ГЛАВА 1. ВВЕДЕНИЕ

Машинное обучение заключается в извлечении знаний из данных. Это научная область, находящаяся на пересечении статистики, искусственного интеллекта и компьютерных наук и также известная как прогнозная аналитика или статистическое обучение. В последние годы применение методов машинного обучения в повседневной жизни стало обыденным явлением. Многие современные веб-сайты и устройства используют алгоритмы машинного обучения, начиная с автоматических рекомендаций по просмотру фильмов, заказа еды или покупки продуктов, и заканчивая персонализированными онлайн-радиотрансляциями и распознаванием друзей на фотографиях. Когда вы видите сложный сайт типа Facebook, Amazon или Netflix, то весьма вероятно, что каждый раздел сайта содержит несколько моделей машинного обучения.

Выйдя за пределы коммерческих приложений, машинное обучение уже оказало огромное влияние на научные исследования, управляемые данными. Инструменты, представленные в этой книге, использовались для решения различных научных задач (исследование звезд, поиск далеких планет, открытие новых частиц, анализ последовательностей ДНК, а также разработка персонализированных методов лечения рака).

Для извлечения прибыли с помощью машинного обучения совсем необязательно, чтобы ваши задачи были столь же крупномасштабными или меняющими мир, как представленные примеры. В этой главе мы объясним, почему машинное обучение стало таким популярным, и обсудим, какие задачи могут быть решены с помощью него. Затем мы покажем вам, как построить свою первую модель машинного обучения, попутно знакомя вас с важными принципами машинного обучения.

Зачем нужно использовать машинное обучение?

На заре появления «интеллектуальных» приложений многие системы использовали жесткие правила «if» и «else» для обработки данных или корректировки информации, введенной пользователем. Вспомните о спам-фильтре, чья работа состоит в том, чтобы переместить соответствующие входящие сообщения электронной почты в папку «Спам». Вы можете составить черный список слов, которые будут идентифицировать письмо как спам. Это пример использования системы экспертных правил для разработки «интеллектуального» приложения. Разработка правил принятия решений в ручном режиме допустимо в некоторых задачах, особенно в тех, где люди четко понимают процесс

моделирования. Однако, использование жестких решающих правил имеет два основных недостатка:

- Логика, необходимая для принятия решения, относится исключительно к одной конкретной области и задачи. Даже несущественное изменение задачи может повлечь за собой переписывание всей системы.
- Разработка правил требует глубокого понимания процесса принятия решения.

Один из примеров, где этот жесткий подход потерпит неудачу – это распознавание лиц на изображениях. На сегодняшний день каждый смартфон может распознать лицо на изображении. Тем не менее, распознавание лиц была нерешенной проблемой, по крайней мере, до 2001 года. Основная проблема заключается в том, что способ, с помощью которого компьютер «воспринимает» пиксели, формирующие изображение на компьютере, очень сильно отличается от человеческого восприятия лица. Эта разница в принципе не позволяет человеку сформулировать подходящий набор правил, описывающих лицо с точки зрения цифрового изображения.

Однако, благодаря машинному обучению, простого предъявления большого количества изображений с лицами будет достаточно для того, чтобы алгоритм определил, какие признаки необходимы для идентификации лица.

Задачи, которые можно решить с помощью машинного обучения

Наиболее успешные алгоритмы машинного обучения – это те, которые автоматизируют процессы принятия решений путем обобщения известных примеров. В этих методах, известных как *обучение с учителем* или *контролируемое обучение* (*supervised learning*), пользователь предоставляет алгоритму пары объект-ответ, а алгоритм находит способ получения ответа по объекту. В частности, алгоритм способен выдать ответ для объекта, которого он никогда не видел раньше, без какой-либо помощи человека. Если вернуться к примеру классификации спама с использованием машинного обучения, пользователь предъявляет алгоритму большое количество писем (объекты) вместе с информацией о том, является ли письмо спамом или нет (ответы). Для нового электронного письма алгоритм вычислит вероятность, с которой это письмо можно отнести к спаму.

Алгоритмы машинного обучения, которые учатся на парах объект-ответ, называются алгоритмами обучения с учителем, так как «учитель» показывает алгоритму ответ в каждом наблюдении, по которому

происходит обучение. Несмотря на то, что создание набора с объектами и ответами – это часто трудоемкий процесс, осуществляемый вручную, алгоритмы обучения с учителем интерпретируемы и качество их работы легко измерить. Если вашу задачу можно сформулировать в виде задачи обучения с учителем, и вы можете создать набор данных, который включает в себя ответы, вероятно, машинное обучение решит вашу проблему.

Примеры задач машинного обучения с учителем:

Определение почтового индекса по рукописным цифрам на конверте
Здесь объектом будет сканированное изображение почерка, а ответ – фактические цифры почтового индекса. Чтобы создать набор данных для построения модели машинного обучения, вам нужно собрать большое количество конвертов. Затем вы можете самостоятельно прочитать почтовые индексы и сохранить цифры в виде ответов.

Определение доброкачественности опухоли на основе медицинских изображений
Здесь объектом будет изображение, а ответом – диагноз, является ли опухоль доброкачественной или нет. Чтобы создать набор данных для построения модели, вам нужна база медицинских изображений. Кроме того, необходимо мнение эксперта, поэтому врач должен просмотреть все изображения и решить, какие опухоли являются доброкачественными, а какие – нет. Помимо анализа изображения может понадобиться дополнительная диагностика для определения доброкачественности опухоли.

Обнаружение мошеннической деятельности в сделках по кредитным картам
Здесь объект – запись о транзакции по кредитной карте, а ответ - информация о том, является ли транзакция мошеннической или нет. Предположим, вы – учреждение, выдающее кредитные карты, сбор данных подразумевает сохранение всех транзакций и запись сообщений клиентов о мошеннических транзакциях.

Приведя эти примеры, интересно отметить что, хотя объекты и ответы выглядят достаточно просто, процесс сбора данных для этих трех задач существенно отличается. Несмотря на то что чтение конвертов является трудоемким занятием, этот процесс прост и дешев. Получение медицинских изображений и проведение диагностики требует не только дорогостоящего оборудования, но и редких, высокооплачиваемых экспертных знаний, не говоря уже об этических проблемах и вопросах конфиденциальности. В примере обнаружения мошенничества с кредитными картами, сбор данных осуществляется намного проще. Ваши

клиенты сами предоставят вам ответы, сообщая о мошенничестве. Все, что вам нужно сделать для получения объектов и ответов, связанных с мошеннической активностью, – это подождать.

