O'REILLY  
Introduction to  
Machine  
Learning  
with Python  
A GUIDE FOR DATA SCIENTISTS  
poweredby  
Jupyter  
Andreas C. Müller& Sarah Guido  
Introduction to Machine Learning with Python  
A Guide for Data Scientists  
Andreas C. Müller and Sarah Guido  
Introduction to Machine Learning with Python  
by Andreas C. Müller and Sarah Guido  
Copyright © 2017 Sarah Guido, Andreas Müller . All rights reserved .  
Printed in the United States of America.  
Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA  
95472 .  
O'Reilly books may be purchased for educational, business, or sales promotional  
use . Online editions are also available for most titles (http://oreilly.com/safari).  
For more information, contact our corporate/institutional sales department : 800-  
998-9938 or corporate@oreilly.com .  
Editor : Dawn Schanafelt  
Production Editor: Kristen Brown  
Copyeditor : Rachel Head  
Proofreader : Jasmine Kwityn  
Indexer : Judy McConville  
Interior Designer : David Futato  
Cover Designer: Karen Montgomery  
Illustrator : Rebecca Demarest  
October 2016: First Edition  
Revision History for the First Edition  
See  
• 2016-09-22 : First Release  
• 2017-01-13 : Second Release  
• 2017-06-09 : Third Release  
• 2018-10-19: Fourth Release  
http://oreilly.com/catalog/errata.csp?isbn=9781449369415 for release details .  
The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. Introduction to  
Machine Learning with Python, the cover image, and related trade dress are  
trademarks of O’Reilly Media, Inc.  
While the publisher and the authors have used good faith efforts to ensure that  
the information and instructions contained in this work are accurate, the  
publisher and the authors disclaim all responsibility for errors or omissions,  
including without limitation responsibility for damages resulting from the use of  
or reliance on this work. Use of the information and instructions contained in this  
work is at your own risk. If any code samples or other technology this work  
contains or describes is subject to open source licenses or the intellectual property  
rights of others, it is your responsibility to ensure that your use thereof complies  
with such licenses and/or rights.  
978-1-449-36941-5  
[LSI]  
Preface  
Machine learning is an integral part of many commercial applications and  
research projects today, in areas ranging from medical diagnosis and treatment to  
finding your friends on social networks. Many people think that machine learning  
can only be applied by large companies with extensive research teams. In this  
book, we want to show you how easy it can be to build machine learning solutions  
yourself, and how to best go about it. With the knowledge in this book, you can  
build your own system for finding out how people feel on Twitter, or making  
predictions about global warming. The applications of machine learning are  
endless and, with the amount of data available today, mostly limited by your  
imagination.  
Who Should Read This Book  
This book is for current and aspiring machine learning practitioners looking to  
implement solutions to real-world machine learning problems. This is an  
introductory book requiring no previous knowledge of machine learning or  
artificial intelligence (AI). We focus on using Python and the scikit-learn library,  
and work through all the steps to create a successful machine learning application.  
The methods we introduce will be helpful for scientists and researchers, as well as  
data scientists working on commercial applications. You will get the most out of  
the book if you are somewhat familiar with Python and the NumPy and matplotlib  
libraries.  
We made a conscious effort not to focus too much on the math, but rather on the  
practical aspects of using machine learning algorithms. As mathematics  
(probability theory, in particular) is the foundation upon which machine learning  
is built, we won’t go into the analysis of the algorithms in great detail. If you are  
interested in the mathematics of machine learning algorithms, we recommend the  
book The Elements of Statistical Learning (Springer) by Trevor Hastie, Robert  
Tibshirani, and Jerome Friedman, which is available for free at the authors’  
website. We will also not describe how to write machine learning algorithms from  
scratch, and will instead focus on how to use the large array of models already  
implemented in scikit-learn and other libraries.  
Why We Wrote This Book  
There are many books on machine learning and AI. However, all of them are meant  
for graduate students or PhD students in computer science, and they’re full of  
advanced mathematics. This is in stark contrast with how machine learning is  
being used, as a commodity tool in research and commercial applications. Today,  
applying machine learning does not require a PhD. However, there are few  
resources out there that fully cover all the important aspects of implementing  
machine learning in practice, without requiring you to take advanced math  
courses. We hope this book will help people who want to apply machine learning  
without reading up on years' worth of calculus, linear algebra, and probability  
theory.  
Navigating This Book  
This book is organized roughly as follows:  
• Chapter 1 introduces the fundamental concepts of machine learning and  
its applications, and describes the setup we will be using throughout the  
book.  
• Chapters 2 and 3 describe the actual machine learning algorithms that are  
most widely used in practice, and discuss their advantages and  
shortcomings .  
• Chapter 4 discusses the importance of how we represent data that is  
processed by machine learning, and what aspects of the data to pay  
attention to .  
• Chapter 5 covers advanced methods for model evaluation and parameter  
tuning, with a particular focus on cross-validation and grid search .  
• Chapter 6 explains the concept of pipelines for chaining models and  
encapsulating your workflow.  
• Chapter 7 shows how to apply the methods described in earlier chapters to  
text data, and introduces some text-specific processing techniques .  
• Chapter 8 offers a high-level overview, and includes references to more  
advanced topics.  
While Chapters 2 and 3 provide the actual algorithms, understanding all of these  
algorithms might not be necessary for a beginner. Ifyou need to build a machine  
learning system ASAP , we suggest starting with Chapter 1 and the opening sections  
of Chapter 2 , which introduce all the core concepts. You can then skip to Section  
2.5 in Chapter 2, which includes a list of all the supervised models that we cover .  
Choose the model that best fits your needs and flip back to read the section  
devoted to it for details. Then you can use the techniques in Chapter 5 to evaluate  
and tune your model.  
Online Resources  
While studying this book, definitely refer to the scikit - learn website for more indepth documentation of the classes and functions, and many examples. There is  
also a video course created by Andreas Müller, "Advanced Machine Learning with  
scikit- learn, " that supplements this book. You can find it at  
http://bit.ly/advanced\_machine\_learning\_scikit-learn .  
Conventions Used in This Book  
The following typographical conventions are used in this book:  
Italic  
Indicates new terms, URLs, email addresses, filenames, and file extensions.  
Constant width  
Used for program listings, as well as within paragraphs to refer to program  
elements such as variable or function names, databases, data types,  
environment variables, statements, and keywords. Also used for commands and  
module and package names.  
Constant width bold  
Shows commands or other text that should be typed literally by the user.  
Constant width italic  
Tip  
Shows text that should be replaced with user-supplied values or by values  
determined by context.  
