MACHINE LEARNING

ETHEM ALPAYDIN

010001010111010001101000
011001010110110100100000
010000010110110001110000
011000010111100101100100
011010010110111000001101
000010100100110101100001
011000110110100001101001
011011100110010100100000
010011000110010101100001
011100100110111001101001
011011100110011100001101
000010100101010001101000
011001010010000001001110
011001010111011100100000
010000010100100100001101
00001010

THE MIT PRESS ESSENTIAL KNOWLEDGE SERIES

MACHINE LEARNING

010001010111010001101000
011001010110110100100000
010000010110110001110000
011000010111100101100100
011010010110111000001101
000010100100110101100001
011000110110100001101001
011011100110010100100000
010011000110010101100001
011100100110111001101001
011011100110011100001101
000010100101010001101000
011001010010000001001110
011001010111011100100000
010000010100100100001101
00001010

The MIT Press Essential Knowledge Series

Auctions, Timothy P. Hubbard and Harry J. Paarsch
Cloud Computing, Nayan Ruparelia
Computing: A Concise History, Paul E. Ceruzzi
The Conscious Mind, Zoltan L. Torey
Crowdsourcing, Daren C. Brabham
Free Will, Mark Balaguer
Information and Society, Michael Buckland
Information and the Modern Corporation, James W. Cortada
Intellectual Property Strategy, John Palfrey
The Internet of Things, Samuel Greengard
Machine Learning: The New AI, Ethem Alpaydın
Memes in Digital Culture, Limor Shifman
Metadata, Jeffrey Pomerantz
The Mind–Body Problem, Jonathan Westphal
MOOCs, Jonathan Haber
Neuroplasticity, Moheb Costandi
Open Access, Peter Suber
Paradox, Margaret Cuonzo
Robots, John Jordan
Self-Tracking, Gina Neff and Dawn Nafus
Sustainability, Kent E. Portney
The Technological Singularity, Murray Shanahan
Understanding Beliefs, Nils J. Nilsson
Waves, Frederic Raichlen

MACHINE LEARNING

THE NEW AI

ETHEM ALPAYDIN

The MIT Press | Cambridge, Massachusetts | London, England

© 2016 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

This book was set in Chaparral and DIN by Toppan Best-set Premedia Limited.
Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Names: Alpaydın, Ethem, author.
Title: Machine learning : the new AI / Ethem Alpaydın.
Description: Cambridge, MA : MIT Press, [2016] | Series: MIT Press essential knowledge series | Includes bibliographical references and index.
Identifiers: LCCN 2016012342 | ISBN 9780262529518 (pbk. : alk. paper)
Subjects: LCSH: Machine learning. | Artificial intelligence.
Classification: LCC Q325.5 .A47 2016 | DDC 006.3/1—dc23 LC record available at https://lccn.loc.gov/2016012342

10 9 8 7 6 5 4 3 2 1

CONTENTS

Series Foreword vii
Preface ix

1 Why We Are Interested in Machine Learning 1
2 Machine Learning, Statistics, and Data Analytics 29
3 Pattern Recognition 55
4 Neural Networks and Deep Learning 85
5 Learning Clusters and Recommendations 111
6 Learning to Take Actions 125
7 Where Do We Go from Here? 141

Notes 169
Glossary 171
References 183
Further Readings 187
Index 189

SERIES FOREWORD

The MIT Press Essential Knowledge series offers accessible, concise, beautifully produced pocket-size books on topics of current interest. Written by leading thinkers, the books in this series deliver expert overviews of subjects that range from the cultural and the historical to the scientific and the technical.

In today's era of instant information gratification, we have ready access to opinions, rationalizations, and superficial descriptions. Much harder to come by is the foundational knowledge that informs a principled understanding of the world. Essential Knowledge books fill that need. Synthesizing specialized subject matter for nonspecialists and engaging critical topics through fundamentals, each of these compact volumes offers readers a point of access to complex ideas.

Bruce Tidor
Professor of Biological Engineering and Computer Science
Massachusetts Institute of Technology

PREFACE

A quiet revolution has been taking place in computer science for the last two decades. Nowadays, more and more, we see computer programs that learn—that is, software that can adapt their behavior automatically to better match the requirements of their task. We now have programs that learn to recognize people from their faces, understand speech, drive a car, or recommend which movie to watch—with promises to do more in the future.

Once, it used to be the programmer who defined what the computer had to do, by coding an algorithm in a programming language. Now for some tasks, we do not write programs but collect data. The data contains instances of what is to be done, and the learning algorithm modifies a learner program automatically in such a way so as to match the requirements specified in the data.

Since the advent of computers in the middle of the last century, our lives have become increasingly computerized and digital. Computers are no longer just the numeric calculators they once were. Databases and digital media have taken the place of printing on paper as the main medium of information storage, and digital communication over computer networks supplanted the post as the main mode of information transfer. First with the personal computer with its easy-to-use graphical interface, and then with the phone and other smart devices, the computer has become a ubiquitous device, a household appliance just like the TV or the microwave. Nowadays, all sorts of information, not only numbers and text but also image, video, audio, and so on, are stored, processed, and—thanks to online connectivity—transferred digitally. All this digital processing results in a lot of data, and it is this surge of data—what we can call a "dataquake"—that is mainly responsible for triggering the widespread interest in data analysis and machine learning.

For many applications—from vision to speech, from translation to robotics—we were not able to devise very good algorithms despite decades of research beginning in the 1950s. But for all these tasks it is easy to collect data, and now the idea is to learn the algorithms for these automatically from data, replacing programmers with learning programs. This is the niche of machine learning, and it is not only that the data continuously has got bigger in these last two decades, but also that the theory of machine learning to process that data to turn it into knowledge has advanced significantly.

Today, in different types of business, from retail and finance to manufacturing, as our systems are computerized, more data is continuously generated and collected. This is also true in various fields of science, from astronomy to biology. In our everyday lives too, as digital technology increasingly infiltrates our daily existence, as our digital footprint deepens, not only as consumers and users but also through social media, an increasingly larger part of our lives is recorded and becomes data. Whatever its source—business, scientific, or personal—data that just lies dormant passively is not of any use, and smart people have been finding new ways to make use of that data and turn it into a useful product or service. In this transformation, machine learning is playing a more significant role.

Our belief is that behind all this seemingly complex and voluminous data, there lies a simple explanation. That although the data is big, it can be explained in terms of a relatively simple model with a small number of hidden factors and their interaction. Think about millions of customers who buy thousands of products online or from their local supermarket every day. This implies a very large database of transactions; but what saves us and works to our advantage is that there is a pattern to this data. People do not shop at random. A person throwing a party buys a certain subset of products, and a person who has a baby at home buys a different subset—there are hidden factors that explain customer behavior. It is this inference of a hidden model—namely, the underlying factors and their interaction—from the observed data that is at the core of machine learning.

Machine learning is not just the commercial application of methods to extract information from data; learning is also a requisite of intelligence. An intelligent system should be able to adapt to its environment; it should learn not to repeat its mistakes but to repeat its successes. Previously, researchers used to believe that for artificial intelligence to become reality, we needed a new paradigm, a new type of thinking, a new model of computation, or a whole new set of algorithms. Taking into account the recent successes in machine learning in various domains, it can now be claimed that what we need is not a set of new specific algorithms but a lot of example data and sufficient computing power to run the learning methods on that much data, bootstrapping the necessary algorithms from data.

It may be conjectured that tasks such as machine translation and planning can be solved with such learning algorithms that are relatively simple but trained on large amounts of example data—recent successes with "deep learning" support this claim. Intelligence seems not to originate from some outlandish formula, but rather from the patient, almost brute force use of simple, straightforward algorithms.

It seems that as technology develops and we get faster computers and more data, learning algorithms will generate a slightly higher level of intelligence, which will find use in a new set of slightly smarter devices and software. It will not be surprising if this type of learned intelligence reaches the level of human intelligence some time before this century is over.

While I was working on this book, one of the most prestigious scientific journals, *Science*, in its July 15, 2015, issue (vol. 349, no. 6245), published a special section on Artificial Intelligence. Though the title announces a focus on artificial intelligence, the dominant theme is machine learning. This is just another indicator that machine learning is now the driving force in artificial intelligence; after the disappointment of logic-based, programmed expert systems in 1980s, it has revived the field, delivering significant results.

The aim of this book is to give the reader an overall idea about what machine learning is, the basics of some important learning algorithms, and a set of example applications. The book is intended for a general readership, and only the essentials of the learning methods are discussed without any mathematical or programming details. The book does not cover any of the machine-learning applications in much detail either; a number of examples are discussed just enough to give the fundamentals without going into the particulars.

For more information on the machine learning algorithms, the reader can refer to my textbook on the topic, on which this book is heavily based: Ethem Alpaydın, *Introduction to Machine Learning*, 3rd ed. (Cambridge, MA: MIT Press, 2014).

The content is organized as follows:

In chapter 1, we discuss briefly the evolution of computer science and its applications, to place in context the current state of affairs that created the interest in machine learning—namely, how the digital technology advanced from number-crunching mainframes to desktop personal computers and later on to smart devices that are online and mobile.

Chapter 2 introduces the basics of machine learning and discusses how it relates to model fitting and statistics on some simple applications.

Most machine learning algorithms are supervised, and in chapter 3, we discuss how such algorithms are used for pattern recognition, such as faces and speech.

Chapter 4 discusses artificial neural networks inspired from the human brain, how they can learn, and how "deep," multilayered networks can learn hierarchies at different levels of abstractions.

Another type of machine learning is unsupervised, where the aim is to learn associations between instances. In chapter 5 we talk about customer segmentation and learning recommendations, as popular applications.

Chapter 6 is on reinforcement learning where an autonomous agent—for example, a self-driving car—learns to take actions in an environment to maximize reward and minimize penalty.

Chapter 7 concludes by discussing some future directions and the newly proposed field of “data science” that also encompasses high-performance cloud computing. We also discuss the ethical and legal implications, such as data privacy and security.

This book aims to give a quick introduction to what is being done nowadays in machine learning, and my hope is to trigger the reader’s interest in thinking about what can be done in the future. Machine learning is certainly one of the most exciting scientific fields today, advancing technology in various domains, and it has already generated a set of impressive applications affecting all walks of life. I have enjoyed very much writing this book; I hope you will enjoy reading it too!

I am grateful to the anonymous reviewers for their constructive comments and suggestions. As always, it has been a pleasure to work with the MIT Press and I would like to thank Kathleen Caruso, Kathleen Hensley, and Marie Lufkin Lee for all their support.

1

WHY WE ARE INTERESTED IN MACHINE LEARNING

The Power of the Digital

Some of the biggest transformations in our lives in the last half century are due to computing and digital technology. The tools, devices, and services we had invented and developed in the centuries before have been increasingly replaced by their computerized “e-” versions, and we in turn have been continuously adapting to this new digital environment.

This transformation has been very fast: once upon a time—fifty years ago is mythical past in the digital realm where things happen at the speed of light—computers were expensive and only very large organizations, such as governments, big companies, universities, research centers, and so on, could afford them. At that time, only they had problems difficult enough to justify the high cost of procuring and maintaining a computer. Computer "centers," in separate floors or buildings, housed those power-hungry behemoths, and inside large halls, magnetic tapes whirred, cards were punched, numbers were crunched, and bugs were real bugs.

As computers became cheaper, they became available to a larger selection of the population and in parallel, their application areas widened. In the beginning, computers were nothing but calculators—they added, subtracted, multiplied, and divided numbers to get new numbers. Probably the major driving force of the computing technology is the realization that every piece of information can be represented as numbers. This in turn implies that the computer, which until then was used to process numbers, can be used to process *all* types of information.

To be more precise, a computer represents every number as a particular sequence of binary digits (bits) of 0 or 1, and such bit sequences can also represent other types of information. For example, "101100" can be used to represent the decimal 44 *and* is also the code for the comma; likewise, "1000001" is both 65 and the uppercase letter 'A'.[1] Depending on the context, the computer program manipulates the sequence according to one of the interpretations.

Actually, such bit sequences can stand for not only numbers and text, but also other types of information—for example, colors in a photo or tones in a song. Even computer programs are sequences of bits. Furthermore, and just as important, operations associated with these types of information, such as making an image brighter or finding a face in a photo, can be converted to commands that manipulate bit sequences.

Computers Store Data

The power of the computer lies in the fact that every piece of information can be represented digitally—that is, as a sequence of bits—and every type of information processing can be written down as computer instructions that manipulate such digital representations.

One consequence of this emerged in the 1960s with *databases*, which are specialized computer programs that store and manipulate *data*, or digitally represented information. Peripheral devices, such as tapes or disks, store bits magnetically, and hence their contents are not erased when the computer is switched off.

With databases, computers have moved beyond processing and have become repositories of information using the digital representation. In time, the digital medium has become so fast, cheap, and reliable that it has supplanted printing on paper as humanity's main means of information storage.

Since the invention of the microprocessor and parallel advances in miniaturization and decreasing costs, starting in the early 1980s, personal computers have become increasingly available. The *personal computer* has made computing accessible to small businesses, but most important the personal computer was small and cheap enough to be a household appliance. You did not need to be a large company; the computer could help with your life too. The personal computer confirmed that everyone had tasks that are computer-worthy, and the growth of applications followed this era of democratization of digital technology.

Graphical user interfaces and the mouse made personal computers easy to use. We do not need to learn programming, or memorize commands with a difficult syntax, to be able to use the computer. The screen is a digital simulacrum of our work environment displaying a virtual desktop, with files, icons, and even a trash can, and the mouse is our virtual hand that picks them up to read or edit them.

The software for the personal computer in parallel has moved from commercial to personal applications by manipulating more types of data and making more of our lives digital. We have a word processor for our letters and other personal documents, a spreadsheet for our household calculations, and software for our hobbies such as music or photography; we can even play games if we want to! Computing has become everyday and fun.

The personal computer with its pleasant and inviting user interface coupled with a palette of everyday applications was a big step in the rapprochement between people and computers, our life as we used to know it, and the digital world. Computers were programmed to fit our lives a little better, and we have adapted a little to accommodate them. In time, using a computer has become a basic skill, like driving.

The personal computer was the first step in making computers accessible to the masses; it made digital technology a larger part of our lives and, most important for our story in this book, allowed more of our lives to be recorded digitally. As such, it was a significant stepping-stone in this process of converting our lives to data, data that we can then analyze and learn from.

Computers Exchange Data

The next major development in computing was in connectivity. Though hooking up computers by data links to exchange information had been done before, commercial systems started to become widespread in 1990s to connect personal computers to each other or to central servers over phone or dedicated lines.

The computer network implies that a computer is no longer isolated, but can exchange data with a computer far away. A user is no longer limited to their own data on their own computer but can access data elsewhere, and if they want, they can make their data available to other users.

The development of computer networks very quickly culminated in the Internet, which is *the* computer network that covers the whole globe. The Internet made it possible for anyone in the world who has access to a computer to send information, such as an email, to anyone else. And because all our data and devices are already digital, the information we can share is more than just text and numbers; we can send images, videos, music, speech, anything.

With computer networks, digitally represented information can be sent at the speed of light to anyone, anywhere. The computer is no longer just a machine where data is stored and processed, but it has also become a means to transfer and share information. Connectivity increased so quickly and digital communication has become so cheap, fast, and reliable that digital transfer has supplanted mail as the main technology for information transfer.

Anyone who is “online” can make their own data on their own computer available over the network to anyone else, which is how the World Wide Web was born. People can “surf” the Web and browse this shared information. Very quickly, secure protocols have been implemented to share confidential information, thereby permitting commercial transactions over the Web, such as online shopping or banking. Online connectivity has further increased the infiltration of digital technology. When we get an online service by using the “www.” portal of the service provider, our computer turns into the digital version of the shop, the bank, the library, or the university; this, in turn, created more data.

Mobile Computing

Every decade we have been seeing computers getting smaller, and with advances in battery technology, in the mid-1990s, portable—laptop or notebook—computers that can also run on batteries started to become widespread; this started the new era of *mobile computing*. Cellular phones also started to become popular around the same time, and roughly around 2005, these two technologies merged in the smartphone.

A *smartphone* is a phone that is also a computer. In time, the smartphone became smarter—more a computer and less a phone—so much so that nowadays, the phone is only one of many apps on a smartphone, and a rarely used one at that. The traditional phone was an acoustic device: you talked into it, and you heard the person on the other end talking. The smartphone today is more of a visual device; it has a large screen and we spend more time looking at this screen or tapping its touch-sensitive surface than talking.

A smartphone is a computer that is always online,[2] and it allows its user to access the Internet for all types of information while mobile. It therefore extends our connectivity in that it permits us greater access—for example, while traveling—to data on other computers, and furthermore, it also makes us, and our data, more accessible to others.

What makes a smartphone special is that it is also a mobile sensing device and because it is always on our person, it continuously records information about its user, most notably their position, and can make this data available. The smartphone is a mobile sensor that makes us detectable, traceable, recordable.

This increased mobility of the computer is new. Once the computer was big and at a "computer center"; it stayed fixed, and we needed to walk to it. We sat in front of a terminal to use the computer—it was called a "terminal" because the computer ended there. Then a smaller computer came to our department, and later a smaller one sat on our desk in our office or in our house, and then an even smaller one was on our lap, and now the computer is in our pocket and with us all the time.

Once there were very few computers, possibly one computer per thousands of people—for example, one per company or campus. This computer-per-person ratio increased very quickly, and the personal computer aimed to have one computer for every person. Today we have many computers per person. Now all our devices are also computers or have computers in them. Your phone is also a computer, your TV is also a computer, your car has many computers inside it for different functions, and your music player is a specialized computer as is your camera or your watch. The *smart device* is a computer that does the digital version of whatever it did originally.