Алгоритмы обучения без учителя или неконтролируемого обучения (unsupervised algorithms) – это еще один вид алгоритмов, который мы рассмотрим в этой книге. В алгоритмах обучения без учителя известны только объекты, а ответов нет. Хотя есть много успешных сфер применения этих методов, их, как правило, труднее интерпретировать и оценить.

Примеры задач машинного обучения без учителя:

Определение тем в наборе постов
Если у вас есть большая коллекция текстовых данных, вы можете агрегировать их и найти распространенные темы. У вас нет предварительной информации о том, какие темы там затрагиваются и сколько их. Таким образом, нет никаких известных ответов.

Сегментирование клиентов на группы с похожими предпочтениями
Имея набор записей о клиентах, вы можете определить группы клиентов со схожими предпочтениями. Для торгового сайта такими группами могут быть «родители», «книгочеи» или «геймеры». Поскольку вы не знаете заранее о существовании этих групп и их количестве, у вас нет ответов.

Обнаружение паттернов аномального поведения на веб-сайте
Чтобы выявить злоупотребления или ошибки, часто бывает полезно найти паттерны поведения, которые отличаются от нормы. Паттерны аномального поведения могут быть разными, и, возможно, у вас не будет зарегистрированных случаев аномального поведения. Поскольку в этом примере вы наблюдаете лишь трафик, и вы не знаете, что представляет собой нормальное и ненормальное поведение, речь идет о задаче обучения без учителя.

Решая задачи машинного обучения с учителем и без, важно представить ваши входные данные в формате, понятном компьютеру. Часто данные представляют в виде таблицы. Каждая точка данных, которую вы хотите исследовать (каждое электронное письмо, каждый клиент, каждая транзакция) является строкой, а каждое свойство, которое описывает эту точку данных (скажем, возраст клиента, сумма или место совершения транзакции), является столбцом. Вы можете описать пользователей по возрасту, полу, дате создания учетной записи и частоте покупок в вашем интернет-магазине. Вы можете описать изображение опухоли с помощью градаций серого цвета для каждого пикселя или с помощью размера, формы и цвета опухоли.

В машинном обучении каждый объект или строка называются *примером* (*sample*) или *точкой данных* (*data point*), а столбцы-свойства, которые описывают эти примеры, называются *характеристиками* или *признаками* (*features*).

Позже в этой книге мы более детально остановимся на теме подготовки данных, которая называется *выделение признаков* (*feature extraction*) или *конструирование признаков* (*feature engineering*). Однако, вы должны иметь в виду, что ни один алгоритм машинного обучения не сможет сделать прогноз по данным, которые не содержат никакой полезной информации. Например, если единственный признак пациента – это его фамилия, алгоритм не сможет предсказать его пол. Этой информации просто нет в данных. Если добавить еще один признак – имя пациента, то дело уже будет обстоять лучше, поскольку часто, зная имя человека, можно судить о его поле.

Постановка задач и знакомство с данными

Вполне возможно, что самая важная часть процесса машинного обучения – это интерпретация данных, с которыми вы работаете, и применимость этих данных к задаче, которую вы хотите решить. Выбрать случайным образом алгоритм и скормить ему свои данные – неэффективное решение. Прежде чем приступить к построению модели, необходимо понять, что представляет собой ваш набор данных. Каждый алгоритм отличается с точки зрения типа обрабатываемых данных и вида решаемых задач. Создавая модель машинного обучения, вы должны ответить, или, по крайней мере, задуматься над следующими вопросами:

- На какой вопрос(ы) я пытаюсь ответить? Собранные данные могут ответить на этот вопрос?
- Как лучше всего сформулировать свой вопрос(ы) с точки зрения задач машинного обучения?
- У меня собрано достаточно данных, чтобы составить представление о задаче, которую я хочу решить?
- Какие признаки я извлек и помогут ли они мне получить правильные прогнозы?
- Как я буду измерять эффективность решения задачи?
- Как решение, полученное с помощью машинного обучения, будет взаимодействовать с другими компонентами моего исследования или бизнес-продукта?

В более широком контексте, алгоритмы и методы машинного обучения являются лишь этапом более крупного процесса, призванного решить конкретную задачу, и поэтому необходимо всегда держать схему

этого процесса в голове. Многие тратят массу времени, создавая сложные модели машинного обучения решения, затем узнавая, что решают задачу неправильно.

Углубляясь в технические аспекты машинного обучения (что мы и будем делать этой книге), легко упустить из виду конечные цели. Несмотря на то, что мы не будем подробно обсуждать вопросы, перечисленные здесь, мы все же рекомендуем вам вспомнить о них, когда вы начнете строить модели машинного обучения.

Почему нужно использовать Python?

Python стал общепринятым языком для многих сфер применения науки о данных (data science). Он сочетает в себе мощь языков программирования с простотой использования предметно-ориентированных скриптовых языков типа MATLAB или R. В Python есть библиотеки для загрузки данных, визуализации, статистических вычислений, обработки естественного языка, обработки изображений и многого другого. Этот обширный набор инструментов предлагает специалистам по работе с данными (data scientists) большой набор инструментов общего и специального назначения. Одним из основных преимуществ использования Python является возможность напрямую работать с программным кодом с помощью терминала или других инструментов типа Jupyter Notebook, который мы рассмотрим ниже. Машинное обучение и анализ данных – это в основном итерационные процессы, в которых данные задают ход анализа. Крайне важно для этих процессов иметь инструменты, которые позволяют оперативно и легко работать.

В качестве языка программирования общего назначения Python позволяет создавать сложные графические пользовательские интерфейсы (GUI) и веб-сервисы, а также легко интегрироваться в уже существующие системы.

scikit-learn

`scikit-learn` – проект с открытым исходным кодом, это означает, что его можно свободно использовать и распространять, и любой человек может легко получить исходный код, чтобы увидеть, что происходит «за кулисами». Проект `scikit-learn` постоянно развивается и совершенствуется, и у него очень активное сообщество пользователей. Он содержит ряд современных алгоритмов машинного обучения, а также полную документацию по каждому алгоритму. `scikit-learn` – очень популярный инструмент и самая известная питоновская библиотека для

машинного обучения. Она широко используется в промышленности и науке, а в интернете имеется богатый выбор обучающих материалов и примеров программного кода. `scikit-learn` прекрасно работает с рядом других научных инструментов Python, которые мы обсудим позже в этой главе.

По мере чтения книги мы рекомендуем также ознакомится с руководством пользователя по `scikit-learn` и документацией по API для получения дополнительной информации о многочисленных параметрах каждого алгоритма. Онлайн-документация является очень подробной, и эта книга познакомит вас со всеми необходимыми основами машинного обучения, чтобы вы научились детально разбираться в нем.