This element signifies a tip or suggestion.  
Note  
This element signifies a general note.  
Warning  
This icon indicates a warning or caution.  
Using Code Examples  
Supplemental material (code examples, IPython notebooks, etc.) is available for  
download at https://github.com/amueller/introduction\_to\_ml\_with\_python.  
This book is here to help you get your job done. In general, if example code is  
offered with this book, you may use it in your programs and documentation. You  
do not need to contact us for permission unless you’re reproducing a significant  
portion of the code. For example, writing a program that uses several chunks of  
code from this book does not require permission. Selling or distributing a CD-ROM  
of examples from O’Reilly books does require permission. Answering a question by  
citing this book and quoting example code does not require permission.  
Incorporating a significant amount of example code from this book into your  
product’s documentation does require permission.  
We appreciate, but do not require, attribution. An attribution usually includes the  
title, author, publisher, and ISBN. For example: “An Introduction to Machine Learning  
with Python by Andreas C. Müller and Sarah Guido (O’Reilly). Copyright 2017 Sarah  
Guido and Andreas Müller, 978-1-449-36941-5.”  
If you feel your use of code examples falls outside fair use or the permission given  
above, feel free to contact us at permissions@oreilly.com.  
O’Reilly Safari  
Safari (formerly Safari Books Online) is a membership-based training and reference  
platform for enterprise, government, educators, and individuals.  
Members have access to thousands of books, training videos, Learning Paths,  
interactive tutorials, and curated playlists from over 250 publishers, including  
O’Reilly Media, Harvard Business Review, Prentice Hall Professional, AddisonWesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Adobe, Focal Press,  
Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt,  
Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett,  
and Course Technology, among others.  
For more information, please visit http://oreilly.com/safari.  
How to Contact Us  
Please address comments and questions concerning this book to the publisher:  
O’Reilly Media, Inc.  
1005 Gravenstein Highway North  
Sebastopol, CA 95472  
800-998-9938 (in the United States or Canada)  
707-829-0515 (international or local)  
707-829-0104 (fax)  
We have a web page for this book, where we list errata, examples, and any  
additional information. You can access this page at http://bit.ly/intro-machinelearning-python.  
To comment or ask technical questions about this book, send email to  
bookquestions@oreilly.com.  
For more information about our books, courses, conferences, and news, see our  
website at http://www.oreilly.com.  
Find us on Facebook: http://facebook.com/oreilly  
Follow us on Twitter: http://twitter.com/oreillymedia  
Watch us on YouTube: http://www.youtube.com/oreillymedia  
Acknowledgments  
From Andreas  
Without the help and support of a large group of people, this book would never  
have existed.  
I would like to thank the editors, Meghan Blanchette, Brian MacDonald, and in  
particular Dawn Schanafelt, for helping Sarah and me make this book a reality.  
I want to thank my reviewers, Thomas Caswell, Olivier Grisel, Stefan van der Walt,  
and John Myles White, who took the time to read the early versions of this book  
and provided me with invaluable feedback—in addition to being some of the  
cornerstones of the scientific open source ecosystem.  
I am forever thankful for the welcoming open source scientific Python community,  
especially the contributors to scikit-learn. Without the support and help from  
this community, in particular from Gael Varoquaux, Alex Gramfort, and Olivier  
Grisel, I would never have become a core contributor to scikit-learn or learned  
to understand this package as well as I do now. My thanks also go out to all the  
other contributors who donate their time to improve and maintain this package.  
I’m also thankful for the discussions with many of my colleagues and peers that  
helped me understand the challenges of machine learning and gave me ideas for  
structuring a textbook. Among the people I talk to about machine learning, I  
specifically want to thank Brian McFee, Daniela Huttenkoppen, Joel Nothman,  
Gilles Louppe, Hugo Bowne-Anderson, Sven Kreis, Alice Zheng, Kyunghyun Cho,  
Pablo Baberas, and Dan Cervone.  
My thanks also go out to Rachel Rakov, who was an eager beta tester and  
proofreader of an early version of this book, and helped me shape it in many ways.  
On the personal side, I want to thank my parents, Harald and Margot, and my  
sister, Miriam, for their continuing support and encouragement. I also want to  
thank the many people in my life whose love and friendship gave me the energy  
and support to undertake such a challenging task.  
From Sarah  
I would like to thank Meg Blanchette, without whose help and guidance this  
project would not have even existed. Thanks to Celia La and Brian Carlson for  
reading in the early days. Thanks to the O’Reilly folks for their endless patience.  
And finally, thanks to DTS, for your everlasting and endless support.  
Chapter 1. Introduction  
Machine learning is about extracting knowledge from data. It is a research field at  
the intersection of statistics, artificial intelligence , and computer science and is  
also known as predictive analytics or statistical learning. The application of  
machine learning methods has in recent years become ubiquitous in everyday life .  
From automatic recommendations of which movies to watch, to what food to order  
or which products to buy, to personalized online radio and recognizing your  
friends in your photos, many modern websites and devices have machine learning  
algorithms at their core . When you look at a complex website like Facebook,  
Amazon, or Netflix, it is very likely that every part of the site contains multiple  
machine learning models .  
Outside of commercial applications, machine learning has had a tremendous  
influence on the way data-driven research is done today. The tools introduced in  
this book have been applied to diverse scientific problems such as understanding  
stars, finding distant planets, discovering new particles, analyzing DNA sequences ,  
and providing personalized cancer treatments.  
Your application doesn't need to be as large-scale or world-changing as these  
examples in order to benefit from machine learning, though. In this chapter, we  
will explain why machine learning has become so popular, and discuss what kind of  
problem can be solved using machine learning. Then, we will show you how to  
build your first machine learning model, introducing important concepts along the  
way.  
1.1 Why Machine Learning?  
In the early days of "intelligent" applications, many systems used handcoded rules  
of "if" and "else" decisions to process data or adjust to user input . Think of a spam  
filter whose job is to move the appropriate incoming email messages to a spam  
folder . You could make up a blacklist of words that would result in an email being  
marked as spam. This would be an example of using an expert-designed rule  
system to design an "intelligent" application . Manually crafting decision rules is  
feasible for some applications, particularly those in which humans have a good  
understanding of the process to model. However, using handcoded rules to make  
decisions has two major disadvantages :  
and domain single a to specific is decision a make to required logic The •  
task. Changing the task even slightly might require a rewrite of the whole  
system .  
• Designing rules requires a deep understanding of how a decision should be  
made by a human expert .  