Ubiquitous computing is a term that is becoming increasingly popular; it means using computers without knowing that you are using one. It means using a lot of computers for all sorts of purposes all the time without explicitly calling them computers. The digital version has its usual advantages, such as speed, accuracy, and easy adaptability. But another advantage is that the digital version of the device now has all its data digitally. And furthermore, if it is online, it can talk to other online computers and make its data available almost instantaneously. We call them "smart objects" or just "things" and talk about the *Internet of Things*.

Social Data

A few thousands of years ago, you needed to be a god or goddess if you wanted to be painted, be sculpted, or have your story remembered and told. A thousand years ago you needed to be a king or queen, and a few centuries ago you needed to be a rich merchant, or in the household of one. Now anybody, even a soup can, can be painted. A similar democratization has also taken place in computing and data. Once only large organizations and businesses had tasks worthy of a computer and hence only they had data; starting with the personal computer, people and even objects became generators of data.

A recent source of data is *social media*, where our social interactions have become digital; these now constitute another type of data that can be collected, stored, and analyzed. Social media replaces discussions in the agora, piazza, market, coffeehouse, and pub, or at the gathering by the spring, the well, and the water cooler.

With social media, each of us is now a celebrity whose life is worth following, and we are our own paparazzi. We are no longer allotted only fifteen minutes of fame, but every time we are online we are famous. The social media allows us to write our digital autobiography as we are living it. In the old times, books and newspapers were expensive and hence scarce; we could keep track of and tell the story of only important lives. Now data is cheap and we are all kings and queens of our little online fiefdoms. A baby born to gadget-loving parents today can generate more data in her first month than it took for Homer to narrate the complete adventures of Odysseus.

All That Data: The Dataquake

The data generated by all our computerized machines and services was once a by-product of digital technology, and computer scientists have done a lot of research on databases to efficiently store and manipulate large amounts of data. We stored data because we had to. Sometime in the last two decades, all this data became a resource; now, more data is a blessing.

Think, for example, of a supermarket chain that sells thousands of goods to millions of customers every day, either at one of the numerous brick-and-mortar stores all over a country or through a virtual store over the Web. The point-of-sales terminals are digital and record the details of each transaction: data, customer id (through some loyalty program), goods bought and at what price, total money spent, and so forth. The stores are connected online, and the data from all the terminals in all the stores can be instantaneously collected in a central database. This amounts to a lot of (and very up-to-date) data every day.

Especially in the last twenty years or so, people have increasingly started to ask themselves what they can do with all this data. With this question the whole direction of computing is reversed. Before, data was what the programs processed and spit out—data was passive. With this question, data starts to drive the operation; it is not the programmers anymore but the data itself that defines what to do next.

One thing that a supermarket chain is always eager to know is which customer is likely to buy which product.

Data starts to drive the operation; it is not the programmers anymore but the data itself that defines what to do next.

With this knowledge, stores can be stocked more efficiently, which will increase sales and profit. It will also increase customer satisfaction because customers will be able to find the products that best match their needs quicker and cheaper.

This task is not evident. We do not know exactly which people are likely to buy this ice cream flavor or the next book of this author, to see this new movie, or to visit this city. Customer behavior changes in time and depends on geographic location.

But there is hope, because we know that customer behavior is not completely random. People do not go to supermarkets and buy things at random. When they buy beer, they buy chips; they buy ice cream in summer and spices for Glühwein in winter. There are certain patterns in customer behavior, and that is where data comes into play.

Though we do not know the customer behavior patterns themselves, we expect to see them occurring in the collected data. If we can find such patterns in past data, assuming that the future, at least the near future, will not be much different from the past when that data was collected, we could expect them to continue to hold, and we can make predictions based on them.

We may not be able to identify the process completely, but we believe we can construct *a good and useful approximation*. That approximation may not explain everything, but may still be able to account for some part of the data.

We believe that though identifying the complete process may not be possible, we can still detect some patterns. We can use those patterns to predict; they may also help us understand the process.

This is called *data mining*. The analogy is that a large volume of earth and raw material is extracted from the mine, which when processed leads to a small amount of very precious material. Similarly in data mining, a large volume of data is processed to construct a simple model with valuable use, such as having high predictive accuracy.

Data mining is one type of machine learning. We do not know the rules (of customer behavior), so we cannot write the program, but the machine—that is, the computer—"learns" by extracting such rules from (customer transaction) data.

Many applications exist where we do not know the rules but have a lot of data. The fact that our businesses use computers and digital technology implies that there are large amounts of data in all sorts of domains. We also use computers or smart machines in our daily social life too, so we have the data for that too.

Learning models are used in pattern recognition, for example, in recognizing images captured by a camera or recognizing speech captured by a microphone. Nowadays we have different types of sensors that we use for different type of applications, from human activity recognition using a smartphone to driving assistance systems in cars.

Another source of data is science. As we build better sensors, we detect more—that is, we get more data—in astronomy, biology, physics, and so on, and we use learning algorithms to make sense of the bigger and bigger data. The Internet itself is one huge data repository, and we need smart algorithms to help us find what we are looking for. One important characteristic of data we have today is that it comes from different modalities—it is multimedia. We have text, we have images or video, we have sound clips, and so on, all somehow related to the same object or event we are interested in, and a major challenge in machine learning today is to combine information coming from these different sources. For example, in consumer data analysis, in addition to past transactions, we also have Web logs—namely, the Web pages that a user has visited recently—and these logs may be quite informative.

With the number of smart machines continuously helping us in our daily lives, we all became producers of data. Every time we buy a product, every time we rent a movie, visit a Web page, write a blog, or post on the social media, even when we just walk or drive around, we are generating data. And that data is valuable for someone who is interested in collecting and analyzing it. The customer is not only always right, but also interesting and worth following.

Each of us is not only a generator but also a consumer of data. We want to have products and services specialized for us. We want our needs to be understood and our interests to be predicted.

Learning versus Programming

To solve a problem on a computer, we need an algorithm.[3] An *algorithm* is a sequence of instructions that are carried out to transform the input to the output. For example, one can devise an algorithm for sorting. The input is a set of numbers and the output is their ordered list. For the same task, there may be various algorithms and we may be interested in finding the most efficient one, the one requiring the least number of instructions, memory, or both.

For some problems, however, we do not have an algorithm. Predicting customer behavior is one; another is diffentiating spam emails from legitimate ones. We know what the input is: an email document that in the simplest case is a text message. We know what the output should be: a yes/no output indicating whether the message is spam or not. But we do not know how to transform the input to the output. What is considered spam changes over time and from individual to individual.

What we lack in knowledge, we make up for in data. We can easily compile thousands of messages, some of which we know to be spam and some of which are not, and what we want is to “learn” what constitutes spam from this sample. In other words, we would like the computer (the machine) to extract automatically the algorithm for this task. There is no need to learn to sort numbers (we already have algorithms for that), but there are many applications for which we do not have an algorithm but have lots of data.

Artificial Intelligence

Machine learning is not just a database or programming problem; it is also a requirement for artificial intelligence. A system that is in a changing environment should have the ability to learn; otherwise, we would hardly call it intelligent. If the system can learn and adapt to such changes, the system designer need not foresee and provide solutions for all possible situations.

For us, the system designer was evolution, and our body shape as well as our built-in instincts and reflexes have evolved over millions of years. We also learn to change our behavior during our lifetime. This helps us cope with changes in the environment that cannot be predicted by evolution. Organisms that have a short life in a well-defined environment may have all their behavior built-in, but instead of hardwiring into us all sorts of behavior for any circumstance that we might encounter in our life, evolution gave us a large brain and a mechanism to learn such that we could update ourselves with experience and adapt to different environments. That is why human beings have survived and prospered in different parts of the globe in very different climates and conditions. When we learn the best strategy in a certain situation that knowledge is stored in our brain, and when the situation arises again—when we recognize (“cognize” means to know) the situation—we recall the suitable strategy and act accordingly.

Each of us, actually every animal, is a data scientist. We collect data from our sensors, and then we process the data to get abstract rules to perceive our environment and control our actions in that environment to minimize pain and/or maximize pleasure. We have memory to store those rules in our brains, and then we recall and use them when needed. Learning is lifelong; we forget rules when they no longer apply or revise them when the environment changes.

Learning has its limits though; there may be things that we can never learn with the limited capacity of our brains, just like we can never “learn” to grow a third arm, or an eye in the back of our head—something that would require a change in our genetic makeup. Roughly speaking, genetics defines the hardware working over thousands of generations, whereas learning defines the software running on (and being constrained by) that hardware during an individual’s lifetime.

Artificial intelligence takes inspiration from the brain. There are cognitive scientists and neuroscientists whose aim is to understand the functioning of the brain, and toward this aim, they build models of neural networks and make simulation studies. But artificial intelligence is a part of computer science and our aim is to build useful systems, as in any domain of engineering. So though the brain inspires us, ultimately we do not care much about the biological plausibility of the algorithms we develop.

We are interested in the brain because we believe that it may help us build better computer systems. The brain is an information-processing device that has some incredible abilities and surpasses current engineering products in many domains—for example, vision, speech recognition, and learning, to name three. These applications have evident economic utility if implemented on machines. If we can understand how the brain performs these functions, we can define solutions to these tasks as formal algorithms and implement them on computers.

Computers were once called "electronic brains," but computers and brains are different. Whereas a computer generally has one or few processors, the brain is composed of a very large number of processing units, namely, neurons, operating in parallel. Though the details are not completely known, the processing units are believed to be much simpler and slower than a typical processor in a computer. What also makes the brain different, and is believed to provide its computational power, is its large connectivity. Neurons in the brain have connections, called synapses, to tens of thousands of other neurons, and they all operate in parallel. In a computer, the processor is active and the memory is separate and passive, but it is believed that in the brain both processing and memory are distributed together over the network; processing is done by the neurons and memory occurs in the synapses between the neurons.

Understanding the Brain

According to Marr (1982), understanding an information processing system works at three *levels of analysis*:

1. *Computational theory* corresponds to the goal of computation and an abstract definition of the task.
2. *Representation and algorithm* is about how the input and the output are represented, and about the specification of the algorithm for the transformation from the input to the output.
3. *Hardware implementation* is the actual physical realization of the system.

The basic idea in these levels of analysis is that for the same computational theory, there may be multiple representations and algorithms manipulating symbols in that representation. Similarly, for any given representation and algorithm, there may be multiple hardware implementations. For any theory, we can use one of various algorithms, and the same algorithm can have different hardware implementations.

Let us take an example: ‘6’, ‘VI’, and ‘110’ are three different representations of the number six; respectively, they are the Arabic, Roman, and binary representations. There is a different algorithm for addition depending on the representation used. Digital computers use the binary representation and have circuitry to add in this representation, which is one particular hardware implementation. Numbers are represented differently, and addition corresponds to a different set of instructions on an abacus, which is another hardware implementation. When we add two numbers “in our head,” we use another representation and an algorithm suitable to that representation, which is implemented by the neurons. But all these different hardware implementations—namely, us, abacus, digital computer—implement the same computational theory: addition.

The classic example is the difference between natural and artificial flying machines. A sparrow flaps its wings; an airplane does not flap its wings but uses jet engines. The sparrow and the airplane are two hardware implementations built for different purposes, satisfying different constraints. But they both implement the same theory, which is aerodynamics.

From this perspective, we can say that the brain is one hardware implementation for learning. If from this particular implementation, we can do reverse engineering and extract the representation and the algorithm used, and if from that in turn we can get the computational theory, we can then use another representation and algorithm, and in turn a hardware implementation more suited to the means and constraints we have. One hopes our implementation will be cheaper, faster and more accurate.

Just as the initial attempts to build flying machines looked a lot like birds until we discovered the theory of aerodynamics, it is also expected that the first attempts to build structures possessing the brain's abilities will look like the brain with networks of large numbers of processing units. Indeed, in chapter 4 we are going to discuss artificial neural networks that are composed of interconnected processing units and how such networks can learn—this is the representation and algorithm level. In time, when we discover the computational theory of intelligence, we may discover that neurons and synapses are implementation details, just as we have realized that feathers are for flying.

Pattern Recognition

In computer science, there are many tasks for which we have tried to devise "expert systems" programmed with manually specified rules and algorithms. Decades of work have led to very limited success. Some of these tasks relate to artificial intelligence in that they are believed to require intelligence. The current approach, in which we have seen tremendous progress recently, is to use machine learning from data.

Let us take the example of recognizing faces: this is a task that we do effortlessly; every day we recognize family members and friends by looking at their faces or from their photographs, despite differences in pose, lighting, hairstyle, and so forth. Face perception is important for us because we want to tell friend from foe. It was important for our survival not only for identification, but also because the face is the dashboard of our internal state. Feelings such as happiness, anger, surprise, and shame can be read from our face, and we have evolved both to display such states as well as to detect it in others.

Though we do such recognition easily, we do it unconsciously and are unable to explain how we do it. Because we are not able to explain our expertise, we cannot write the corresponding computer program.

By analyzing different face images of a person, a learning program captures the pattern specific to that person and then checks for that pattern in a given image. This is one example of *pattern recognition*.

The reason we can do this is because we know that a face image, just like any natural image, is not just a random collection of pixels (a random image would be like a snowy TV). A face has structure. It is symmetric. The eyes, the nose, and the mouth are located in certain places on the face. Each person's face is a pattern composed of a particular combination of these. When the illumination or pose changes, when we grow our hair or put on glasses, or when we age, certain parts of the face image change but some parts do not. This is similar to customer behavior in that there are items we buy regularly and also impulse buys. The learning algorithm finds those unchanging *discriminatory* features and the way they are combined to define a particular person's face by going over a number of images of that person.

What We Talk about When We Talk about Learning

In machine learning, the aim is to construct a program that fits the given data. A learning program is different from an ordinary computer program in that it is a *general template with modifiable parameters*, and by assigning different values to these parameters the program can do different things. The learning algorithm *adjusts the parameters* of the template—which we call a *model*—by *optimizing a performance criterion* defined on the data.

For example, for a face recognizer, the parameters are adjusted so that we get the highest prediction accuracy on a set of training images of a person. This learning is generally repetitive and incremental. The learning program sees a lot of example images, one after the other, and the parameters are updated slightly at each example, so that in time the performance improves gradually. After all, this is what learning is: as we learn a task, we get better at it, be it tennis, geometry, or a foreign language.

In chapter 2, we are going to talk in more detail about what the template is (actually as we will see, we have different templates depending on the type of the task) and the different learning algorithms that adjust the parameters so as to get the best performance.

In building a learner, there are a number of important considerations:

First, we should keep in mind that just because we have a lot of data, it does not mean that there are underlying rules that can be learned. We should make sure that there are dependencies in the underlying process and that the collected data provides enough information for them to be learned with acceptable accuracy. Let's say we have a phone book containing people's names and their phone numbers. It does not make sense to believe that there is an overall relationship between names and phone numbers so that after training on some given phone book (no matter how large), we can later predict the corresponding phone number when we see a new name.

Second, the learning algorithm itself should be efficient, because generally we have a lot of data and we want learning to be as fast as possible, using computation and memory effectively. In many applications, the underlying characteristics of the problem may change in time; in such a case, previously collected data becomes obsolete and the need arises to continuously and efficiently update the trained model with new data.

Third, once a learner has been built and we start using it for prediction, it should be efficient in terms of memory and computation as well. In certain applications, the efficiency of the final model may be as important as its predictive accuracy.

History

Almost all of science is fitting models to data. Scientists—such as Galileo, Newton, and Mendel—designed experiments, made observations, and collected data. They then tried to extract knowledge by devising theories, that is, building models to explain the data they observed.[4] They then used these theories to make predictions and if they didn't work, they collected more data and revised the theories. This process of data collection and theory/model building continued until they got models that had enough explanation power.

We are now at a point where this type of data analysis can no longer be done manually, because people who can do such analysis are rare; furthermore, the amount of data is huge and manual analysis is not possible. There is thus a growing interest in computer programs that can analyze data and extract information automatically from them—in other words, learn.

The methods that we discuss have their origins in different scientific domains. It was not uncommon that sometimes the same or very similar algorithms were independently invented in different fields following different historical paths.

The main theory underlying machine learning comes from statistics, where going from particular observations to general descriptions is called *inference* and learning is called *estimation*. Classification is called *discriminant analysis* in statistics. Statisticians used to work on small samples and, being mathematicians, mostly worked on simple models that could be analyzed mathematically. In engineering, classification is called pattern recognition and the approach is more empirical.

In computer science, as part of work on artificial intelligence, research was done on learning algorithms; a parallel but almost independent line of study was called *knowledge discovery in databases*. In electrical engineering, research in signal processing resulted in adaptive image processing and speech recognition programs.

In the mid-1980s, a huge explosion of interest in artificial neural network models from various disciplines took place. These disciplines included physics, statistics, psychology, cognitive science, neuroscience, and linguistics, not to mention computer science, electrical engineering, and adaptive control. Perhaps the most important contribution of research on artificial neural networks is this synergy that bridged various disciplines, especially statistics and computer science. The fact that neural network research, which later led to the field of machine learning, started in the 1980s is not accidental. At that time, with advances in VLSI (very-large-scale integration) technology, we gained the capacity to build parallel hardware containing thousands of processors, and artificial neural networks was of interest as a possible theory to distribute computation over a large number of processing units, all running in parallel. Furthermore, because they could learn, they would not need programming.

Research in these different communities followed different paths in the past with different emphases. Our aim in this book is to bring them together to give a unified introductory treatment of the field together with some interesting applications.

2

MACHINE LEARNING, STATISTICS, AND DATA ANALYTICS

Learning to Estimate the Price of a Used Car

We saw in the previous chapter that we use machine learning when we believe there is a relationship between observations of interest but do not know exactly how. Because we do not know its exact form, we cannot just go ahead and write down the computer program. So our approach is to collect data of example observations and to analyze it to discover the relationship. Now, let us discuss further what we mean by a relationship and how we extract it from data; as always, it is a good idea to use an example to make the discussion concrete.

