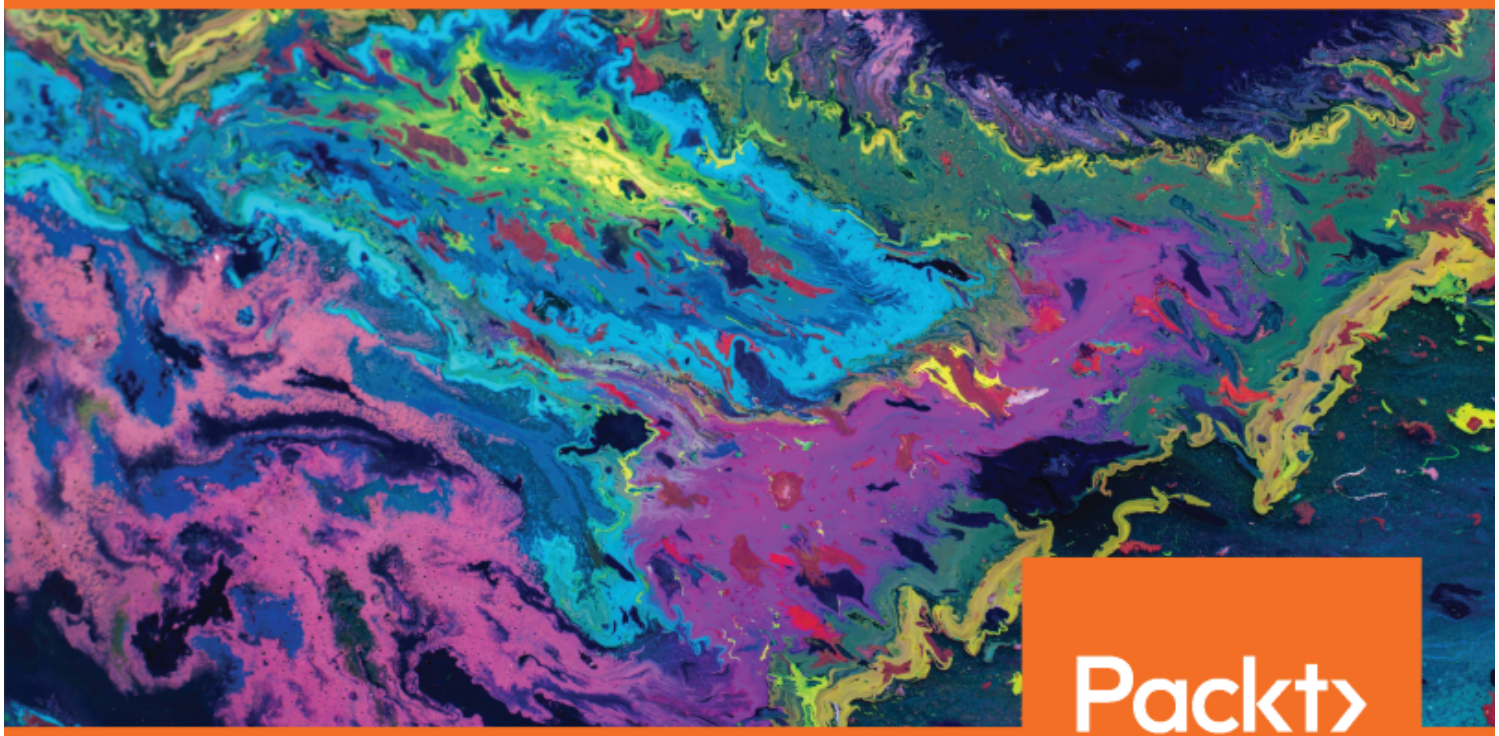
Python Machine Learning By Example

Second Edition

Implement machine learning algorithms and techniques to build intelligent systems



Yuxi (Hayden) Liu

Packt
www.packt.com

Python Machine Learning By Example
Second Edition

Implement machine learning algorithms and techniques to build intelligent systems

Yuxi (Hayden) Liu

Packt

BIRMINGHAM - MUMBAI

Python Machine Learning By Example
Second Edition

Copyright © 2019 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Commissioning Editor: Sunith Shetty
Acquisition Editor: Reshma Amare
Content Development Editor: Athikho Sapuni Rishana
Technical Editor: Vibhuti Gawde
Copy Editor: Safis Editing
Project Coordinator: Kirti Pisat
Proofreader: Safis Editing
Indexer: Pratik Shirodkar
Graphics: Jisha Chirayil
Production Coordinator: Jisha Chirayil

First published: May 2017
Second edition: February 2019

Production reference: 1270219

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78961-672-9

www.packtpub.com

mapt.io

Mapt is an online digital library that gives you full access to over 5,000 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
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

Packt.com

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.

I would like to thank all of the great people that made this book possible. Without any of you, this book would only exist in my mind. I would like to especially thank all of my editors at Packt Publishing: Vibhuti, Athikho, Martin, and so many more, as well as my reviewer, Vadim, a renowned machine learning researcher from MIT. I would also like to thank all of the reviewers of the code of this book. Without them, this book would be harder to read and apply to real-world problems. Last but not least, I'd like to thank all the readers for the support they provided, which encourages me to continue with the second edition of this book.

Foreword

This book is a deep dive into the exciting world of machine learning. What's unique about this book is the clarity with which it explains concepts from first principles and teaches by example in a way that is accessible to a wide audience. You will learn how to implement key algorithms from scratch and compare your code against proven machine learning libraries.

The discussion in this book is backed by mathematical principles and includes from-scratch coding exercises that help you gain a deeper understanding of the subject. By reading this book, you will be learning something new, whether you are a beginner or an experienced machine learning practitioner.

True to its title, you will learn about a number of interesting applications, such as predicting click-through rates for targeted advertisements, mining text data for patterns, and predicting the stock price of a major exchange index. Throughout the book, you will find exercises and links to help you better understand the material.

I encourage you to turn the page and dive into the exciting world of machine learning.

Vadim Smolyakov

Contributors

About the author

Yuxi (Hayden) Liu is an author of a series of machine learning books and an education enthusiast. His first book, the first edition of *Python Machine Learning By Example*, was a #1 bestseller in Amazon India in 2017 and 2018. His other books include *R Deep Learning Projects* and *Hands-On Deep Learning Architectures with Python* published by Packt.

He is an experienced data scientist who's focused on developing machine learning and deep learning models and systems. He has worked in a variety of data-driven domains and has applied his machine learning expertise to computational advertising, recommendation, and network anomaly detection. He published five first-authored IEEE transaction and conference papers during his master's research at the University of Toronto.

About the reviewer

Vadim Smolyakov is currently pursuing his PhD at MIT in the areas of computer science and artificial intelligence. His primary research interests include Bayesian inference, deep learning, and optimization. Prior to coming to MIT, Vadim received his undergraduate degree in engineering science at the University of Toronto. He previously worked as a data scientist in the e-commerce space. Vadim is passionate about machine learning and data science, and is interested in making the field accessible to a broad audience and inspiring readers to innovate and pursue research in artificial intelligence.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Preface

The surge in interest in machine learning is due to the fact that it revolutionizes automation by learning patterns in data and using them to make predictions and decisions. If you're interested in machine learning, this book will serve as your entry point.

This edition of *Python Machine Learning By Example* begins with an introduction to important concepts and implementations using Python libraries. Each chapter of the book walks you through an industry-adopted application. You'll implement machine learning techniques in areas such as exploratory data analysis, feature engineering, and natural language processing (NLP) in a clear and easy-to-follow way.