One example of where this handcoded approach will fail is in detecting faces in  
images . Today, every smartphone can detect a face in an image . However, face  
detection was an unsolved problem until as recently as 2001. The main problem is  
that the way in which pixels (which make up an image in a computer) are  
"perceived " by the computer is very different from how humans perceive a face .  
This difference in representation makes it basically impossible for a human to  
come up with a good set of rules to describe what constitutes a face in a digital  
image .  
Using machine learning, however, simply presenting a program with a large  
collection of images of faces is enough for an algorithm to determine what  
characteristics are needed to identify a face .  
1.1.1 Problems Machine Learning Can Solve  
The most successful kinds of machine learning algorithms are those that automate  
decision-making processes by generalizing from known examples. In this setting,  
which is known as supervised learning, the user provides the algorithm with pairs of  
inputs and desired outputs, and the algorithm finds a way to produce the desired  
output given an input. In particular , the algorithm is able to create an output for  
an input it has never seen before without any help from a human. Going back to  
our example of spam classification, using machine learning, the user provides the  
algorithm with a large number of emails (which are the input), together with  
information about whether any of these emails are spam (which is the desired  
output) . Given a new email, the algorithm will then produce a prediction as to  
whether the new email is spam.  
Machine learning algorithms that learn from input/output pairs are called  
supervised learning algorithms because a "teacher" provides supervision to the  
algorithms in the form of the desired outputs for each example that they learn  
from. While creating a dataset of inputs and outputs is often a laborious manual  
process, supervised learning algorithms are well understood and their  
performance is easy to measure. Ifyour application can be formulated as a  
supervised learning problem, and you are able to create a dataset that includes  
the desired outcome , machine learning will likely be able to solve your problem .  
Examples of supervised machine learning tasks include :  
Identifying the zip code from handwritten digits on an envelope  
Here the input is a scan of the handwriting, and the desired output is the  
actual digits in the zip code . To create a dataset for building a machine  
learning model, you need to collect many envelopes. Then you can read the zip  
codes yourself and store the digits as your desired outcomes.  
Determining whether a tumor is benign based on a medical image  
Here the input is the image , and the output is whether the tumor is benign. To  
create a dataset for building a model, you need a database of medical images .  
You also need an expert opinion, so a doctor needs to look at all of the images  
and decide which tumors are benign and which are not . It might even be  
necessary to do additional diagnosis beyond the content of the image to  
determine whether the tumor in the image is cancerous or not .  
Detecting fraudulent activity in credit card transactions  
Here the input is a record of the credit card transaction, and the output is  
whether it is likely to be fraudulent or not . Assuming that you are the entity  
distributing the credit cards, collecting a dataset means storing all transactions  
and recording if a user reports any transaction as fraudulent.  
An interesting thing to note about these examples is that although the inputs and  
outputs look fairly straightforward, the data collection process for these three  
tasks is vastly different . While reading envelopes is laborious, it is easy and cheap .  
Obtaining medical imaging and diagnoses, on the other hand, requires not only  
expensive machinery but also rare and expensive expert knowledge , not to  
mention the ethical concerns and privacy issues. In the example of detecting  
credit card fraud, data collection is much simpler. Your customers will provide  
you with the desired output, as they will report fraud. All you have to do to obtain  
the input/output pairs of fraudulent and nonfraudulent activity is wait .  
Unsupervised algorithms are the other type of algorithm that we will cover in this  
book. In unsupervised learning, only the input data is known, and no known output  
data is given to the algorithm. While there are many successful applications of  
these methods, they are usually harder to understand and evaluate . Examples of  
unsupervised learning include:  
Identifying topics in a set of blog posts  
Ifyou have a large collection of text data, you might want to summarize it and  
find prevalent themes in it . You might not know beforehand what these topics  
are , or how many topics there might be . Therefore , there are no known  
outputs.  
Segmenting customers into groups with similar preferences  
Given a set of customer records, you might want to identify which customers  
are similar, and whether there are groups of customers with similar  
preferences . For a shopping site, these might be "parents," "bookworms, ” or  
"gamers . " Because you don't know in advance what these groups might be , or  
even how many there are , you have no known outputs.  
Detecting abnormal access patterns to a website  
To identify abuse or bugs, it is often helpful to find access patterns that are  
different from the norm . Each abnormal pattern might be very different , and  
you might not have any recorded instances of abnormal behavior . Because in  
this example you only observe traffic, and you don't know what constitutes  
normal and abnormal behavior, this is an unsupervised problem.  
For both supervised and unsupervised learning tasks, it is important to have a  
representation ofyour input data that a computer can understand. Often it is  
helpful to think of your data as a table . Each data point that you want to reason  
about (each email, each customer, each transaction) is a row, and each property  
that describes that data point (say, the age of a customer or the amount or location  
of a transaction) is a column. You might describe users by their age , their gender ,  
when they created an account, and how often they have bought from your online  
shop . You might describe the image of a tumor by the grayscale values of each  
pixel , or maybe by using the size , shape , and color of the tumor .  
Each entity or row here is known as a sample (or data point) in machine learning,  
while the columns-the properties that describe these entities are called features .  
Later in this book we will go into more detail on the topic of building a good  
representation ofyour data, which is called feature extraction or feature engineering .  
You should keep in mind, however, that no machine learning algorithm will be  
able to make a prediction on data for which it has no information. For example , if  
the only feature that you have for a patient is their last name , no algorithm will be  
able to predict their gender . This information is simply not contained in your data .  
Ifyou add another feature that contains the patient's first name , you will have  
much better luck, as it is often possible to tell the gender by a person's first name .  
1.1.2 Knowing Your Task and Knowing Your Data  
Quite possibly the most important part in the machine learning process is  
understanding the data you are working with and how it relates to the task you  
want to solve. It will not be effective to randomly choose an algorithm and throw  
your data at it. It is necessary to understand what is going on in your dataset  
before you begin building a model. Each algorithm is different in terms of what  
kind of data and what problem setting it works best for . While you are building a  
machine learning solution, you should answer, or at least keep in mind, the  
following questions:  
• What question(s)am I tryingto answer? Do I thinkthe data collected can  
answer that question?  
• What is the best way to phrase my question(s) as a machine learning  
problem?  
• Have I collected enough data to represent the problem I want to solve?  
What features of the data did I extract, and will these enable the right  
predictions?  
• •  
How will I measure success in my application?  
• How will the machine learning solution interact with other parts of my  
research or business product?  