Consider the problem of estimating the price of a used car. This is a good example of a machine learning application because we do not know the exact formula for this; at the same time, we know that there should be some rules:

the price depends on the properties of the car, such as its brand; it depends on usage, such as mileage, and it even depends on things that are not directly related to the car, such as the current state of the economy.

Though we can identify these as the factors, we cannot determine how they affect the price. For example, we know that as mileage increases price decreases, but we do not know how quickly this occurs. How these factors are combined to determine the price is what we do not know and want to learn. Toward this aim, we collect a data set where we look at a number of cars currently in the market, recording their attributes and how much they go for, and then we try to learn the specifics of the relationship between the attributes of a car and the price (see figure 2.1).

In doing that, the first question is to decide what to use as the *input representation*, that is, the attributes that we believe have an effect on the price of a used car. Those that immediately come to mind are the make and model of the car, its year of production, and its mileage. You can think of others too, but such attributes should be easy to record.

One important fact to always keep in mind is that there can be two different cars having exactly the same values for these attributes, yet they can still go for different prices. This is because there are other factors that have an effect, such as accessories. There may also be factors that we cannot directly observe and hence cannot include in the

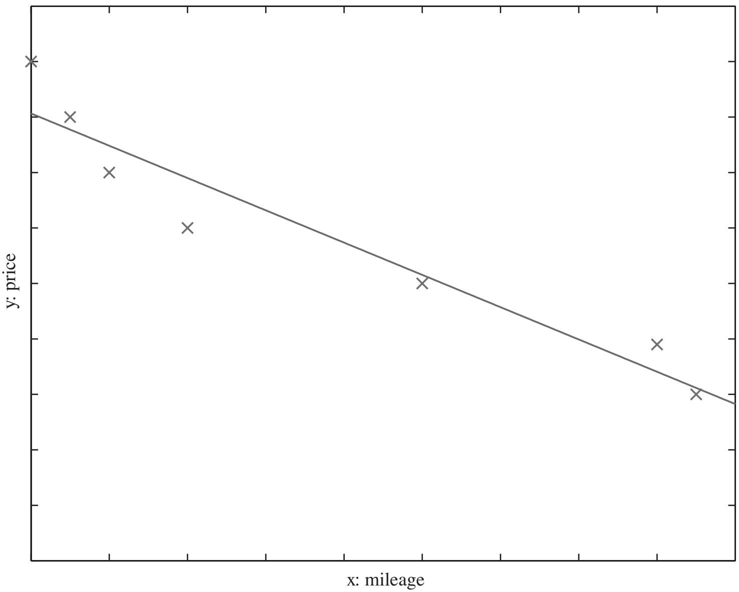

Figure 2.1 Estimating the price of a used car as a regression task. Each cross represents one car where the horizontal x-axis is the mileage and the vertical y-axis is the price (in some units). They constitute the training set. In estimating the price of a used car, we want to learn a model that fits (passes as close as possible to) these data points; an example linear fit is shown. Once such a model is fit, it can be used to predict the price of any car given its mileage.

input—for example, all the conditions under which the car has been driven in the past and how well the car has been maintained.

The crucial point is that no matter how many properties we list as input, there are always other factors that affect the output; we cannot possibly record and take all of them as input, and all these other factors that we neglect introduce uncertainty.

The effect of this uncertainty is that we can no longer estimate an exact price, but we can estimate an *interval* in which the unknown price is likely to lie, and the length of this interval depends on the amount of uncertainty—it defines how much the price can vary due to those factors that we do not, or cannot, take as input.

Randomness and Probability

In mathematics and engineering, we model uncertainty using *probability theory*. In a deterministic system, given the inputs, the output is always the same; in a random process, however, the output depends also on uncontrollable factors that introduce randomness.

Consider the case of tossing a coin. It can be claimed that if we have access to knowledge such as the exact composition of the coin, its initial position, the amount, position, and the direction of the force applied to the coin when tossing it, where and how it is caught, and so forth, the outcome of the toss can be predicted exactly; but because all this information is hidden from us, we can only talk about the probability of the outcomes of the toss. We do not know if the outcome is heads or tails, but we can say something about the probability of each outcome, which is a measure of our belief in how likely that outcome is. For example, if a coin is *fair*, the probability of heads and the probability of tails are equal—if we toss it many times, we expect to see roughly as many heads as tails.

If we do not know those probabilities and want to *estimate* them, then we are in the realm of *statistics*. We follow the common terminology and call each data instance an “example” and reserve the word “sample” for the *collection* of such examples. The aim is to build a *model* to calculate the values we want to estimate using the sample. In the coin tossing case, we collect a sample by tossing the coin a number of times and record the outcomes—heads or tails—as our observations. Then, our estimator for the probability of heads can simply be the proportion of heads in the sample—if we toss the coin six times and see four heads and two tails in our sample, we estimate the probability of heads as 2/3 (and hence the probability of tails as 1/3). Then if we are asked to guess the outcome of the next toss, our estimate will be heads because it is the more probable outcome.

This type of uncertainty underlies games of chance, which makes gambling a thrill for some people. But most of us do not like uncertainty, and we try to avoid it in our lives, at some cost if necessary. For example, if the stakes are high, we buy insurance—we prefer the certainty of never losing a large amount of money (due to accidental loss of something of high worth) to the cost of paying a small premium, even if the event is very unlikely.

The price of a used car is similar in that there are uncontrollable factors that make the depreciation of a car a random process. Two cars following one another on the production line are exactly the same at that point and hence are worth exactly the same. Once they are sold and start being used, all sorts of factors start affecting them: one of the owners may be more meticulous, one of the cars may be driven in better weather conditions, one car may have been in an accident, and so on. Each of these factors is like a random coin toss that varies the price.

A similar argument can also be made for customer behavior. We expect customers in general to follow certain patterns in their purchases depending on factors such as the composition of their household, their tastes, their income, and so on. Still, there are always additional random factors that introduce variance: vacation, change in weather, some catchy advertisement, and so on. As a result of this randomness, we cannot estimate exactly which items will be purchased next, but we can calculate the probability that an item will be purchased. Then if we want to make predictions, we can just choose the items whose probabilities are the highest.

Learning a General Model

Whenever we collect data, we need to collect it in such a way as to learn general trends. For example, in representing a car, if we use the brand as an input attribute we define a very specific car. But if we instead use general attributes such as the number of seats, engine power, trunk volume, and so forth, we can learn a more general estimator. This is because different models and makes of cars all appeal to the same type of customer, called a customer segment, and we would expect cars in the same segment to depreciate similarly. Ignoring the brand and concentrating on the basic attributes that define the type is equivalent to using the same, albeit noisy, data instance for all such cars of the same type; it effectively increases the size of our data.

A similar argument can also be made for the output. Instead of estimating the price as is, it makes more sense to estimate the percentage of its original price, that is, the effect of depreciation. This again allows us to learn a more general model.

Though of course it is good to learn models that are general, we should not try to learn models that are too general. For example, cars and trucks have very different characteristics, and it is better to collect data separately and learn different models for the two than to collect data and try to learn a single model for both.

Another important fact is that the underlying task may change in time. For example, the price of a car depends not only on the attributes of the car, the attributes representing its past usage, or the attributes of the owner, but also on the state of the economy, that is, the price of other things. If the economy, which is the environment in which we do all our buying and selling, undergoes significant changes, previous trends no longer apply. Statistically speaking, the properties of the random process that underlie the data have changed; we are given a new set of coins to toss. In this case, the previously learned model does not hold anymore, and we need to collect new data and learn again; or, we need to have a mechanism for getting feedback about our performance and fine-tune the model as we continue to use it.

Model Selection

One of the most critical points in learning is the *model* that defines the template of relationship between the inputs and the output. For example, if we believe that we can write the output as a weighed sum of the attributes, we can use a *linear model* where attributes have an additive effect—for example, each additional seat increases the value of the car by *X* dollars and each additional thousand miles driven decreases the value by *Y* dollars, and so on.

The weight of each attribute (*X* and *Y* above) can be calculated from the sample. A weight can be positive or negative—that is, more of the corresponding attribute increases or decreases the price respectively. If a weight is estimated to be very close to zero, we can conclude that the corresponding attribute is not important and eliminate it from the model. These weights are the *parameters* of the model and are fine-tuned using data. The model is always fixed; it is the parameters that are adjustable, and it is this process of adjustment to better match the data that we call learning.

The linear model is very popular because it is simple; it has few parameters and it is easy to calculate a weighted sum. It is easy to understand and interpret. Furthermore, it works surprisingly well for a lot of different tasks.

No matter how we vary its parameters, each model can only be used to learn a set of problems and *model selection* refers to the task of choosing between models. Selecting the right model is a more difficult task than optimizing its parameters once a model is fixed, and information about the application is helpful.

For instance here, in estimating car prices, the linear model may not be applicable. It has been seen empirically that the effect of the age is not arithmetic but geometric: each additional year does not decrease the price by the same amount, but a typical vehicle loses 15 percent of its value each year.[1] In later sections, we discuss some machine learning algorithms that use nonlinear models that are more powerful, in the sense that they can be used in a larger variety of applications.

Supervised Learning

This task of estimating an output value from a set of input values is called *regression* in statistics; for the linear model, we have linear regression. In machine learning, regression is one type of *supervised learning*. There is a supervisor who can provide us with the desired output—namely, the price—for each input car. When we collect data by looking at the cars currently sold in the market, we are able to observe both the attributes of the cars and also their prices.

The linear model with its weight parameters is not the only possible one. Each model corresponds to a certain type of dependency assumption between the inputs and the output. Learning corresponds to adjusting the parameters so that the model makes the most accurate predictions on the data. In the general case, learning implies getting better according to a performance criterion, and in regression, performance depends on how close the model predictions are to the observed output values in the training data. The assumption here is that the training data reflects sufficiently well the characteristics of the underlying task, so a model that works accurately on the training data can be said to have learned the task.

The different machine learning algorithms we have in the literature either differ in the models they use, or in the performance criteria they optimize, or in the way the parameters are adjusted during this optimization.

At this point, we should remember that the aim of machine learning is rarely to replicate the training data but the correct prediction of new cases. If there were only a certain number of possible cars in the market and if we knew the price for all of them, then we could simply store all those values and do a table lookup; this would be *memorization*. But frequently (and this is what makes learning interesting), we see only a small subset of all possible instances and from that data, we want to *generalize*—that is, we want to learn a general model that goes beyond the training examples to also make good predictions for inputs not seen during training.

Having seen only a small subset of all possible cars, we would like to be able to predict the right price for a car outside the training set, one for which the correct output was not given in the training set. How well a model trained on the training set predicts the right output for such new instances is called the *generalization ability* of the model and the learning algorithm.

The basic assumption we make here (and it is this assumption that makes learning possible) is that similar cars have similar prices, where similarity is measured in terms of the input attributes we choose to use. As the values of these attributes change slowly—for example, as mileage changes—price is also expected to change slowly. There is smoothness in the output in terms of the input, and that is what makes generalization possible. Without such regularity, we cannot go from particular cases to a general model, as then there would be no basis in the belief that there can be a general model that is applicable to all cases, both inside and outside the training set.

Not only for the task of estimating the price of a used car, but for many tasks where data is collected from the real world, be they for business applications, pattern recognition, or science, we see this smoothness. Machine learning, and prediction, is possible because the world has regularities. Things in the world change smoothly. We are not “beamed” from point A to point B, but we need to pass through a sequence of intermediate locations. Objects occupy a continuous block of space in the world. Nearby points in our visual field belong to the same object and hence mostly have shades of the same color. Sound too, whether in song or speech, changes smoothly. Discontinuities correspond to boundaries, and they are rare. Most

Machine learning, and prediction, is possible because the world has regularities. Things in the world change smoothly. We are not “beamed” from point A to point B, but we need to pass through a sequence of intermediate locations.

of our sensory systems make use of this smoothness; what we call visual illusions, such as the Kanizsa triangle, are due to the smoothness assumptions of our sensory organs and brain.

Such an assumption is necessary because collected data is not enough to find a unique model—learning, or fitting a model to data, is an *ill-posed problem*. Every learning algorithm makes a set of assumptions about the data to find a unique model, and this set of assumptions is called the *inductive bias* of the learning algorithm (Mitchell 1997).

This ability of generalization is the basic power of machine learning; it allows going beyond the training instances. Of course, there is no guarantee that a machine learning model generalizes correctly—it depends on how suitable the model is for the task, how much training data there is, and how well the model parameters are optimized—but if it does generalize well, we have a model that is much more than the data. A student who can solve only the exercises that the teacher previously solved in class has not fully mastered the subject; we want them to acquire a sufficiently general understanding from those examples so that they can also solve new questions about the same topic.

Learning a Sequence

Let us see a very simple example. You are given a sequence of numbers and are asked to find the next number in the sequence. Let us say the sequence is

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55

You probably noticed that this is the *Fibonacci sequence*. The first two terms are 0 and 1, and every term that follows is the sum of its two preceding terms. Once you identify the model, you can use it to make a prediction and guess that the next number will be 89. You can then keep predicting using the same model and generate a sequence as long as you like.

The reason we come up with this answer is that we are unconsciously trying to find a *simple explanation* to this data. This is what we always do. In philosophy, *Occam's razor* tells us to prefer simpler explanations, eliminating unnecessary complexity. For this sequence, a linear rule where two preceding terms are added is simple enough.

If the sequence were shorter,

0, 1, 1, 2.

you would not immediately go for the Fibonacci sequence—my prediction would be 2. With short sequences, there are many possible, and simpler, rules. As we see each successive term, those rules whose next value does not match are eliminated. Model fitting is basically a process of elimination: each extra observation (training example) is a constraint that eliminates all those candidates that do not obey it. And once we run out of the simple ones, we need to move on to complicated explanations incrementally to cover all the terms.

The complexity of the model is defined using *hyperparameters*. Here the fact that the model is linear and that only the previous two terms are used are hyperparameters.

Now let us say that the sequence is

0, 1, 1, 2, 3, 6, 8, 13, 20, 34, 55

Maybe you can also find a rule that explains this sequence, but I imagine it will be a complicated one. The alternative is to say that this is the Fibonacci sequence *with two errors* (6 instead of 5 and 20 instead of 21) and still predict 89 as the next number—or we can predict that the next number will lie in the interval [88,90].

Instead of a complex model that explains this sequence exactly, a noisy Fibonacci may be a more likely explanation if we believe that there may be errors (remember our earlier discussion on random effects due to unknown factors). And indeed errors are likely. Most of our sensors are far from perfect, typists make typos all the time, and though we like to believe that we act reasonably and rationally, we also act on a whim and buy/read/listen/click/travel on impulse. Human behavior is sometimes as much Dionysian as it is Apollonian.

Learning also performs *compression*. Once you learn the rule underlying the sequence, you do not need the data anymore. By fitting a rule to the data, we get an explanation that is simpler than the data, requiring less memory to store and less computation to process. Once we learn the rules of multiplication, we do not need to remember the product of every possible pair of numbers.

Credit Scoring

Let us now see another application to help us discuss another type of machine learning algorithm. A *credit* is an amount of money loaned by a financial institution such as a bank, to be paid back with interest, generally in installments. It is important for the bank to be able to predict in advance the risk associated with a loan, which is a measure of how likely it is that the customer will default and not pay the whole amount back. This is both to make sure that the bank will make a profit and also to not inconvenience a customer with a loan over their financial capacity.

In *credit scoring*, the bank calculates the risk given the amount of credit and the information about the customer.

This information includes data we have access to and is relevant in calculating the customer's financial capacity—namely, income, savings, collateral, profession, age, past financial history, and so forth. Again, there is no known rule for calculating the score; it changes from place to place and time to time. So the best approach is to collect data and try to learn it.

Credit scoring can be defined as a regression problem; historically the linear model, where the score of a customer was written as a weighted sum of different attributes, was frequently used. Each additional thousand dollars in salary increases the score by *X* points, and each additional thousand dollars of debt decreases the score by *Y* points. Depending on the estimated score, different actions can be taken—for example, a customer with a higher score may have a higher limit on their credit card.

Instead of regression, credit scoring can also be defined as a *classification* problem, where there are the two classes of customers: low-risk and high-risk. Classification is another type of supervised learning where the output is a class code, as opposed to the numeric value we have in regression.

A *class* is a set of instances that share a common property, and in defining this as a classification problem, we are assuming that all high-risk customers share some common characteristics not shared by low-risk customers, and that there exists a formulation of the class in terms of those

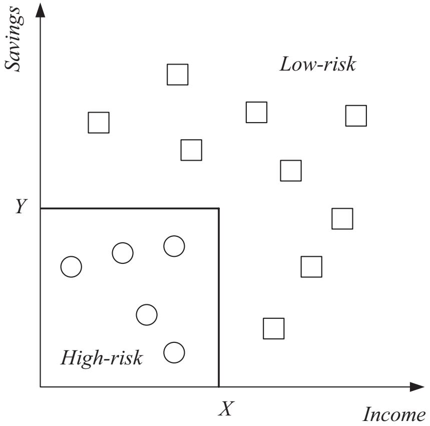

Figure 2.2 Separating the low- and high-risk customers as a classification problem. The two axes are the income and savings, each in its unit (for example, in thousands of dollars). Each customer, depending on their income and savings, is represented by one point in this two-dimensional space, and their class is represented by shape—a circle represents a high-risk customer and a square represents a low-risk customer. All the high-risk customers have their income less than *X* and savings less than *Y*, and hence this condition can be used as a discriminant, shown in bold.

characteristics, called a *discriminant*. We can visualize the discriminant as a boundary separating examples of the two classes in the space defined by the customer attributes.

As usual, we are interested in the case where we do not know the underlying discriminant but have a sample of example data, and we want to learn the discriminant from data.

In preparing the data, we look at our past transactions, and we label the customers who paid back their loans as low-risk and those who defaulted as high-risk. Analyzing this data, we would like to learn the class of high-risk customers so that in the future, when there is a new application for a loan, we can check whether or not the customer matches that description and reject or accept the application accordingly.

