Python Machine Learning
Third Edition

Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2

Sebastian Raschka

Vahid Mirjalili

Packt

BIRMINGHAM - MUMBAI

Python Machine Learning
Third Edition

Acquisition Editor: Jonathan Malysiak
Acquisition Editor – Peer Reviews: Suresh Jain
Content Development Editors: Joanne Lovell, Chris Nelson
Technical Editor: Saby D'silva
Project Editor: Radhika Atitkar
Proofreader: Safis Editing
Indexer: Tejal Daruwale Soni
Presentation Designer: Sandip Tadge

First published: September 2015

Second edition: September 2017

Third edition: December 2019

Production reference: 1091219

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78995-575-0

www.packt.com

packt.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Learn better with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.Packt.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.Packt.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributors

About the authors

Sebastian Raschka received his doctorate from Michigan State University, where he focused on developing methods at the intersection of computational biology and machine learning. In the summer of 2018, he joined the University of Wisconsin-Madison as Assistant Professor of Statistics. His research activities include the development of new deep learning architectures to solve problems in the field of biometrics.

Sebastian has many years of experience with coding in Python and has given several seminars on the practical applications of data science, machine learning, and deep learning over the years, including a machine learning tutorial at SciPy, the leading conference for scientific computing in Python.

Among Sebastian's achievements is his book *Python Machine Learning*, which is a bestselling title at Packt and on Amazon.com. The book received the ACM Best of Computing award in 2016 and was translated into many different languages, including German, Korean, Chinese, Japanese, Russian, Polish, and Italian.

In his free time, Sebastian loves to contribute to open source projects, and methods that he implemented are now successfully used in machine learning competitions such as Kaggle.

I would like to take this opportunity to thank the great Python community and the developers of open source packages who helped me create the perfect environment for scientific research and data science. Also, I want to thank my parents, who always encouraged and supported me in pursuing the path and career that I was so passionate about.

Special thanks to the core developers of scikit-learn and TensorFlow. As a contributor and user, I had the pleasure of working with great people who are not only very knowledgeable when it comes to machine learning and deep learning but are also excellent programmers.

Vahid Mirjalili obtained his PhD in mechanical engineering working on novel methods for large-scale, computational simulations of molecular structures at Michigan State University. Being passionate about the field of machine learning, he joined the iPRoBe lab at Michigan State University, where he worked on applying machine learning in the computer vision and biometrics domains. After several productive years at the iPRoBe lab and many years in academia, Vahid recently joined **3M** Company as a research scientist, where he can use his expertise and apply state-of-the-art machine learning and deep learning techniques to solve real-world problems in various applications to make life better.

I would like to thank my wife, Taban Eslami, who has been very supportive and encouraged me on my career path. Also, special thanks to my advisors, Nikolai Priezjev, Michael Feig, and Arun Ross, for supporting me during my PhD studies, as well as my professors, Vishnu Boddeti, Leslie Kuhn, and Xiaoming Liu, who have taught me so much and encouraged me to pursue my passion.

About the reviewers

Raghav Bali is a senior data scientist at one of the world's largest healthcare organizations. His work involves the research and development of enterprise-level solutions based on machine learning, deep learning, and natural language processing for healthcare- and insurance-related use cases. In his previous role at Intel, he was involved in enabling proactive data-driven IT initiatives using natural language processing, deep learning, and traditional statistical methods. He has also worked in the finance domain with American Express, solving digital engagement and customer retention use cases.

Raghav has also authored multiple books with leading publishers, the most recent being on the latest advancements in transfer learning research.

Raghav has a master's degree (gold medalist) in information technology from the International Institute of Information Technology, Bangalore. Raghav loves reading and is a shutterbug, capturing moments when he isn't busy solving problems.

Motaz Saad holds a PhD in computer science from the University of Lorraine. He loves data and he likes to play with it. He has over 10 years of professional experience in natural language processing, computational linguistics, data science, and machine learning. He currently works as an assistant professor at the faculty of Information Technology, IUG.

Table of Contents

Preface

Through exposure to the news and social media, you are probably very familiar with the fact that machine learning has become one of the most exciting technologies of our time. Large companies, such as Google, Facebook, Apple, Amazon, and IBM, heavily invest in machine learning research and applications for good reason. While it may seem that machine learning has become the buzzword of our age, it is certainly not just hype. This exciting field opens up the way to new possibilities and has become indispensable to our daily lives. Think about talking to the voice assistant on our smartphones, recommending the right product for our customers, preventing credit card fraud, filtering out spam from our email inboxes, and detecting and diagnosing medical diseases; the list goes on and on.

Get started with machine learning

If you want to become a machine learning practitioner or a better problem solver, or maybe you are even considering a career in machine learning research, then this book is for you! For a novice, the theoretical concepts behind machine learning can be quite overwhelming, but the many practical books that have been published in recent years will help you to get started in machine learning by implementing powerful learning algorithms.

Practice and theory

Being exposed to practical code examples and working through example applications of machine learning are great ways to dive into this field. Also, concrete examples help to illustrate the broader concepts by putting the learned material directly into action. However, remember that with great power comes great responsibility!

In addition to offering hands-on experience with machine learning using the Python programming language and Python-based machine learning libraries, this book introduces the mathematical concepts behind machine learning algorithms, which are essential for using machine learning successfully. Thus, this book is different from a purely practical book; this is a book that discusses the necessary details regarding machine learning concepts and offers intuitive, yet informative, explanations on how machine learning algorithms work, how to use them, and, most importantly, how to avoid the most common pitfalls.

Why Python?

Before we dive deeper into the machine learning field, let's answer your most important question: "Why Python?" The answer is simple: it is powerful, yet very accessible. Python has become the most popular programming language for data science because it allows us to forget the tedious parts of programming and offers us an environment where we can quickly jot down our ideas and put concepts directly into action.

Explore the machine learning field

If you type "machine learning" as a search term into Google Scholar, it will return an overwhelmingly large number—3,250,000 publications. Of course, we cannot discuss all the nitty-gritty details of all the different algorithms and applications that have emerged in the last 60 years. However, in this book, we will embark on an exciting journey, covering all the essential topics and concepts to give you a head start in this field. If you find that your thirst for knowledge is not satisfied, you can use the many useful resources that this book references to follow up on the essential breakthroughs in this field.

We, the authors, can truly say that the study of machine learning made us better scientists, thinkers, and problem solvers. In this book, we want to share this knowledge with you. Knowledge is gained by learning, the key to this is enthusiasm, and the real mastery of skills can only be achieved through practice.

The road ahead may be bumpy on occasions, and some topics may be more challenging than others, but we hope that you will embrace this opportunity and focus on the reward. Remember that we are on this journey together, and throughout this book, we will add many powerful techniques to your arsenal that will help you to solve even the toughest problems the data-driven way.

Who this book is for

If you have already studied machine learning theory in detail, this book will show you how to put your knowledge into practice. If you have used machine learning techniques before and want to gain more insight into how machine learning actually works, this book is also for you.

Don't worry if you are completely new to the machine learning field; you have even more reason to be excited! This is a promise that machine learning will change the way you think about the problems you want to solve and show you how to tackle them by unlocking the power of data. If you want to find out how to use Python to start answering critical questions about your data, pick up *Python Machine Learning*. Whether you want to start from scratch or extend your data science knowledge, this is an essential and unmissable resource.

What this book covers

Chapter 1, *Giving Computers the Ability to Learn from Data*, introduces the main subareas of machine learning used to tackle various problem tasks. In addition, it discusses the essential steps for creating a typical machine learning model-building pipeline that will guide us through the following chapters.

Chapter 2, *Training Simple Machine Learning Algorithms for Classification*, goes back to the origin of machine learning and introduces binary perceptron classifiers and adaptive linear neurons. This chapter is a gentle introduction to the fundamentals of pattern classification and focuses on the interplay of optimization algorithms and machine learning.