Установка scikit-learn

`scikit-learn` требует наличия еще двух пакетов Python – NumPy и SciPy. Для построения графиков и интерактивной работы необходимо также установить `matplotlib`, IPython и Jupyter Notebook. Мы рекомендуем использовать один из нижеперечисленных дистрибутивов Python, которые уже включают все необходимые пакеты:

Anaconda

Дистрибутив Python, предназначенный для крупномасштабной обработки данных, прогнозной аналитики и научных вычислений. Anaconda уже включает NumPy, SciPy, `matplotlib`, `pandas`, IPython, Jupyter Notebook и `scikit-learn`. Есть версии для Mac OS, Windows и Linux. Это очень удобное решение и это тот дистрибутив, который мы рекомендуем пользователям, у которых еще не установлены пакеты Python для научных вычислений. Кроме того, сейчас Anaconda включает в себя коммерческую библиотеку Intel MKL, которой можно пользоваться бесплатно. Использование MKL (это происходит автоматически при установке Anaconda) может дать значительный прирост скорости при выполнении различных алгоритмов в `scikit-learn`.

Enthought Canopy

Еще один дистрибутив Python для научных вычислений. Он уже содержит NumPy, SciPy, `matplotlib`, `pandas` и IPython, но бесплатная версия не включает `scikit-learn`. Если вы являетесь учебным заведением, вы можете запросить учебную лицензию и получить свободный доступ к платной версии Enthought Canopy. Enthought Canopy доступен для Python 2.7.x и работает на Mac OS, Windows и Linux.

Python (x, y)
Свободный дистрибутив Python для научных вычислений, специально предназначенный для Windows. Python (x, y) включает NumPy, SciPy, `matplotlib`, `pandas`, IPython и `scikit-learn`.

Если у вас уже стоит Python, вы можете использовать `pip` для установки всех этих пакетов:

```
$ pip install numpy scipy matplotlib ipython scikit-learn pandas
```

Основные библиотеки и инструменты

Понимание принципов работы и использования `scikit-learn` – важно, но есть несколько других библиотек, которые расширят ваш опыт. `scikit-learn` базируется на двух питоновских библиотеках для научных вычислений NumPy и SciPy. Помимо NumPy и SciPy мы будем использовать `pandas` и `matplotlib`. Кроме того, мы познакомимся с Jupyter Notebook, который представляет собой интерактивную среду программирования на основе браузера. Если коротко, то ниже приводится информация о перечисленных инструментах, которыми вы должны овладеть, чтобы получить максимальную отдачу от `scikit-learn`.[1]

Jupyter Notebook

Jupyter Notebook представляет собой интерактивную среду для запуска программного кода в браузере. Это отличный инструмент для разведочного анализа данных и широко используется специалистами по анализу данных. Несмотря на то что Jupyter Notebook поддерживает множество языков программирования, нам нужна лишь поддержка Python. Jupyter Notebook позволяет легко интегрировать программный код, текст и изображения, и вся эта книга была фактически написана в формате Jupyter Notebook. Все примеры программного кода, приведенные в этой книги, можно загрузить с GitHub.

NumPy

NumPy – это один из основных пакетов для научных вычислений в Python. Он содержит функциональные возможности для работы с многомерными массивами, высокоуровневыми математическими

[1] Если вы не знакомы с NumPy или `matplotlib`, мы рекомендуем прочитать первую главу SciPy Lectures Notes.

функциями (операции линейной алгебры, преобразование Фурье, генератор псевдослучайных чисел).

В `scikit-learn` массив NumPy – это основная структура данных. `scikit-learn` принимает данные в виде массивов NumPy. Любые данные, которые вы используете, должны быть преобразованы в массив NumPy. Базовый функционал NumPy – это класс `ndarray`, многомерный (n-мерный) массив. Все элементы массива должны быть одного и того же типа. Массив NumPy выглядит следующим образом:

```
In[2]:
import numpy as np

x = np.array([[1, 2, 3], [4, 5, 6]])
print("x:\n{}".format(x))

Out[2]:
x:
[[1 2 3]
 [4 5 6]]
```

Мы будем *очень много* использовать NumPy в этой книге, и будем называть объекты класса `ndarray` «массивами NumPy» или просто «массивами».

SciPy

SciPy – это набор функций для научных вычислений в Python. Помимо всего прочего он предлагает продвинутые процедуры линейной алгебры, математическую оптимизацию функций, обработку сигналов, специальные математические функции и статистические функции. `scikit-learn` использует набор функций SciPy для реализации своих алгоритмов. Для нас наиболее важной частью SciPy является пакет `scipy.sparse`: с помощью него мы получаем *разреженные матрицы* (*sparse matrices*), которые представляют собой еще один формат данных, который используется в `scikit-learn`. Разреженные матрицы используются всякий раз, когда нам нужно сохранить 2D массив, который содержит в основном нули:

```
In[3]:
from scipy import sparse

# Создаем 2D массив NumPy с единицами по главной диагонали и нулями в остальных ячейках
eye = np.eye(4)
print("массив NumPy:\n{}".format(eye))

Out[3]:
массив NumPy:
[[ 1. 0. 0. 0.]
 [ 0. 1. 0. 0.]
 [ 0. 0. 1. 0.]
 [ 0. 0. 0. 1.]]
```

```
In[4]:
# Преобразовываем массив NumPy в разреженную матрицу SciPy в формате CSR
# Сохраняем лишь ненулевые элементы
sparse_matrix = sparse.csr_matrix(eye)
print("\nразреженная матрица SciPy в формате CSR:\n{}".format(sparse_matrix))

Out[4]:
разреженная матрица SciPy в формате CSR:
(0, 0) 1.0
(1, 1) 1.0
(2, 2) 1.0
(3, 3) 1.0
```

Обычно невозможно плотно записать разреженные данные (поскольку они не уместились бы в памяти), поэтому нам нужно непосредственно создать разреженные матрицы. Ниже приводится способ, которой позволяет создать такую же разреженную матрицу, что была приведена выше, но этот раз с использованием формата COO[2]:

```
In[5]:
data = np.ones(4)
row_indices = np.arange(4)
col_indices = np.arange(4)
eye_coo = sparse.coo_matrix((data, (row_indices, col_indices)))
print("формат COO:\n{}".format(eye_coo))

Out[5]:
формат COO:
(0, 0) 1.0
(1, 1) 1.0
(2, 2) 1.0
(3, 3) 1.0
```

Более подробную информацию о разреженных матрицах SciPy можно найти в SciPy Lecture Notes.