The information about a customer makes up the input to the classifier whose task is to assign the input to one of the two classes. Using our knowledge of the application, let us say that we decide to take a customer's income and savings as input (see figure 2.2). We observe them because we have reason to believe that they give us sufficient information about the credibility of a customer.

We do not have access to complete knowledge of all the factors that play a role, including the state of economy in full detail and full knowledge about the customer; so whether someone will be a low-risk or high-risk customer cannot be deterministically calculated. These are nonobservables and as such introduce randomness, and so, with what we can observe, we cannot say exactly whether a new customer is low- or high-risk, but we can calculate the probabilities of the two classes and then choose the one with the higher probability.

One possible model defines the discriminant in the form of *if-then rules*:

```
IF income < X AND savings < Y THEN high-risk
ELSE low-risk
```

where *X* and *Y* are the parameters fine-tuned to the data, to best match the prediction with what the data tells us (see figure 2.2). In this model the parameters are these thresholds, not weights as we have in the linear model.

Each if-then rule specifies one composite condition made up of terms, each of which is a simple condition on one of the input attributes. The antecedent of the rule is an expression containing terms connected with `AND`, namely, a conjunction; that is, all the conditions should hold for the rule to apply.

We understand from the rule that among the subset of customers that satisfies the antecedent—namely, those whose income is less than *X and* whose savings is less than *Y*—there are more high-risk than low-risk customers, so the probability of high risk for them is higher; that is why the consequent of the rule has high-risk as its label.

In this simple example, there is a single way of being high-risk and all the remaining cases are low-risk. In another application, there may be a *rule base* that is composed of several if-then rules, each of which delimits a certain region, and each class is specified using a disjunction of such rules. There are different ways of being high-risk, each of which is specified by one rule and satisfaction of any of the rules is enough.

Learning such rules from data allows *knowledge extraction*. The rule is a simple model that explains the data, and looking at this model we have an explanation about the process underlying the data. For example, once we learn the discriminant separating the low-risk and high-risk customers, we have knowledge about the properties of low-risk customers. We can then use this information to target potential low-risk customers more efficiently, such as through customized advertising.

Expert Systems

Before machine learning was the norm, expert systems existed. Proposed in 1970s and used in 1980s,[2] they were computer programs that aided humans in decision making.

An *expert system* is composed of a *knowledge base* and an *inference engine*. The knowledge is represented as a set of if-then rules, and the inference engine uses logical inference rules for deduction. The rules are programmed after consultation with domain experts, and they are fixed. This process of converting domain knowledge to if-then rules was difficult and costly. The inference engine was programmed in specialized programming languages such as Lisp and Prolog, which are especially suited for logical inference.

For a time in the 1980s, expert systems were quite popular around the world, not only in the United States (where Lisp was used), but also in Europe (where Prolog was used). Japan had a Fifth Generation Computer Systems Project for massively parallel architectures for expert systems and artificial intelligence (AI). There were applications, but in rather limited domains, such as MYCIN for diagnosing infectious diseases (Buchanan and Shortliffe 1984); commercial systems also existed.

Despite the research and the wide interest, expert systems never took off. There are basically two reasons for this. First, the knowledge base needed to be created manually through a very laborious process; there was no learning from data. The second reason was the unsuitability of logic to represent the real world. In real life, things are not true or false, but have grades of truth: a person is not either old or not old, but oldness increases gradually with age. The logical rules too may apply with different degrees of certainty: "If *X* is a bird, *X* can fly" is mostly true but not always.

To represent degrees of truth, *fuzzy logic* was proposed with fuzzy memberships, fuzzy rules, and inference, and since its inception, had some success in a variety of applications (Zadeh 1965). Another way to represent uncertainty is to use probability theory, as we do in this book.

Machine learning systems that we discuss in this book are extensions of expert systems in decision making in two ways: first, they need not be programmed but can learn from examples, and second, because they use probability theory, they are better in representing real-world settings with all the concomitant noise, exceptions, ambiguities, and resulting uncertainty.

Expected Values

When we make a decision—for example, when we choose one of the classes—we may be correct or wrong. It may be the case that correct decisions are not equally good and wrong decisions are not equally bad. When making a decision about a loan applicant, a financial institution should take into account both the potential gain as well as the potential loss. An accepted low-risk applicant increases profit, while a rejected high-risk applicant decreases loss. A high-risk applicant who is erroneously accepted causes loss, and an erroneously rejected low-risk applicant is a missed chance for profit.

The situation is much more critical and far from symmetrical in other domains, such as *medical diagnosis*. Here, the inputs are the relevant information we have about the patient and the classes are the illnesses. The inputs contain the patient's age, gender, past medical history and current symptoms. Some tests may not have been applied to the patient, and these inputs would be missing. Tests take time, are costly, and may inconvenience the patient, so we do not want to apply them unless we believe they will give us valuable information.

In the case of medical diagnosis, a wrong decision may lead to wrong or no treatment, and the different types of error are not equally bad. Let us say we have a system that collects information about a patient and based on those, we want to decide whether the patient has a certain disease (say a certain type of cancer) or not. There are two possibilities: either the patient has cancer—let us call it the positive class—or not—the negative class.

Similarly, there are two types of error. If the system predicts cancer but in fact the patient does not have it, this is a *false positive*—the system chooses the positive class wrongly. This is bad because it will cause unnecessary treatment, which is both costly and also inconvenient for the patient. If the system predicts no disease when in fact the patient has it, this is a *false negative*.

A false negative has a higher cost than a false positive because then the patient will not get the necessary treatment. Because the cost of a false negative is so much larger than the cost of a false positive, we would choose the positive class—to start a closer investigation—even if the probability of the positive class is relatively small. This is not like predicting a coin toss where we choose the outcome—heads or tails—whose probability is higher than 1/2.

This is the basis of *expected value* calculation where not only do we decide by using probabilities, but we also take into account the possible loss or gain we may face as a result of our decision. Though expected value calculation is frequently done in many domains, such as in insurance, it is known that people do not always act rationally; if that were the case, no one would buy a lottery ticket!

In Max Frisch's novel *Homo Faber*, the mother of a girl who was bitten by a snake is told not to worry because the mortality from snakebites is only 3–10 percent. The woman gets angry and says, "If I had a hundred daughters … then I should lose only three to ten daughters. Amazingly few! You're quite right," and then she continues, "I've only got one child." We need to be careful in using expected value calculation when ethical matters are involved.[3]

If both the false positive and the false negative have high costs, a possible third action is to *reject* and defer decision. For example, if computer-based diagnostics cannot choose between two outcomes, it can opt to reject and the case can be decided manually; the human expert can make use of additional information not directly available to the system. In an automated mail sorter, if the system cannot recognize the numeric zip code on an envelope, the postal worker can also read the address.

3

PATTERN RECOGNITION

Learning to Read

Different automatic visual recognition tasks have different complexities. One of the simplest is the reading of *barcodes* where information is represented in terms of lines of different widths, which are shapes that are easy to recognize. The barcode is a simple and efficient technology. It is easy to print barcodes, and it is also easy to build scanners to read them; that is why they are still widely used. But the barcode is not a natural representation, and the information capacity is limited; recently two-dimensional matrix barcodes have been proposed where more information can be coded in a smaller area.

There is always a trade-off in engineering. When the task is difficult to solve, we can devise more efficient solutions by constraining it. For example, the wheel is a very good solution for transportation, but it requires flat surfaces and so roads too have to be built. The controlled environment makes the task easier. Legs work in a variety of terrains, but they are more difficult to build and control, and they can carry a significantly less heavy load.

Optical character recognition is recognizing printed or written characters from their images. This is more natural than barcodes because no extra coding (in terms of bars) is used. If a single font is used, there is a single way of writing each character; there are standardized fonts such as OCR-A, defined specifically to make automatic recognition easier.

With barcodes or a single font, a single template exists for each class and there is no need for learning. For each character, we have a single prototype that we can simply store. It is the ideal image for that character, and we compare the seen input with all the prototypes one by one and choose the class with the best matching prototype—this is called *template matching*. There may be errors in printing or sensing, but we can do recognition by finding the closest match.

If we have many fonts or handwritings, we have multiple ways of writing the same character, and we cannot possibly store all of them as possible templates. Instead, we want to “learn” the class by going over all the different examples of the same character and find some general description that covers all of them.

It is interesting that though writing is a human invention, we do not have a formal description of 'A' that covers all 'A's and none of the non-'A's. Not having it, we take samples from different writers and fonts, and learn a definition of 'A'-ness from these examples. But though we do not know what it is that makes an image an instance of the class 'A', we are certain that all those distinct 'A's have something in common, which is what we want to extract from the examples.

We know that a character image is not just a collection of random dots and strokes of different orientations, but it has a regularity that we believe we can capture by using a learning program. For each character, we see examples in different fonts (for printed text) or writings (for handwritten text) and generalize; that is, we find a description that is shared by all of the examples of a character: 'A' is one way of combining a certain set of strokes, 'B' is another way, and so on.

Recognition of printed characters is relatively easy with the Latin alphabet and its variants; it is trickier with alphabets where there are more characters, accents, and writing styles. In cursive handwriting, characters are connected and there is the additional problem of segmentation.

Many different fonts exist, and people have different handwriting styles. Characters may also be small or large, slanted, printed in ink or written with a pencil, and as a result, many possible images can correspond to the same character. Despite all the research, there is still no computer program today that is as accurate as humans for this task. That is why captchas are still used, a *captcha* being a corrupted image of words or numbers that needs to be typed to prove that the user is a human and not a computer program.

Matching Model Granularity

In machine learning, the aim is to fit a model to the data. In the ideal case, we have one single, *global* model that applies to all of the instances. For all cars, as we saw in chapter 2, we have a single regression model that we can use to estimate the price. In such a case, the model is trained with the whole training data and all the instances have an effect on the model parameters. In statistics, this is called *parametric estimation*.

The parametric model is good because it is simple—we store and calculate a single model—and it is trained with the whole data. Unfortunately, it may be restrictive in the sense that this assumption of a single model applicable to all cases may not hold in all applications. In certain tasks, we may have a set of *local* models, each of which is applicable to a certain type of instances. This is *semi-parametric estimation*. We still have a model that maps the input to the output but is valid only locally, and for different type of inputs we have different models.

For example, in estimating the price of used cars, we may have one model for sedans, another model for sports cars, and another for luxury cars, if we have reason to believe that for these different types of cars, the depreciation behavior may be different. In such an approach, given only the data, the grouping of data into localities and the training of the models in each local region are done together in a coupled manner, and each local model is trained only with the training data that falls within its scope—the number of local models is the hyperparameter defining the model flexibility and hence complexity.

In certain applications, even the semi-parametric assumption may not hold; that is, the data may lack a clear structure and it cannot be explained in terms of a few local models. In such a case, we use the other extreme of *nonparametric estimation* where we assume no simple model, either globally or locally. The only information we use is the most basic assumption—namely, that similar inputs have similar outputs. In such a case, we do not have an explicit training process that converts training data to model parameters; instead, we just keep the training data as the sample of past cases.

Given an instance, we find the training instances that are most similar to the query and we calculate an output in terms of the known outputs of these past similar instances.

For example, given a car whose price we want to estimate, we find among all the training instances the three cars that are most similar—in terms of the attributes we use—and then calculate the average of the prices of these three cars as our estimate. Because those are the cars that are most similar in their attributes, it makes sense that their prices should be similar too. This is called *k-nearest neighbor estimation* where here *k* is three. Since those are the three most similar past "cases," this approach is sometimes called *case-based reasoning*. The nearest-neighbor algorithm is intuitive: similar instances mean similar things. We all love our neighbors because they are so much like us—or we hate them, as the case may be, for exactly the same reason.

Generative Models

An approach that has recently become very popular in data analysis is to consider a *generative model* that represents our belief as to how the data is generated. We assume that there is a hidden model with a number of *hidden*, or *latent*, *causes* that interact to generate the data we observe. Though the data we observe may seem big and complicated, it is produced through a process that is controlled by a few variables, which are the hidden factors, and if we can somehow infer these, the data can be represented and understood in a much simpler way. Such a simple model can also make accurate predictions.

Consider optical character recognition. Generatively speaking, we can say that each character image is composed of two types of factors: there is the identity, namely the label of the character, and there is the appearance, those that are due to the process of writing or printing (possibly also when it is scanned/perceived).

In a printed text, the appearance part may be due to the font; for example, characters in Times Roman font have serifs and strokes that are not all of the same width. The font is an aesthetic concern; in calligraphy, it is the aesthetic part that becomes especially prominent. But these added characteristics due to appearance should not be large enough to cause confusion about the identity. Just like the choice of font in printed text, the handwriting style of the writer introduces variance in written text. The appearance also depends on the material the writer is using (for example, pen versus pencil) and also on the medium (for example, paper versus marble slab).[1]

The printed or written character may be large or small, and this is generally handled at a preprocessing stage of normalization where the character image is converted to a fixed size—we know that the size does not affect the identity. This is called *invariance*. We want invariance to size (whether the text is 12pt or 18pt, the content is the same) or invariance to slant (as when the text is in italics) or invariance to the width of strokes (as in bold characters). But, for example, we do not want full rotation invariance: *q* is a rotated *b*.

In recognizing the class, we need to focus on the identity, and we should find attributes that represent the identity, and learn how to combine them to represent the character. We treat all the attributes that relate to the appearance, namely the writer, aesthetics, medium, and sensing, as irrelevant and learn to ignore them. But note that in a different task, those may be the important ones; for example, in authentication of handwriting or in signature recognition, it is the writer-specific attributes that become important.

The generative model is *causal* in that it explains how the data is generated by hidden factors that cause it. Once we have such a model trained, we may want to use it for *diagnostics*, which implies going in the opposite direction, that is, from observation to cause. Medical domain is a good example here: the diseases are the causes and they are hidden; the symptoms are the observed attributes of the patient, such as the results of medical tests. Going from disease to symptom is the causal direction—that is what the disease does; going from symptom to disease is diagnostics—that is what the doctor does. In the general case, diagnostics is the inference of hidden factors from observed variables.

A generative model can be represented as a graph composed of nodes that correspond to hidden and observed variables, and the arcs between nodes represent dependencies between them, such as causalities. Such *graphical models* are interesting in that they allow a visual representation of the problem, and statistical inference and estimation procedures can be mapped to well-known and efficient graph operations (Koller and Friedman 2009).

For example, a causal link goes from a hidden factor to an observed symptom, while a diagnostics effectively inverts the direction of the link. We use conditional probability to model the dependency, and for example, when we talk about the conditional probability that a patient has a runny nose given that they have the flu, we go in the causal direction: the flu causes the runny nose (with a certain probability).

If we have a patient and we know they have a runny nose, we need to calculate the conditional probability in the other direction—namely, the probability that they have the flu given that they have a runny nose (see figure 3.1). In probability, the two conditional probabilities are related because of the *Bayes' rule*,[2] and that is why graphical models are sometimes also called *Bayesian networks*. In a later section, we return to Bayesian estimation; we see that we can also include the model parameters in such networks and that this allows additional flexibility.

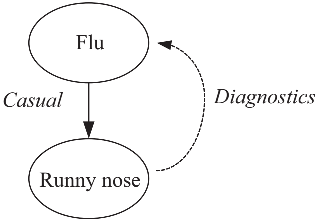

Figure 3.1 The graphical model showing that the flu is the cause of a runny nose. If we know that the patient has a runny nose and want to check the probability that they have the flu, we are doing diagnostics by making inference in the opposite direction (using Bayes' rule). We can form larger graphs by adding more nodes and links to show increasingly complicated dependencies.

If we are reading a text, one factor we can make use of is the language information. A word is a sequence of characters, and we rarely write an arbitrary sequence of characters; we choose a word from the lexicon of the language. This has the advantage that even if we cannot recognize a character, we can still read t?e word. Such contextual dependencies may also occur at higher levels, between words and sentences as defined by the syntactic and semantic rules of the language. Machine learning algorithms help us learn such dependencies for natural language processing, as we discuss shortly.

Face Recognition

In the case of *face recognition*, the input is the image captured by a camera and the classes are the people to be recognized. The learning program should learn to match the face images to their identities. This problem is more difficult than optical character recognition because the input image is larger, a face is almost three-dimensional, and differences in pose and lighting cause significant changes in the image. Certain parts of the face may also be occluded; glasses may hide the eyes and eyebrows, and a beard may hide the chin.

Just as in character recognition, we can think of two sets of factors that affect the face image: there are the features that define the identity, and there are features that have no effect on the identity but affect appearance, such as hairstyle; or expression (namely, neutral, smiling, angry, and so forth). These appearance features may also be due to hidden factors that affect the captured face image, such as the source of illumination or the pose. If we are interested in the identity, we want to learn a face description that uses only the first type of features, learning to be invariant to features of the second type.

However, we may be interested in the second type of features for other tasks. Recognizing facial expressions allows us to recognize mood or emotions, as opposed to identity. For example, during a video monitoring a meeting, we may want to keep track of the mood of the participants. Likewise, in online education, it is important to understand whether the student is confused or gets frustrated, to better adjust the speed of presenting the material. In *affective computing*, which is a field that is rapidly becoming popular, the aim is to have computer systems that adapt themselves to the mood of the user.

If the aim is identification or authentication of people—for example, for security purposes—using the face image is only one of the possibilities. *Biometrics* is recognition or authentication of people using their physiological and/or behavioral characteristics. In addition to the face, examples of physiological characteristics are the fingerprint, iris, and palm; examples of behavioral characteristics include the dynamics of signature, voice, gait, and keystroke. For more accurate decisions, inputs from different modalities can be integrated. When there are many different inputs—as opposed to the usual identification procedures of photo, printed signature, or password—forgeries (spoofing) becomes more difficult and the system more accurate, hopefully without too much inconvenience to the users.