Chapter 3, *A Tour of Machine Learning Classifiers Using scikit-learn*, describes the essential machine learning algorithms for classification and provides practical examples using one of the most popular and comprehensive open source machine learning libraries, scikit-learn.

Chapter 4, *Building Good Training Datasets – Data Preprocessing*, discusses how to deal with the most common problems in unprocessed datasets, such as missing data. It also discusses several approaches to identify the most informative features in datasets and how to prepare variables of different types as proper inputs for machine learning algorithms.

Chapter 5, *Compressing Data via Dimensionality Reduction*, describes the essential techniques to reduce the number of features in a dataset to smaller sets, while retaining most of their useful and discriminatory information. It also discusses the standard approach to dimensionality reduction via principal component analysis and compares it to supervised and nonlinear transformation techniques.

Chapter 6, *Learning Best Practices for Model Evaluation and Hyperparameter Tuning*, discusses the dos and don'ts for estimating the performance of predictive models. Moreover, it discusses different metrics for measuring the performance of our models and techniques for fine-tuning machine learning algorithms.

Chapter 7, *Combining Different Models for Ensemble Learning*, introduces the different concepts of combining multiple learning algorithms effectively. It explores how to build ensembles of experts to overcome the weaknesses of individual learners, resulting in more accurate and reliable predictions.

Chapter 8, *Applying Machine Learning to Sentiment Analysis*, discusses the essential steps for transforming textual data into meaningful representations for machine learning algorithms to predict the opinions of people based on their writing.

Chapter 9, *Embedding a Machine Learning Model into a Web Application*, continues with the predictive model from the previous chapter and walks through the essential steps of developing web applications with embedded machine learning models.

Chapter 10, *Predicting Continuous Target Variables with Regression Analysis*, discusses the essential techniques for modeling linear relationships between target and response variables to make predictions on a continuous scale. After introducing different linear models, it also talks about polynomial regression and tree-based approaches.

Chapter 11, *Working with Unlabeled Data – Clustering Analysis*, shifts the focus to a different subarea of machine learning, unsupervised learning. It covers algorithms from three fundamental families of clustering algorithms that find groups of objects that share a certain degree of similarity.

Chapter 12, *Implementing a Multilayer Artificial Neural Network from Scratch*, extends the concept of gradient-based optimization, which we first introduced in *Chapter 2*, *Training Simple Machine Learning Algorithms for Classification*. In this chapter, we will build powerful, multilayer **neural networks** (**NNs**) based on the popular backpropagation algorithm in Python.

Chapter 13, *Parallelizing Neural Network Training with TensorFlow*, builds upon the knowledge from the previous chapter to provide a practical guide for training NNs more efficiently. The focus of this chapter is on TensorFlow 2.0, an open source Python library that allows us to utilize multiple cores of modern graphics processing units (GPUs) and construct deep NNs from common building blocks via the user-friendly Keras API.

Chapter 14, *Going Deeper – The Mechanics of TensorFlow*, picks up where the previous chapter left off and introduces the more advanced concepts and functionality of TensorFlow 2.0. TensorFlow is an extraordinarily vast and sophisticated library, and this chapter walks through concepts such as compiling code into a static graph for faster execution and defining trainable model parameters. In addition, this chapter provides additional hands-on experience of training deep neural networks using TensorFlow's Keras API, as well as TensorFlow's pre-made Estimators.

Chapter 15, *Classifying Images with Deep Convolutional Neural Networks*, introduces **convolutional neural networks** (**CNNs**). A CNN represents a particular type of deep NN architecture that is particularly well suited for image datasets. Due to their superior performance compared to traditional approaches, CNNs are now widely used in computer vision to achieve state-of-the-art results for various image recognition tasks. Throughout this chapter, you will learn how convolutional layers can be used as powerful feature extractors for image classification.

Chapter 16, *Modeling Sequential Data Using Recurrent Neural Networks*, introduces another popular NN architecture for deep learning that is especially well suited to working with text and other types of sequential data and time series data. As a warm-up exercise, this chapter introduces recurrent NNs for predicting the sentiment of movie reviews. Then, the chapter covers teaching recurrent networks to digest information from books in order to generate entirely new text.

Chapter 17, *Generative Adversarial Networks for Synthesizing New Data*, introduces a popular adversarial training regime for NNs that can be used to generate new, realistic-looking images. The chapter starts with a brief introduction to autoencoders, a particular type of NN architecture that can be used for data compression. The chapter then shows how to combine the decoder part of an autoencoder with a second NN that can distinguish between real and synthesized images. By letting two NNs compete with each other in an adversarial training approach, you will implement a generative adversarial network that generates new handwritten digits. Lastly, after introducing the basic concepts of generative adversarial networks, the chapter introduces improvements that can stabilize the adversarial training, such as using the Wasserstein distance metric.

Chapter 18, *Reinforcement Learning for Decision Making in Complex Environments*, covers a subcategory of machine learning that is commonly used for training robots and other autonomous systems. This chapter starts by introducing the basics of **reinforcement learning** (**RL**) to make you familiar with agent/environment interactions, the reward process of RL systems, and the concept of learning from experience. The chapter covers the two main categories of RL, model-based and model-free RL. After learning about basic algorithmic approaches, such as Monte Carlo- and temporal distance-based learning, you will implement and train an agent that can navigate a grid world environment using the Q-learning algorithm.

Finally, this chapter introduces the deep Q-learning algorithm, which is a variant of Q-learning that uses deep NNs.

What you need for this book

The execution of the code examples provided in this book requires an installation of Python 3.7.0 or newer on macOS, Linux, or Microsoft Windows. We will make frequent use of Python's essential libraries for scientific computing throughout this book, including SciPy, NumPy, scikit-learn, Matplotlib, and pandas.

The first chapter will provide you with instructions and useful tips to set up your Python environment and these core libraries. We will add additional libraries to our repertoire, and installation instructions are provided in the respective chapters, for example, the NLTK library for natural language processing in *Chapter 8*, *Applying Machine Learning to Sentiment Analysis*, the Flask web framework in *Chapter 9*, *Embedding a Machine Learning Model into a Web Application*, and TensorFlow for efficient NN training on GPUs in *Chapter 13* to *Chapter 18*.

To get the most out of this book

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Download the example code files

You can download the example code files from your account at `http://www.packtpub.com` for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `http://www.packt.com`.
2. Select the **Support** tab.
3. Click on **Code Downloads**.
4. Enter the name of the book in the **Search** box and follow the on-screen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows

- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

Alternatively, if you have obtained a copy of the book from elsewhere or do not wish to create an account at Packt, all code examples are also available for download through GitHub at `https://github.com/rasbt/python-machine-learning-book-3rd-edition`.

All code in this book is also available in the form of Jupyter notebooks, and a short introduction can be found in the code directory of *Chapter 1*, *Giving Computers the Ability to Learn from Data*, at `https://github.com/rasbt/python-machine-learning-book-3rd-edition/tree/master/ch01#pythonjupyter-notebook`. For more information about the general Jupyter Notebook GUI, please see the official documentation at `https://jupyter-notebook.readthedocs.io/en/stable/`.

While we recommend using Jupyter Notebook for executing code interactively, all code examples are available in both a Python script (for example, `ch02/ch02.py`) and a Jupyter Notebook format (for example, `ch02/ch02.ipynb`). Furthermore, we recommend that you view the `README.md` file that accompanies each individual chapter for additional information and updates (for example, `https://github.com/rasbt/python-machine-learning-book-3rd-edition/blob/master/ch01/README.md`).

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you to better understand the changes in the output. You can download this file from `https://static.packt-cdn.com/downloads/9781789955750_ColorImages.pdf`. In addition, lower resolution color images are embedded in the code notebooks of this book that come bundled with the example code files.

Conventions used

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text are shown as follows: "And already installed packages can be updated via the `--upgrade` flag."