With the help of this extended and updated edition, you'll learn how to tackle data-driven problems and implement your solutions with the powerful yet simple Python language, and popular Python packages and tools such as TensorFlow, scikit-learn, Gensim, and Keras. To aid your understanding of popular machine learning algorithms, this book covers interesting and easy-to-follow examples such as news topic modeling and classification, spam email detection, and stock price forecasting.

By the end of the book, you'll have put together a broad picture of the machine learning ecosystem and will be well-versed with the best practices of applying machine learning techniques to make the most out of new opportunities.

Who this book is for

If you're a machine learning aspirant, data analyst, or a data engineer who's highly passionate about machine learning and wants to begin working on machine learning assignments, this book is for you. Prior knowledge of Python coding is assumed, and basic familiarity with statistical concepts will be beneficial, although not necessary.

What this book covers

`Chapter 1`, *Getting Started with Machine Learning and Python*, will be the starting point for readers who are looking forward to entering the field of machine learning with Python. It will introduce the essential concepts of machine learning, which we will dig deeper into throughout the rest of the book. In addition, it will discuss the basics of Python for machine learning and explain how to set it up properly for the upcoming examples and projects.

`Chapter 2`, *Exploring the 20 Newsgroups Dataset with Text Analysis Techniques*, will start developing the first project of the book, exploring and mining the 20 newsgroups dataset, which will be split into two parts—`Chapter 2`, *Exploring the 20 Newsgroups Dataset with Text Analysis Techniques*, and `Chapter 3`, *Mining the 20 Newsgroups Dataset with Clustering and Topic Modeling Algorithms*. In this chapter, readers will get familiar with NLP and various NLP libraries that will be used for this project. We will explain several important NLP techniques implementing them in NLTK. We will also cover the dimension reduction technique, especially t-SNE and its use in text data visualization.

`Chapter 3`, *Mining the 20 Newsgroups Dataset with Clustering and Topic Modeling Algorithms*, will continue our newsgroups project after exploring the 20 newsgroups dataset. In this chapter, readers will learn about unsupervised learning and clustering algorithms, as well as some advanced NLP techniques, such as LDA and word embedding. We will cluster the newsgroups data using the k-means algorithm, and detect topics using NMF and LDA.

`Chapter 4`, *Detecting Spam Emails with Naive Bayes*, will start our supervised learning journey. In this chapter, we focus on classification with Naïve Bayes, and we'll look at an in-depth implementation. We will also cover other important machine learning concepts, such as classification performance evaluation, model selection and tuning, and cross-validation. Examples including spam email detection will be demonstrated.

`Chapter 5`, *Classifying Newsgroup Topics with a Support Vector Machine*, will reuse the newsgroups dataset we used in `Chapter 2`, *Exploring the 20 Newsgroups Dataset with Text Analysis Techniques*, and `Chapter 3`, *Mining the 20 Newsgroups Dataset with Clustering and Topic Modeling Algorithms*. We will cover multiclass classification, as well as SVM and how they are applied in topic classification. Other important concepts, such as kernel machines, overfitting, and regularization, will be discussed as well.

`Chapter 6`, *Predicting Online Ad Click-Through with Tree-Based Algorithms*, will introduce and explain decision trees and random forests in depth throughout the course of solving the advertising click-through rate problem. Important concepts of tree-based models such as ensemble, feature importance, and feature selection will also be covered.

`Chapter 7`, *Predicting Online Ads Click-Through with Logistic Regression*, will introduce and explain logistic regression classifiers on the same project from the previous chapters. We will also cover other concepts, such as categorical variable encoding, L1 and L2 regularization, feature selection, online learning and stochastic gradient descent, and, of course, how to work with large datasets.

`Chapter 8`, *Scaling Up Prediction to Terabyte Click Logs*, covers online advertising click-through prediction, where we have millions of labeled samples in a typical large-scale machine learning problem. In this chapter, we will explore a more scalable solution than the previous chapters, utilizing powerful parallel computing tools such as Apache Hadoop and Spark. We will cover the essential concepts of Spark, such installation, RDD, and core programming, as well as its machine learning components. We will work with the entire dataset of millions of samples, explore the data, build classification models, perform feature engineering, and performance evaluation using Spark, which scales up the computation.

`Chapter 9`, *Stock Price Prediction with Regression Algorithms*, introduces the aim of this project, which is to analyze and predict stock market prices using the Yahoo/Google Finance data, and maybe additional data.

We will start the chapter by covering the challenges in finance and looking at a brief explanation of the related concepts. The next step is to obtain and explore the dataset and start feature engineering after exploratory data analysis. The core section, looking at regression and regression algorithms, linear regression, decision tree regression, SVR, and neural networks, will follow. Readers will also practice solving regression problems using scikit-learn and the TensorFlow API.

`Chapter 10`, *Machine Learning Best Practices*, covers best practices in machine learning. After covering multiple projects in this book, you will have gathered a broad picture of the machine learning ecosystem using Python. However, there will be issues once you start working on projects in the real world. This chapter aims to foolproof your learning and get you ready for production by providing 21 best practices throughout the entire machine learning workflow.

To get the most out of this book

You are expected to have basic knowledge of Python, the basic machine learning algorithms, and some basic Python libraries, such as TensorFlow and Keras, to create smart cognitive actions for your projects.

Download the example code files

You can download the example code files for this book from your account at `www.packt.com`. If you purchased this book elsewhere, you can visit `www.packt.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 `www.packt.com`.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the onscreen 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

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Python-Machine-Learning-By-Example-Second-Edition`. In case there's an update to the code, it will be updated on the existing GitHub repository.

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 a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://www.packtpub.com/sites/default/files/downloads/9781789616729_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Then, we'll load the `en_core_web_sm` model and parse the sentence using this model."