In a larger context, the algorithms and methods in machine learning are only one  
part of a greater process to solve a particular problem, and it is good to keep the  
big picture in mind at all times. Many people spend a lot of time building complex  
machine learning solutions, only to find out they don't solve the right problem.  
When going deep into the technical aspects of machine learning (as we will in this  
book), it is easy to lose sight of the ultimate goals. While we will not discuss the  
questions listed here in detail , we still encourage you to keep in mind all the  
assumptions that you might be making, explicitly or implicitly, when you start  
building machine learning models.  
1.2 Why Python?  
Python has become the lingua franca for many data science applications. It  
combines the power of general-purpose programming languages with the ease of  
use of domain-specific scripting languages like MATLAB or R. Python has libraries  
for data loading, visualization, statistics, natural language processing, image  
processing, and more . This vast toolbox provides data scientists with a large array  
of general- and special-purpose functionality. One of the main advantages of using  
Python is the ability to interact directly with the code, using a terminal or other  
tools like the Jupyter Notebook, which we'll look at shortly. Machine learning and  
data analysis are fundamentally iterative processes, in which the data drives the  
analysis . It is essential for these processes to have tools that allow quick iteration  
and easy interaction.  
As a general -purpose programming language, Python also allows for the creation  
of complex graphical user interfaces (GUIs) and web services, and for integration  
into existing systems .  
1.3 scikit-learn  
scikit - learn is an open source project, meaning that it is free to use and  
distribute , and anyone can easily obtain the source code to see what is going on  
behind the scenes. The scikit - learn project is constantly being developed and  
improved, and it has a very active user community. It contains a number of stateof-the- art machine learning algorithms, as well as comprehensive documentation  
about each algorithm. scikit - learn is a very popular tool, and the most  
prominent Python library for machine learning. It is widely used in industry and  
academia, and a wealth of tutorials and code snippets are available online .  
scikit - learn works well with a number of other scientific Python tools, which we  
will discuss later in this chapter .  
While reading this, we recommend that you also browse the scikit - learn user  
guide and API documentation for additional details on and many more options for  
each algorithm. The online documentation is very thorough, and this book will  
provide you with all the prerequisites in machine learning to understand it in  
detail.  
1.3.1 Installing scikit-learn  
scikit-learn depends on two other Python packages, NumPy and SciPy. For  
plotting and interactive development, you should also install matplotlib, IPython,  
and the Jupyter Notebook. We recommend using one of the following prepackaged  
Python distributions, which will provide the necessary packages:  
Anaconda  
A Python distribution made for large-scale data processing, predictive  
analytics, and scientific computing. Anaconda comes with NumPy, SciPy,  
matplotlib, pandas, IPython, Jupyter Notebook, and scikit-learn. Available  
on Mac OS, Windows, and Linux, it is a very convenient solution and is the one  
we suggest for people without an existing installation of the scientific Python  
packages.  
Enthought Canopy  
Another Python distribution for scientific computing. This comes with NumPy,  
SciPy, matplotlib, pandas, and IPython, but the free version does not come  
with scikit-learn. If you are part of an academic, degree-granting institution,  
you can request an academic license and get free access to the paid  
subscription version of Enthought Canopy. Enthought Canopy is available for  
Python 2.7.x, and works on Mac OS, Windows, and Linux.  
Python(x,y)  
A free Python distribution for scientific computing, specifically for Windows.  
Python(x,y) comes with NumPy, SciPy, matplotlib, pandas, IPython, and  
scikit-learn.  
If you already have a Python installation set up, you can use pip to install all of  
these packages:  
$ pip install numpy scipy matplotlib ipython scikit-learn pandas pillow  
For the tree visualizations in Chapter 2, you also need the graphviz packages; see  
the accompanying code for instructions. For Chapter 7, you will also need the nltk  
and spacy libraries; see the instructions in that chapter.  
1.4 Essential Libraries and Tools  
Understanding what scikit-learn is and how to use it is important, but there are  
a few other libraries that will enhance your experience. scikit-learn is built on  
top of the NumPy and SciPy scientific Python libraries. In addition to NumPy and  
SciPy, we will be using pandas and matplotlib. We will also introduce the Jupyter  
Notebook, which is a browser-based interactive programming environment.  
Briefly, here is what you should know about these tools in order to get the most out  
of scikit-learn.1  
1.4.1 Jupyter Notebook  
The Jupyter Notebook is an interactive environment for running code in the  
browser. It is a great tool for exploratory data analysis and is widely used by data  
scientists. While the Jupyter Notebook supports many programming languages, we  
only need the Python support. The Jupyter Notebook makes it easy to incorporate  
code, text, and images, and all of this book was in fact written as a Jupyter  
Notebook. All of the code examples we include can be downloaded from  
https://github.com/amueller/introduction\_to\_ml\_with\_python.  
1.4.2 NumPy  
NumPy is one of the fundamental packages for scientific computing in Python. It  
contains functionality for multidimensional arrays, high-level mathematical  
functions such as linear algebra operations and the Fourier transform, and  
pseudorandom number generators.  
In scikit-learn, the NumPy array is the fundamental data structure. scikitlearn takes in data in the form of NumPy arrays. Any data you’re using will have  
to be converted to a NumPy array. The core functionality of NumPy is the ndarray  
class, a multidimensional (n-dimensional) array. All elements of the array must be  
of the same type. A NumPy array looks like this:  
In[1]:  
import numpy as np  
x = np.array([[1, 2, 3], [4, 5, 6]])  
print("x:\n{}".format(x))  
Out[1]:  
x:  
[[1 2 3]  
[4 5 6]]  
We will be using NumPy a lot in this book, and we will refer to objects of the NumPy  
ndarray class as “NumPy arrays” or just “arrays.”  
1.4.3 SciPy  
SciPy is a collection of functions for scientific computing in Python. It provides,  
among other functionality, advanced linear algebra routines, mathematical  
function optimization, signal processing, special mathematical functions, and  
statistical distributions. scikit-learn draws from SciPy’s collection of functions  
for implementing its algorithms. The most important part of SciPy for us is  
scipy.sparse: this provides sparse matrices, which are another representation that  
is used for data in scikit-learn. Sparse matrices are used whenever we want to  
store a 2D array that contains mostly zeros:  
In[2]:  
from scipy import sparse  
# Create a 2D NumPy array with a diagonal of ones, and zeros everywhere else  
eye = np.eye(4)  
print("NumPy array:\n", eye)  
Out[2]:  
NumPy array:  
[[1. 0. 0. 0.]  
[0. 1. 0. 0.]  
[0. 0. 1. 0.]  