matplotlib

`matplotlib` – это основная библиотека для построения научных графиков в Python. Она включает функции для создания высококачественных визуализаций типа линейных диаграмм, гистограмм, диаграмм разброса и т.д. Визуализация данных и различных аспектов вашего анализа может дать вам важную информацию, и мы будем использовать `matplotlib` для всех наших визуализаций. При работе в Jupyter Notebook, вы можете вывести рисунок прямо в браузере с помощью встроенных команд `%matplotlib notebook` и `%matplotlib inline`. Мы рекомендуем использовать `%matplotlib notebook`, который предлагает интерактивное окружение (хотя при написании этой книги мы использовали `%matplotlib inline`). Например, нижеприведенный программный код строит график, изображенный на рис. 1.1:

[2] COO (coordinate format) – координатный формат хранения разреженных матриц: хранятся только ненулевые элементы матрицы и их координаты (номера строк и столбцов). – *Прим. пер.*

```
In[6]:
%matplotlib inline
import matplotlib.pyplot as plt

# Генерируем последовательность чисел от -10 до 10 с 100 шагами
x = np.linspace(-10, 10, 100)
# Создаем второй массив с помощью синуса
y = np.sin(x)
# Функция создает линейный график на основе двух массивов
plt.plot(x, y, marker="x")
```

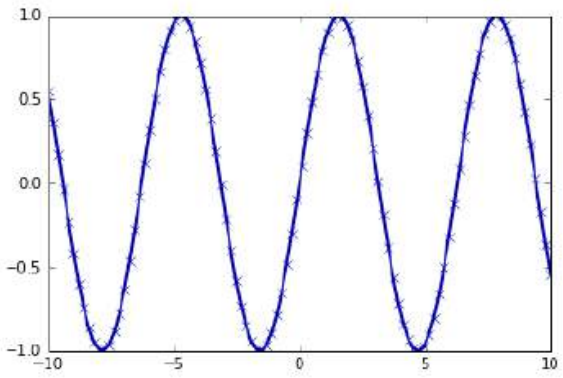

Рис. 1.1 Простой линейный график синусоидальной функции с использованием matplotlib

pandas

`pandas` – библиотека Python для обработки и анализа данных. Она построена на основе структуры данных, называемой `DataFrame` и смоделированной по принципу датафреймов среды статистического программирования R. Проще говоря, `DataFrame` библиотеки `pandas` представляет собой таблицу, похожую на электронную таблицу Excel. Библиотека `pandas` предлагает большой спектр методов по работе с этой таблицей, в частности, она позволяет выполнять SQL-подобные запросы и присоединения таблиц. В отличие от NumPy, который требует, чтобы все записи в массиве были одного и того же типа, в `pandas` каждый столбец может иметь отдельный тип (например, целые числа, даты, числа с плавающей точкой и строки). Еще одним преимуществом библиотеки `pandas` является ее способность работать с различными форматами файлов и баз данных, например, с файлами SQL, Excel и CSV. Детальное рассмотрение функционала `pandas` выходит за рамки этой книги. Тем не менее, *Python for Data Analysis* Уэса МакКинни (O'Reilly, 2012) является замечательным руководством. Ниже приводится небольшой пример создания `DataFrame` с помощью словаря:

```
In[7]:
import pandas as pd

# создаем простой набор данных с характеристиками пользователей
data = {'Name': ["John", "Anna", "Peter", "Linda"],
        'Location' : ["New York", "Paris", "Berlin", "London"],
        'Age' : [24, 13, 53, 33]
       }

data_pandas = pd.DataFrame(data)
# IPython.display позволяет "красиво напечатать" датафреймы
# в Jupyter notebook
display(data_pandas)
```

Приведенный код генерирует следующий вывод:

	Age	Location	Name
0	24	New York	John
1	13	Paris	Anna
2	53	Berlin	Peter
3	33	London	Linda

Существует несколько способов осуществить запрос к таблице. Например:

```
In[8]:
# Выбрать все строки, в которых значение столбца age больше 30
display(data_pandas[data_pandas.Age > 30])
```

	Age	Location	Name
2	53	Berlin	Peter
3	33	London	Linda

mglearn

Эта книга содержит сопутствующий программный код, который вы можете найти на GitHub. Программный код включает в себя не только все примеры, приведенные в этой книге, но и библиотеку `mglearn`. Она представляет собой библиотеку, включающие разные полезные функции. Мы написали ее для этой книги, чтобы не перегружать листинги подробной информацией о построении графиков и загрузке данных. Если вам интересно, вы можете посмотреть все эти функции в репозитории, но детали `mglearn` не очень важны для понимания материала этой книги. Если вы видите вызов `mglearn` в программном коде, то речь, как правило,

идет о быстром способе построить красивую картинку или загрузить некоторые интересные данные.[3]

На протяжении всей книги мы будем достаточно много использовать NumPy, `matplotlib` и `pandas`. Поэтому убедитесь в импорте следующих библиотек:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn
```

Кроме того, мы полагаем, что вы будете запускать программный код в Jupyter Notebook, используя замечательные возможности `%matplotlib notebook` или `%matplotlib inline` для построения графиков. Если вы не используете notebook или эти замечательные команды, вам придется вызвать `plt.show`, чтобы вывести эти графики.
Некоторые примеры используют функцию `display` оболочки `IPython`, поэтому если при выполнении программного кода вы получаете ошибку, в которой так или иначе упоминается `display`, запустите следующую строку:

```
from IPython.display import display
```

Для решения проблемы корректного отображения русских надписей в графиках `matplotlib` воспользуйтесь следующим программным кодом:

```
plt.rc('font', family='Verdana')
```

Таким образом, первый блок программного кода каждой главы должен выглядеть следующим образом:

```
[ln1]
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import mglearn
from IPython.display import display
plt.rc('font', family='Verdana')
```

[3] Самый простой способ воспользоваться библиотекой `mglearn`, скачать папку `mglearn` с GitHub и в переменной окружения PATH прописать полный путь к ней. В Windows 7 для этого нажмите кнопку **Пуск**, выберите **Панель управления**. Дважды нажмите на **Система**, затем выберите **Дополнительные параметры системы**. Во вкладке **Дополнительно** нажмите на **Переменные среды**. Выберите **Path** и нажмите на **Изменить**. В поле **Значение переменной** введите путь к папке `mglearn` (например, C:\Anaconda3\mglearn). – *Прим. пер.*

Сравнение Python 2 и Python 3

Существуют две основные версии Python, которые широко используются на данный момент: Python 2 (точнее, 2.7) и Python 3 (последняя версия 3.5 на момент написания книги). Иногда это приводит к некоторой путанице. Python 2 уже активно не развивается, а поскольку Python 3 содержит существенные изменения, код, написанный для Python 2, как правило, не запустится в Python 3. Если вы новичок в Python или запускаете новый проект с нуля, мы настоятельно рекомендуем использовать последнюю версию Python 3. Если у вас есть большой фрагмент программного кода, написанный для Python 2, вам не нужно что-либо менять. Однако вы должны попытаться перейти на Python 3 как можно скорее. Вообще, при написании нового программного кода, как правило, довольно легко написать код, который будет работать и под Python 2 и под Python 3.[4] Если у вас нет необходимости использовать устаревшее программное обеспечение, вы должны обязательно использовать Python 3. Весь программный код в этой книге написан таким образом, что работает в обеих версиях. Однако некоторые детали вывода могут отличаться в этих версиях.

Версии библиотек, используемые в этой книге

В этой книге мы используем следующие версии ранее упомянутых библиотек:

```
In[9]:
import sys
print("версия Python: {}".format(sys.version))

import pandas as pd
print("версия pandas: {}".format(pd.__version__))

import matplotlib
print("версия matplotlib: {}".format(matplotlib.__version__))

import numpy as np
print("версия NumPy: {}".format(np.__version__))

import scipy as sp
print("версия SciPy: {}".format(sp.__version__))

import IPython
print("версия IPython: {}".format(IPython.__version__))

import sklearn
print("версия scikit-learn: {}".format(sklearn.__version__))
```

```
версия Python: 3.5.2 |Anaconda 4.1.1 (64-bit)| (default, Jul 2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
версия pandas: 0.18.1
версия matplotlib: 1.5.1
версия NumPy: 1.11.1
версия SciPy: 0.17.1
```

[4] Для решения этой задачи очень удобен пакет six.

```
версия IPython: 5.1.0
версия scikit-learn: 0.18
```

Несмотря на то, что точное соответствие вышеприведенным версиям неважно, у вас должна быть установлена версия `scikit-learn`, которая была последней по крайней мере на момент написания книги.

Теперь, когда мы все установили, давайте в первый раз применим машинное обучение.

Эта книга предполагает, что у вас установлена `scikit-learn` версии 0.18 или более свежая. Модуль `model_selection` появился в версии 0.18, и если вы используете более раннюю версию `scikit-learn`, вам нужно обновить `scikit-learn`, чтобы воспользоваться этим модулем.[5]

Первый пример: классификация сортов ириса

В этом разделе мы рассмотрим простой пример применения машинного обучения и построим нашу первую модель. В процессе изложения материала мы познакомим вас с некоторыми основными принципами и терминами.

Предположим, что ботаник-любитель хочет классифицировать сорта ирисов, которые он собрал. Он измерил в сантиметрах некоторые характеристики ирисов: длину и ширину лепестков, а также длину и ширину чашелистиков (см. рис. 1.2).

Кроме того, у него есть измерения этих же характеристик ирисов, которые ранее позволили опытному эксперту отнести их к сортам *setosa*, *versicolor* и *virginica*. Относительно этих ирисов ботаник-любитель уверенно может сказать, к какому сорту принадлежит каждый ирис. Давайте предположим, что перечисленные сорта являются единственными сортами, которые ботаник-любитель может встретить в дикой природе.

Наша цель заключается в построении модели машинного обучения, которая сможет обучиться на основе характеристик ирисов, уже классифицированных по сортам, и затем предскажет сорт для нового цветка ириса.

[5] Например, если вы установили пакет Anaconda для Windows, воспользуйтесь менеджером `conda`: `conda install -c anaconda scikit-learn=0.18.1`

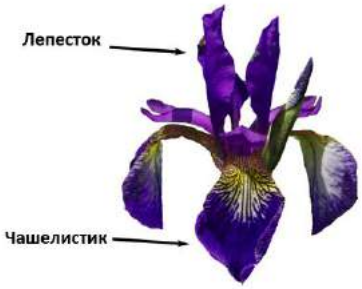

Рис. 1.2 Структура цветка ириса

Поскольку у нас есть примеры, по которых мы уже знаем правильные сорта ириса, решаемая задача является задачей обучения с учителем. В этой задаче нам нужно спрогнозировать один из сортов ириса. Это пример задачи *классификации* (*classification*). Возможные ответы (различные сорта ириса) называются *классами* (*classes*). Каждый ирис в наборе данных принадлежит к одному из трех классов, таким образом решаемая задача является задачей трехклассовой классификации.

Ответом для отдельной точки данных (ириса) является тот или иной сорт этого цветка. Сорт, к которому принадлежит цветок (конкретная точка данных), называется *меткой* (*label*).

Загружаем данные

Данные, которые мы будем использовать для этого примера, – это набор данных Iris, классический набор данных в машинном обучении и статистике. Он уже включен в модуль `datasets` библиотеки `scikit-learn`. Мы можем загрузить его, вызвав функцию `load_iris`:

```
In[10]:
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

Объект `iris`, возвращаемый `load_iris`, является объектом `Bunch`, который очень похож на словарь. Он содержит ключи и значения:

```
In[11]:
print("Ключи iris_dataset: \n{}".format(iris_dataset.keys()))
```

```
Out[11]:
Ключи iris_dataset:
dict_keys(['target_names', 'feature_names', 'DESCR', 'data', 'target'])
```

Значение ключа `DESCR` – это краткое описание набора данных. Здесь мы покажем начало описания (оставшуюся часть описания вы можете посмотреть самостоятельно):

```
In[12]:
print(iris_dataset['DESCR'][:193] + "\n...")


Out[12]:
Iris Plants Database
====================
Notes
----
Data Set Characteristics:
:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive att
...
----
```

Значение ключа `target_names` – это массив строк, содержащий сорта цветов, которые мы хотим предсказать:

```
In[13]:
print("Названия ответов: {}".format(iris_dataset['target_names']))

Out[13]:
Названия ответов: ['setosa' 'versicolor' 'virginica']
```

Значение `feature_names` – это список строк с описанием каждого признака:

```
In[14]:
print("Названия признаков: \n{}".format(iris_dataset['feature_names']))

Out[14]:
Названия признаков:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
 'petal width (cm)']
```

Сами данные записаны в массивах `target` и `data`. `data` – массив NumPy, который содержит количественные измерения длины чашелистиков, ширины чашелистиков, длины лепестков и ширины лепестков:

```
In[15]:
print("Тип массива data: {}".format(type(iris_dataset['data'])))

Out[15]:
Тип массива data: <class 'numpy.ndarray'>
```

Строки в массиве `data` соответствуют цветам ириса, а столбцы представляют собой четыре признака, которые были измерены для каждого цветка:

```
In[16]:
print("Форма массива data: {}".format(iris_dataset['data'].shape))

Out[16]:
Форма массива data: (150, 4)
```

Мы видим, что массив содержит измерения для 150 различных цветов по 4 признакам. Вспомним, что в машинном обучении отдельные элементы называются *примерами* (*samples*), а их свойства –

характеристиками или *признаками* (*feature*). *Форма* (*shape*) массива данных определяется количеством примеров, умноженным на количество признаков. Это является общепринятым соглашением в `scikit-learn`, и ваши данные всегда будут представлены в этой форме. Ниже приведены значения признаков для первых пяти примеров:

```
In[17]:
print("Первые пять строк массива data:\n{}".format(iris_dataset['data'][:5]))

Out[17]:
Первые пять строк массива data:
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]]
```

Взглянув на эти данные, мы видим, что все пять цветов имеют ширину лепестка 0.2 см и первый цветок имеет самую большую длину чашелистика, 5.1 см.