A block of code is set as follows:

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> y = df.iloc[0:100, 4].values
>>> y = np.where(y == 'Iris-setosa', -1, 1)
>>> X = df.iloc[0:100, [0, 2]].values
>>> plt.scatter(X[:50, 0], X[:50, 1],
...             color='red', marker='x', label='setosa')
>>> plt.scatter(X[50:100, 0], X[50:100, 1],
...             color='blue', marker='o', label='versicolor')
>>> plt.xlabel('sepal length')
>>> plt.ylabel('petal length')
>>> plt.legend(loc='upper left')
>>> plt.show()
```

Any command-line input or output is written as follows:

```
> dot -Tpng tree.dot -o tree.png
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Clicking the **Next** button moves you to the next screen."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us to improve subsequent versions of this book. If you find any errata, please report them by visiting `www.packtpub.com/support/errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the **Errata** section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

1
Giving Computers the Ability to Learn from Data

In my opinion, **machine learning**, the application and science of algorithms that make sense of data, is the most exciting field of all the computer sciences! We are living in an age where data comes in abundance; using self-learning algorithms from the field of machine learning, we can turn this data into knowledge. Thanks to the many powerful open source libraries that have been developed in recent years, there has probably never been a better time to break into the machine learning field and learn how to utilize powerful algorithms to spot patterns in data and make predictions about future events.

In this chapter, you will learn about the main concepts and different types of machine learning. Together with a basic introduction to the relevant terminology, we will lay the groundwork for successfully using machine learning techniques for practical problem solving.

In this chapter, we will cover the following topics:

- The general concepts of machine learning
- The three types of learning and basic terminology
- The building blocks for successfully designing machine learning systems
- Installing and setting up Python for data analysis and machine learning

Building intelligent machines to transform data into knowledge

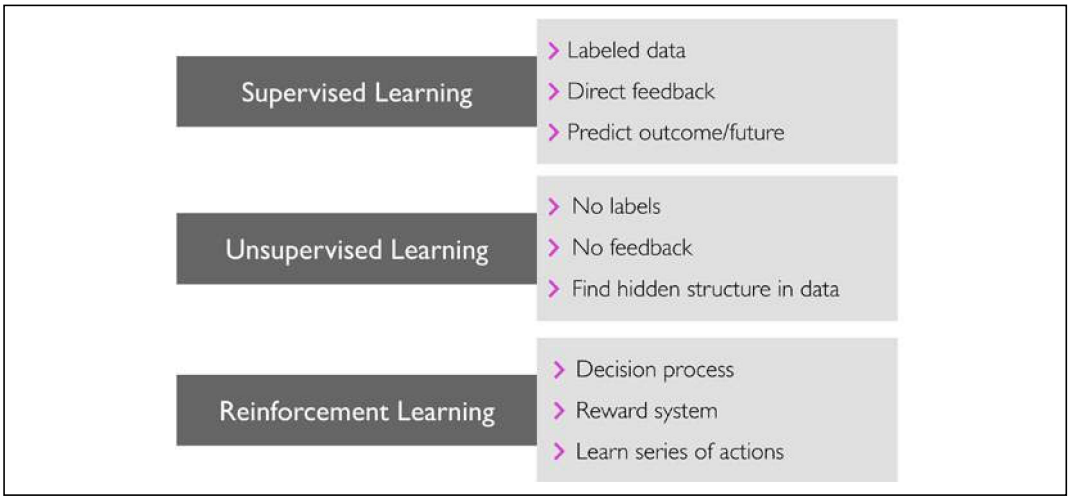
In this age of modern technology, there is one resource that we have in abundance: a large amount of structured and unstructured data. In the second half of the 20th century, machine learning evolved as a subfield of **artificial intelligence** (**AI**) involving self-learning algorithms that derive knowledge from data in order to make predictions.

Instead of requiring humans to manually derive rules and build models from analyzing large amounts of data, machine learning offers a more efficient alternative for capturing the knowledge in data to gradually improve the performance of predictive models and make data-driven decisions.

Not only is machine learning becoming increasingly important in computer science research, but it is also playing an ever-greater role in our everyday lives. Thanks to machine learning, we enjoy robust email spam filters, convenient text and voice recognition software, reliable web search engines, and challenging chess-playing programs. Hopefully soon, we will add safe and efficient self-driving cars to this list. Also, notable progress has been made in medical applications; for example, researchers demonstrated that deep learning models can detect skin cancer with near-human accuracy (`https://www.nature.com/articles/nature21056`). Another milestone was recently achieved by researchers at DeepMind, who used deep learning to predict 3D protein structures, outperforming physics-based approaches for the first time (`https://deepmind.com/blog/alphafold/`).

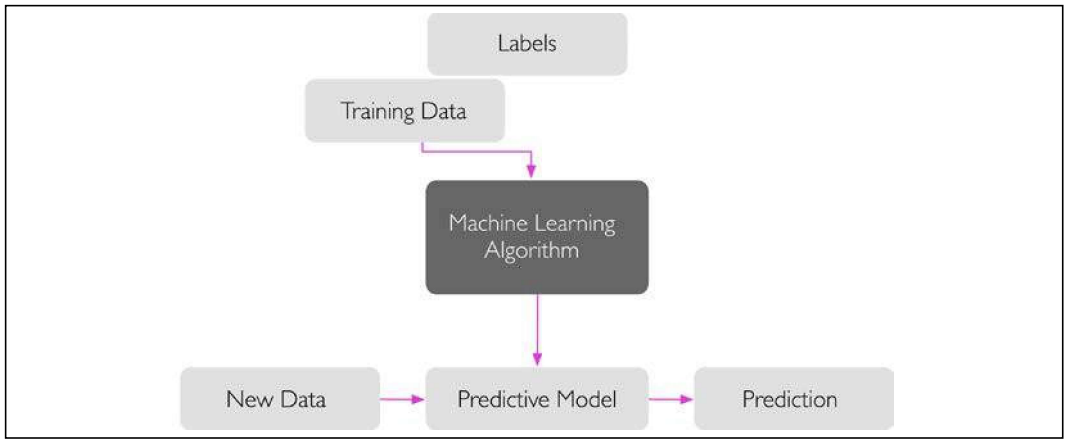
The three different types of machine learning

In this section, we will take a look at the three types of machine learning: **supervised learning**, **unsupervised learning**, and **reinforcement learning**. We will learn about the fundamental differences between the three different learning types and, using conceptual examples, we will develop an understanding of the practical problem domains where they can be applied:



Making predictions about the future with supervised learning

The main goal in supervised learning is to learn a model from labeled training data that allows us to make predictions about unseen or future data. Here, the term "supervised" refers to a set of training examples (data inputs) where the desired output signals (labels) are already known. The following figure summarizes a typical supervised learning workflow, where the labeled training data is passed to a machine learning algorithm for fitting a predictive model that can make predictions on new, unlabeled data inputs:

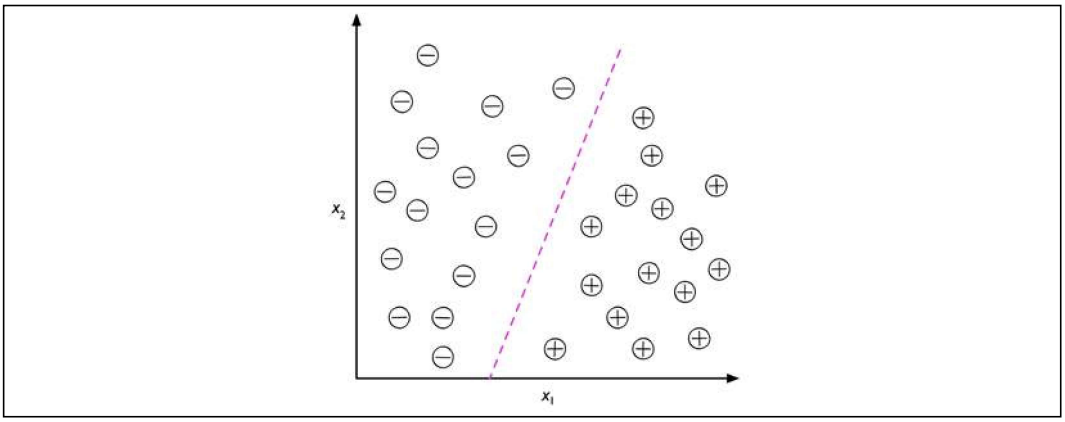


Considering the example of email spam filtering, we can train a model using a supervised machine learning algorithm on a corpus of labeled emails, which are correctly marked as spam or non-spam, to predict whether a new email belongs to either of the two categories. A supervised learning task with discrete class labels, such as in the previous email spam filtering example, is also called a **classification task**. Another subcategory of supervised learning is **regression**, where the outcome signal is a continuous value.