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

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

```
sudo pip install –U nltk
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "A new window will pop up and ask us which collections (the **Collections** tab in the following screenshot) or corpus (the identifiers in the **Corpora** tab in the following screenshot) to download and where to keep the data."

Warnings or important notes appear 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 have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packt.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

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
Section 1: Fundamentals of Machine Learning

In this section, readers will learn about the essential concepts in machine learning, including types of machine learning tasks, the core of machine learning, and an overview of data processing and modeling. Readers will also have a chance to set up the working environment of the rest of the book, and will learn how to install Python machine learning packages properly.

The following chapter is in this section:

- `Chapter 1`, *Getting Started with Machine Learning and Python*

1
Getting Started with Machine Learning and Python

We kick off our Python and machine learning journey with the basic, yet important, concepts of machine learning. We'll start with what machine learning is about, why we need it, and its evolution over a few decades. We'll then discuss typical machine learning tasks and explore several essential techniques of working with data and working with models. It's a great starting point for the subject and we'll learn it in a fun way. Trust me. At the end, we'll also set up the software and tools needed for this book.

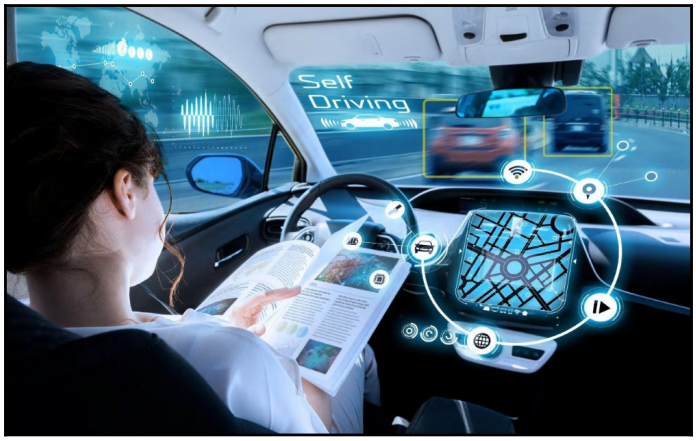
We'll go into detail on the following topics:

- Overview of machine learning and the importance of machine learning
- The core of machine learning—generalizing with data
- Overfitting
- Underfitting
- Bias variance trade-off
- Techniques to avoid overfitting
- Techniques for data preprocessing
- Techniques for feature engineering
- Techniques for model aggregation
- Software installing
- Python package setup

Defining machine learning and why we need it

Machine learning is a term coined around 1960, composed of two words—**machine** corresponds to a computer, robot, or other device, and **learning** refers to an activity intended to acquire or discover event patterns, which we humans are good at.

So, why do we need machine learning and why do we want a machine to learn as a human? First and foremost, of course, computers and robots can work 24/7 and don't get tired, need breaks, call in sick, or go on strike. Their maintenance is much lower than a human's and costs a lot less in the long run. Also, for sophisticated problems that involve a variety of huge datasets or complex calculations, for instance, it's much more justifiable, not to mention intelligent, to let computers do all of the work. Machines driven by algorithms designed by humans are able to learn latent rules and inherent patterns and to fulfill tasks desired by humans. Learning machines are better suited than humans for tasks that are routine, repetitive, or tedious. Beyond that, automation by machine learning can mitigate risks caused by fatigue or inattention. Self-driving cars, as shown in the following photograph, are a great example: a vehicle capable of navigating by sensing its environment and making its decision without human input. Another example is the use of robotic arms in production lines, capable of causing a significant reduction in injuries and costs:

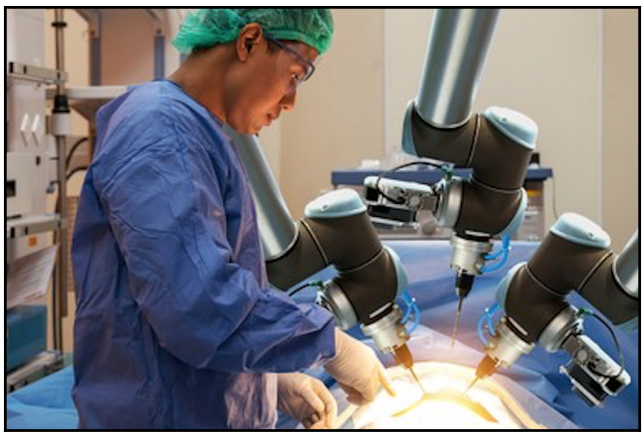


Assume humans don't fatigue or we have resources to hire enough shift workers, would machine learning still have a place? Of course it would; there are many cases, reported and unreported, where machines perform comparably or even better than domain experts. As algorithms are designed to learn from the ground truth, and the best-thought decisions made by human experts, machines can perform just as well as experts. In reality, even the best expert makes mistakes. Machines can minimize the chance of making wrong decisions by utilizing collective intelligence from individual experts. A major study that found machines are better than doctors at diagnosing some types of cancer proves this philosophy, for instance. **AlphaGo** is probably the best known example of machines beating human masters. Also, it's much more scalable to deploy learning machines than to train individuals to become experts, economically and socially. We can distribute thousands of diagnostic devices across the globe within a week but it's almost impossible to recruit and assign the same number of qualified doctors.

Now you may argue: what if we have sufficient resources and capacity to hire the best domain experts and later aggregate their opinions—would machine learning still have a place? Probably not—learning machines might not perform better than the joint efforts of the most intelligent humans. However, individuals equipped with learning machines can outperform the best group of experts. This is actually an emerging concept called **AI-based Assistance** or **AI Plus Human Intelligence**, which advocates combining the efforts of machine learners and humans. We can summarize the previous statement in the following inequality:

$$human + machine\ learning \rightarrow most\ intelligent\ tireless\ human \geq machine\ learning > human$$

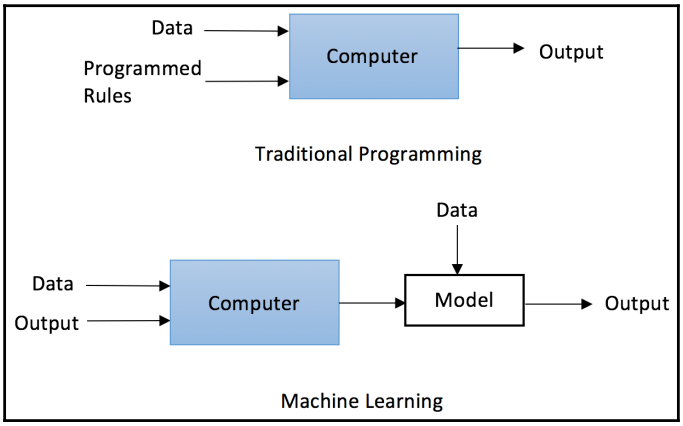
A medical operation involving robots is one example of the best human and machine learning synergy. The following photograph presents robotic arms in an operation room alongside the surgery doctor:



So, does machine learning simply equate to automation that involves the programming and execution of human-crafted or human-curated rule sets? A popular myth says that the majority of code in the world has to do with simple rules possibly programmed in Common Business Oriented Language (COBOL), which covers the bulk of all of the possible scenarios of client interactions. So, if the answer to that question is yes, why can't we just hire many software programmers and continue programming new rules or extending old rules?

One reason is that defining, maintaining, and updating rules becomes more and more expensive over time. The number of possible patterns for an activity or event could be enormous and, therefore, exhausting all enumeration isn't practically feasible. It gets even more challenging when it comes to events that are dynamic, ever-changing, or evolving in real time. It's much easier and more efficient to develop learning algorithms that command computers to learn and extract patterns and to figure things out themselves from abundant data.

The difference between machine learning and traditional programming can be described using the following diagram:



Another reason is that the volume of data is exponentially growing. Nowadays, the floods of textual, audio, image, and video data are hard to fathom. The **Internet of Things** (**IoT**) is a recent development of a new kind of internet, which interconnects everyday devices. The IoT will bring data from household appliances and autonomous cars to the forefront. The average company these days has mostly human clients but, for instance, social media companies tend to have many bot accounts. This trend is likely to continue and we'll have more machines talking to each other. Besides the quantity, the quality of data available has kept increasing in the past years due to cheaper storage. This has empowered the evolution of machine learning algorithms and data-driven solutions.