Массив `target` содержит сорта уже измеренных цветов, тоже записанные в виде массива NumPy:

```
In[18]:
print("Тип массива target: {}".format(type(iris_dataset['target'])))

Out[18]:
Тип массива target: <class 'numpy.ndarray'>
```

`target` представляет собой одномерный массив, по одному элементу для каждого цветка:

```
In[19]:
print("Форма массива target: {}".format(iris_dataset['target'].shape))

Out[19]:
Форма массива target: (150,)
```

Сорта кодируются как целые числа от 0 до 2:

```
In[20]:
print("Ответы:\n{}".format(iris_dataset['target']))

Out[20]:

Ответы:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Значения чисел задаются массивом `iris['target_names']`: 0 – *setosa*, 1 – *versicolor*, а 2 – *virginica*.

Метрики эффективности: обучающий и тестовый наборы

На основе этих данных нам нужно построить модель машинного обучения, которая предскажет сорта ириса для нового набора измерений. Но прежде, чем мы применить нашу модель к новому набору, мы должны убедиться в том, что модель на самом деле работает и ее прогнозам можно доверять.

К сожалению, для оценки качества модели мы не можем использовать данные, которые были взяты нами для построения модели. Это обусловлено тем, что наша модель просто запомнит весь обучающий набор и поэтому она всегда будет предсказывать правильную метку для любой точки данных в обучающем наборе. Это «запоминание» ничего не говорит нам об обобщающей способности модели (другими словами, мы не знаем, будет ли эта модель так же хорошо работать на новых данных).

Для оценки эффективности модели, мы предъявляем ей новые размеченные данные (размеченные данные, которые она не видела раньше). Обычно это делается путем разбиения собранных размеченных данных (в данном случае 150 цветов) на две части. Одна часть данных используется для построения нашей модели машинного обучения и называется *обучающими данными* (*training data*) или *обучающим набором* (*training set*). Остальные данные будут использованы для оценки качества модели, их называют *тестовыми данными* (*test data*), *тестовым набором* (*test set*) или *контрольным набором* (*hold-out set*).

В библиотеке `scikit-learn` есть функция `train_test_split`, которая перемешивает набор данных и разбивает его на две части. Эта функция отбирает в обучающий набор 75% строк данных с соответствующими метками. Оставшиеся 25% данных с метками объявляются тестовым набором. Вопрос о том, сколько данных отбирать в обучающий и тестовый наборы, является дискуссионным, однако использование тестового набора, содержащего 25% данных, является хорошим правилом.

В `scikit-learn` данные, как правило, обозначаются заглавной `X`, тогда как метки обозначаются строчной `y`. Это навеяно стандартной математической формулой $f(x)=y$, где x является аргументом функции, а y – выводом. В соответствии с некоторыми математическими соглашениями мы используем заглавную `X`, потому что данные представляют собой двумерный массив (матрицу) и строчную `y`, потому что целевая переменная – это одномерный массив (вектор).

Давайте вызовем функцию `train_test_split` для наших данных и зададим обучающие данные, обучающие метки, тестовые данные, тестовые метки, используя вышеупомянутые буквы:

```
In[21]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)
```

Перед разбиением функция `train_test_split` перемешивает набор данных с помощью генератора псевдослучайных чисел. Если мы просто возьмем последние 25% наблюдений в качестве тестового набора, все точки данных будет иметь метку 2, поскольку все точки данных отсортированы по меткам (смотрите вывод для `iris['target']`, показанный ранее). Используя тестовый набор, содержащий только один из трех классов, вы не сможете объективно судить об обобщающей способности модели, таким образом, мы перемешиваем наши данные, чтобы тестовые данные содержали все три класса.

Чтобы в точности повторно воспроизвести полученный результат, мы воспользуемся генератором псевдослучайных чисел с фиксированным стартовым значением, которое задается с помощью параметра `random_state`. Это позволит сделать результат воспроизводим, поэтому вышеприведенный программный код будет генерировать один и тот же результат. Мы всегда будем задавать `random_state` при использовании рандомизированных процедур в этой книге.

Выводом функции `train_test_split` являются `X_train`, `X_test`, `y_train` и `y_test`, которые все являются массивами Numpy. `X_train` содержит 75% строк набора данных, а `X_test` содержит оставшиеся 25%:

```
In[22]:
print("форма массива X_train: {}".format(X_train.shape))
print("форма массива y_train: {}".format(y_train.shape))

Out[22]:
форма массива X_train: (112, 4)
форма массива y_train: (112,)

In[23]:
print("форма массива X_test: {}".format(X_test.shape))
print("форма массива y_test: {}".format(y_test.shape))