Speech Recognition

In *speech recognition*, the input is the acoustic signal captured by a microphone and the classes are the words that can be uttered. This time the association to be learned is between an acoustic signal and a word of some language.

Just as we can consider each character image to be composed of basic primitives like strokes of different orientations, a word is considered to be a sequence of *phonemes*, which are the basic speech sounds. In the case of speech, the input is *temporal*; words are uttered in time as a sequence of these phonemes, and some words are longer than others.

Different people, because of differences in age, gender, or accent, pronounce the same word differently, and again, we may consider each word sound to be composed of two sets of factors, those that relate to the word and those that relate to the speaker. Speech recognition uses the first type of features, whereas speaker authentication uses the second. Incidentally, this second type of features (those relating to the speaker) is not easy to recognize or to artificially generate—that is why the output of speech synthesizers still sounds "robotic."[3]

Just as in biometrics, researchers here rely on the idea of combining multiple sources. In addition to the acoustic information, we can also use the video image of the speaker's lips and the shape of the mouth as they speak the words.

Natural Language Processing and Translation

In speech recognition, as in optical character recognition, the integration of a *language model* taking contextual information into account helps significantly. Decades of research on programmed rules in computational linguistics have revealed that the best way to come up with a language model (defining the lexical, syntactic, and semantic rules of the language) is by learning it from some large corpus of example data. The applications of machine learning to *natural language processing* are constantly increasing; see Hirschberg and Manning 2015 for a recent survey.

One of the easier applications is *spam filtering*, where spam generators on one side and filters on the other side keep finding more and more ingenious ways to outdo each other. This is a classification problem with two classes, spam and legitimate emails. A similar application is *document categorization* where we want to assign text documents to one of several categories, such as, arts, culture, politics, and so on.

A face is an image and a spoken sentence is an acoustic signal, but what is in a text? A text is a sequence of characters, but characters are defined by an alphabet and the relationship between a language and the alphabet is not straightforward. The human language is a very complex form of information representation with lexical, syntactic, and semantic rules at different levels, together with its subtleties such as humor and sarcasm, not to mention the fact that a sentence almost never stands or should be interpreted alone, but is part of some dialogue or general context.

The most popular method for representing text is the *bag of words* representation where we predefine a large vocabulary of words and then we represent each document by using a list of the words that appear *anywhere* in the document. That is, of the words we have chosen, we note which ones appear in the document and which ones do not. We lose the position of the words in the text, which may be good or bad depending on the application. In choosing a vocabulary, we choose words that are indicative of the task; for example, in spam filtering, words such as "opportunity" and "offer" are discriminatory. There is a preprocessing step where suffixes (for example, "-ing," "-ed") are removed, and where noninformative words (for example, "the," "of") are ignored.

Recently, analyzing messages on social media has become an important application area of machine learning. Analyzing blogs or posts to extract *trending topics* is one: this implies a certain novel combination of words that has suddenly started to appear in print a lot. Another task is mood recognition, that is, determining whether a customer is happy or not with a product (for example, a politician). For this, one can define a vocabulary containing words indicative of the two classes—happy versus not happy—using the bag of words representation and learn how they affect the class descriptions.

Perhaps the most impressive application of machine learning is—or rather, would be—*machine translation*. After decades of research on hand-coded translation rules, it has become apparent that the most promising approach is to provide a very large sample of pairs of texts in both languages and to have a learning program automatically figure out the rules to map one to the other. In bilingual countries such as Canada, and in the European Union with its many official languages, it is relatively easy to find the same text carefully translated in two or more languages. Such data is heavily used by machine learning approaches to translation.

In chapter 4, we discuss *deep learning*, which shows a lot of promise for this task, in automatically learning the different layers of abstraction that are necessary for processing natural language.

Combining Multiple Models

In any application, you can use any one of various learning algorithms and instead of trying to choose the single best one, a better approach may be to use them all and combine their predictions. This smooths out the randomness in the training and may lead to better performance.

The aim is to find a set of models that are diverse, who complement each other. One way to get this is by having them look at different sources of information. We already saw this in biometrics where we look at different characteristics—for example, face, fingerprints, and so on—and in speech recognition where in addition to the acoustic speech signal we also keep track of the speaker's lip.

Nowadays, most of our data is multimedia, and *multi-view models* can be used in a variety of contexts where we have different sensors providing different but complementary information. In image retrieval, in addition to the image itself, we may also have a text description or a set of tag words. Using both sources together leads to better retrieval performance. Our smart devices, such as smart watches and smartphones, are equipped with sensors, and their readings can be combined for the purpose of, for example, *activity recognition*.

Outlier Detection

Another application area of machine learning is *outlier detection*, where the aim this time is to find instances that do not obey the general rule—those are the exceptions that are informative in certain contexts. The idea is that typical instances share characteristics that can be simply stated, and instances that do not have them are atypical.

In *Anna Karenina*, Tolstoy writes, “All happy families resemble one another, but each unhappy family is unhappy in its own way.” This holds true in many domains, and not only for the case of nineteenth-century Russian families. For example, in medical diagnosis, we can similarly say that all healthy people are alike and that there are different ways of being unhealthy—each one of them is one disease.

In such a case, the model covers the typical instances and then any instance that falls outside is an exception. An *outlier* is an instance that is very different from other instances in the sample. An outlier may indicate an abnormal behavior of the system; for example, for a credit card transaction, it may indicate *fraud*; in an image, an outlier may indicate an *anomaly* requiring attention, for example, a tumor; in the case of network traffic, it may be an intrusion attempt by a hacker; in a health-care scenario, it may indicate a significant deviation from a patient’s normal behavior. Outliers may also be recording errors (for example, due to faulty sensors) that should be detected and discarded to get reliable statistics. An outlier may also be a novel, previously unseen but valid case, which is where the related term, *novelty detection*, comes into play. For example, it may be a new type of profitable customer, indicating a new niche in the market waiting to be exploited by the company.

Dimensionality Reduction

In any application, observed data attributes that we believe contain information are taken as inputs and are used for decision-making. However, it may be the case that some of these features actually are not informative at all, and they can be discarded; for example, it may turn out that the color of a used car does not have a significant effect on its price. Or, it may be the case that two different attributes are correlated and say basically the same thing (for example, the production year and mileage of a used car are highly correlated), so keeping one may be enough.

We are interested in *dimensionality reduction* in a separate preprocessing step for a number of reasons:

First, in most learning algorithms, both the complexity of the model and the training algorithm depend on the number of input attributes. Here, complexity is of two types: the time complexity, which is how much calculation we do, and the space complexity, which is how much memory we need. Decreasing the number of inputs always decreases both, but how much they decrease depends on the particular model and the learning algorithm.

Second, when an input is deemed unnecessary, we save the cost of measuring it. For example, in medical diagnosis, if it turns out that a certain test is not needed, we do not do it, thereby eliminating both the monetary cost and the patient discomfort.

Third, simpler models are more robust on small data sets; that is, they can be trained with fewer data; or when trained with the same amount of data, they have smaller variance (uncertainty).

Fourth, when data can be explained with fewer features, we have a simpler model that is easier to interpret.

Fifth, when data can be represented in few (for example, two) dimensions, it can be plotted and analyzed visually, for structure and outliers, which again helps facilitate knowledge extraction from data. A plot is worth a thousand dots, and if we can find a good way to display the data, our visual cortex can do the rest, without any need for model fitting calculation.

There are basically two ways to achieve dimensionality reduction, namely, through feature selection and feature extraction. In *feature selection*, we keep the important features and discard the unimportant ones. It is basically a process of subset selection where we want to choose the smallest subset of the set of input attributes leading to maximum performance. The most widely used method for feature selection is the *wrapper* approach, where we iteratively add features until there is no further improvement. The feature selector is “wrapped around” the basic classifier or regressor that is trained and tested with each subset.

In *feature extraction*, we define new features that are calculated from the original features. These newly calculated features are fewer in number but still preserve the information in the original features. Those few synthesized features explain the data better than any of the original attributes, and sometimes they may be interpreted as hidden or abstract concepts.

In projection methods, each new feature is a linear combination of the original features; one such method is *principal component analysis* where we find new features that preserve the maximum amount of variance of the data. If the variance is large, the data has large spread making the differences between the instances most apparent, whereas if the variance is small, we lose the differences between data instances. The other method, *linear discriminant analysis* is a form of supervised feature extraction where the aim is to find new features that maximize the separation between classes.

Whether one should use feature selection or extraction depends on the application and the granularity of the features. If we are doing credit scoring and have features such as customer age, income, profession, and so on, feature selection makes sense. For each feature, we can say whether it is informative or not by itself. But a feature projection does not make sense: what does a linear combination (weighted sum) of age, income, and profession mean? On the other hand, if we are doing face recognition and the inputs are pixels, feature selection does not make sense—an individual pixel by itself does not carry discriminative information. It makes more sense to look at particular *combinations* of pixels in defining a face, as is done by feature extraction.

Nonlinear dimensionality reduction methods go beyond a linear combination and can find better features; this is one of the hottest topics in machine learning. The ideal feature set best represents the (classification or regression) information in the data set using the fewest numbers, and it is a process of encoding. It may also be considered as a process of abstraction because these new features can correspond to higher-level features representing the data in a more concise manner. In chapter 4, we talk about autoencoder networks and deep learning where this type of nonlinear feature extraction is learned in artificial neural networks.

Decision Trees

Previously we discussed if-then rules and one way to learn such rules is by decision trees. The *decision tree* is one of the oldest methods in machine learning and though simple in both training and prediction, it is accurate in many domains. Trees use the famous "divide and conquer" strategy popular since Caesar where we divide a complex task—for example, governing Gaul—into simpler, regional tasks. Trees are used in computer science frequently for the same reason, namely to decrease complexity, in all sorts of applications.

Previously we talked about nonparametric estimation where, as you will remember, the main idea is to find a subset of the neighboring training examples that are most similar to the new query. In *k* nearest-neighbor algorithms, we do this by storing all the training data in memory, calculating one by one the similarity between the new test query and all the training instances, and choosing the *k* most similar ones. This is rather a complex calculation when the training data is large, and it may be infeasible when the data is big.

The decision tree finds the most similar training instances by a sequence of tests on different input attributes. The tree is composed of decision nodes and leaves; starting from the root, each decision node applies a splitting test to the input and depending on the outcome, we take one

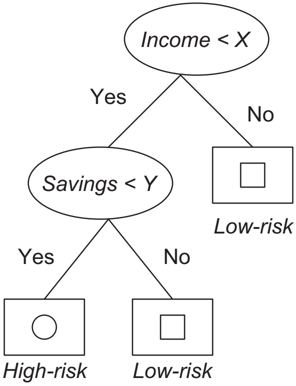

Figure 3.2 A decision tree separating low- and high-risk customers. This tree implements the discriminant shown in figure 2.2.

of the branches. When we get to a leaf, the search stops and we understand that we have found the most similar training instances, and we interpolate from those (see figure 3.2).

Each path from the root to a leaf corresponds to a conjunction of test conditions in the decision nodes on the path and such a path can be written as an if-then rule. That is one of the advantages of the decision tree: that a tree can be converted to a rule base of if-then rules and that those rules are easy to interpret. The tree is trained with a given training data where splits are placed to delimit regions that have the highest "purity," in the sense that each region contains instances that are similar in terms of their output.

Decision tree learning is nonparametric—the tree grows as needed and its size depends on the complexity of the problem underlying the data; for a simple task, the tree is small, whereas a difficult task may grow a large tree.

There are different decision tree models and learning algorithms depending on the splitting test used in the decision nodes and the interpolation done at the leaves; one very popular approach nowadays is the *random forest*, where we train many decision trees on random subsets of the training data and we take a vote on their predictions (to get a smoother estimate).

Trees are used successfully in various machine learning applications, and together with the linear model, the decision tree should be taken as one of the basic benchmark methods before any more complex learning algorithm is tried.

Active Learning

In learning, it is critical that the learner also knows what it knows and what it does not know. When a trained model makes a prediction, it is helpful if it can also indicate its certainty in that prediction. As we discussed before, this can be in the form of a confidence interval where a smaller interval indicates less uncertainty.

Generally more data means more information, and hence more data tends to decrease uncertainty. But data points are not created equal, and if the trained model knows where it has high uncertainty, it can actively ask the supervisor to label examples in there. This is called *active learning*. The model generates inquiries by synthesizing new inputs and asks for them to be labeled, rather like a student asking a question during a lecture.

For example, very early on in artificial intelligence, it was realized that the most informative examples are those that lie closest to the current estimate of the class boundary: a *near miss* is an instance that looks very much like a positive example but is actually a negative example (Winston 1975).

A related research area in machine learning is called *computational learning theory,* where work is done to find theoretical bounds for learning algorithms that hold in general, independent of the particular learning task. For example, for a given model and learning algorithm, we may want to know the minimum number of training instances needed to guarantee at most a certain error with high enough probability—this is called *probably approximately correct learning* (Valiant 1984).

Learning to Rank

Ranking is an application area of machine learning that is different from regression or classification, and that is sort of between the two. In classification and regression, for each instance, we have a desired absolute value for the output; in ranking we train on pairs of instances and are asked to have the outputs for the two in the correct order.

Let us say we want to learn a *recommendation model* to make movie recommendations. For this task, the input is composed of the attributes of the movie and the attributes of the customer. The output is a numeric score that is a measure of how much we believe that a particular customer will enjoy a particular movie. To train such a model, we use past customer ratings. If we know that the customer liked movie *A* more than movie *B* in the past, we do our training such that for that customer the estimated score for *A* is indeed a higher value than the estimated score for *B*. Then, later on when we use that model to make a recommendation based on the highest scores, we expect to choose a movie that is more similar to *A* than to *B*.

There is no required numeric value for the score, as we have for the price of a used car, for example. The scores can be in any range as long as the ordering is correct. The training data is not given in terms of absolute values but in terms of such rank constraints (Liu 2011).

We can note here the advantage and difference of a ranker over a classifier or a regressor. If users rate the movies they have seen as enjoyed versus not enjoyed, this will be a two-class classification problem and a classifier can be used, but taste is nuanced and a binary rating is hard to come by. On the other hand, if people rate their enjoyment of each movie on a scale of, say, from 1 to 10, this will be a regression problem, but such values are difficult to assign. It is more natural and easier for people to say of the two movies they watched which one they enjoyed more. After the ranker is trained with all such pairs, it is expected to generate numeric scores satisfying all these constraints.

Ranking has many applications. In search engines, we want to retrieve the most relevant documents when given a query. When we retrieve and display the current top ten candidates, if the user clicks the third one but skips the first two, we understand that the third should have been ranked higher than the first and the second. Such *click logs* are used to train rankers.

Bayesian Methods

In certain applications and with certain models, we may have some *prior belief* about the possible values of parameters. When we toss a coin, we expect it to be a fair coin or close to being fair, so we expect the probability of heads to be close to 1/2. In estimating the price of a car, we expect mileage to have a negative effect on the price. Bayesian methods allow us to take such prior beliefs into account in estimating the parameters.[4]

The idea in *Bayesian estimation* is to use that prior knowledge together with the data to calculate a *posterior distribution* for the parameters. The Bayesian approach is especially useful when the data set is small. One advantage to getting a posterior distribution is that we know how likely each parameter value is, so not only we can choose the particular parameter that is most likely—the *maximum a posteriori* (MAP) *estimator*—but we can also average over all values of the parameter, or a number of highly probable ones, thereby averaging out the uncertainty in estimating the parameter.

Though the Bayesian approach is flexible and interesting, it has the disadvantage that except for simple scenarios under restrictive assumptions, the necessary calculation is too complex. One possibility is that of *approximation* where instead of the real posterior distribution that we cannot easily handle, we use one that is the most similar to the distributions we can. Another possibility is *sampling* where instead of using the distribution itself, we generate representative instances from the distribution and make our inferences based on them. The popular methods for these, namely, *variational approximation* for the former and *Markov chain Monte Carlo* (MCMC) *sampling* for the latter, are among the most important current research directions in machine learning.

The Bayesian approach allows us to incorporate our prior beliefs in training. For example, we have a prior belief that the underlying problem is smooth, which makes us prefer simpler models. In *regularization*, we penalize complexity, and during training, in addition to maximizing our fit to the data, we also try to minimize the model complexity. We get rid of those parameters that make the model unnecessarily complex and the output too variant. This implies a learning scheme that involves not only the adjustment of parameters but also changes to the model structure. Or we can go in the other direction and add complexity when we suspect we have a model that is too simple for the data.

The use of such *nonparametric* approaches in Bayesian estimation is especially interesting because we are no longer constrained by some parametric model class, but the model complexity also changes dynamically to match the complexity of the task in the data (Orbanz and Teh 2010). This implies a model of “infinite size,” because it can be as complex as we want—it grows when it learns.

4

NEURAL NETWORKS AND DEEP LEARNING

Artificial Neural Networks

Our brains make us intelligent; we see or hear, learn and remember, plan and act thanks to our brains. In trying to build machines to have such abilities then, our immediate source of inspiration is the human brain, just as birds were the source of inspiration in our early attempts to fly. What we would like to do is to look at how the brain works and try to come up with an understanding of how it does what it does. But we want to have an explanation that is independent of the particular implementation details—this is what we called the computational theory when we talked about levels of analysis in chapter 1. If we can extract such an abstract, mathematical, and computational description, we can later implement it with what we have at our disposal as engineers—for example, in silicon and running on electricity.

Early attempts to build flying machines failed until we understood the theory of aerodynamics; only then we could build airplanes. Nowadays, we see birds and airplanes as two different ways of flying—we call them airplanes now, not artificial birds, and they can do more than birds can; they cover longer distances and carry passengers or cargo. The idea is to accomplish the same for intelligence, and we start by getting inspired by the brain.