Jack Ma, co-founder of the e-commerce company Alibaba, explained in a speech that IT was the focus of the past 20 years but, for the next 30 years, we'll be in the age of **Data Technology** (**DT**). During the age of IT, companies grew larger and stronger thanks to computer software and infrastructure. Now that businesses in most industries have already gathered enormous amounts of data, it's presently the right time to exploit DT to unlock insights, derive patterns, and boost new business growth. Broadly speaking, machine learning technologies enable businesses to better understand customer behavior, engage with customers, and optimize operations management. As for us individuals, machine learning technologies are already making our lives better every day.

An application of machine learning with which we're all familiar is spam email filtering. Another is online advertising, where ads are served automatically based on information advertisers have collected about us. Stay tuned for the next chapters, where we'll learn how to develop algorithms in solving these two problems and more. A search engine is an application of machine learning we can't imagine living without. It involves information retrieval, which parses what we look for, queries related to records, and applies contextual ranking and personalized ranking, which sorts pages by topical relevance and user preference. E-commerce and media companies have been at the forefront of employing recommendation systems, which help customers to find products, services, and articles faster. The application of machine learning is boundless and we just keep hearing new examples everyday: credit card fraud detection, disease diagnosis, presidential election prediction, instant speech translation, and robot advisors—you name it!

In the 1983 *War Games* movie, a computer made life-and-death decisions that could have resulted in Word War III. As far as we know, technology wasn't able to pull off such feats at the time. However, in 1997, the Deep Blue supercomputer did manage to beat a world chess champion. In 2005, a Stanford self-driving car drove by itself for more than 130 kilometers in a desert. In 2007, the car of another team drove through regular traffic for more than 50 kilometers. In 2011, the Watson computer won a quiz against human opponents. In 2016, the AlphaGo program beat one of the best Go players in the world. If we assume that computer hardware is the limiting factor, then we can try to extrapolate into the future. Ray Kurzweil did just that and, according to him, we can expect human level intelligence around 2029. What's next?

A very high-level overview of machine learning technology

Machine learning mimicking human intelligence is a subfield of AI—a field of computer science concerned with creating systems. Software engineering is another field in computer science. Generally, we can label Python programming as a type of software engineering. Machine learning is also closely related to linear algebra, probability theory, statistics, and mathematical optimization. We usually build machine learning models based on statistics, probability theory, and linear algebra, then optimize the models using mathematical optimization. The majority of you reading this book should have a good, or at least sufficient, command of Python programming. Those who aren't feeling confident about mathematical knowledge might be wondering how much time should be spent learning or brushing up on the aforementioned subjects. Don't panic: we'll get machine learning to work for us without going into any mathematical details in this book. It just requires some basic 101 knowledge of probability theory and linear algebra, which helps us to understand the mechanics of machine learning techniques and algorithms. And it gets easier as we'll be building models both from scratch and with popular packages in Python, a language we like and are familiar with.

For those who want to learn or brush up on probability theory and linear algebra, feel free to search for *basic probability theory* and *basic linear algebra*. There are a lot of resources online, for example, `https://people.ucsc.edu/~abrsvn/intro_prob_1.pdf` on probability 101 and `http://www.maths.gla.ac.uk/~ajb/dvi-ps/2w-notes.pdf` about basic linear algebra.

Those who want to study machine learning systematically can enroll into computer science, **Artificial Intelligence** (**AI**), and, more recently, data science masters programs. There are also various data science boot camps. However, the selection for boot camps is usually stricter as they're more job-oriented and the program duration is often short, ranging from 4 to 10 weeks. Another option is the free **Massive Open Online Courses** (**MOOCs**), Andrew Ng's popular course on machine learning. Last but not least, industry blogs and websites are great resources for us to keep up with the latest developments.

Machine learning isn't only a skill but also a bit of sport. We can compete in several machine learning competitions, such as Kaggle (`www.kaggle.com`)—sometimes for decent cash prizes, sometimes for joy, and most of the time to play our strengths. However, to win these competitions, we may need to utilize certain techniques, which are only useful in the context of competitions and not in the context of trying to solve a business problem. That's right, the **no free lunch** theorem applies here.

Types of machine learning tasks

A machine learning system is fed with input data—this can be numerical, textual, visual, or audiovisual. The system usually has an output—this can be a floating-point number, for instance, the acceleration of a self-driving car, or can be an integer representing a category (also called a **class**), for example, a cat or tiger from image recognition.

The main task of machine learning is to explore and construct algorithms that can learn from historical data and make predictions on new input data. For a data-driven solution, we need to define (or have it defined to us by an algorithm) an evaluation function called **loss** or **cost function**, which measures how well the models are learning. In this setup, we create an optimization problem with the goal of learning in the most efficient and effective way.

Depending on the nature of the learning data, machine learning tasks can be broadly classified into the following three categories:

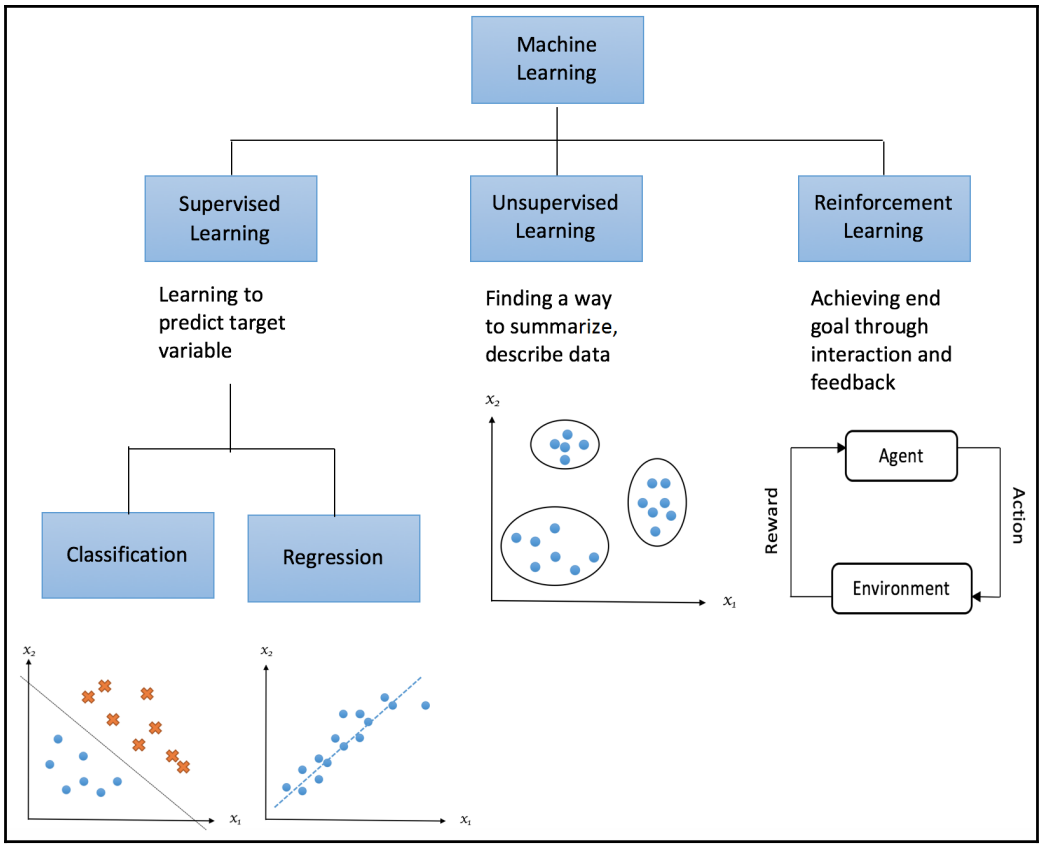
- **Unsupervised learning**: When the learning data only contains indicative signals without any description attached, it's up to us to find the structure of the data underneath, to discover hidden information, or to determine how to describe the data. This kind of learning data is called **unlabeled** data. Unsupervised learning can be used to detect anomalies, such as fraud or defective equipment, or to group customers with similar online behaviors for a marketing campaign.
- **Supervised learning**: When learning data comes with a description, targets, or desired output besides indicative signals, the learning goal becomes to find a general rule that maps input to output. This kind of learning data is called **labeled** data. The learned rule is then used to label new data with unknown output. The labels are usually provided by event-logging systems and human experts. Besides, if it's feasible, they may also be produced by members of the public, through crowd-sourcing, for instance. Supervised learning is commonly used in daily applications, such as face and speech recognition, products or movie recommendations, and sales forecasting.

  We can further subdivide supervised learning into regression and classification. **Regression** trains on and predicts continuous-valued response, for example, predicting house prices, while **classification** attempts to find the appropriate class label, such as analyzing a positive/negative sentiment and prediction loan default.

If not all learning samples are labeled, but some are, we'll have **semi-supervised learning**. It makes use of unlabeled data (typically a large amount) for training, besides a small amount of labeled data. Semi-supervised learning is applied in cases where it's expensive to acquire a fully labeled dataset and more practical to label a small subset. For example, it often requires skilled experts to label hyperspectral remote sensing images and lots of field experiments to locate oil at a particular location, while acquiring unlabeled data is relatively easy.

