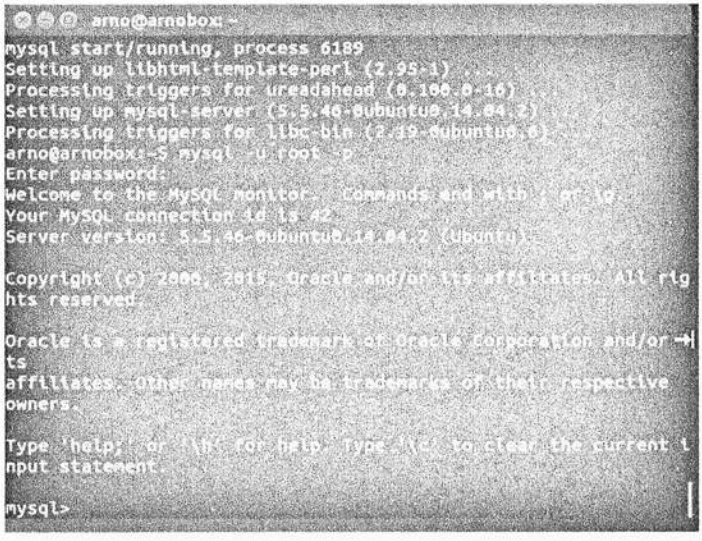


Рис. B.4. Консоль MySQL в Linux

7. Создайте схему, которая будет использоваться в учебном примере главы 4:

```
Create database test;
```

Приложение Г

Установка Anaconda в виртуальной среде

Anaconda — программный пакет Python, особенно полезный в области data science. Вариант установки по умолчанию содержит многие инструменты, которые могут понадобиться специалисту data science. В этой книге используется 32-разрядная версия, потому что она более стабильно работает со многими пакетами Python (особенно относящимися к SQL).

Хотя мы рекомендуем использовать Anaconda, это ни в коей мере не обязательно. В этом приложении рассматривается процедура установки и настройки Anaconda. В приложении приводятся инструкции по установке как для Linux, так и для Windows, а за ними следуют инструкции по настройке среды. Если вы разбираетесь в использовании пакетов Python, ничто не мешает вам сделать то же самое по-своему. Например, вы можете использовать библиотеки virtualenv и pip.

Г.1. Установка в Linux

Чтобы установить Anaconda в Linux, выполните следующие действия:

1. Откройте страницу *https://www.continuum.io/downloads* и загрузите программу установки для 32-разрядной версии Anaconda на базе Python 2.7 для Linux.
2. Когда загрузка будет завершена, установите Anaconda следующей командой:

```
bash Anaconda2-2.4.0-Linux-x86_64.sh
```

3. Команда conda должна работать в приглашении командной строки Linux. Anaconda спросит, нужно ли включить эту возможность; выберите ответ Yes.

Г.2. Установка в Windows

Чтобы установить Anaconda в Windows, выполните следующие действия:

1. Откройте страницу *https://www.continuum.io/downloads* и загрузите программу установки для 32-разрядной версии Anaconda на базе Python 2.7 для Windows.
2. Запустите программу установки.

Г.3. Настройка среды

После того как установка будет завершена, следует заняться настройкой среды. Интересная схема соответствия команд conda и pip находится по адресу *http://conda.pydata.org/docs/_downloads/conda-pip-virtualenv-translator.html*.

1. Введите следующую команду в командной строке вашей операционной системы. Замените *имя_среды* фактическим именем, которое вы хотите связать со своей средой.

   ```
   conda create -n имя_среды anaconda
   ```

2. Подтвердите продолжение настройки вводом «y» в конце списка (рис. Г.1). Через некоторое время все будет готово к работе.