The human brain is composed of a very large number of processing units, called *neurons*, and each neuron is connected to a large number of other neurons through connections called *synapses*. Neurons operate in parallel and transfer information among themselves over these synapses. It is believed that the processing is done by the neurons and memory is in the synapses, that is, in the way the neurons are connected and influence each other.

Research on *neural networks* as models for analog computation—neuron outputs are not 0 or 1—started as early as research on digital computation (McCulloch and Pitts 1943) but, after the quick success and widespread use of digital computers, went largely unnoticed for a long time.

In the 1960s, the *perceptron* model was proposed as a model for pattern recognition (Rosenblatt 1962). It is a network composed of artificial neurons and synaptic connections, where each neuron has an activation value, and a connection from neuron *A* to neuron *B* has a weight that defines the effect of *A* on *B*. If the synapse is excitatory, when *A* is active it also tries to activate *B*; if the synapse is inhibitory, when *A* is active it tries to suppress *B*.

During operation, each neuron sums up the activations from all the neurons that make a synapse with it, weighted by their synaptic weights, and if the total activation is larger than a threshold value, the neuron "fires" and its output corresponds to the value of this activation; otherwise the neuron is silent. If the neuron fires, it sends its activation value in turn down to all the neurons with which it makes a synapse (see figure 4.1).

The perceptron basically calculates a weighted sum before making a decision, and this can be seen as one way of implementing a variant of the linear model we discussed before. Such neurons can be organized as layers where all the neurons in a layer take input from all the neurons in the previous layer and calculate their value in parallel, and these values together are fed to all the neurons in the layer that follows—this is called a *multilayer perceptron*.

Some of the neurons are sensory neurons and take their values from the environment (for example, from the sensed image), similar to the receptors in the retina. These then are given to other neurons that do some more processing over them in successive layers as activation propagates over the network. Finally, there are the output neurons that make the final decision and carry out the

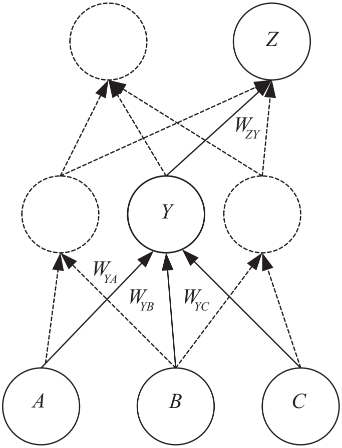

Figure 4.1 An example of a neural network composed of neurons and synaptic connections between them. Neuron *Y* takes its inputs from neurons *A*, *B*, and *C*. The connection from *A* to *Y* has weight W_{YA} that determines the effect of *A* on *Y*. *Y* calculates its total activation by summing the effect of its inputs weighted by their corresponding connection weights. If this is large enough, *Y* fires and sends its value to the neurons after it—for example, *Z*—through the connection with weight W_{ZY}.

actions through actuators—for example, to move an arm, utter a word, and so on.

Neural Network Learning Algorithms

In a neural network, learning algorithms adjust the connection weights between neurons. An early algorithm was proposed by Hebb (1949) and is known as the *Hebbian learning* rule: the weight between two neurons gets reinforced if the two are active at the same time—the synaptic weight effectively learns the correlation between the two neurons.

Let us say we have one neuron that checks whether there is a circle in the visual field and another neuron that checks whether there is the digit six, ‘6’, in the visual field. Whenever we see a six—or are told that it is a six when we are learning to read—we also see a circle, so the connection between them is reinforced, but the connection between the circle neuron and, say, the neuron for digit seven, ‘7’, is not reinforced because when we see one, we do not see the other. So the next time we see a circle in the visual field, this will increase the activation of the neuron for the digit six but will diminish the activation of the neuron for the digit seven, making six a more likely hypothesis than seven.

In some applications, certain neurons in the network are explicitly designated as input units and certain of them as output units. We have a training set that contains a sample of inputs and their corresponding correct output values, as specified by a supervisor—for example, in estimating the price of a used car, we have the car attributes as the input and their prices as the output. In this case of supervised learning, we clamp the input units to the input values in the training set, let the activity propagate through the network depending on the weights and the network structure, and then we look at the values calculated at the output units.

We define an *error function* as the sum of the differences between the actual outputs the network estimates for an input and their required values specified by the supervisor in the training set; and in neural network training, for each training example, we update the connection weights slightly, in such a way as to decrease the error for that instance. Decreasing the error implies that the next time we see the same or similar input, estimated outputs will be closer to their correct values. *Theoretically speaking*, this is nothing but the good old regression we discussed in chapter 2, except that here the model is *implemented* as a neural network of neurons and connections.

This is one important characteristic of neural network learning algorithms, namely that they can learn *online*, by doing small updates on the connection weights as we see training instances one at a time. In *batch* learning however, we have the whole data set and do training all at once using the whole data. A popular approach today involves *mini batches*, where we use small sets of instances in each update.

Nowadays with data sets getting larger, online learning is attractive because it does not require the collection and the storage of the whole data; we can just learn by using one example or a few examples at a time in a streaming data scenario. Furthermore, if the underlying characteristics of the data change slowly—as they generally do—online learning can adapt seamlessly, without needing to stop, collect new data, and retrain.

What a Perceptron Can and Cannot Do

Though the perceptron was successful in many tasks—remember that the linear model works reasonably well in many domains—there are certain tasks that cannot be implemented by a perceptron (Minsky and Papert 1969). The most famous of these is the exclusive OR (XOR) problem:

In logic, there are two types of OR, the inclusive OR and the exclusive OR. In everyday speech, when we say "To go to the airport, I will take the bus or the train," what we mean is the exclusive OR. There are two cases and only one of them can be true at one time. To represent the inclusive OR, we use the construct "and/or," as in "This fall, I will take Math 101 and/or Phys 101." In other words, I will take Math 101, Phys 101, or both.

Though the inclusive OR can be implemented by a perceptron, the exclusive OR cannot. It is not difficult to see why: if you have two cases, for example, the bus and the train, and if you want either to be enough, you need to give each of them a weight larger than the threshold so that the neuron fires when any one of them is true. But then when both of them are true, the overall activation will be twice as high and cannot be less than the threshold.

Though it was known at that time that tasks like XOR *can* be implemented using multiple layers of perceptrons, it was not known how to train such networks; and the fact that the perceptron cannot implement a task as straightforward as XOR—which can easily be implemented by a few (digital) logic gates—led to disappointment and the abandonment of neural network research for a long time, except for a few places around the world. It was only in the mid-1980s when the *backpropagation* algorithm was proposed to train multilayer perceptrons—the idea had been around since the 1960s and 1970s but had gone largely unnoticed—that interest in it was revived (Rumelhart, Hinton, and Williams 1986).

Not all artificial neural networks are feedforward; there are also *recurrent networks* where in addition to connections between layers, neurons also have connections to neurons in the same layer (including themselves), or even to neurons in the layers that precede them. Each synapse causes a certain delay so the neuron activations through the recurrent connections act as a *short-term memory* for contextual information and let the network remember the past.

Let us say that input neuron A is connected to neuron X and that there is also a recurrent connection from X to itself. So at time t, the value of X will depend on input A at time t and will also depend on the value of X at time $t - 1$ because of the recurrent connection from X to itself. In the next time step, X at time $t + 1$ will depend on input A at time $t + 1$ and also on X at time t (previously calculated using A at time t and X at time $t - 1$), and so on. In this way, the value of X at any time will depend on all the inputs seen until then.

If we define the state of a network as the collection of the values of all the neurons at a certain time, recurrent connections allow the current state to depend not only on the current input but also on the network state in the previous time steps calculated from the previous inputs. So, for example, if we are seeing a sentence one word at a time, the recurrence allows the previous words in the sentence to be kept in this short-term memory in a condensed and abstract form and hence taken into account while processing the current word. The architecture of the network and the way recurrent connections are set define how far back and in what way the past influences the current output.

Recurrent neural networks are used in many tasks where the time dimension is important, as in speech or language processing, where what we would like to

If we define the state of a network as the collection of the values of all the neurons at a certain time, recurrent connections allow the current state to depend not only on the current input but also on the network state in the previous time steps calculated from the previous inputs.

recognize are sequences. In a translation of text from one language to another, not only the input but also the output is a sequence.

Connectionist Models in Cognitive Science

Artificial neural network models are known as *connectionist* or *parallel distributed processing* (PDP) models in cognitive psychology and cognitive science (Feldman and Ballard 1982; Rumelhart and McClelland and the PDP Research Group 1986). The idea is that neurons correspond to concepts and that the activation of a neuron corresponds to our current belief in the truth of that concept. Connections correspond to constraints or dependencies between concepts. A connection has a positive weight and is excitatory if the two concepts occur simultaneously—for example, between the neurons for circle and '6'—and has a negative weight and is inhibitory if the two concepts are mutually exclusive—for example, between the neurons for circle and '7'.

Neurons whose values are observed—for example, by sensing the environment—affect the neurons they are connected to, and this activity propagation throughout the network results in a state of neuron outputs that satisfies the constraints defined by the connections.

The basic idea in connectionist models is that intelligence is an *emergent property* and high-level tasks, such as recognition or association between patterns, arise automatically as a result of this activity propagation by the rather elemental operations of interconnected simple processing units. Similarly, learning is done at the connection level through simple operations, for instance, according to the Hebbian rule, without any need for a higher-level programmer.

Connectionist networks care about biological plausibility but are still abstract models of the brain; for example, it is very unlikely that there is actually a neuron for every concept in the brain—this is the *grandmother cell* theory, which states that I have a neuron in my brain that is activated only when I see or think of my grandmother—that is a *local representation*. It is known that neurons die and new neurons are born in the brain, so it makes more sense to believe that the concepts have a *distributed representation* on a cluster of neurons, with enough redundancy for concepts to survive despite physical changes in the underlying neuronal structure.

Neural Networks as a Paradigm for Parallel Processing

Since the 1980s, computer systems with thousands of processors have been commercially available. The software for such parallel architectures, however, has not advanced as quickly as hardware. The reason for this is that almost all our theory of computation up to that point was based on serial, single-processor machines. We are not able to use the parallel machines in their full capacity because we cannot program them efficiently.

There are mainly two paradigms for *parallel processing*. In *single instruction, multiple data* (SIMD) machines, all processors execute the same instruction but on different pieces of data. In *multiple instruction, multiple data* (MIMD) machines, different processors may execute different instructions on different data. SIMD machines are easier to program because there is only one program to write. However, problems rarely have such a regular structure that they can be parallelized over a SIMD machine. MIMD machines are more general, but it is not an easy task to write separate programs for all the individual processors; additional problems arise that are related to synchronization, data transfer between processors, and so forth. SIMD machines are also easier to build, and machines with more processors can be constructed if they are SIMD. In MIMD machines, processors are more complex, and a more complex communication network must be constructed for the processors to exchange data arbitrarily.

Assume now that we can have machines where processors are a little bit more complex than SIMD processors but not as complex as MIMD processors. Assume that we have simple processors with a small amount of local memory where some parameters can be stored. Each processor implements a fixed function and executes the same instructions as SIMD processors; but by loading different values into its local memory, each processor can be doing different things and the whole operation can be distributed over such processors. We will then have what we can call *neural instruction, multiple data* (NIMD) machines, where each processor corresponds to a neuron, local parameters correspond to its synaptic weights, and the whole structure is a neural network. If the function implemented in each processor is simple and if the local memory is small, then many such processors can be fit on a single chip.

The problem now is to distribute a task over a network of such processors and to determine the local parameter values. This is where learning comes into play: we do not need to program such machines and determine the parameter values ourselves if such machines can learn from examples.

Thus, artificial neural networks are a way to make use of the parallel hardware we can build with current technology and—thanks to learning—they need not be programmed. Therefore, we also save ourselves the effort of programming them.

Hierarchical Representations in Multiple Layers

Before, we mentioned that a single layer of perceptron cannot implement certain tasks, such as XOR, and that such limitations do not apply when there are multiple layers. Actually it has been proven that the multilayer perceptron is a *universal approximator*, that is, it can approximate any function with desired accuracy given enough neurons—through training it to do that is not always straightforward.

The perceptron algorithm can train only single-layer networks, but in the 1980s the *backpropagation algorithm* was invented to train multilayer perceptrons, and this caused a flurry of applications in various domains significantly accelerating neural network research in many fields, from cognitive science to computer science and engineering.

The multilayer network is intuitive because it corresponds to layers of operation where we start from the raw input and incrementally perform a more complicated transformation, until we get to an abstract output representation.

For example, in image recognition, we have image pixels as the basic input and as input to the first layer. The neurons in the next layer combine these to detect basic image descriptors such as strokes and edges of different orientations. A later layer combines these to form longer lines, arcs, and corners. Layers that follow combine them to learn more complex shapes such as circles, squares, and so on. These in turn are combined with some more layers of processing to represent the objects we want to learn, such as faces or handwritten characters.

Each neuron in a layer defines a more complex feature in terms of the simpler patterns detected in the layer below it. These intermediate feature-detecting units are called *hidden units* because they correspond to hidden attributes not directly observed but are defined in terms of what is observed. These successive layers of hidden units correspond to increasing layers of abstraction, where we start from raw data such as pixels and end up in abstract concepts such as a digit or a face.

It is interesting to note that a similar mechanism seems to be operating in the visual cortex. In their experiments on cats, Hubel and Wiesel, who were later awarded the 1981 Nobel Prize for their work on visual neurophysiology, have shown that there are *simple cells* that respond to lines of particular orientations in particular positions in the visual field, and these in turn feed to *complex* and *hypercomplex cells* for detecting more complicated shapes (Hubel 1995)—though not much is known about what happens in later layers.

Imposing such a structure on the network implies making assumptions, such as dependencies, about the input. For example, in vision we know that nearby pixels are correlated and there are local features like edges and corners. Any object, such as a handwritten digit, may be defined as a combination of such primitives. We know that because the visual scene changes smoothly, nearby pixels tend to belong to the same object, and where there is sudden change—an edge—is informative because it is rare.

Similarly, in speech, locality is in time, and inputs close in time can be grouped as speech primitives. By combining these primitives, longer utterances, namely speech phonemes, can be defined. They in turn can be combined to define words, and these in turn can be combined as sentences.

In such cases, when designing the connections between layers, units are not connected to all of the input units because not all inputs are dependent. Instead, we define units that define a window over the input space and are connected to only a small *local* subset of the inputs. This decreases the number of connections and therefore the number of parameters to be learned. Such a structure is called a *convolutional neural network* where the operation of each unit is considered to be a convolution—that is, a matching—of its input with its weight (Le Cun et al. 1989). An earlier similar architecture is the *neocognitron* (Fukushima 1980).

The idea is to repeat this in successive layers where each layer is connected to a small number of local units below. Each layer of *feature extractors* checks for slightly more complicated features by combining the features below in a slightly larger part of the input space, until we get to the output layer that looks at the whole input. Feature extraction also implements *dimensionality reduction* because although the raw attributes that we observe may be many in number, the important hidden features that we extract from data and that we use to calculate the output are generally much fewer.

This multilayered network is an example of a *hierarchical cone* where features get more complex, abstract, and fewer in number as we go up the network until we get to classes (see figure 4.2).

A special type of multilayer network is the *autoencoder*, where the desired output is set to be equal to the input, and there are fewer hidden units in the intermediate layers than there are in the input. The first part, from the input to the hidden layer, implements an encoder stage where a high-dimensional input is compressed and represented by the values of the fewer hidden units. The second part, from the hidden layer to the output, implements a decoder stage that takes that low-dimensional representation in the hidden layer and reconstructs the higher dimensional input back again at the output.

For the network to be able to reconstruct the input at its output units, those few hidden units that act as a

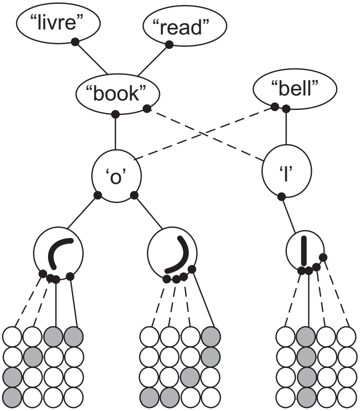

Figure 4.2 A very simplified example of hierarchical processing. At the lowest level are pixels, and they are combined to define primitives such as arcs and line segments. The next layer combines them to define letters, and the next combines them to define words. The representation becomes more abstract as we go up. Continuous lines denote positive (excitatory) connections, and dashed lines denote negative (inhibitory) connections. The letter *o* exists in "book" but not in "bell." At higher levels, activity may propagate using more abstract relationships such as the relationship between "book" and "read," and in a multilingual context, between "book" and "livre," the French word for book.

bottleneck should be able to extract the best features that preserve information maximally. The autoencoder is unsupervised; those hidden units learn to find a good encoding of the input, a short compressed description, extracting the most important features and ignoring what is irrelevant, namely, noise.

Deep Learning

In computer vision in the last half century, significant research has been done to find the best features for accurate classification, and many different image filters, transforms, and convolutions have been proposed to implement such feature extractors manually.

Though these approaches have had some success, learning algorithms are achieving higher accuracy recently with big data and powerful computers. With few assumptions and little manual interference, structures similar to the hierarchical cone are being automatically learned from large amounts of data. These learning approaches are especially interesting in that, because they learn, they are not fixed for any specific task, and they can be used in a variety of applications. They learn both the hidden feature extractors and also how they are best combined to define the output.

This is the idea behind *deep neural networks* where, starting from the raw input, each hidden layer combines the values in its preceding layer and learns more complicated functions of the input. The fact that the hidden unit values are not 0 or 1 but continuous allows a finer and graded representation of similar inputs. Successive layers correspond to more abstract representations until we get to the final layer where the outputs are learned in terms of these most abstract concepts.

With few assumptions and little manual interference, structures similar to the hierarchical cone are being automatically learned from large amounts of data. These learning approaches are especially interesting in that, because they learn, they are not fixed for any specific task, and they can be used in a variety of applications.