Classification for predicting class labels

Classification is a subcategory of supervised learning where the goal is to predict the categorical class labels of new instances, based on past observations. Those class labels are discrete, unordered values that can be understood as the group memberships of the instances. The previously mentioned example of email spam detection represents a typical example of a binary classification task, where the machine learning algorithm learns a set of rules in order to distinguish between two possible classes: spam and non-spam emails.

The following figure illustrates the concept of a binary classification task given 30 training examples; 15 training examples are labeled as the negative class (minus signs) and 15 training examples are labeled as the positive class (plus signs). In this scenario, our dataset is two-dimensional, which means that each example has two values associated with it: x_1 and x_2. Now, we can use a supervised machine learning algorithm to learn a rule—the decision boundary represented as a dashed line—that can separate those two classes and classify new data into each of those two categories given its x_1 and x_2 values:



However, the set of class labels does not have to be of a binary nature. The predictive model learned by a supervised learning algorithm can assign any class label that was presented in the training dataset to a new, unlabeled instance.

A typical example of a **multiclass classification** task is handwritten character recognition. We can collect a training dataset that consists of multiple handwritten examples of each letter in the alphabet. The letters ("A," "B," "C," and so on) will represent the different unordered categories or class labels that we want to predict. Now, if a user provides a new handwritten character via an input device, our predictive model will be able to predict the correct letter in the alphabet with certain accuracy. However, our machine learning system will be unable to correctly recognize any of the digits between 0 and 9, for example, if they were not part of the training dataset.

Regression for predicting continuous outcomes

We learned in the previous section that the task of classification is to assign categorical, unordered labels to instances. A second type of supervised learning is the prediction of continuous outcomes, which is also called **regression analysis**. In regression analysis, we are given a number of predictor (**explanatory**) variables and a continuous response variable (**outcome**), and we try to find a relationship between those variables that allows us to predict an outcome.

Note that in the field of machine learning, the predictor variables are commonly called "features," and the response variables are usually referred to as "target variables." We will adopt these conventions throughout this book.

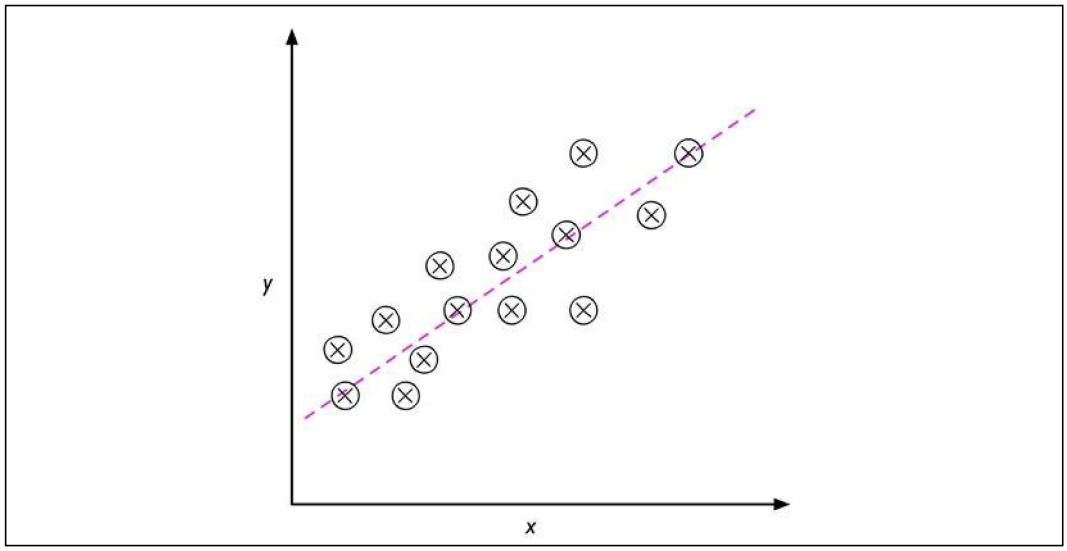
For example, let's assume that we are interested in predicting the math SAT scores of students. If there is a relationship between the time spent studying for the test and the final scores, we could use it as training data to learn a model that uses the study time to predict the test scores of future students who are planning to take this test.

Regression toward the mean

The term "regression" was devised by Francis Galton in his article *Regression towards Mediocrity in Hereditary Stature* in 1886. Galton described the biological phenomenon that the variance of height in a population does not increase over time.

He observed that the height of parents is not passed on to their children, but instead, their children's height regresses toward the population mean.

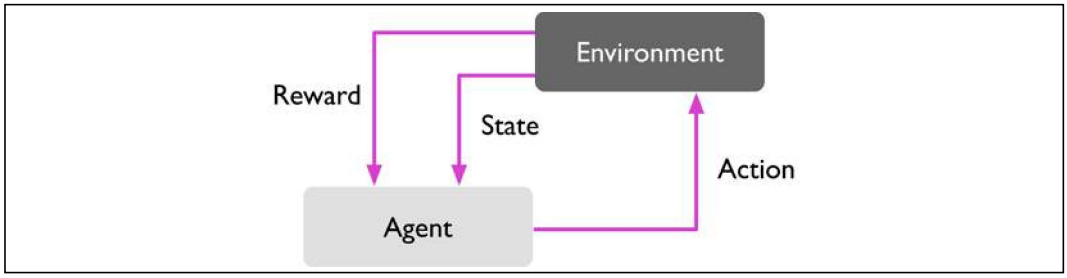
The following figure illustrates the concept of linear regression. Given a feature variable, x, and a target variable, y, we fit a straight line to this data that minimizes the distance—most commonly the average squared distance—between the data points and the fitted line. We can now use the intercept and slope learned from this data to predict the target variable of new data:



Solving interactive problems with reinforcement learning

Another type of machine learning is **reinforcement learning**. In reinforcement learning, the goal is to develop a system (**agent**) that improves its performance based on interactions with the environment. Since the information about the current state of the environment typically also includes a so-called **reward signal**, we can think of reinforcement learning as a field related to supervised learning. However, in reinforcement learning, this feedback is not the correct ground truth label or value, but a measure of how well the action was measured by a reward function. Through its interaction with the environment, an agent can then use reinforcement learning to learn a series of actions that maximizes this reward via an exploratory trial-and-error approach or deliberative planning.

A popular example of reinforcement learning is a chess engine. Here, the agent decides upon a series of moves depending on the state of the board (the environment), and the reward can be defined as **win** or **lose** at the end of the game:



There are many different subtypes of reinforcement learning. However, a general scheme is that the agent in reinforcement learning tries to maximize the reward through a series of interactions with the environment. Each state can be associated with a positive or negative reward, and a reward can be defined as accomplishing an overall goal, such as winning or losing a game of chess. For instance, in chess, the outcome of each move can be thought of as a different state of the environment.

To explore the chess example further, let's think of visiting certain configurations on the chess board as being associated with states that will more likely lead to winning—for instance, removing an opponent's chess piece from the board or threatening the queen. Other positions, however, are associated with states that will more likely result in losing the game, such as losing a chess piece to the opponent in the following turn. Now, in the game of chess, the reward (either positive for winning or negative for losing the game) will not be given until the end of the game. In addition, the final reward will also depend on how the opponent plays. For example, the opponent may sacrifice the queen but eventually win the game.