Out[23]:
форма массива X_test: (38, 4)
форма массива y_test: (38,)
```

Сперва посмотрите на Ваши данные

Перед тем как строить модель машинного обучения, неплохо было бы исследовать данные, чтобы понять, можно ли легко решить поставленную задачу без машинного обучения или содержится ли нужная информация в данных.

Кроме того, исследование данных – это хороший способ обнаружить аномалии и особенности. Например, вполне возможно, что некоторые из ваших ирисов измерены в дюймах, а не в сантиметрах. В реальном мире нестыковки в данных и неожиданности очень распространены.

Один из лучших способов исследовать данные – визуализировать их. Это можно сделать, используя *диаграмму рассеяния* (*scatter plot*). В диаграмме рассеяния один признак откладывается по оси x, а другой признак – по оси y, каждому наблюдению соответствует точка. К сожалению, экран компьютера имеют только два измерения, что позволяет разместить на графике только два (или, возможно, три) признака одновременно. Таким образом, трудно разместить на графике наборы данных с более чем тремя признаками. Один из способов решения этой проблемы – построить *матрицу диаграмм рассеяния* (*scatterplot matrix*) или *парные диаграммы рассеяния* (*pair plots*), на которых будут изображены все возможные пары признаков. Если у вас есть небольшое число признаков, например, четыре, как здесь, то использование матрицы диаграмм рассеяния будет вполне разумным. Однако, вы должны помнить, что матрица диаграмм рассеяния не показывает взаимодействие между всеми признаками сразу, поэтому некоторые интересные аспекты данных не будут выявлены с помощью этих графиков.

Рис. 1.3 представляет собой матрицу диаграмм рассеяния для признаков обучающего набора. Точки данных окрашены в соответствии с сортами ириса, к которым они относятся. Чтобы построить диаграммы, мы сначала преобразовываем массив NumPy в `DataFrame` (основной тип данных в библиотеке `pandas`). В `pandas` есть функция для создания парных диаграмм рассеяния под названием `scatter_matrix`. По диагонали этой матрицы располагаются гистограммы каждого признака:

In[24]:
```
# создаем dataframe из данных в массиве X_train
# маркируем столбцы, используя строки в iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# создаем матрицу рассеяния из dataframe, цвет точек задаем с помощью y_train
grr = pd.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15), marker='o',
                        hist_kwds={'bins': 20}, s=60, alpha=.8, cmap=mglearn.cm3)
```

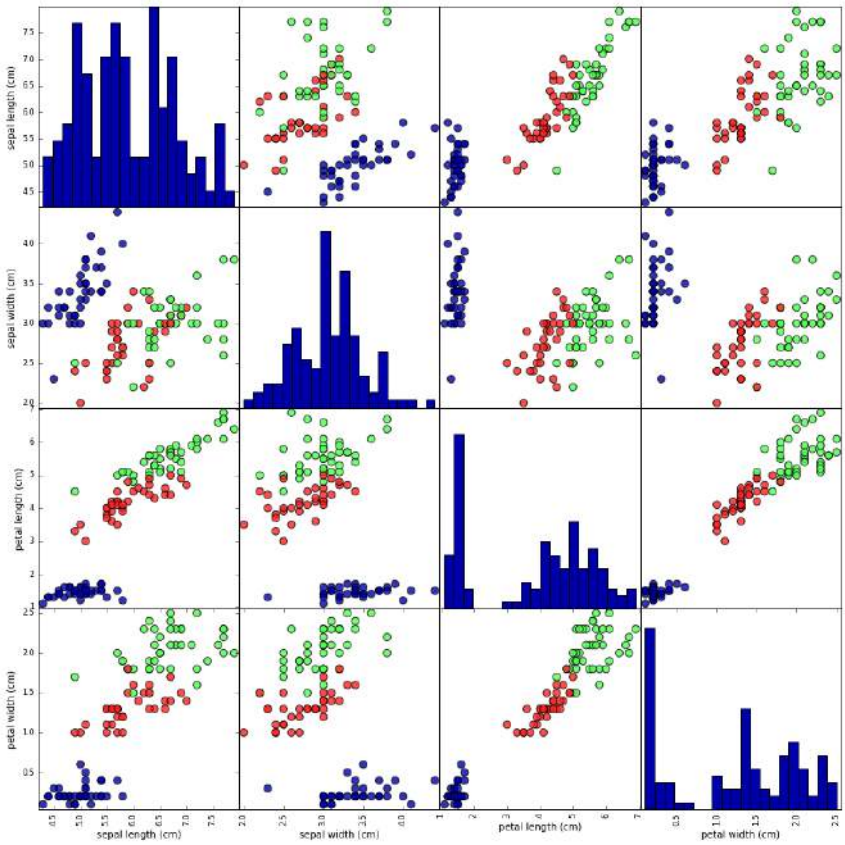

Рис. 1.3 Матрица диаграмм рассеяния для набора данных Iris, цвет точек данных определяется метками классов

Взглянув на график, мы можем увидеть, что, похоже, измерения чашелистиков и лепестков позволяют относительно хорошо разделить три класса. Это означает, что модель машинного обучения, вероятно, сможет научиться разделять их.

Построение вашей первой модели: метод *k* ближайших соседей

Теперь мы можем начать строить реальную модель машинного обучения. В библиотеке `scikit-learn` имеется довольно много алгоритмов классификации, которые мы могли бы использовать для построения модели. В данном примере мы будем использовать классификатор на основе метода *k* ближайших соседей, который легко интерпретировать. Построение этой модели заключается лишь в запоминании обучающего набора. Для того, чтобы сделать прогноз для новой точки данных, алгоритм находит точку в обучающем наборе, которая находится ближе всего к новой точке. Затем он присваивает метку, принадлежащую этой точке обучающего набора, новой точке данных.

k в методе *k* ближайших соседей означает, что вместо того, чтобы использовать лишь ближайшего соседа новой точки данных, мы в ходе обучения можем рассмотреть любое фиксированное число (*k*) соседей (например, рассмотреть ближайшие три или пять соседей). Тогда мы можем сделать прогноз для точки данных, используя класс, которому принадлежит большинство ее соседей. Подробнее мы поговорим об этом в главе 2, а в данный момент мы будем использовать только одного соседа.

В `scikit-learn` все модели машинного обучения реализованы в собственных классах, называемых классами `Estimator`. Алгоритм классификации на основе метода *k* ближайших соседей реализован в классификаторе `KNeighborsClassifier` модуля `neighbors`. Прежде чем использовать эту модель, нам нужно создать объект-экземпляр класса. Это произойдет, когда мы зададим параметры модели. Самым важным параметром `KNeighborsClassifier` является количество соседей, которые мы установим равным 1:

```
In[25]:
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

Объект `knn` включает в себя алгоритм, который будет использоваться для построения модели на обучающих данных, а также алгоритм, который сгенерирует прогнозы для новых точек данных. Он также будет содержать информацию, которую алгоритм извлек из обучающих данных. В случае с `KNeighborsClassifier` он будет просто хранить обучающий набор.