We saw an example of this in the convolutional neural network where starting from pixels, we get to edges, and then to corners, and so on, until we get to a digit. In such a network, some user knowledge is necessary to define the connectivity and the overall architecture. Consider a face recognizer network where inputs are the image pixels. If each hidden unit is connected to *all* the pixels, the network has no knowledge that the inputs are face images or even that the input is two-dimensional—the input is just a set of values. Using a convolutional network where hidden units are fed with localized two-dimensional patches is a way to feed this locality information such that correct abstractions can be learned.

In *deep learning*, the idea is to learn feature levels of increasing abstraction with minimum human contribution (Schmidhuber 2015; LeCun, Bengio, and Hinton 2015). This is because in most applications, we do not know what structure there is in the input, especially as we go up, and the corresponding concepts become "hidden." So any sort of dependency should be automatically discovered during training from a large sample of examples. It is this extraction of hidden dependencies, or patterns, or regularities from data that allows abstraction and learning general descriptions.

Training a network with multiple hidden layers is difficult and slow because the error at the output needs to be propagated back to update the weights in all the preceding layers, and there is interference when there are many parameters. In a convolutional network, each unit is fed to only a small subset of the units before and feeds to only a small subset of units after, so the interference is less and training can be done faster.

A deep neural network can be trained one layer at a time. The aim of each layer is to extract the salient features in its input, and a method such as the autoencoder can be used for this purpose. There is the extra advantage that we can use unlabeled data—the autoencoder is unsupervised and hence does not need labeled data. So starting from the raw input, we train an autoencoder, and the encoded representation learned in its hidden layer is then used as input to train the next autoencoder, and so on, until we get to the final layer trained in a supervised manner with the labeled data. Once all the layers are trained in this way one by one, they are all assembled one after the other and the whole network of stacked autoencoders can be fine-tuned with the labeled data.

If a lot of labeled data and a lot of computational power are available, the whole deep network can be trained in a supervised manner, but using an unsupervised method to initialize the weights works much better than random initialization; besides, learning can be done much faster and with fewer labeled data.

Deep learning methods are attractive mainly because they need less manual interference. We do not need to craft the right features or the suitable transformations. Once we have data—and nowadays we have "big" data—and sufficient computation available—and nowadays we have data centers with thousands of processors—we just wait and let the learning algorithm discover all that is necessary by itself.

The idea of multiple layers of increasing abstraction that underlies deep learning is intuitive. Not only in vision—in handwritten digits or face images—but also in many applications we can think of such layers of abstraction. Discovering these abstract representations is useful, not only for prediction but also because abstraction allows a better description and understanding of the problem.

Another good example is natural language processing where the need for good feature extractors, that is, good hidden representations, is most apparent. Researchers have worked on predefined databases, called *ontologies*, for representing relationships between words in a language and such databases work with some success; but it is best to learn such relationships from a lot of data. Deep networks that learn hierarchies at different levels of abstraction can be a way of doing this. Autoencoders and recursive autoencoders are good candidates for the building blocks of such deep architectures. A recursive autoencoder is trained so that the learned hidden representation depends not only on the current input but also on the previous inputs.

Consider machine translation. Starting with an English sentence, in multiple levels of processing and abstraction that are learned automatically from a very large English corpus to code the lexical, syntactic, and semantic rules of the English language, we would get to the most abstract representation. Now consider the same sentence in French. The levels of processing learned this time from a French corpus would be different, but if the two sentences mean the same, at the most abstract, language-independent level, they should have very similar representations.

Language understanding is a process of encoding where from a given sentence we extract this high-level abstract representation, and language generation is a process of decoding where we synthesize a natural language sentence from such a high-level representation. In translation, we encode in the source language and decode in the target language. In a dialogue system, we first encode the question to an abstract level and process it to form a response in the abstract level, which we then decode as the response sentence.

One deep network does not a brain make; deep networks still work in relatively constrained domains, but we are seeing more impressive results every day as the networks get larger and are trained with more data.

5

LEARNING CLUSTERS AND RECOMMENDATIONS

Finding Groups in Data

Previously we covered supervised learning where there is an input and an output—for example, car attributes and price—and the aim is to learn a mapping from the input to the output. A supervisor provides the correct values, and the parameters of a model are updated so that its output gets as close as possible to these desired outputs.

We are now going to discuss *unsupervised learning*, where there is no predefined output, and hence no such supervisor; we have only the input data. The aim in unsupervised learning is to find the regularities in the input, to see what normally happens. There is a structure to the input space such that certain patterns occur more often than others, and we want to see what generally happens and what does not.

One method for unsupervised learning is *clustering*, where the aim is to find clusters or groupings of input. In statistics, these are called *mixture models*.

In the case of a company, the customer data contains demographic information, such as age, gender, zip code, and so on, as well as past transactions with the company. The company may want to see the distribution of the profile of its customers, to see what type of customers frequently occur. In such a case, a clustering model allocates customers similar in their attributes to the same group, providing the company with natural groupings of its customers; this is called *customer segmentation*. Once such groups are found, the company may decide strategies, for example, services and products, specific to different groups; this is known as *customer relationship management* (CRM).

Such a grouping also allows a company to identify those who are outliers, namely, those who are different from other customers, which may imply a niche in the market that can be further exploited by the company, or those customers who require further investigation, for example, churning customers; see figure 5.1.

We expect to see regularities and patterns repeated with minor variations in many different domains. Detecting them as primitives and ignoring the irrelevant parts (namely, noise) is also a way of doing compression. For example, in an image, the input is made up of pixels, but we can identify regularities by analyzing repeated image

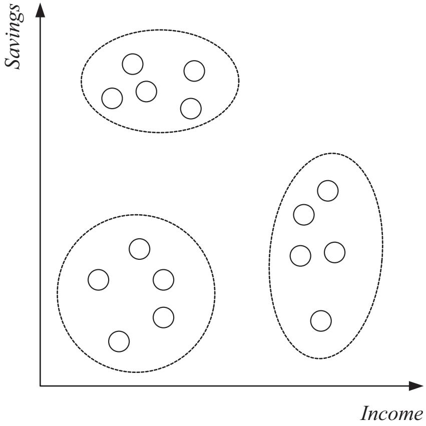

Figure 5.1 Clustering for customer segmentation. For each customer, we have the income and savings information. Here, we see that there are three customer segments. Such a grouping allows us to understand the characteristics of the different segments so that we can define different interactions with each segment; this is called customer relationship management.

patterns, such as texture, objects, and so forth. This allows a higher-level, simpler, and a more useful description of the scene and achieves better compression than compressing at the pixel level. A scanned document page does not have random on/off pixels but bitmap images of characters; there is structure in the data, and we make use of this redundancy by finding a shorter description of the data in terms of strokes of different orientations. Going further, if we can discover that those strokes combine in certain ways to make up characters, we can use just the code of a character, which is shorter than its image.

In *document clustering*, the aim is to group similar documents. For example, news reports can be subdivided into those related to politics, sports, fashion, arts, and so on. We can represent the document as a bag of words using a lexicon that reflects such document types, and then documents are grouped depending on the number of shared words. It is of course critical how the lexicon is chosen.

Unsupervised learning methods are also used in *bioinformatics*. DNA in our genome is the "blueprint of life" and is a sequence of bases, namely, A, G, C, and T. RNA is transcribed from DNA, and proteins are translated from RNA. Proteins are what the living body is and does. Just as DNA is a sequence of bases, a protein is a sequence of amino acids (as defined by bases). One application area of computer science in molecular biology is *alignment*, which is matching one sequence to another. This is a difficult string-matching problem because strings may be quite long, there are many template strings to match against, and there may be deletions, insertions, or substitutions.

Clustering is used in learning *motifs*, which are sequences of amino acids that occur repeatedly in proteins. Motifs are of interest because they may correspond to structural or functional elements within the sequences they characterize. The analogy is that if the amino acids are letters and proteins are sentences, motifs are like words, namely, a string of letters with a particular meaning occurring frequently in different sentences.

Clustering may be used as an *exploratory data analysis* technique where we identify groups naturally occurring in the data. We can then, for example, label those groups as classes and later on try to classify them. A company may cluster its customers and find segments, and then toward a certain aim—for example, churning—can label them and train a classifier to predict the behavior of new customers. But the important point is that there may be a cluster or clusters that no expert could have foreseen, and that is the power of unsupervised data-driven analysis.

Sometimes a class is made up of multiple groups. Consider the case of optical character recognition. There are two ways of writing the digit seven; the American version is ‘7’, whereas the European version has a horizontal bar in the middle (to tell it apart from the European ‘1’, which keeps the small stroke on top in handwriting). In such a case, when the sample contains examples from both continents, the class for seven should be represented as the union/disjunction/mixture of two groups.

A similar example occurs in speech recognition where the same word can be uttered in different ways, due to differences in pronunciation, accent, gender, age, and so on—“I say to-may-to, you say to-mah-to.” Thus when there is not a single, universal way, all these different ways should be represented as equally valid alternatives to be statistically correct.

Clustering algorithms group instances in terms of their similarities calculated in terms of their input representation, which is a list of input attributes, and the similarity between instances is measured by combining similarities in these attributes. In certain applications, we can define a similarity measure between instances directly, in terms of the original data structure, without explicitly generating such a list of attributes and calculating similarities for each.

Consider clustering Web pages. In addition to the text field, we can also use the similarity of meta (or header) information such as titles or keywords, or the number of common Web pages that link to or are linked from those two. This gives us a much better similarity measure than what is calculated using the bag of words representation on the text of the Web pages. Using a similarity measure that is better suited to the application—if one can be defined—leads to better clustering results; this is the basic idea in *spectral clustering*.

Such application-specific similarity representations are also popular in supervised learning applications typically grouped under the name *kernel function*. The *support vector machine* (Vapnik 1998) is one such learning algorithm used for both classification and regression.

It is also possible to do *hierarchical clustering*, where instead of a flat list of clusters, we generate a tree structure with clusters at different levels of granularity and clusters higher in the tree that are subdivided into smaller clusters. We are familiar with such trees of clusters from studies in biology—most famously, the taxonomy by Linnaeus—or human languages. One explanation of the splitting up of clusters into smaller clusters is due to *phylogeny*, that is, to evolutionary changes—small mutations are gradually accumulated over time until a species splits into two—but in other applications, the reason of similarity may be different.

The aim in clustering in particular, or unsupervised learning in general, is to find structure in the data. In the case of supervised learning (for example, in classification), this structure is imposed by the supervisor who defines the different classes and labels the instances in the training data by these classes. This additional information provided by the supervisor is of course useful, but we should always make sure that it does not become a source of bias or impose artificial boundaries. There is also the risk that there is error in labeling, which is called "teacher noise."

Unsupervised learning is an important research area because unlabeled data is a lot easier and cheaper to find. For speech recognition, a talk radio station is a source of unlabeled speech data. The idea is to extract the basic characteristics from unlabeled data and learn what is typical, which can then later be labeled for different purposes. A baby spends their first few years looking around when they see things, objects, faces repeatedly under a variety of conditions, during which presumably they learn their basic feature extractors and how they typically combine to form objects. Later on when that baby learns language, they learn the names for those.

Recommendation Systems

In chapter 1, we discussed *recommendation systems* for predicting customer behavior as an application of machine learning. Given a large data set of customer transactions, we can find *association rules* of the form “People who buy X are also likely to buy Y.” Such a rule implies that among the customers who buy X, a large percentage have also bought Y. So if we find a customer who has bought X but has not bought Y, we can target them as a potential Y customer. X and Y can be products, authors, cities to be visited, videos to be watched, and so on; we see many examples of this type of recommendation every day, especially while surfing online.

Though this targeting approach is used frequently and efficient algorithms have been proposed to learn such rules from very large data sets, interesting algorithms that make use of generative models are being proposed nowadays.

Remember that while constructing a generative model, we think about how we believe the data is generated. In customer behavior therefore, we consider the causes that affect this behavior. We know that people do not buy things at random. Their purchases depend on a number of factors, such as their household composition—that is, how many people they live with, their gender, ages—and their income, their taste (which in turn is a result of other factors such as the place of origin), and so on. Though some companies have loyalty cards and collect some of this information, in practice, most of these factors are not known, are hidden, and need to be inferred from the observed data.

Note however that an overreliance on these factors can be misguided because they are often wrong or incomplete; there may also be factors that we cannot immediately think of or factors that are not as important as we think, which is why it is always best to learn (discover) them from data.

There may be many products whose values—for example, the amounts bought—are observable, but their purchase is influenced by a small number of hidden factors. The idea is that if we can estimate those factors for a customer, we can make more accurate predictions about that customer's later purchases.

Extracting such hidden causes will build a much better model than trying to learn associations among products. For example, a hidden factor may be “baby at home,” which will lead to the purchase of different items such as diapers, milk, baby formula, wipes, and so on. So instead of learning association rules between pairs or triples of these items, if we can estimate the hidden baby factor based on past purchases, this will trigger an estimation of whatever it is that has not been bought yet.

In practice, there are many such factors; each customer is affected (or defined) by a number of these, and each factor triggers a subset of the products. The factor values are not 0 or 1 but continuous, and this distributed representation provides a richness when it comes to representing customer instances.

This approach aims to find structure by decomposing data into two parts. The first one, the mapping between customers and factors, defines a customer in terms of the factors (with different weights). The second one, the mapping between factors and products, defines a factor in terms of the products (with different weights). In mathematics, we model data using matrices, which is why this approach is called *matrix decomposition*.

Such a generative approach with hidden factors makes sense in many other applications. Let us take the case of movie recommendations (see figure 5.2). We have customers who have rented a number of movies; we also have a lot

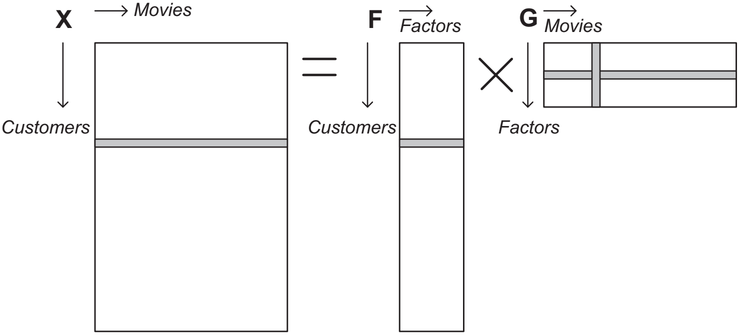

Figure 5.2 Matrix decomposition for movie recommendations. Each row of the data matrix *X* contains the scores given by one customer for the movies, most of which will be missing (because the customer hasn't watched that movie). It is factored into two matrices *F* and *G* where each row of *F* is one customer defined as a vector of factors and each row of *G* defines the effect of one factor over the movies; each column of *G* is one movie defined in terms of the factors. The number of factors is typically much smaller than the number of customers or movies; in other words, it is the number of factors that defines the complexity of the data, named the rank of the data matrix *X*.

of movies they have not watched, and from those we want to make a recommendation.

The first characteristic of this problem is that we have many customers and many movies, but the data is *sparse*. Every customer has watched only a small percentage of the movies, and most movies have been watched by only a small percentage of the customers. Based on these facts, the learning algorithm needs to be able to generalize and predict successfully, even when new movies or new customers are added to the data.

In this case too, we can think of hidden factors, such as the age and gender of the customer, which makes certain genres, such as action, comedy, and so on, a more likely choice. Using decomposition, we can define each customer in terms of such factors (in different proportions), and each such factor triggers certain movies (with different probabilities). This is better—easier, cheaper—than trying to come up with rules between pairs of movies. Note again that such factors are not predefined but are automatically discovered during learning; they may not always be easy to interpret or assign a meaning to.

Another possible application area is *document categorization* (Blei 2012). Let us say we have a lot of documents, and each is written using a certain bag of words. Again the data is sparse; each document uses only a small number of words. Here, we can interpret hidden factors as topics. When a reporter writes a report, they want to write about certain topics, so each document is a combination of certain topics, and each topic is written using a small subset of all possible words. This is called *latent semantic indexing*. It is clear that this makes more sense than trying to come up with rules such as "People who use the word *X* also use the word *Y*."

Thinking of how the data is generated through hidden factors and how we believe they combine to generate the observable data is important, and it can make the estimation process much easier. What we discuss here is an additive model where we take a sum of the effects of the hidden factors. Models are not always linear—for example, a factor may inhibit another factor—and learning nonlinear generative models from data is one of the important current research directions in machine learning.

6

LEARNING TO TAKE ACTIONS

Reinforcement Learning

Let us say we want to build a machine that learns to play chess. Assume we have a camera to see the positions of the pieces on the board, ours and our opponent's, and the aim is to decide on our moves so that we win the game.

In this case, we cannot use a supervised learner for two reasons. First, it is very costly to have a teacher who will take us through many games, indicating the best move for each board state. Second, in many cases, there is no such thing as the best move; how good a move is depends on the moves that follow. A single move does not count; a sequence of moves is good if after playing them we win the game. The only real feedback is at the end of the game when we win or lose the game.

Another example is a robot that is placed in a maze to find a goal location. The robot can move in one of the four compass directions and should make a sequence of movements to reach the goal. As long as the robot moves around, there is no feedback and the robot tries many moves until it reaches the goal; only then does it get a reward. In this case there is no opponent, but we can have a preference for shorter trajectories—the robot may be running on a battery—which implies that in this case we are playing against time.