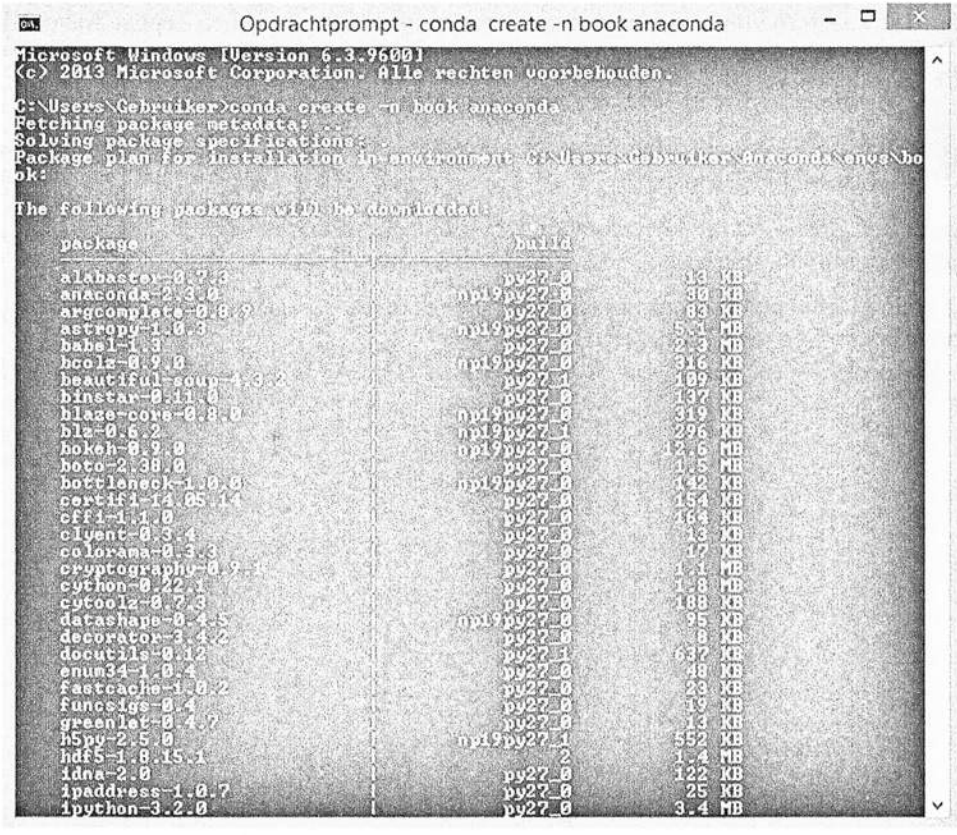


Рис. Г.1. Создание виртуальной среды Anaconda в командной строке Windows

Anaconda создает среду в папке по умолчанию, но если вы захотите изменить ее местонахождение, для этого существуют специальные параметры.

3. После того как среда будет создана, ее можно активизировать в командной строке:
   - В Windows введите команду `activate` *имя_среды*.
   - В Linux введите команду `source activate` *имя_среды*.

Также можно указать среду в среде разработки (IDE) Python.

4. Если среда активизируется в командной строке, вы можете запустить Jupiter (IPython) IDE следующей командой:

```
Ipython notebook
```

   Jupiter (формальное название IPython) — интерактивный интерфейс разработки Python, работающий в браузере.

5. Для каждого упомянутого в книге пакета, не установленного в среде Anaconda по умолчанию:
   - Активизируйте среду в командной строке.
   - Введите команду conda install *имя_библиотеки* или pip install *имя_библиотеки* в командной строке.

За дополнительной информацией об установке pip обращайтесь по адресу *http://python-packaging-user-guide.readthedocs.org/en/latest/installing/*.

За дополнительной информацией об установке conda обращайтесь по адресу *http://conda.pydata.org/docs/intro.html*.

Дэви Силен, Арно Мейсман, Мохамед Али

Основы Data Science и Big Data. Python и наука о данных

Перевел с английского Е. Матвеев

Завсдующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*Н. Римицан*
Литературный редактор	*И. Карпова*
Художник	*С. Заматевская*
Корректоры	*Н. Викторова, И. Тимофеева*
Верстка	*Л. Егорова*

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12.000 — Книги печатные профессиональные, технические и научные.

Подписано в печать 19.12.16. Формат 70×100/16. Бумага офсетная. Усл. п. л. 27,090. Тираж 1200. Заказ 9247

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

САЛД

САНКТ-ПЕТЕРБУРГСКАЯ
АНТИВИРУСНАЯ
ЛАБОРАТОРИЯ
ДАНИЛОВА

www.SALD.ru
8 (812) 336-3739

Антивирусные
программные продукты

Data Science — это совокупность понятий и методов, позволяющих придать смысл и понятный вид огромным объемам данных.

Каждая из глав этой книги посвящена одному из самых интересных аспектов анализа и обработки данных. Вы начнете с теоретических основ, затем перейдете к алгоритмам машинного обучения, работе с огромными массивами данных, NoSQL, потоковым данным, глубокому анализу текстов и визуализации информации. В многочисленных практических примерах использованы сценарии Python.

Обработка и анализ данных — одна из самых горячих областей IT, где постоянно требуются разработчики, которым по плечу проекты любого уровня, от социальных сетей до обучаемых систем. Надеемся, книга станет отправной точкой для вашего путешествия в увлекательный мир Data Science.

ISBN 978-5-496-02517-1

ПИТЕР®

Заказ книг:
тел.: (812) 703-73-74
books@piter.com

WWW.PITER.COM
каталог книг
и интернет-магазин

vk.com/piter_publishing
instagram.com/piterbooks
facebook.com/idpiter