[0. 0. 0. 1.]]  
In[3]:  
# Convert the NumPy array to a SciPy sparse matrix in CSR format  
# Only the nonzero entries are stored  
sparse\_matrix = sparse.csr\_matrix(eye)  
print("\nSciPy sparse CSR matrix:\n", sparse\_matrix)  
Out[3]:  
SciPy sparse CSR matrix:  
(0,0) 1.0  
(1, 1) 1.0  
(2, 2) 1.0  
(3, 3) 1.0

|  |  |  |  |
| --- | --- | --- | --- |
| Usually | not possible | create dense representations of sparse | (as they |
| to | data |  |  |

it is would not fit into memory), so we need to create sparse representations directly.  
Here is a way to create the same sparse matrix as before, using the COO format:  
In[4]:  
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("COO representation:\n", eye\_coo)  
Out[4]:  
COO representation:  
(0,0) 1.0  
(1, 1) 1.0  
(2, 2) 1.0  
(3, 3) 1.0  
More details on SciPy sparse matrices can be found in the SciPy Lecture Notes.  
1.4.4 matplotlib  
matplotlib is the primary scientific plotting library in Python. It provides  
functions for making publication-quality visualizations such as line charts,  
histograms, scatter plots, and so on. Visualizing your data and different aspects of  
your analysis can give you important insights, and we will be using matplotlib for  
all our visualizations. When working inside the Jupyter Notebook, you can show  
figures directly in the browser by using the %matplotlib notebook and  
%matplotlib inline commands. We recommend using %matplotlib notebook,  
which provides an interactive environment (though we are using %matplotlib  
inline to produce this book). For example, this code produces the plot in Figure 1-  
1:  
In[5]:  
%matplotlib inline  
import matplotlib.pyplot as plt  
# Generate a sequence of numbers from -10 to 10 with 100 steps in between  
x = np.linspace(-10, 10, 100)  
# Create a second array using sine  
y = np.sin(x)  
# The plot function makes a line chart of one array against another  
plt.plot(x, y, marker="x")  
1.4.5 pandas Figure 1-1. Simple line plot of the sine function using matplotlib  
pandas is a Python library for data wrangling and analysis. It is built around a data  
structure called the DataFrame that is modeled after the R DataFrame. Simply put,  
a pandas DataFrame is a table, similar to an Excel spreadsheet. pandas provides a  
great range of methods to modify and operate on this table; in particular, it allows  
SQL-like queries and joins of tables. In contrast to NumPy, which requires that all  
entries in an array be of the same type, pandas allows each column to have a  
separate type (for example, integers, dates, floating-point numbers, and strings).  
Another valuable tool provided by pandas is its ability to ingest from a great  
variety of file formats and databases, like SQL, Excel files, and comma-separated  
values (CSV) files. Going into detail about the functionality of pandas is out of the  
scope of this book. However, Python for Data Analysis by Wes McKinney (O’Reilly,  
2012) provides a great guide. Here is a small example of creating a DataFrame  
using a dictionary:  
In[6]:  
import pandas as pd  
# create a simple dataset of people  
data = {'Name': ["John", "Anna", "Peter", "Linda"],  
}  
'Location' : ["New York", "Paris", "Berlin", "London"],  
'Age' : [24, 13, 53, 33]  
data\_pandas = pd.DataFrame(data)  
# IPython.display allows "pretty printing" of dataframes  
# in the Jupyter notebook  
display(data\_pandas)  
This produces the following output:  
Age Location Name  
0 24 New York John  
1 13 Paris Anna  
2 53 Berlin Peter  
3 33 London Linda  
There are several possible ways to query this table . For example :  
In [7 ] :  
# Select all rows that have an age column greater than 30  
display ( data\_pandas [ data\_pandas.Age > 30 ] )  
This produces the following result:  
Age Location Name  
2 53 Berlin Peter  
3 33 London Linda  
1.4.6 mglearn  
This book comes with accompanying code, which you can find on  
https://github.com/amueller/introduction\_to\_ml\_with\_python . The accompanying code  
includes not only all the examples shown in this book, but also the mglearn library .  
This is a library of utility functions we wrote for this book, so that we don't clutter  
up our code listings with details of plotting and data loading. Ifyou're interested,  
you can look up all the functions in the repository, but the details of the mglearn  
module are not really important to the material in this book. Ifyou see a call to  
mglearn in the code, it is usually a way to make a pretty picture quickly, or to get  
our hands on some interesting data. Ifyou run the notebooks published on GitHub,  
the mglearn package is already in the right place and you don't have to worry  
about it . If you want to call mglearn functions from any other place, the easiest way  
to install it is by calling pip install mglearn .  
Note  
Throughout the book we make ample use of NumPy, matplotlib and pandas . All  
the code will assume the following imports:  
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
import mglearn  
from IPython.display import display  
We also assume that you will run the code in a Jupyter Notebook with the  
%matplotlib notebook or %matplotlib inline magic enabled to show plots. If you  
are not using the notebook or these magic commands, you will have to call  
plt.show to actually show any of the figures.  
1.5 Python 2 Versus Python 3  
There are two major versions of Python that are widely used at the moment:  
Python 2 (more precisely, 2.7) and Python 3 (with the latest release being 3.7 at the  
time of writing). This sometimes leads to some confusion. Python 2 is no longer  
actively developed, but because Python 3 contains major changes, Python 2 code  
usually does not run on Python 3. If you are new to Python, or are starting a new  
project from scratch, we highly recommend using the latest version of Python 3. If  
you have a large codebase that you rely on that is written for Python 2, you are  
excused from upgrading for now. However, you should try to migrate to Python 3  
as soon as possible. When writing any new code, it is for the most part quite easy to  
write code that runs under Python 2 and Python 3. If you don’t have to interface  
with legacy software, you should definitely use Python 3. All the code in this book  
is written in a way that works for both versions. However, the exact output might  
differ slightly under Python 2. You should also note that many packages such as  
matplotlib, numpy, and scikit-learn are no longer releasing new features under  
Python 2.7; you need to upgrade to Python 3.7 to get the benefit of the  
improvements that come with newer versions.  