- **Reinforcement learning**: Learning data provides feedback so that the system adapts to dynamic conditions in order to achieve a certain goal in the end. The system evaluates its performance based on the feedback responses and reacts accordingly. The best known instances include self-driving cars and the chess master, AlphaGo.

The following diagram depicts types of machine learning tasks:



Feeling a little bit confused by the abstract concepts? Don't worry. We'll encounter many concrete examples of these types of machine learning tasks later in this book. In `Chapter 2`, *Exploring the 20 Newsgroups Dataset with Text Analysis Techniques*, and `Chapter 3`, *Mining the 20 Newsgroups Dataset with Clustering and Topic Modeling Algorithms*, we'll explore unsupervised techniques and algorithms; in `Chapter 4`, *Detecting Spam Email with Naive Bayes*, and `Chapter 8`, *Scaling Up Prediction to Terabyte Click Logs*, we'll work on supervised learning tasks and several classification algorithms; in `Chapter 9`, *Stock Price Prediction with Regression Algorithms*, we'll continue with another supervised learning task, regression, and assorted regression algorithms.

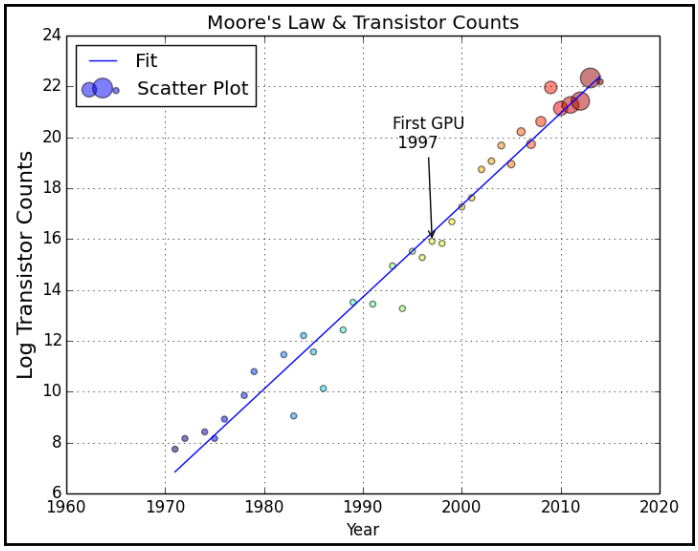
A brief history of the development of machine learning algorithms

In fact, we have a whole zoo of machine learning algorithms that have experienced varying popularity over time. We can roughly categorize them into four main approaches such as logic-based learning, statistical learning, artificial neural networks, and genetic algorithms.

The logic-based systems were the first to be dominant. They used basic rules specified by human experts and, with these rules, systems tried to reason using formal logic, background knowledge, and hypotheses. In the mid-1980s, **artificial neural networks** (**ANNs**) came to the foreground, to be then pushed aside by statistical learning systems in the 1990s. ANNs imitate animal brains and consist of interconnected neurons that are also an imitation of biological neurons. They try to model complex relationships between input and output values and to capture patterns in data. Genetic algorithms (GA) were popular in the 1990s. They mimic the biological process of evolution and try to find the optimal solutions using methods such as mutation and crossover.

We are currently seeing a revolution in **deep learning**, which we might consider a rebranding of neural networks. The term deep learning was coined around 2006 and refers to deep neural networks with many layers. The breakthrough in deep learning is caused by the integration and utilization of **Graphical Processing Units** (**GPUs**), which massively speed up computation. GPUs were originally developed to render video games and are very good in parallel matrix and vector algebra. It's believed that deep learning resembles the way humans learn, therefore, it may be able to deliver on the promise of sentient machines.

Some of us may have heard of **Moore's law**—an empirical observation claiming that computer hardware improves exponentially with time. The law was first formulated by Gordon Moore, the co-founder of Intel, in 1965. According to the law, the number of transistors on a chip should double every two years. In the following diagram, you can see that the law holds up nicely (the size of the bubbles corresponds to the average transistor count in GPUs):



The consensus seems to be that Moore's law should continue to be valid for a couple of decades. This gives some credibility to Ray Kurzweil's predictions of achieving true machine intelligence in 2029.

Core of machine learning – generalizing with data

The good thing about data is that there's a lot of it in the world. The bad thing is that it's hard to process this data. The challenges stem from the diversity and noisiness of the data. We humans usually process data coming into our ears and eyes. These inputs are transformed into electrical or chemical signals. On a very basic level, computers and robots also work with electrical signals. These electrical signals are then translated into ones and zeroes. However, we program in Python in this book and, on that level, normally we represent the data either as numbers, images, or texts. Actually, images and text aren't very convenient, so we need to transform images and text into numerical values.

Especially in the context of supervised learning, we have a scenario similar to studying for an exam. We have a set of practice questions and the actual exams. We should be able to answer exam questions without knowing the answers to them. This is called **generalization**—we learn something from our practice questions and, hopefully, are able to apply the knowledge to other similar questions. In machine learning, these practice questions are called **training sets** or **training samples**. They're where the models derive patterns from. And the actual exams are **testing sets** or **testing samples**. They're where the models eventually apply and how compatible they are is what it's all about. Sometimes, between practice questions and actual exams, we have mock exams to assess how well we'll do in actual ones and to aid revision. These mock exams are called **validation sets** or **validation samples** in machine learning. They help us to verify how well the models will perform in a simulated setting, then we fine-tune the models accordingly in order to achieve greater hits.

An old-fashioned programmer would talk to a business analyst or other expert, then implement a rule that adds a certain value multiplied by another value corresponding, for instance, to tax rules. In a machine learning setting, we give the computer example input values and example output values. Or if we're more ambitious, we can feed the program the actual tax texts and let the machine process the data further, just like an autonomous car doesn't need a lot of human input.

This means implicitly that there's some function, for instance, a tax formula, we're trying to figure out. In physics, we have almost the same situation. We want to know how the universe works and formulate laws in a mathematical language. Since we don't know the actual function, all we can do is measure the error produced and try to minimize it. In supervised learning tasks, we compare our results against the expected values. In unsupervised learning, we measure our success with related metrics. For instance, we want clusters of data to be well defined; the metrics could be how similar the data points within one cluster are, and how different the data points from two clusters are. In reinforcement learning, a program evaluates its moves, for example, using some predefined function in a chess game.

Other than the normal generalizing with data, there can be two levels of generalization, over and under generalization, which we'll explore in the next section.

Overfitting, underfitting, and the bias-variance trade-off

Overfitting is a very important concept, hence, we're discussing it here, early in this book.

If we go through many practice questions for an exam, we may start to find ways to answer questions that have nothing to do with the subject material. For instance, given only five practice questions, we find that if there are two occurrences of *potatoes*, one *tomato*, and three occurrences of *banana* in a question, the answer is always *A* and if there is one *potato*, three occurrences of *tomato*, and two occurrences of banana in a question, the answer is always *B*, then we conclude this is always true and apply such a theory later on, even though the subject or answer may not be relevant to potatoes, tomatoes, or bananas. Or even worse, you may memorize the answers to each question verbatim. We can then score high on the practice questions; we do so with the hope that the questions in the actual exams will be the same as the practice questions. However, in reality, we'll score very low on the exam questions as it's rare that the exact same questions will occur in the exams.