Reinforcement learning is concerned with learning to choose a series of actions that maximizes the total reward, which could be earned either immediately after taking an action or via *delayed* feedback.

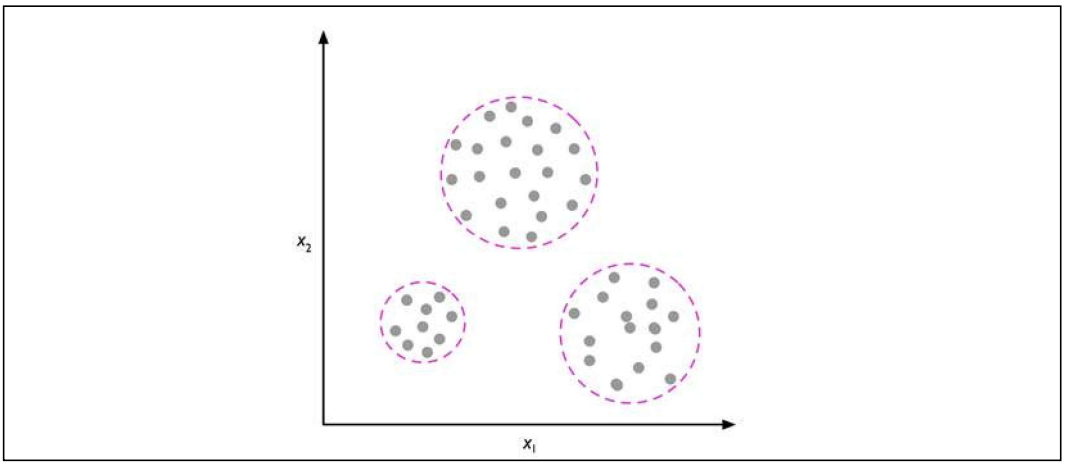
Discovering hidden structures with unsupervised learning

In supervised learning, we know the right answer beforehand when we train a model, and in reinforcement learning, we define a measure of reward for particular actions carried out by the agent. In unsupervised learning, however, we are dealing with unlabeled data or data of unknown structure. Using unsupervised learning techniques, we are able to explore the structure of our data to extract meaningful information without the guidance of a known outcome variable or reward function.

Finding subgroups with clustering

Clustering is an exploratory data analysis technique that allows us to organize a pile of information into meaningful subgroups (**clusters**) without having any prior knowledge of their group memberships. Each cluster that arises during the analysis defines a group of objects that share a certain degree of similarity but are more dissimilar to objects in other clusters, which is why clustering is also sometimes called **unsupervised classification**. Clustering is a great technique for structuring information and deriving meaningful relationships from data. For example, it allows marketers to discover customer groups based on their interests, in order to develop distinct marketing programs.

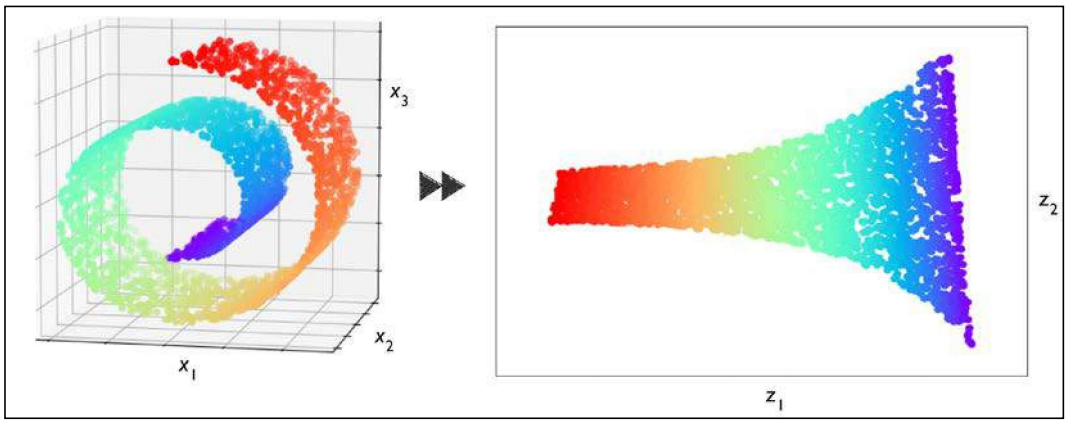
The following figure illustrates how clustering can be applied to organizing unlabeled data into three distinct groups based on the similarity of their features, x_1 and x_2:



Dimensionality reduction for data compression

Another subfield of unsupervised learning is **dimensionality reduction**. Often, we are working with data of high dimensionality—each observation comes with a high number of measurements—that can present a challenge for limited storage space and the computational performance of machine learning algorithms. Unsupervised dimensionality reduction is a commonly used approach in feature preprocessing to remove noise from data, which can also degrade the predictive performance of certain algorithms, and compress the data onto a smaller dimensional subspace while retaining most of the relevant information.

Sometimes, dimensionality reduction can also be useful for visualizing data; for example, a high-dimensional feature set can be projected onto one-, two-, or three-dimensional feature spaces in order to visualize it via 2D or 3D scatterplots or histograms. The following figure shows an example where nonlinear dimensionality reduction was applied to compress a 3D Swiss Roll onto a new 2D feature subspace:



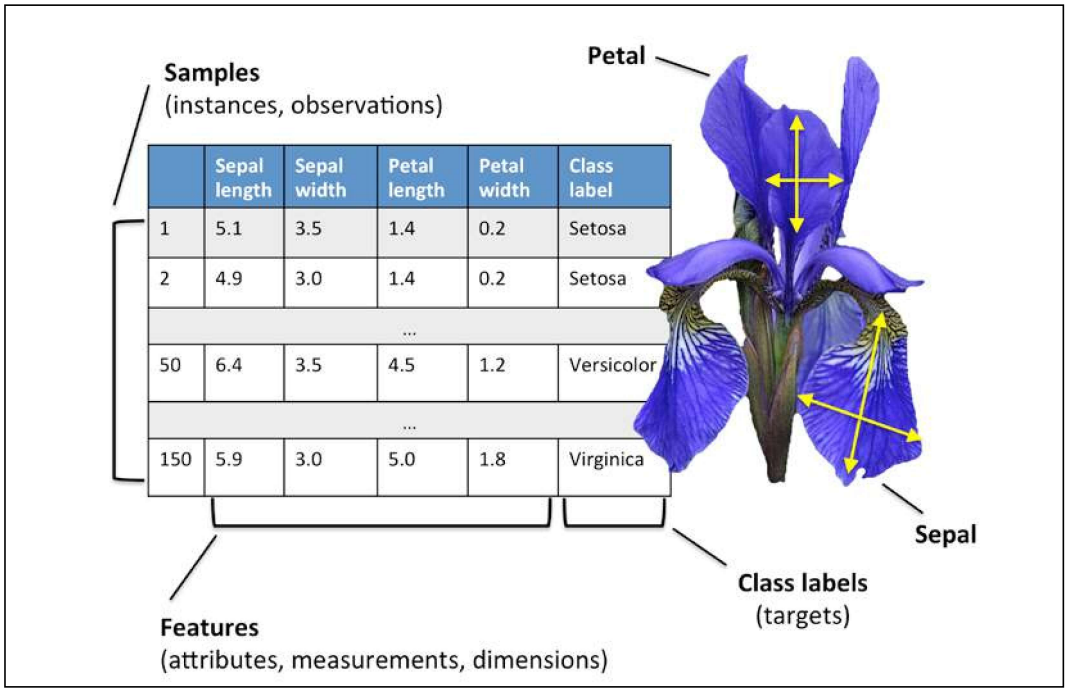
Introduction to the basic terminology and notations

Now that we have discussed the three broad categories of machine learning—supervised, unsupervised, and reinforcement learning—let's have a look at the basic terminology that we will be using throughout this book. The following subsection covers the common terms we will be using when referring to different aspects of a dataset, as well as the mathematical notation to communicate more precisely and efficiently.