2  
1.6 Versions Used in this Book  
We are using the following versions of the previously mentioned libraries in this  
book:  
In[8]:  
import sys  
print("Python version:", sys.version)  
import pandas as pd  
print("pandas version:", pd.\_\_version\_\_)  
import matplotlib  
print("matplotlib version:", matplotlib.\_\_version\_\_)  
import numpy as np  
print("NumPy version:", np.\_\_version\_\_)  
import scipy as sp  
print("SciPy version:", sp.\_\_version\_\_)  
import IPython  
print("IPython version:", IPython.\_\_version\_\_)  
import sklearn  
print("scikit-learn version:", sklearn.\_\_version\_\_)  
Out[8]:  
Python version: 3.7.0 (default, Jun 28 2018, 13:15:42)  
[GCC 7.2.0]  
pandas version: 0.23.4  
matplotlib version: 3.0.0  
NumPy version: 1.15.2  
SciPy version: 1.1.0  
IPython version: 6.4.0  
scikit-learn version: 0.20.0  
While it is not important to match these versions exactly, you should have a  
version of scikit-learn that is as least as recent as the one we used.  
Note  
When using the code in this book, you might sometimes see DeprecationWarnings or  
FutureWarnings from scikit-learn. These inform you about behavior in scikitlearn that will change in the future or will be removed. While going through this  
book, you can safely ignore these. If you are running a machine learning algorithm  
in production, you should carefully consider each warning, as they might inform  
you about functionality being removed in the future or outcomes of predictions  
changing.  
Now that we have everything set up, let’s dive into our first application of machine  
learning.  
1.7 A First Application: Classifying Iris Species  
In this section, we will go through a simple machine learning application and  
create our first model. In the process, we will introduce some core concepts and  
terms.  
Let’s assume that a hobby botanist is interested in distinguishing the species of  
some iris flowers that she has found. She has collected some measurements  
associated with each iris: the length and width of the petals and the length and  
width of the sepals, all measured in centimeters (see Figure 1-2).  
She also has the measurements of some irises that have been previously identified  
by an expert botanist as belonging to the species setosa, versicolor, or virginica. For  
these measurements, she can be certain of which species each iris belongs to. Let’s  
assume that these are the only species our hobby botanist will encounter in the  
wild.  
Our goal is to build a machine learning model that can learn from the  
measurements of these irises whose species is known, so that we can predict the  
species for a new iris .  
Petal  
Sepal \_  
Figure 1-2. Parts of the iris flower  
Because we have measurements for which we know the correct species of iris, this  
is a supervised learning problem. In this problem, we want to predict one of  
several options (the species of iris). This is an example of a classification problem.  
The possible outputs (different species of irises) are called classes. Every iris in the  
dataset belongs to one of three classes, so this problem is a three- class  
classification problem .  
The desired output for a single data point (an iris) is the species of this flower. For  
a particular data point, the species it belongs to is called its label.  
1.7.1 Meet the Data  
The data we will use for this example is the Iris dataset, a classical dataset in  
machine learning and statistics. It is included in scikit - learn in the dataset  
module. We can load it by calling the load\_iris function:  
In [ 9 ] :  
from sklearn.datasets import load\_iris  
iris\_dataset = load\_iris()  
The iris object that is returned by load\_iris is a Bunch object, which is very  
similar to a dictionary. It contains keys and values:  
In[10]:  
print("Keys of iris\_dataset:\n", iris\_dataset.keys())  
Out[10]:  
Keys of iris\_dataset:  
dict\_keys(['data', 'target', 'target\_names', 'DESCR', 'feature\_names',  
'filename'])  
The value of the key DESCR is a short description of the dataset. We show the  
beginning of the description here (feel free to look up the rest yourself):  
In[11]:  
print(iris\_dataset['DESCR'][:193] + "\n...")  
Out[11]:  
Iris Plants Database  
====================  
Notes  
----  
Data Set Characteristics:  
:Number of Instances: 150 (50 in each of three classes)  
:Number of Attributes: 4 numeric, predictive att  
...  
----  
The value of the key target\_names is an array of strings, containing the species of  
flower that we want to predict:  
In[12]:  
print("Target names:", iris\_dataset['target\_names'])  
Out[12]:  
Target names: ['setosa' 'versicolor' 'virginica']  
The value of feature\_names is a list of strings, giving the description of each  
feature:  
In[13]:  
print("Feature names:\n", iris\_dataset['feature\_names'])  
Out[13]:  
Feature names:  
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',  
'petal width (cm)']  
The data itself is contained in the target and data fields. data contains the  
numeric measurements of sepal length, sepal width, petal length, and petal width  
in a NumPy array:  
In[14]:  
print("Type of data:", type(iris\_dataset['data']))  
Out[14]:  
Type of data: <class 'numpy.ndarray'>  
The rows in the data array correspond to flowers, while the columns represent the  
four measurements that were taken for each flower:  
In[15]:  
print("Shape of data:", iris\_dataset['data'].shape)  
Out[15]:  
Shape of data: (150, 4)  
We see that the array contains measurements for 150 different flowers. Remember  
that the individual items are called samples in machine learning, and their  
properties are called features. The shape of the data array is the number of samples  
times the number of features. This is a convention in scikit-learn, and your data  
will always be assumed to be in this shape. Here are the feature values for the first  
five samples:  
In[16]:  
print("First five rows of data:\n", iris\_dataset['data'][:5])  
Out[16]:

|  |  |  |  |
| --- | --- | --- | --- |
| First five | rows | of | data: |
| [[5.1 3.5 | 1.4 | 0.2] |  |

|  |  |  |  |
| --- | --- | --- | --- |
| [4.9  [4.7 | 3.  3.2 | 1.4  1.3 | 0.2] 0.2] |

[4.6 3.1 1.5 0.2]  
[5. 3.6 1.4 0.2]]  
From this data, we can see that all of the first five flowers have a petal width of 0.2  
cm and that the first flower has the longest sepal, at 5.1 cm.  
The target array contains the species of each of the flowers that were measured,  
also as a NumPy array:  
In[17]:  
print("Type of target:", type(iris\_dataset['target']))  
Out[17]:  
Type of target: <class 'numpy.ndarray'>  
target is a one-dimensional array, with one entry per flower:  
In[18]:  
print("Shape of target:", iris\_dataset['target'].shape)  
Out[18]:  
Shape of target: (150,)  
The species are encoded as integers from 0 to 2:  
In[19]:  
print ( " Target : \n " , iris\_dataset [ ' target ' ] )  
Out[ 19 ] :  
Target :  
0000000000000000000000000000000000000 ]  
0000000000000 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
1111111111111111111111111 1 2 2 2 2 2 2 2 2 2 2 2  
2222222222222222222222222222222222222  
2 2 ]  
The meanings of the numbers are given by the iris [ ' target\_names ' ] array: 0  
means setosa, 1 means versicolor, and 2 means virginica.  