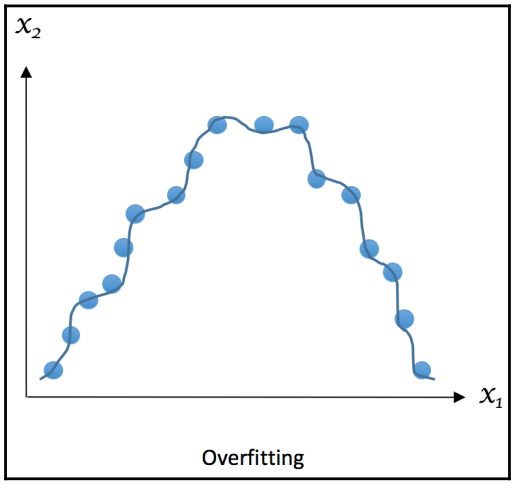
The phenomenon of memorization can cause overfitting. This can occur when we're over extracting too much information from the training sets and making our model just work well with them, which is called **low bias** in machine learning. In case you need a quick recap of bias, here it is: *Bias* is the difference between the average prediction and the true value. It's computed as follows:

$$Bias[\hat{y}] = E[\hat{y} - y]$$

Here, $\hat{y}$ is the prediction. At the same time, however, overfitting won't help us to generalize with data and derive true patterns from it. The model, as a result, will perform poorly on datasets that weren't seen before. We call this situation **high variance** in machine learning. Again, a quick recap of variance: *Variance* measures the spread of the prediction, which is the variability of the prediction. It can be calculated as follows:

$$Variance[\hat{y}] = E[\hat{y}^2] - E[\hat{y}]^2$$

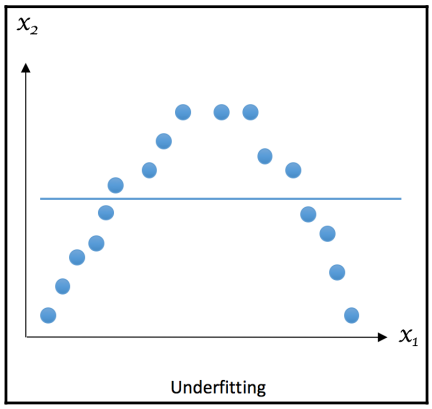
The following example demonstrates what a typical instance of overfitting looks like, where the regression curve tries to flawlessly accommodate all samples:



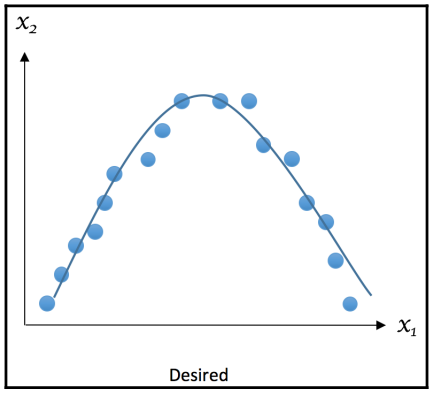
Overfitting occurs when we try to describe the learning rules based on too many parameters relative to the small number of observations, instead of the underlying relationship, such as the preceding example of potato and tomato where we deduced three parameters from only five learning samples. Overfitting also takes place when we make the model excessively complex so that it fits every training sample, such as memorizing the answers for all questions, as mentioned previously.

The opposite scenario is **underfitting**. When a model is underfit, it doesn't perform well on the training sets and won't do so on the testing sets, which means it fails to capture the underlying trend of the data. Underfitting may occur if we aren't using enough data to train the model, just like we'll fail the exam if we don't review enough material; it may also happen if we're trying to fit a wrong model to the data, just like we'll score low in any exercises or exams if we take the wrong approach and learn it the wrong way. We call any of these situations **high bias** in machine learning; although its variance is low as performance in training and test sets are pretty consistent, in a bad way.

The following example shows what a typical underfitting looks like, where the regression curve doesn't fit the data well enough or capture enough of the underlying pattern of the data:



After the overfitting and underfitting example, let's look at what a well-fitting example should look like:



We want to avoid both overfitting and underfitting. **Recall bias** is the error stemming from incorrect assumptions in the learning algorithm; high bias results in underfitting, and variance measures how sensitive the model prediction is to variations in the datasets. Hence, we need to avoid cases where either bias or variance is getting high. So, does it mean we should always make both bias and variance as low as possible? The answer is yes, if we can. But, in practice, there's an explicit trade-off between them, where decreasing one increases the other. This is the so-called **bias-variance trade-off**. Sounds abstract? Let's take a look at the next example.

Let's say we're asked to build a model to predict the probability of a candidate being the next president based on phone poll data. The poll was conducted using zip codes. We randomly choose samples from one zip code and we estimate there's a 61% chance the candidate will win. However, it turns out he loses the election. Where did our model go wrong? The first thing we think of is the small size of samples from only one zip code. It's a source of high bias also, because people in a geographic area tend to share similar demographics, although it results in a low variance of estimates. So, can we fix it simply by using samples from a large number of zip codes? Yes, but don't get happy so early. This might cause an increased variance of estimates at the same time. We need to find the optimal sample size—the best number of zip codes to achieve the lowest overall bias and variance.

Minimizing the total error of a model requires a careful balancing of bias and variance. Given a set of training samples $x_1, x_2, \ldots, x_n$ and their targets $y_1, y_2, \ldots, y_n$, we want to find a regression function $\hat{y}(x)$ that estimates the true relation $y(x)$ as correctly as possible. We measure the error of estimation, how good (or bad) the regression model is **mean squared error** (**MSE**):

$$MSE = E\left[\left(y(x) - \hat{y}(x)\right)^2\right]$$

The *E* denotes the expectation. This error can be decomposed into bias and variance components following the analytical derivation as shown in the following formula (although it requires a bit of basic probability theory to understand):

$$\begin{aligned} MSE &= E\left[(y - \hat{y})^2\right] \\ &= E\left[(y - E[\hat{y}] + E[\hat{y}] - \hat{y})^2\right] \\ &= E\left[\left(y - E[\hat{y}]\right)^2\right] + E\left[\left(E[\hat{y}] - \hat{y}\right)^2\right] + E\left[2\left(y - E[\hat{y}]\right)\left(E[\hat{y}] - \hat{y}\right)\right] \\ &= E\left[\left(y - E[\hat{y}]\right)^2\right] + E\left[\left(E[\hat{y}] - \hat{y}\right)^2\right] + 2\left(y - E[\hat{y}]\right)\left(E[\hat{y}] - E[\hat{y}]\right) \\ &= \left(E[\hat{y} - y]\right)^2 + E[\hat{y}^2] - E[\hat{y}]^2 \\ &= Bias[\hat{y}]^2 + Variance[\hat{y}] \end{aligned}$$

The *Bias* term measures the error of estimations and the *Variance* term describes how much the estimation $\hat{y}$ moves around its mean. The more complex the learning model $\hat{y}(x)$ is and the larger size of training samples, the lower the bias will be. However, these will also create more shift on the model in order to better fit the increased data points. As a result, the variance will be lifted.

We usually employ cross-validation technique as well as regularization and feature reduction to find the optimal model balancing bias and variance and to diminish overfitting.

You may ask why we only want to deal with overfitting: how about underfitting? This is because underfitting can be easily recognized: it occurs as long as the model doesn't work well on a training set. And we need to find a better model or tweak some parameters to better fit the data, which is a must under all circumstances. On the other hand, overfitting is hard to spot. Sometimes, when we achieve a model that performs well on a training set, we're overly happy and think it ready for production right away. This happens all of the time despite how dangerous it could be. We should instead take extra step to make sure the great performance isn't due to overfitting and the great performance applies to data excluding the training data.

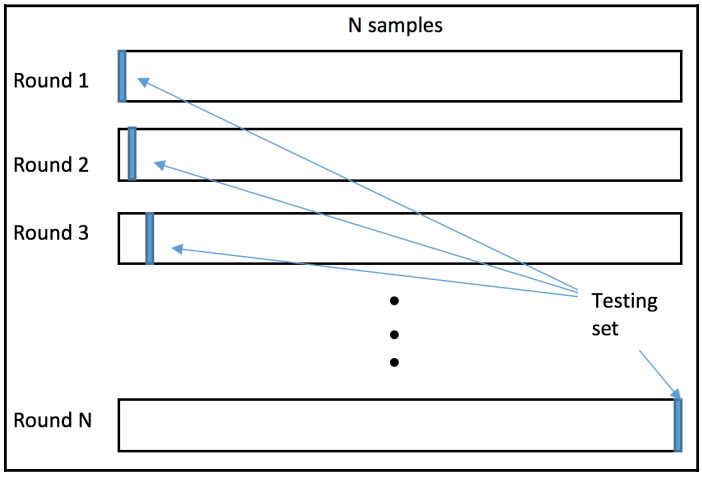
Avoiding overfitting with cross-validation

Recall that between practice questions and actual exams, there are mock exams where we can assess how well we'll perform in actual exams and use that information to conduct necessary revision. In machine learning, the validation procedure helps evaluate how the models will generalize to independent or unseen datasets in a simulated setting. In a conventional validation setting, the original data is partitioned into three subsets, usually 60% for the training set, 20% for the validation set, and the rest (20%) for the testing set. This setting suffices if we have enough training samples after partitioning and we only need a rough estimate of simulated performance. Otherwise, cross-validation is preferable.