As machine learning is a vast field and very interdisciplinary, you are guaranteed to encounter many different terms that refer to the same concepts sooner rather than later. The second subsection collects many of the most commonly used terms that are found in machine learning literature, which may be useful to you as a reference section when reading more diverse machine learning literature.

Notation and conventions used in this book

The following table depicts an excerpt of the Iris dataset, which is a classic example in the field of machine learning. The Iris dataset contains the measurements of 150 Iris flowers from three different species—Setosa, Versicolor, and Virginica. Here, each flower example represents one row in our dataset, and the flower measurements in centimeters are stored as columns, which we also call the **features** of the dataset:



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

To keep the notation and implementation simple yet efficient, we will make use of some of the basics of linear algebra. In the following chapters, we will use a matrix and vector notation to refer to our data. We will follow the common convention to represent each example as a separate row in a feature matrix, $\mathbf{X}$, where each feature is stored as a separate column.

The Iris dataset, consisting of 150 examples and four features, can then be written as a 150×4 matrix, $\boldsymbol{X} \in \mathbb{R}^{150\times4}$:

$$\begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(150)} & x_2^{(150)} & x_3^{(150)} & x_4^{(150)} \end{bmatrix}$$

Notational conventions

For the rest of this book, unless noted otherwise, we will use the superscript *i* to refer to the *i*th training example, and the subscript *j* to refer to the *j*th dimension of the training dataset.

We will use lowercase, bold-face letters to refer to vectors ($\boldsymbol{x} \in \mathbb{R}^{n\times1}$) and uppercase, bold-face letters to refer to matrices ($\boldsymbol{X} \in \mathbb{R}^{n\times m}$). To refer to single elements in a vector or matrix, we will write the letters in italics ($x^{(n)}$ or $x_m^{(n)}$, respectively).

For example, $x_1^{(150)}$ refers to the first dimension of flower example 150, the *sepal length*. Thus, each row in this feature matrix represents one flower instance and can be written as a four-dimensional row vector, $\boldsymbol{x}^{(i)} \in \mathbb{R}^{1\times4}$:

$$\boldsymbol{x}^{(i)} = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & x_3^{(i)} & x_4^{(i)} \end{bmatrix}$$

And each feature dimension is a 150-dimensional column vector, $\boldsymbol{x}^{(i)} \in \mathbb{R}^{150\times1}$. For example:

$$\boldsymbol{x}_j = \begin{bmatrix} x_j^{(1)} \\ x_j^{(2)} \\ \dots \\ x_j^{(150)} \end{bmatrix}$$

Similarly, we will store the target variables (here, class labels) as a 150-dimensional column vector:

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ \dots \\ y^{(150)} \end{bmatrix} (\text{y} \in \{\text{Setosa}, \text{Versicolor}, \text{Virginica}\})$$

Machine learning terminology

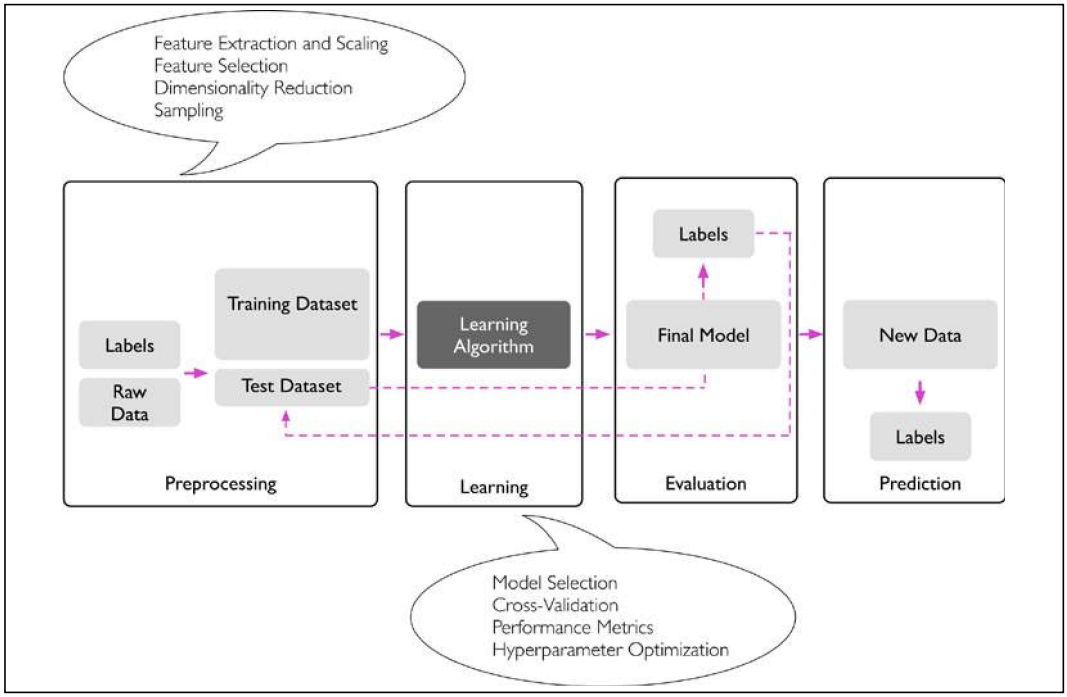
Machine learning is a vast field and also very interdisciplinary as it brings together many scientists from other areas of research. As it happens, many terms and concepts have been rediscovered or redefined and may already be familiar to you but appear under different names. For your convenience, in the following list, you can find a selection of commonly used terms and their synonyms that you may find useful when reading this book and machine learning literature in general:

- Training example: A row in a table representing the dataset and synonymous with an observation, record, instance, or sample (in most contexts, sample refers to a collection of training examples).
- Training: Model fitting, for parametric models similar to parameter estimation.
- Feature, abbrev. x: A column in a data table or data (design) matrix. Synonymous with predictor, variable, input, attribute, or covariate.
- Target, abbrev. y: Synonymous with outcome, output, response variable, dependent variable, (class) label, and ground truth.
- Loss function: Often used synonymously with a *cost* function. Sometimes the loss function is also called an *error* function. In some literature, the term "loss" refers to the loss measured for a single data point, and the cost is a measurement that computes the loss (average or summed) over the entire dataset.

A roadmap for building machine learning systems

In previous sections, we discussed the basic concepts of machine learning and the three different types of learning. In this section, we will discuss the other important parts of a machine learning system accompanying the learning algorithm.

The following diagram shows a typical workflow for using machine learning in predictive modeling, which we will discuss in the following subsections:



Preprocessing – getting data into shape

Let's begin with discussing the roadmap for building machine learning systems. Raw data rarely comes in the form and shape that is necessary for the optimal performance of a learning algorithm. Thus, the preprocessing of the data is one of the most crucial steps in any machine learning application.

If we take the Iris flower dataset from the previous section as an example, we can think of the raw data as a series of flower images from which we want to extract meaningful features. Useful features could be the color, hue, and intensity of the flowers, or the height, length, and width of the flowers.

Many machine learning algorithms also require that the selected features are on the same scale for optimal performance, which is often achieved by transforming the features in the range [0, 1] or a standard normal distribution with zero mean and unit variance, as we will see in later chapters.

Some of the selected features may be highly correlated and therefore redundant to a certain degree. In those cases, dimensionality reduction techniques are useful for compressing the features onto a lower dimensional subspace. Reducing the dimensionality of our feature space has the advantage that less storage space is required, and the learning algorithm can run much faster. In certain cases, dimensionality reduction can also improve the predictive performance of a model if the dataset contains a large number of irrelevant features (or noise); that is, if the dataset has a low signal-to-noise ratio.