1.7.2 Measuring Success : Training and Testing Data  
We want to build a machine learning model from this data that can predict the  
species of iris for a new set of measurements . But before we can apply our model to  
new measurements, we need to know whether it actually works-that is, whether  
we should trust its predictions.  
Unfortunately, we cannot use the data we used to build the model to evaluate it .  
This is because our model can always simply remember the whole training set , and  
will therefore always predict the correct label for any point in the training set .  
This "remembering" does not indicate to us whether our model will generalize well  
(in other words, whether it will also perform well on new data) .  
To assess the model's performance , we show it new data (data that it hasn't seen  
before) for which we have labels. This is usually done by splitting the labeled data  
we have collected (here, our 150 flower measurements) into two parts. One part of  
the data is used to build our machine learning model, and is called the training data  
or training set . The rest of the data will be used to assess how well the model works;  
this is called the test data, test set, or hold-out set.  
scikit - learn contains a function that shuffles the dataset and splits it for you: the  
train\_test\_split function. This function extracts 75% of the rows in the data as  
the training set , together with the corresponding labels for this data. The  
remaining 25% of the data, together with the remaining labels, is declared as the  
test set . Deciding how much data you want to put into the training and the test set  
respectively is somewhat arbitrary, but using a test set containing 25% of the data  
is a good rule of thumb .  
In scikit - learn , data is usually denoted with a capital X, while labels are denoted  
by a lowercase y. This is inspired by the standard formulation f(x)=y in  
mathematics, where x is the input to a function and y is the output. Following more  
conventions from mathematics, we use a capital X because the data is a two  
dimensional array (a matrix) and a lowercase y because the target is a onedimensional array (a vector).  
Let’s call train\_test\_split on our data and assign the outputs using this  
nomenclature:  
In[20]:  
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)  
Before making the split, the train\_test\_split function shuffles the dataset using  
a pseudorandom number generator. If we just took the last 25% of the data as a test  
set, all the data points would have the label 2, as the data points are sorted by the  
label (see the output for iris['target'] shown earlier). Using a test set  
containing only one of the three classes would not tell us much about how well our  
model generalizes, so we shuffle our data to make sure the test data contains data  
from all classes.  
To make sure that we will get the same output if we run the same function several  
times, we provide the pseudorandom number generator with a fixed seed using  
the random\_state parameter. This will make the outcome deterministic, so this  
line will always have the same outcome. We will always fix the random\_state in  
this way when using randomized procedures in this book.  
The output of the train\_test\_split function is X\_train, X\_test, y\_train, and  
y\_test, which are all NumPy arrays. X\_train contains 75% of the rows of the  
dataset, and X\_test contains the remaining 25%:  
In[21]:  
print("X\_train shape:", X\_train.shape)  
print("y\_train shape:", y\_train.shape)  
Out[21]:  
X\_train shape: (112, 4)  
y\_train shape: (112,)  
In[22]:  
print("X\_test shape:", X\_test.shape)  
print("y\_test shape:", y\_test.shape)  
Out[22]:  
X\_test shape: (38,4)  
y\_test shape: (38,)  
1.7.3 First Things First: Look at Your Data  
Before building a machine learning model it is often a good idea to inspect the  
data, to see if the task is easily solvable without machine learning, or if the desired  
information might not be contained in the data.  
Additionally, inspecting your data is a good way to find abnormalities and  
peculiarities. Maybe some of your irises were measured using inches and not  
centimeters, for example. In the real world, inconsistencies in the data and  
unexpected measurements are very common.  
One of the best ways to inspect data is to visualize it. One way to do this is by using  
a scatter plot. A scatter plot of the data puts one feature along the x-axis and  
another along the y-axis, and draws a dot for each data point. Unfortunately,  
computer screens have only two dimensions, which allows us to plot only two (or  
maybe three) features at a time. It is difficult to plot datasets with more than  
three features this way. One way around this problem is to do a pair plot, which  
looks at all possible pairs of features. If you have a small number of features, such  
as the four we have here, this is quite reasonable. You should keep in mind,  
however, that a pair plot does not show the interaction of all of features at once, so  
some interesting aspects of the data may not be revealed when visualizing it this  
way.  
Figure 1-3 is a pair plot of the features in the training set. The data points are  
colored according to the species the iris belongs to. To create the plot, we first  
convert the NumPy array into a pandas DataFrame. pandas has a function to create  
pair plots called scatter\_matrix. The diagonal of this matrix is filled with  
histograms of each feature:  
In[23]:  
# create dataframe from data in X\_train  
# label the columns using the strings in iris\_dataset.feature\_names  
iris\_dataframe = pd.DataFrame(X\_train, columns=iris\_dataset.feature\_names)  
# create a scatter matrix from the dataframe, color by y\_train  
pd.plotting.scatter\_matrix(iris\_dataframe, c=y\_train, figsize=(15, 15),  
marker='o', hist\_kwds={'bins': 20}, s=60,  
alpha=.8, cmap=mglearn.cm3)  
Figure 1-3. Pair plot of the Iris dataset, colored by class label  
From the plots, we can see that the three classes seem to be relatively well  
separated using the sepal and petal measurements. This means that a machine  
learning model will likely be able to learn to separate them.  
1.7.4 Building Your First Model: k-Nearest Neighbors  
Now we can start building the actual machine learning model. There are many  
classification algorithms in scikit-learn that we could use. Here we will use a knearest neighbors classifier, which is easy to understand. Building this model only  
consists of storing the training set. To make a prediction for a new data point, the  
algorithm finds the point in the training set that is closest to the new point. Then  
it assigns the label of this training point to the new data point.  
The k in k-nearest neighbors signifies that instead of using only the closest  
neighbor to the new data point, we can consider any fixed number k of neighbors  
in the training (for example, the closest three or five neighbors). Then, we can  
make a prediction using the majority class among these neighbors. We will go into  
more detail about this in Chapter 2; for now, we’ll use only a single neighbor.  
All machine learning models in scikit-learn are implemented in their own  
classes, which are called Estimator classes. The k-nearest neighbors classification  
algorithm is implemented in the KNeighborsClassifier class in the neighbors  
module. Before we can use the model, we need to instantiate the class into an  
object. This is when we will set any parameters of the model. The most important  
parameter of KNeighborsClassifier is the number of neighbors, which we will set  
to 1:  
In[24]:  
from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier(n\_neighbors=1)  
The knn object encapsulates the algorithm that will be used to build the model  
from the training data, as well the algorithm to make predictions on new data  
points. It will also hold the information that the algorithm has extracted from the  
training data. In the case of KNeighborsClassifier, it will just store the training  
set.  