In one round of cross-validation, the original data is divided into two subsets, for **training** and **testing** (or **validation**) respectively. The testing performance is recorded. Similarly, multiple rounds of cross-validation are performed under different partitions. Testing results from all rounds are finally averaged to generate a more reliable estimate of model prediction performance. Cross-validation helps to reduce variability and, therefore, limit overfitting.

When the training size is very large, it's often sufficient to split it into training, validation, and testing (three subsets) and conduct a performance check on the latter two. Cross-validation is less preferable in this case since it's computationally costly to train a model for each single round. But if you can afford it, there's no reason not to use cross-validation. When the size isn't so large, cross-validation is definitely a good choice.

There are mainly two cross-validation schemes in use, exhaustive and non-exhaustive. In the exhaustive scheme, we leave out a fixed number of observations in each round as testing (or validation) samples and the remaining observations as training samples. This process is repeated until all possible different subsets of samples are used for testing once. For instance, we can apply **Leave-One-Out-Cross-Validation** (**LOOCV**) and let each datum be in the testing set once. For a dataset of the size *n*, LOOCV requires *n* rounds of cross-validation. This can be slow when *n* gets large. This following diagram presents the workflow of LOOCV:



A non-exhaustive scheme, on the other hand, as the name implies, doesn't try out all possible partitions. The most widely used type of this scheme is **k-fold cross-validation**. The original data first randomly splits the data into **k equal-sized** folds. In each trial, one of these folds becomes the testing set, and the rest of the data becomes the training set. We repeat this process *k* times, with each fold being the designated testing set once. Finally, we average the *k* sets of test results for the purpose of evaluation. Common values for *k* are 3, 5, and 10. The following table illustrates the setup for five-fold:

Round	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
1	Testing	Training	Training	Training	Training
2	Training	Testing	Training	Training	Training
3	Training	Training	Testing	Training	Training
4	Training	Training	Training	Testing	Training
5	Training	Training	Training	Training	Testing

K-fold cross-validation often has a lower variance compared to LOOCV, since we're using a chunk of samples instead a single one for validation.

We can also randomly split the data into training and testing sets numerous times. This is formally called the **holdout** method. The problem with this algorithm is that some samples may never end up in the testing set, while some may be selected multiple times in the testing set.

Last but not the least, **nested cross-validation** is a combination of cross-validations. It consists of the following two phases:

- **Inner cross-validation**: This phase is conducted to find the best fit and can be implemented as a k-fold cross-validation
- **Outer cross-validation**: This phase is used for performance evaluation and statistical analysis

We'll apply cross-validation very intensively throughout this entire book. Before that, let's look at cross-validation with an analogy next, which will help us to better understand it.

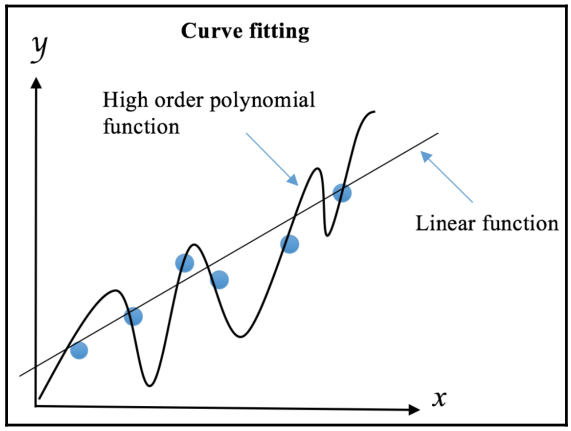
A data scientist plans to take his car to work and his goal is to arrive before 9 a.m. every day. He needs to decide the departure time and the route to take. He tries out different combinations of these two parameters on some Mondays, Tuesdays, and Wednesdays and records the arrival time for each trial. He then figures out the best schedule and applies it every day. However, it doesn't work quite as well as expected. It turns out the scheduling **model** is overfit with data points gathered in the first three days and may not work well on Thursdays and Fridays. A better solution would be to test the best combination of parameters derived from Mondays to Wednesdays on Thursdays and Fridays and similarly repeat this process based on different sets of learning days and testing days of the week. This analogized cross-validation ensures the selected schedule works for the whole week.

In summary, cross-validation derives a more accurate assess of model performance by combining measures of prediction performance on different subsets of data. This technique not only reduces variances and avoids overfitting, but also gives an insight into how the model will generally perform in practice.

Avoiding overfitting with regularization

Another way of preventing overfitting is **regularization.** Recall that the unnecessary complexity of the model is a source of overfitting. Regularization adds extra parameters to the error function we're trying to minimize, in order to penalize complex models.

According to the principle of Occam's Razor, simpler methods are to be favored. William Occam was a monk and philosopher who, in around the year 1320, came up with the idea that the simplest hypothesis that fits data should be preferred. One justification is that we can invent fewer simple models than complex models. For instance, intuitively, we know that there are more high-polynomial models than linear ones. The reason is that a line ($y=ax+b$) is governed by only two parameters—the intercept b and slope a. The possible coefficients for a line span two-dimensional space. A quadratic polynomial adds an extra coefficient for the quadratic term, and we can span a three-dimensional space with the coefficients. Therefore, it is much easier to find a model that perfectly captures all training data points with a **High order polynomial function**, as its search space is much larger than that of a linear function. However, these easily obtained models generalize worse than linear models, which are more prompt to overfitting. And, of course, simpler models require less computation time. The following diagram displays how we try to fit a **Linear function** and a **High order polynomial function** respectively to the data:



The linear model is preferable as it may generalize better to more data points drawn from the underlying distribution. We can use regularization to reduce the influence of the high orders of polynomial by imposing penalties on them. This will discourage complexity, even though a less accurate and less strict rule is learned from the training data.

We'll employ regularization quite often starting from `Chapter 7`, *Predicting Online Ads Click-Through with Logistic Regression*. For now, next let's see an analogy to help us to understand it better.

A data scientist wants to equip his robotic guard dog with the ability to identify strangers and his friends. He feeds it with the the following learning samples:

Male	Young	Tall	With glasses	In grey	**Friend**
Female	Middle	Average	Without glasses	In black	**Stranger**
Male	Young	Short	With glasses	In white	**Friend**
Male	Senior	Short	Without glasses	In black	**Stranger**
Female	Young	Average	With glasses	In white	**Friend**
Male	Young	Short	Without glasses	In red	**Friend**

The robot may quickly learn the following rules:

- Any middle-aged female of average height without glasses and dressed in black is a stranger
- Any senior short male without glasses and dressed in black is a stranger
- Anyone else is his friend

Although these perfectly fit the training data, they seem too complicated and unlikely to generalize well to new visitors. In contrast, the data scientist limits the learning aspects. A loose rule that can work well for hundreds of other visitors could be: anyone without glasses dressed in black is a stranger.