To determine whether our machine learning algorithm not only performs well on the training dataset but also generalizes well to new data, we also want to randomly divide the dataset into a separate training and test dataset. We use the training dataset to train and optimize our machine learning model, while we keep the test dataset until the very end to evaluate the final model.

Training and selecting a predictive model

As you will see in later chapters, many different machine learning algorithms have been developed to solve different problem tasks. An important point that can be summarized from David Wolpert's famous *No free lunch theorems* is that we can't get learning "for free" (*The Lack of A Priori Distinctions Between Learning Algorithms*, D.H. Wolpert, 1996; *No free lunch theorems for optimization*, D.H. Wolpert and W.G. Macready, 1997). We can relate this concept to the popular saying, "*I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail*" (Abraham Maslow, 1966). For example, each classification algorithm has its inherent biases, and no single classification model enjoys superiority if we don't make any assumptions about the task. In practice, it is therefore essential to compare at least a handful of different algorithms in order to train and select the best performing model. But before we can compare different models, we first have to decide upon a metric to measure performance. One commonly used metric is classification accuracy, which is defined as the proportion of correctly classified instances.

One legitimate question to ask is this: *how do we know which model performs well on the final test dataset and real-world data if we don't use this test dataset for the model selection, but keep it for the final model evaluation?* In order to address the issue embedded in this question, different techniques summarized as "cross-validation" can be used. In cross-validation, we further divide a dataset into training and validation subsets in order to estimate the generalization performance of the model. Finally, we also cannot expect that the default parameters of the different learning algorithms provided by software libraries are optimal for our specific problem task. Therefore, we will make frequent use of hyperparameter optimization techniques that help us to fine-tune the performance of our model in later chapters.

We can think of those hyperparameters as parameters that are not learned from the data but represent the knobs of a model that we can turn to improve its performance. This will become much clearer in later chapters when we see actual examples.

Evaluating models and predicting unseen data instances

After we have selected a model that has been fitted on the training dataset, we can use the test dataset to estimate how well it performs on this unseen data to estimate the so-called generalization error. If we are satisfied with its performance, we can now use this model to predict new, future data. It is important to note that the parameters for the previously mentioned procedures, such as feature scaling and dimensionality reduction, are solely obtained from the training dataset, and the same parameters are later reapplied to transform the test dataset, as well as any new data instances—the performance measured on the test data may be overly optimistic otherwise.

Using Python for machine learning

Python is one of the most popular programming languages for data science and thanks to its very active developer and open source community, a large number of useful libraries for scientific computing and machine learning have been developed.

Although the performance of interpreted languages, such as Python, for computation-intensive tasks is inferior to lower-level programming languages, extension libraries such as NumPy and SciPy have been developed that build upon lower-layer Fortran and C implementations for fast vectorized operations on multidimensional arrays.

For machine learning programming tasks, we will mostly refer to the scikit-learn library, which is currently one of the most popular and accessible open source machine learning libraries. In the later chapters, when we focus on a subfield of machine learning called deep learning, we will use the latest version of the TensorFlow library, which specializes in training so-called deep neural network models very efficiently by utilizing graphics cards.

Installing Python and packages from the Python Package Index

Python is available for all three major operating systems—Microsoft Windows, macOS, and Linux—and the installer, as well as the documentation, can be downloaded from the official Python website: `https://www.python.org`.

This book is written for Python version 3.7 or higher, and it is recommended that you use the most recent version of Python 3 that is currently available. Some of the code may also be compatible with Python 2.7, but as the official support for Python 2.7 ends in 2019, and the majority of open source libraries have already stopped supporting Python 2.7 (`https://python3statement.org`), we strongly advise that you use Python 3.7 or newer.

The additional packages that we will be using throughout this book can be installed via the `pip` installer program, which has been part of the Python Standard Library since Python 3.3. More information about `pip` can be found at `https://docs.python.org/3/installing/index.html`.

After we have successfully installed Python, we can execute pip from the terminal to install additional Python packages:

```
pip install SomePackage
```

Already installed packages can be updated via the `--upgrade` flag:

```
pip install SomePackage --upgrade
```

Using the Anaconda Python distribution and package manager

A highly recommended alternative Python distribution for scientific computing is Anaconda by Continuum Analytics. Anaconda is a free—including commercial use—enterprise-ready Python distribution that bundles all the essential Python packages for data science, math, and engineering into one user-friendly, cross-platform distribution. The Anaconda installer can be downloaded at `https://docs.anaconda.com/anaconda/install/`, and an Anaconda quick start guide is available at `https://docs.anaconda.com/anaconda/user-guide/getting-started/`.

After successfully installing Anaconda, we can install new Python packages using the following command:

```
conda install SomePackage
```

Existing packages can be updated using the following command:

```
conda update SomePackage
```

Packages for scientific computing, data science, and machine learning

Throughout this book, we will mainly use NumPy's multidimensional arrays to store and manipulate data. Occasionally, we will make use of pandas, which is a library built on top of NumPy that provides additional higher-level data manipulation tools that make working with tabular data even more convenient. To augment your learning experience and visualize quantitative data, which is often extremely useful to make sense of it, we will use the very customizable Matplotlib library.

The version numbers of the major Python packages that were used to write this book are mentioned in the following list. Please make sure that the version numbers of your installed packages are equal to, or greater than, these version numbers to ensure that the code examples run correctly:

- NumPy 1.17.4
- SciPy 1.3.1
- scikit-learn 0.22.0
- Matplotlib 3.1.0
- pandas 0.25.3

Summary

In this chapter, we explored machine learning at a very high level and familiarized ourselves with the big picture and major concepts that we are going to explore in the following chapters in more detail. We learned that supervised learning is composed of two important subfields: classification and regression. While classification models allow us to categorize objects into known classes, we can use regression analysis to predict the continuous outcomes of target variables. Unsupervised learning not only offers useful techniques for discovering structures in unlabeled data, but it can also be useful for data compression in feature preprocessing steps.

We briefly went over the typical roadmap for applying machine learning to problem tasks, which we will use as a foundation for deeper discussions and hands-on examples in the following chapters. Finally, we set up our Python environment and installed and updated the required packages to get ready to see machine learning in action.

Later in this book, in addition to machine learning itself, we will introduce different techniques to preprocess a dataset, which will help you to get the best performance out of different machine learning algorithms. While we will cover classification algorithms quite extensively throughout the book, we will also explore different techniques for regression analysis and clustering.

We have an exciting journey ahead, covering many powerful techniques in the vast field of machine learning. However, we will approach machine learning one step at a time, building upon our knowledge gradually throughout the chapters of this book. In the following chapter, we will start this journey by implementing one of the earliest machine learning algorithms for classification, which will prepare us for *Chapter 3*, *A Tour of Machine Learning Classifiers Using scikit-learn*, where we will cover more advanced machine learning algorithms using the scikit-learn open source machine learning library.

2

Training Simple Machine Learning Algorithms for Classification

In this chapter, we will make use of two of the first algorithmically described machine learning algorithms for classification: the perceptron and adaptive linear neurons. We will start by implementing a perceptron step by step in Python and training it to classify different flower species in the Iris dataset. This will help us to understand the concept of machine learning algorithms for classification and how they can be efficiently implemented in Python.