To build the model on the training set, we call the fit method of the knn object,  
which takes as arguments the NumPy array X\_train containing the training data  
and the NumPy array y\_train of the corresponding training labels:  
In[25]:  
knn.fit(X\_train, y\_train)  
Out[25]:  
KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',  
metric\_params=None, n\_jobs=None, n\_neighbors=1, p=2,  
weights='uniform')  
The fit method returns the knn object itself (and modifies it in place), so we get a  
string representation of our classifier. The representation shows us which  
parameters were used in creating the model. Nearly all of them are the default  
values, but you can also find n\_neighbors=1, which is the parameter that we  
passed. Most models in scikit-learn have many parameters, but the majority of  
them are either speed optimizations or for very special use cases. You don’t have  
to worry about the other parameters shown in this representation. Printing a  
scikit-learn model can yield very long strings, but don’t be intimidated by these.  
We will cover all the important parameters in Chapter 2. In the remainder of this  
book, we will not usually show the output of fit because it doesn’t contain any new  
information.  
1.7.5 Making Predictions  
We can now make predictions using this model on new data for which we might not  
know the correct labels. Imagine we found an iris in the wild with a sepal length of  
5 cm, a sepal width of 2.9 cm, a petal length of 1 cm, and a petal width of 0.2 cm.  
What species of iris would this be? We can put this data into a NumPy array, again  
by calculating the shape—that is, the number of samples (1) multiplied by the  
number of features (4):  
In[26]:  
X\_new = np.array([[5, 2.9, 1, 0.2]])  
print("X\_new.shape:", X\_new.shape)  
Out[26]:  
X\_new.shape: (1, 4)  
Note that we made the measurements of this single flower into a row in a twodimensional NumPy array, as scikit-learn always expects two-dimensional arrays  
for the data.  
To make a prediction, we call the predict method of the knn object:  
In[27]:  
prediction = knn.predict(X\_new)  
print("Prediction:", prediction)  
print("Predicted target name:",  
iris\_dataset['target\_names'][prediction])  
Out[27]:  
Prediction: [0]  
Predicted target name: ['setosa']  
Our model predicts that this new iris belongs to the class 0, meaning its species is  
setosa. But how do we know whether we can trust our model? We don’t know the  
correct species of this sample, which is the whole point of building the model!  
1.7.6 Evaluating the Model  
This is where the test set that we created earlier comes in. This data was not used  
to build the model, but we do know what the correct species is for each iris in the  
test set .  
Therefore , we can make a prediction for each iris in the test data and compare it  
against its label (the known species) . We can measure how well the model works by  
computing the accuracy, which is the fraction of flowers for which the right species  
was predicted:  
In [ 28 ] :  
y\_pred = knn.predict (X\_test )  
print ( " Test set predictions : \ n " , y\_pred )  
Out [ 28 ] :  
Test set predictions :  
[ 210202 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 22102 ]  
In [ 29 ] :  
print( " Test set score : { : .2f} " . format ( np.mean ( y\_pred == y\_test ) ) )  
Out [ 29 ] :  
Test set score : 0.97  
We can also use the score method of the knn object, which will compute the test  
set accuracy for us :  
In [ 30 ] :  
print( " Test set score : { : .2f} " . format ( knn.score (X\_test , y\_test ) ) )  
Out [ 30 ] :  
Test set score : 0.97  
For this model, the test set accuracy is about 0.97, which means we made the right  
prediction for 97% of the irises in the test set . Under some mathematical  
assumptions, this means that we can expect our model to be correct 97% of the  
time for new irises. For our hobby botanist application, this high level of accuracy  
means that our model may be trustworthy enough to use. In later chapters we will  
discuss how we can improve performance, and what caveats there are in tuning a  
model.  
1.8 Summary and Outlook  
Let’s summarize what we learned in this chapter. We started with a brief  
introduction to machine learning and its applications, then discussed the  
distinction between supervised and unsupervised learning and gave an overview  
of the tools we’ll be using in this book. Then, we formulated the task of predicting  
which species of iris a particular flower belongs to by using physical measurements  
of the flower. We used a dataset of measurements that was annotated by an expert  
with the correct species to build our model, making this a supervised learning  
task. There were three possible species, setosa, versicolor, or virginica, which made  
the task a three-class classification problem. The possible species are called classes  
in the classification problem, and the species of a single iris is called its label.  
The Iris dataset consists of two NumPy arrays: one containing the data, which is  
referred to as X in scikit-learn, and one containing the correct or desired  
outputs, which is called y. The array X is a two-dimensional array of features, with  
one row per data point and one column per feature. The array y is a onedimensional array, which here contains one class label, an integer ranging from 0  
to 2, for each of the samples.  
We split our dataset into a training set, to build our model, and a test set, to evaluate  
how well our model will generalize to new, previously unseen data.  
We chose the k-nearest neighbors classification algorithm, which makes  
predictions for a new data point by considering its closest neighbor(s) in the  
training set. This is implemented in the KNeighborsClassifier class, which  
contains the algorithm that builds the model as well as the algorithm that makes a  
prediction using the model. We instantiated the class, setting parameters. Then we  
built the model by calling the fit method, passing the training data (X\_train) and  
training outputs (y\_train) as parameters. We evaluated the model using the score  
method, which computes the accuracy of the model. We applied the score method  
to the test set data and the test set labels and found that our model is about 97%  
accurate, meaning it is correct 97% of the time on the test set.  
This gave us the confidence to apply the model to new data (in our example, new  
flower measurements) and trust that the model will be correct about 97% of the  
time.  
Here is a summary of the code needed for the whole training and evaluation  
procedure:  
In[31]:  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
iris\_dataset['data'], iris\_dataset['target'], random\_state=0)  
knn.fit(X\_train, = KNeighborsClassifier(n\_neighbors=1)  
y\_train)  
print("Test set score: {:.2f}".format(knn.score(X\_test, y\_test)))  
Out[31]:  
Test set score: 0.97  
This snippet contains the core code for applying any machine learning algorithm  
using scikit-learn. The fit, predict, and score methods are the common  
interface to supervised models in scikit-learn, and with the concepts introduced  
in this chapter, you can apply these models to many machine learning tasks. In the  
next chapter, we will go into more depth about the different kinds of supervised  
models in scikit-learn and how to apply them successfully.  
1 If you are unfamiliar with NumPy or matplotlib, we recommend reading the first  
chapter of the SciPy Lecture Notes.  
2 The six package can be very handy for that.