Besides penalizing complexity, we can also stop a training procedure early as a form of regularization. If we limit the time a model spends learning or we set some internal stopping criteria, it's more likely to produce a simpler model. The model complexity will be controlled in this way and hence overfitting becomes less probable. This approach is called **early stopping** in machine learning.

Last but not least, it's worth noting that regularization should be kept at a moderate level or, to be more precise, fine-tuned to an optimal level. Too small a regularization doesn't make any impact; too large a regularization will result in underfitting, as it moves the model away from the ground truth. We'll explore how to achieve optimal regularization in `Chapter 7`, *Predicting Online Ads Click-Through with Logistic Regression*, and `Chapter 9`, *Stock Price Prediction with Regression Algorithms*.

Avoiding overfitting with feature selection and dimensionality reduction

We typically represent data as a grid of numbers (a **matrix**). Each column represents a variable, which we call a **feature** in machine learning. In supervised learning, one of the variables is actually not a feature, but the label that we're trying to predict. And in supervised learning, each row is an example that we can use for training or testing.

The number of features corresponds to the dimensionality of the data. Our machine learning approach depends on the number of dimensions versus the number of examples. For instance, text and image data are very high dimensional, while stock market data has relatively fewer dimensions.

Fitting high-dimensional data is computationally expensive and is prone to overfitting due to the high complexity. Higher dimensions are also impossible to visualize, and therefore we can't use simple diagnostic methods.

Not all of the features are useful and they may only add randomness to our results. It's therefore often important to do good feature selection. **Feature selection** is the process of picking a subset of significant features for use in better model construction. In practice, not every feature in a dataset carries information useful for discriminating samples; some features are either redundant or irrelevant, and hence can be discarded with little loss.

In principle, feature selection boils down to multiple binary decisions about whether to include a feature or not. For *n* features, we get 2^n feature sets, which can be a very large number for a large number of features. For example, for 10 features, we have 1,024 possible feature sets (for instance, if we're deciding what clothes to wear, the features can be temperature, rain, the weather forecast, where we're going, and so on). At a certain point, brute force evaluation becomes infeasible. We'll discuss better methods in `Chapter 6`, *Predicting Online Ads Click-Through with Tree-Based Algorithms*. Basically, we have two options: we either start with all of the features and remove features iteratively or we start with a minimum set of features and add features iteratively. We then take the best feature sets for each iteration and compare them.

We'll explore how to perform feature selection mainly in `Chapter 7`, *Predicting Online Ads Click-Through with Logistic Regression*.

Another common approach of reducing dimensionality is to transform high-dimensional data in lower-dimensional space. It's called **dimensionality reduction** or **feature projection**. This transformation leads to information loss, but we can keep the loss to a minimum.

We'll talk about and implement dimensionality reduction in `Chapter 2`, *Exploring the 20 Newsgroups Dataset with Text Analysis Techniques*, `Chapter 3`, *Mining the 20 Newsgroups Dataset with Clustering and Topic Modeling Algorithms, and* `chapter 10`, *Machine Learning Best Practices*

Preprocessing, exploration, and feature engineering

Data mining, a buzzword in the 1990s, is the predecessor of data science (the science of data). One of the methodologies popular in the data mining community is called **Cross-Industry Standard Process for Data Mining** (**CRISP-DM**). CRISP-DM was created in 1996 and is still used today. I'm not endorsing CRISP-DM, however, I do like its general framework.

The CRISP DM consists of the following phases, which aren't mutually exclusive and can occur in parallel:

- **Business understanding**: This phase is often taken care of by specialized domain experts. Usually, we have a business person formulate a business problem, such as selling more units of a certain product.
- **Data understanding**: This is also a phase that may require input from domain experts, however, often a technical specialist needs to get involved more than in the business understanding phase. The domain expert may be proficient with spreadsheet programs, but have trouble with complicated data. In this book, it's usually termed as **phase exploration**.
- **Data preparation**: This is also a phase where a domain expert with only Microsoft Excel knowledge may not be able to help you. This is the phase where we create our training and test datasets. In this book, it's usually termed as **phase preprocessing**.
- **Modeling**: This is the phase most people associate with machine learning. In this phase, we formulate a model and fit our data.
- **Evaluation**: In this phase, we evaluate how well the model fits the data to check whether we were able to solve our business problem.
- **Deployment**: This phase usually involves setting up the system in a production environment (it's considered good practice to have a separate production system). Typically, this is done by a specialized team.

When we learn, we require high-quality learning material. We can't learn from gibberish, so we automatically ignore anything that doesn't make sense. A machine learning system isn't able to recognize gibberish, so we need to help it by cleaning the input data. It's often claimed that cleaning the data forms a large part of machine learning. Sometimes cleaning is already done for us, but you shouldn't count on it.

To decide how to clean the data, we need to be familiar with the data. There are some projects that try to automatically explore the data and do something intelligent, such as produce a report. For now, unfortunately, we don't have a solid solution, so you need to do some manual work.

We can do two things, which aren't mutually exclusive: first, scan the data and second, visualize the data. This also depends on the type of data we're dealing with—whether we have a grid of numbers, images, audio, text, or something else. In the end, a grid of numbers is the most convenient form, and we'll always work toward having numerical features. Let's pretend that we have a table of numbers in the rest of this section.

We want to know whether features have missing values, how the values are distributed, and what type of features we have. Values can approximately follow a normal distribution, a binomial distribution, a Poisson distribution, or another distribution altogether. Features can be binary: either yes or no, positive or negative, and so on. They can also be categorical: pertaining to a category, for instance, continents (Africa, Asia, Europe, Latin America, North America, and so on). Categorical variables can also be ordered, for instance, high, medium, and low. Features can also be quantitative, for example, temperature in degrees or price in dollars.

Feature engineering is the process of creating or improving features. It's more of a dark art than a science. Features are often created based on common sense, domain knowledge, or prior experience. There are certain common techniques for feature creation, however, there's no guarantee that creating new features will improve your results. We're sometimes able to use the clusters found by unsupervised learning as extra features. **Deep neural networks** are often able to derive features automatically. We'll briefly look at several techniques such as polynomial features, power transformations, and binning, as appetizers in this chapter.