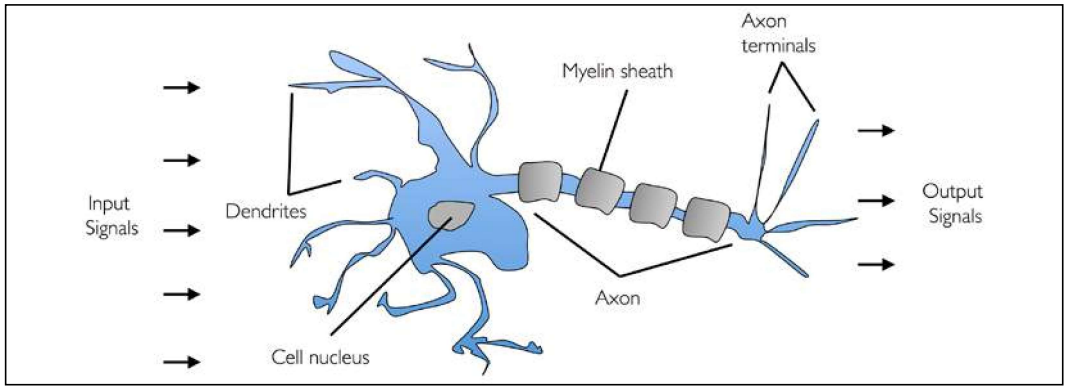
Discussing the basics of optimization using adaptive linear neurons will then lay the groundwork for using more sophisticated classifiers via the scikit-learn machine learning library in *Chapter 3*, *A Tour of Machine Learning Classifiers Using scikit-learn*.

The topics that we will cover in this chapter are as follows:

- Building an understanding of machine learning algorithms
- Using pandas, NumPy, and Matplotlib to read in, process, and visualize data
- Implementing linear classification algorithms in Python

Artificial neurons – a brief glimpse into the early history of machine learning

Before we discuss the perceptron and related algorithms in more detail, let's take a brief tour of the beginnings of machine learning. Trying to understand how the biological brain works, in order to design artificial intelligence (AI), Warren McCulloch and Walter Pitts published the first concept of a simplified brain cell, the so-called **McCulloch-Pitts** (**MCP**) neuron, in 1943 (*A Logical Calculus of the Ideas Immanent in Nervous Activity*, *W. S. McCulloch* and *W. Pitts*, *Bulletin of Mathematical Biophysics*, 5(4): 115-133, *1943*). Biological neurons are interconnected nerve cells in the brain that are involved in the processing and transmitting of chemical and electrical signals, which is illustrated in the following figure:



McCulloch and Pitts described such a nerve cell as a simple logic gate with binary outputs; multiple signals arrive at the dendrites, they are then integrated into the cell body, and, if the accumulated signal exceeds a certain threshold, an output signal is generated that will be passed on by the axon.

Only a few years later, Frank Rosenblatt published the first concept of the perceptron learning rule based on the MCP neuron model (*The Perceptron: A Perceiving and Recognizing Automaton*, *F. Rosenblatt*, *Cornell Aeronautical Laboratory*, *1957*). With his perceptron rule, Rosenblatt proposed an algorithm that would automatically learn the optimal weight coefficients that would then be multiplied with the input features in order to make the decision of whether a neuron fires (transmits a signal) or not. In the context of supervised learning and classification, such an algorithm could then be used to predict whether a new data point belongs to one class or the other.

The formal definition of an artificial neuron

More formally, we can put the idea behind **artificial neurons** into the context of a binary classification task where we refer to our two classes as *1* (positive class) and *–1* (negative class) for simplicity. We can then define a decision function ($\phi(z)$) that takes a linear combination of certain input values, $\boldsymbol{x}$, and a corresponding weight vector, $\boldsymbol{w}$, where z is the so-called net input $z = w_1x_1 + w_2x_2 + \ldots + w_mx_m$:

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \qquad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

Now, if the net input of a particular example, $\boldsymbol{x}^{(i)}$, is greater than a defined threshold, θ, we predict class *1*, and class *–1* otherwise. In the perceptron algorithm, the decision function, $\phi(\cdot)$, is a variant of a **unit step function**:

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq \theta, \\ -1 & \text{otherwise.} \end{cases}$$

For simplicity, we can bring the threshold, θ, to the left side of the equation and define a weight-zero as $w_0 = -\theta$ and $x_0 = 1$ so that we write z in a more compact form:

$$z = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \boldsymbol{w}^T\boldsymbol{x}$$

And:

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

In machine learning literature, the negative threshold, or weight, $w_0 = -\theta$, is usually called the **bias unit**.

Linear algebra basics: dot product and matrix transpose

In the following sections, we will often make use of basic notations from linear algebra. For example, we will abbreviate the sum of the products of the values in $\boldsymbol{x}$ and $\boldsymbol{w}$ using a vector dot product, whereas superscript T stands for transpose, which is an operation that transforms a column vector into a row vector and vice versa:

$$z = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \sum_{j=0}^{m} x_j w_j = \boldsymbol{w}^T \boldsymbol{x}$$

For example:

$$[1 \quad 2 \quad 3] \times \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 1 \times 4 + 2 \times 5 + 3 \times 6 = 32$$

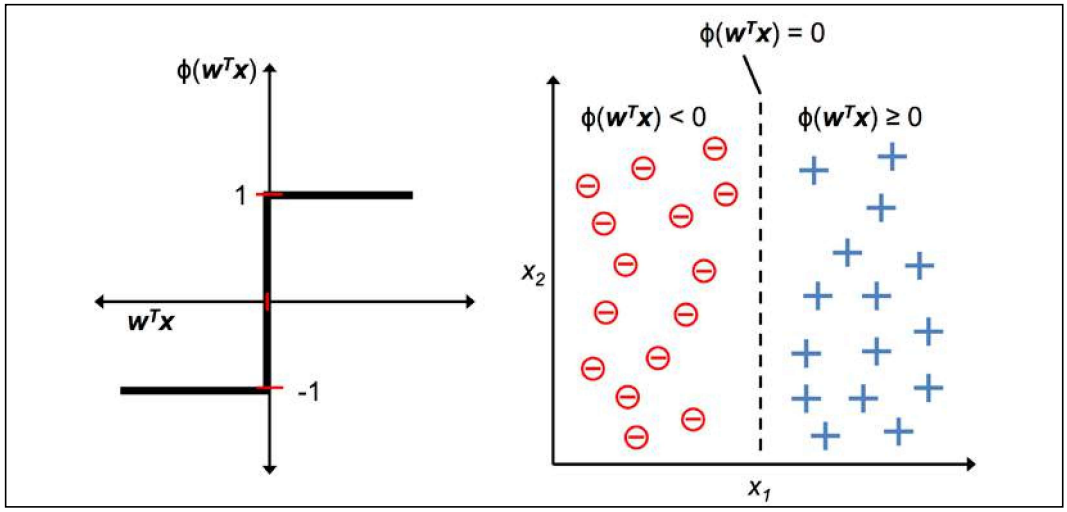
Furthermore, the transpose operation can also be applied to matrices to reflect it over its diagonal, for example:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

Please note that the transpose operation is strictly only defined for matrices; however, in the context of machine learning, we refer to $n \times 1$ or $1 \times m$ matrices when we use the term "vector."

In this book, we will only use very basic concepts from linear algebra; however, if you need a quick refresher, please take a look at Zico Kolter's excellent *Linear Algebra Review and Reference*, which is freely available at `http://www.cs.cmu.edu/~zkolter/course/linalg/linalg_notes.pdf`.

The following figure illustrates how the net input $z = \boldsymbol{w}^T \boldsymbol{x}$ is squashed into a binary output (–1 or 1) by the decision function of the perceptron (left subfigure) and how it can be used to discriminate between **two linearly separable classes** (right subfigure):



The perceptron learning rule

The whole idea behind the MCP neuron and Rosenblatt's *thresholded* perceptron model is to use a reductionist approach to mimic how a single neuron in the brain works: it either *fires* or it doesn't. Thus, Rosenblatt's initial perceptron rule is fairly simple, and the perceptron algorithm can be summarized by the following steps:

1. Initialize the weights to 0 or small random numbers.
2. For each training example, $\boldsymbol{x}^{(i)}$:
    a. Compute the output value, $\hat{y}$.
    b. Update the weights.

Here, the output value is the class label predicted by the unit step function that we defined earlier, and the simultaneous update of each weight, w_j, in the weight vector, $\boldsymbol{w}$, can be more formally written as:

$$w_j := w_j + \Delta w_j$$