Для построения модели на обучающем наборе, мы вызываем метод `fit` объекта `knn`, который принимает в качестве аргументов массив NumPy `X_train`, содержащий обучающие данные, и массив NumPy `y_train`, соответствующий обучающим меткам:

```
In[26]:
knn.fit(X_train, y_train)

Out[26]:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

Метод `fit` возвращает сам объект `knn` (и изменяет его), таким образом, мы получаем строковое представление нашего классификатора. Оно показывает нам, какие параметры были использованы при создании модели. Почти все параметры имеют значения по умолчанию, но вы также можете обнаружить параметр `n_neighbor=1`, заданный нами. Большинство моделей в `scikit-learn` имеют массу параметров, но большая часть из них связана с оптимизацией скорости вычислений или

предназначена для особых случаев использования. Вам не нужно беспокоиться о других параметрах, приведенных здесь. Вывод модели в `scikit-learn` может быть очень длинным, но не нужно пугаться его. Мы рассмотрим все важные параметры в главе 2. В оставшейся части этой книги мы не будем приводить вывод метода `fit`, поскольку он не содержит никакой новой информации.

Получение прогнозов

Теперь мы можем получить прогнозы, применив эту модель к новым данным, по которым мы еще не знаем правильные метки. Представьте, что мы нашли в дикой природе ирис с длиной чашелистика 5 см, шириной чашелистика 2.9 см, длиной лепестка 1 см и шириной лепестка 0.2 см. К какому сорту ириса нужно отнести этот цветок? Мы можем поместить эти данные в массив NumPy, снова вычисляя форму массива, т.е. количество примеров (1), умноженное на количество признаков (4):

```
In[27]:
X_new = np.array([[5, 2.9, 1, 0.2]])
print("форма массива X_new: {}".format(X_new.shape))

Out[27]:
форма массива X_new: (1, 4)
```

Обратите внимание, что мы записали измерения по одному цветку в двумерный массив NumPy, поскольку `scikit-learn` работает с двумерными массивами данных.

Чтобы сделать прогноз, мы вызываем метод `predict` объекта `knn`:

```
In[28]:
prediction = knn.predict(X_new)
print("Прогноз: {}".format(prediction))
print("Спрогнозированная метка: {}".format(
       iris_dataset['target_names'][prediction]))

Out[28]:
Прогноз: [0]
Спрогнозированная метка: ['setosa']
```

Наша модель предсказывает, что этот новый цветок ириса принадлежит к классу 0, что означает сорт *setosa*. Но как узнать, можем ли мы доверять нашей модели? Правильный сорт ириса для этого примера нам неизвестен, а ведь именно получение правильных прогнозов и является главной задачей построения модели!

Оценка качества модели

Это тот самый момент, когда нам понадобится созданный ранее тестовый набор. Эти данные не использовались для построения модели, но мы знаем правильные сорта для каждого ириса в тестовом наборе.

Таким образом, мы можем сделать прогноз для каждого ириса в тестовом наборе и сравнить его с фактической меткой (уже известным сортом). Мы можем оценить качество модели, вычислив *правильность* (*accuracy*) – процент цветов, для которых модель правильно спрогнозировала сорта:

```
In[29]:
y_pred = knn.predict(X_test)
print("Прогнозы для тестового набора:\n {}".format(y_pred))

Out[29]:
Прогнозы для тестового набора:
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0 2]

In[30]:
print("Правильность на тестовом наборе: {:.2f}".format(np.mean(y_pred == y_test)))

Out[30]:
Правильность на тестовом наборе: 0.97
```

Кроме того, мы можем использовать метод `score` объекта `knn`, который вычисляет правильность модели для тестового набора:

```
In[31]:
print("Правильность на тестовом наборе: {:.2f}".format(knn.score(X_test, y_test)))

Out[31]:
Правильность на тестовом наборе: 0.97
```

Правильность этой модели для тестового набора составляет около 0.97, что означает, что мы дали правильный прогноз для 97% ирисов в тестовом наборе. При некоторых математических допущениях, это означает, что мы можем ожидать, что наша модель в 97% случаев даст правильный прогноз для новых ирисов. Для нашего ботаника-любителя этот высокий уровень правильности означает, что наша модель может быть достаточно надежной в использовании. В следующих главах мы обсудим, как можно улучшить эффективность модели, и с какими подводными камнями можно столкнуться при настройке модели.

Выводы и перспективы

Давайте подытожем то, чему мы научились в этой главе. Мы начали с краткого введения в машинного обучение и сфер его применения, затем обсудили различие между обучением с учителем и обучением без учителя и дали краткий обзор инструментов, которые мы будем

использовать в этой книге. Затем мы сформулировали задачу классификации ирисов на основе проведенных измерений характеристик цветов. Мы использовали набор данных, в котором эксперт уже предварительно классифицировал ирисы для построения модели, таким образом, мы решали задачу обучения с учителем. Было три возможных сорта ирисов – *setosa*, *versicolor* и *virginica*, что делало нашу задачу задачей 3-классовой классификации. В задаче классификации возможные сорта ирисов называются *классами* (*classes*) а сами названия сортов – *метками* (*labels*).

Набор данных Iris состоит из двух массивов NumPy: один содержит данные и в `scikit-learn` обозначается как `X`, другой содержит правильные или нужные ответы и обозначается как `y`. Массив `X` представляет собой двумерный массив признаков, в котором одна строка соответствует одной точке данных, а один столбец – одному признаку. Массив y представляет собой одномерный массив, который для каждого примера содержит метку класса, целое число от 0 до 2.

Мы разделили наш набор данных на *обучающий набор* (*training set*), чтобы построить нашу модель, а также *тестовый набор* (*test set*), чтобы оценить, насколько хорошо наша модель будет классифицировать новые, ранее незнакомые ей данные.

Мы выбрали алгоритм классификации *k* ближайших соседей, который генерирует прогноз для новой точки данных, рассматривая ее ближайшего соседа(ей) в обучающем наборе. Все это реализовано в классе `KNeighborsClassifier`, который содержит алгоритм, строящий модель, а также алгоритм, который дает прогнозы, используя построенную модель. Мы создали объект-экземпляр класса, задав параметры. Затем мы построили модель, вызвав метод `fit` и передав обучающие данные (`X_train`) и обучающие ответы (`y_train`) в качестве параметров. Мы оценили качество модели с использованием метода `score`, который вычисляет правильность модели. Мы применили метод `score` к тестовым данным и тестовым ответам и обнаружили, что наша модель демонстрирует правильность около 97%. Это означает, что модель выдает правильные прогнозы для 97% наблюдений тестового набора.

Это убедило нас в том, что модель можно применить к новым данным (в нашем примере это измерения характеристик новых цветов), и мы надеемся, что эта модель даст правильные прогнозы в 97% случаев.

Ниже приводится краткое изложение программного кода, необходимого для всей процедуры обучения и оценки модели:

```
In[32]:
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
```

```
print("Правильность на тестовом наборе: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Out[32]:
Правильность на тестовом наборе: 0.97
```

Этот фрагмент содержит базовый код, необходимый для применения любого алгоритма машинного обучения с помощью `scikit-learn`. Методы `fit`, `predict` и `score` являются общими для моделей контролируемого обучения в `scikit-learn` и, используя принципы, приведенные в этой главе, вы можете применить эти модели для решения различных задач машинного обучения. В следующей главе мы подробнее рассмотрим различные модели машинного обучения с учителем, имеющиеся в `scikit-learn`, и расскажем, как успешно применять их.