These two applications have a number of points in common. There is a decision maker, called the *agent*, which is placed in an *environment* (see figure 6.1). In the first case, the chessboard is the environment of the game-playing agent; in the second case, the maze is the environment of the robot. At any time, the environment is in a

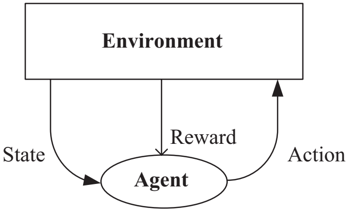

Figure 6.1 Basic setting for reinforcement learning where the agent interacts with its environment. At any state of the environment, the agent takes an action and the action changes the state and may or may not return a reward.

certain *state*, which means the position of the pieces on the board and the position of the robot in the maze, respectively. The decision maker has a set of *actions* possible: the legal movement of pieces on the chessboard and the movement of the robot in various directions without hitting any obstacle. Once an action is chosen and taken, the state changes.

The solution to the task requires a sequence of actions, and we get feedback in the form of a *reward*. What makes learning challenging is that the reward comes rarely and generally only after the complete sequence has been carried out—we win or lose the game after a long sequence of moves. The reward defines the aim of the task and is necessary if we want learning. The agent learns the best sequence of actions to solve the task where "best" is quantified as the sequence of actions that returns the maximum reward as early as possible. This is the setting of *reinforcement learning* (Sutton and Barto 1998).

Reinforcement learning is different from the learning methods we've already discussed in a number of respects. It is called "learning with a critic," as opposed to the learning with a teacher that we have in supervised learning. A critic differs from a teacher in that they do not tell us what to do, but only how well we have been doing in the past. The critic never informs in advance! The feedback from the critic is scarce and when it comes, it comes late. This leads to the *credit assignment* problem. After taking several actions and getting the reward, we would like to assess the individual actions we did in the past and find the moves that led us to win the reward so that we can record and recall them later on.

Actually, what a reinforcement learning program does is generate an *internal value* for the intermediate states or actions in terms of how good they are in leading us to the goal and getting us the real reward. Once such an internal reward mechanism is learned, the agent can just take the local actions to maximize it. The solution to the task requires a *sequence* of actions chosen in this way that cumulatively generates the highest real reward.

Unlike the previous scenarios we discussed, here there is no external process that provides the training data. It is the agent that actively generates data by trying out actions in the environment and receiving feedback (or not) in the form of a reward. It then uses this feedback to update its knowledge so that in time it learns to do actions that return the highest reward.

***K*-Armed Bandit**

We start with a simple example. The *K-armed bandit* is a hypothetical slot machine with *K* levers. The action is to choose and pull one of the levers to win a certain amount of money, which is the reward associated with the lever (action). The task is to decide which lever to pull to maximize the reward.

This is a classification problem where we choose one of K. If this were supervised learning, the teacher would tell us the correct class, namely, the lever leading to maximum earning. In this case of reinforcement learning, we can only try different levers and keep track of the best.

Initially estimated values for all levers are zero. To explore the environment, we can choose one of the levers at random and observe a reward. If that reward is higher than zero, we can just store it as our internal reward estimate of that action. Then, when we need to choose a lever again, we can keep on pulling that lever and receiving positive rewards. But it may be the case that another lever leads to a higher reward, so even after finding a lever with a positive reward we want to try out the other levers; we need to make sure that we have done a thorough enough exploration of the alternatives before we become set in our ways. Once we try out all levers and know everything there is to know, we can then choose the action with the maximum value.

The setting here assumes that rewards are deterministic, that we always receive the same reward for a lever. In a real slot machine, the reward is a matter of chance, and the same lever may lead to different reward values in different trials. In such a case, we want to maximize our

If this were supervised learning, the teacher would tell us the correct class, namely, the lever leading to maximum earning. In this case of reinforcement learning, we can only try different levers and keep track of the best.

expected reward, and our internal reward estimate for the action is the average of all rewards in the same situation. This implies that doing an action once is not enough to learn how good it is; we need to do many trials and collect many observations (rewards) to calculate a good estimate of the average.

The *K*-armed bandit is a simplified reinforcement learning problem because there is only one state—one slot machine. In the general case, when the agent chooses an action, not only does it receive a reward or not, but its state also changes. This next state of the agent may also be probabilistic because of the hidden factors in the environment, and this may lead to different rewards and next states for the same action.

For example, there is randomness in games of chance: in some games there are dice, or we draw randomly from a deck in card games. In a game like chess, there are no dice or decks of cards, but there is an opponent whose behavior is unpredictable—another source of uncertainty. In a robotic environment, the obstacles may move or there may be other mobile agents that can occlude perception or limit movement. Sensors may be noisy and motors that control the actuators may be far from perfect. A robot may want to go ahead, but because of wear and tear may swerve to the right or left. All these are hidden factors that introduce uncertainty, and as usual, we estimate expected values to average out the effect of uncertainty.

Another reason the *K*-armed bandit is simplified is because we get a reward after a single action; the reward is not delayed and we immediately see the value of our action. In a game of chess or with a robot whose task is to find the goal location in a room, the reward arrives only at the very end, after many actions during which we receive no reward or any other feedback.

In reinforcement learning, what we want is to be able to predict how good any intermediate action is in taking us to the real reward—this is our *internal reward estimate* for the action. Initially, this reward estimate for all actions is zero because we do not yet know anything. We need data to learn, so we need to do some *exploration* where we try out certain actions and observe whether we get any reward; we then update our internal estimates using this information.

As we explore more we collect more data, and we learn more about the environment and how good our actions are. When we believe we have reached a level where our reward estimates of actions are good enough, we can start *exploitation*. We do this by taking the actions that generate the highest reward according to our internal reward estimates. In the beginning when we do not know much, we try out actions at random; as we learn more, we gradually move from exploration to exploitation by moving from random choices to those influenced by our internal reward estimates.

Temporal Difference Learning

For any state and action, we want to learn the expected cumulative reward starting from that state with that action. This is an *expected* value because it is an average over all sources of randomness in the rewards and the states to come. The expected cumulative rewards of two consecutive state-action pairs are related through the *Bellman equation*, and we use it to *back up* the rewards from later actions to earlier actions, as follows.

Let us consider the final move of the robot that leads to the goal; because we reach the goal, we receive a reward of, say, 100 units (see figure 6.2). Now consider the state and action immediately before that. In that state we did an action, which, though it did not give us an immediate reward (because we were still one step away from the goal), led us to the state where with one more action we got the full reward of 100. So we discount this reward, let us say by a factor of 0.9, because the reward is in the future and the future is never certain—in other words, "A bird in the hand is worth two in the bush." So we can say that the state-action pair one step before the goal has an *internal reward* of 90.

Note that the real external, real reward there is still zero, because we still have not reached the goal, but we internally reward ourselves for having arrived at a state that is only one step away from the goal. Similarly, the one

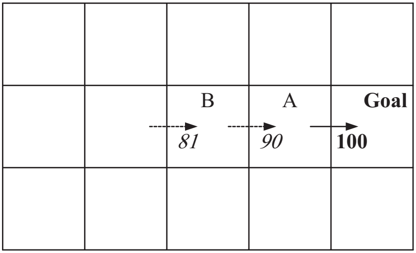

Figure 6.2 Temporal difference learning through reward backup. When we are in state A, if we go right, we get the real reward of 100. In state B just before that, if we do the correct action (namely, go right), we get to A where with one more action we can get the real reward, so it is as if going right in B also has a reward. But it is discounted (here by a factor of 0.9) because it is one step before, and it is a simulated internal reward, not a real one. The real reward for going from B to A is zero; the internal reward of 90 indicates how close we are to getting the real reward.

before that action gets an internal reward of 81, and we can continue assigning internal values to all the previous actions in that sequence. Of course, this is for only one trial episode. We need to do many trials where in each, because of the uncertainties, we observe different rewards and visit different next states, and we average over all those internal reward estimates. This is called *temporal difference (TD) learning*; the internal reward estimate for each state-action pair is denoted by *Q*, and the algorithm that updates them is called *Q learning*.

Note that only the final action gets us the real reward; all the values for the intermediate actions are simulated rewards. They are not the aim; they only help us to find the actions that eventually lead us to the real reward. Just like in a school, a student gets grades based on their performance in different courses, but those grades are only simulated rewards indicating how likely it is the student will get the real reward, which they will get only when they graduate and become a productive member of their community.

In certain applications, the environment is *partially observable*, and the agent does not know the state exactly. It is equipped with sensors that return an observation, which it uses to estimate the state of the environment. Let us say we have a robot that navigates in a room. The robot may not know its exact location in the room, or what else is in the room. The robot may have a camera, but an image does not tell the robot the environment's state exactly; it only gives some indication about the likely state. For example, the robot may only know that there is an obstacle to its left.

In such a case, based on the observation, the agent predicts its state; or more accurately speaking, it predicts the probability that it is in each state given the observation and then does the update for all probable states weighted by their probabilities. This additional uncertainty makes the task much more difficult and the problem harder to learn (Thrun, Burgard, and Fox 2005).

Reinforcement Learning Applications

One of the early applications of reinforcement learning is the *TD-Gammon* program that learns to play backgammon by playing against itself (Tesauro 1995). This program is superior to the previous *NeuroGammon* program also developed by Tesauro, which was trained in a supervised manner based on plays by experts. Backgammon is a complex task with approximately 10^{20} states; it features an opponent and extra randomness due to the roll of dice. Using a version of the temporal difference learning, the program achieves master-level play after playing 1,500,000 games against a copy of itself.

Though reinforcement learning algorithms are slower than supervised learning algorithms, it is clear that they have a wider variety of application and have the potential to construct better learning machines (Ballard 1997). They do not need any supervision, and this may actually be better since there will not be any teacher bias. For example, Tesauro's TD-Gammon program in certain circumstances came up with moves that turned out to be superior to those made by the best players.

A recent impressive work combines reinforcement learning with deep learning to play arcade games (Mnih et al. 2015). The *Deep Q Network* is composed of a convolutional neural network that takes directly the 84*84 image of the screen (from games from the 1980s when image

Though reinforcement learning algorithms are slower than supervised learning algorithms, it is clear that they have a wider variety of application and have the potential to construct better learning machines. … They do not need any supervision, and this may actually be better since there will not be any teacher bias.

resolution was low) and learns to play the game using only the image and the score information. The training is end-to-end, from pixels to actions. The same network with the same algorithm, network architecture, and hyperparameters can learn any of a number of arcade games. The program, which can play as well as a human player, comes up with interesting strategies not foreseen or expected by its programmers.

Very recently, the same group developed the *AlphaGo* program (Silver et al. 2016) that again combines deep convolutional networks with reinforcement learning, this time to play the game of Go. The input is the 19*19 Go board, and there is the policy network that is trained to select the best move and the value network trained to evaluate how good it is in winning the game. The policy network is first trained with a very large database of expert games and then further improved through reinforcement learning by playing against itself. *AlphaGo* defeated the European Go champion, 5 games to 0, in 2015 and defeated one of the greatest Go players in the world, 4 games to 1, in March 2016.

Scaling these approaches up to more complex scenarios with more complex input and larger action sets is one of the current challenges in reinforcement learning. The most important and impressive characteristic of these systems is that training is done end-to-end, from raw input to actions, without any assumed intermediate representation between the two parts of perception and action. For example, one can imagine that applying the same approach to chess will be difficult because though the board is smaller, unlike in Go, all pieces are not the same.

It will also be interesting to see whether this type of end-to-end training can also be used in tasks other than game playing where any intermediate processing or representation is automatically learned from data. For example, consider a smartphone with a translation app where you speak in English and the person on the other end hears your sentence uttered in French.

7

WHERE DO WE GO FROM HERE?

Make Them Smart, Make Them Learn

Machine learning has already proved itself to be a viable technology, and its applications in many domains are increasing every day. This trend of collecting and learning from data is expected to continue even stronger in the near future (Jordan and Mitchell 2015). Analysis of data allows us to both understand the process that underlies the past data—just like scientists have been doing in different domains of science for hundreds of years—and also predict the behavior of the process in the future.

A few decades ago, computing hardware used to advance one microprocessor at a time; every new microprocessor—first size 8, then 16, then 32 bits—could do slightly more computation in unit time and could use slightly more memory, and this translated to slightly better computers. Similarly, computer software used to advance one programming language at a time. Each new language made some new type of computation easier to program. When computers were used for number crunching, we programmed in Fortran; when we used them for business applications, we used Cobol; later on, when computers started to process all types of and more complex types of information, we developed object-oriented languages that allowed us to define more complicated data structures together with the specialized algorithms that manipulate them.

Then, computing started to advance one operating system at a time; each new version made computers easier to use and supported a new set of applications. Nowadays, computing is advancing one smart device or one smart app at a time. Once, the key person who defined the advance of computing was the hardware designer, then it was the software engineer, then it was the user sitting in front of their computer, and now it is anyone while doing anything.

Nobody eagerly awaits a new microprocessor anymore, and neither a new programming language nor a new operating system version is newsworthy. Now we wait for the next new device or app, smart either because of its designer, or because it learns to be smart.

More and more of our lives are being projected in the digital domain, and as a result we are creating more and more data. The earliest hard disks for personal computers had a capacity of five megabytes; now a typical computer comes with five hundred gigabytes—this is one hundred thousand times more storage capacity in roughly thirty years. A reasonably large database now stores hundreds of terabytes, and we have already started using the petabyte as a measure; very soon, we will jump to the next measure, the exabyte, which is one thousand petabytes or one million terabytes. Together with an increase in storage capacity, processing has also become cheaper and faster thanks to advances in technology that deliver both faster computer chips and parallel architectures containing thousands of processors that run simultaneously, each solving one part of a large problem.

The trend of moving from all-purpose personal computers to specialized smart devices is also expected to accelerate. We discussed in chapter 1 how in the past organizations moved from a computer center to a distributed scheme with many interconnected computers and storage devices; now a similar transformation is taking place for a single user. A person no longer has one personal computer that holds all their data and does all their processing; instead, their data is stored in the "cloud," in some remote offsite data center, but in such a way as to be accessible from all their smart devices, each of which accesses the part it needs.

What we call the *cloud* is a virtual computer center that handles all our computing needs. We do not need to worry about where and how the processing is done or where and how the data is stored as long as we can keep accessing it whenever we want. This used to be called *grid computing*, analogous to the electrical grid made up of an interconnected set of power generators and consumers; as consumers, we plug our TV in the nearest outlet without a second thought as to where the electricity comes from.

This also implies connectivity with larger bandwidths. Streaming music and video is already a feasible technology today. CDs and DVDs we keep on our shelves (that had once supplanted the nondigital LPs and videotapes) have now in turn become useless and are replaced by some invisible source that stores *all* the songs and the movies. E-book and digital subscription services are quickly replacing the printed book and the bookstore, and search engines have long ago made window stoppers out of thick encyclopedias.

With smart devices, there is no longer any need for millions of people to keep separate copies of the same song/movie/book locally all the time. The motto now is "Do not buy it, rent it!" Buy the smart device, or the app, or the subscription to the service, and the bandwidth that allows you to access it when you need it.

This change and ease in accessibility also offers new ways to "package" and sell products. For example, traditionally LPs and CDs corresponded to an "album" that is made up of a number of songs; it is now possible to rent individual songs. Similarly, it is now possible to purchase a single short story without buying the book of collected stories.

In chapter 5, we discussed the use of machine learning in recommendation systems. With more shared data and streaming, there will be more data to analyze, and furthermore, the data will be more detailed. For example, now we also know how many times a person listened to a song or how far they read into a novel, and such information can be used as a measure of how much the person enjoyed the product.

With advances in mobile technology, there is continuing interest in wearable devices. The smartphone, a wearable device, is now much more than a phone; it also acts as an intermediary for smaller smart “things,” such as a watch or glasses, by putting them online. The phone may become even smarter in the near future, for example, with an app for real-time translation: you’ll speak in your own language on one end and the person on the other end will hear it automatically translated into their own tongue, not only with the content syntactically and semantically correct but also in your voice and uttered with correct emphasis and intonation.

Machine learning will help us make sense of an increasingly complex world. Already we are exposed to more data than what our sensors can cope with or our brains can process. Information repositories available online today contain massive amounts of digital text and are now so big

Machine learning will help us make sense of an increasingly complex world. Already we are exposed to more data than what our sensors can cope with or our brains can process.

that they cannot be processed manually. Using machine learning for this purpose is called *machine reading*.

We need search engines that are smarter than the ones that use just keywords. Nowadays we have information distributed in different sources or mediums, so we need to query them all and merge the responses in an intelligent manner. These different sources may be in different languages—for example, a French source may contain more information on the topic even though your query is in English. A query may also trigger a search in an image or video database. And still, the overall result should be summarized and condensed enough to be digestible by a user.

Web scraping is when programs automatically surf the web and extract information from web pages. These web pages may be social media, and accumulated information can be analyzed by learning algorithms, for instance, to track trending topics and the detection of sentiment, opinions, and beliefs about products and people—for example, politicians in election times. Another important research area where machine learning is used in social media is to identify the "social networks" of people who are connected. Analyzing such networks allows us to find cliques of like-minded individuals, or to track how information propagates over social media.

One of the current research directions is in adding smartness—that is, the ability to collect and process data, as well as to share it with other online devices—to all sorts of traditional tools and devices, including traditional wearables such as glasses and watches. When more devices are smart, there will be more data to analyze and make meaningful inferences from. Different devices and sensors collect different aspects of the task, and it is critical how we will combine and integrate these multiple modalities. This implies all sorts of new interesting scenarios and applications where learning algorithms can be used.

Smart devices can help us both at work and at home. Machine learning helps us in building systems that can learn their environment and adapt to their users, to be able to work with minimum supervision and maximum user satisfaction.

Important work is being done in the field of smart cars. Cars (or buses, trucks, etc.) that are online allow their passengers to be online and can deliver all types of online services, such as streaming video, over their digital *infotainment* systems. Cars that are online can also exchange data for maintenance purposes and access real-time information about the road and weather conditions. If you are driving under difficult conditions, a car that is a mile ahead of you is a sensor that is a mile ahead of you.

But more important than being online is when cars will be smart enough to help with the driving itself. Cars already have assistance systems for cruise control, self-parking, and lane keeping, but soon they will become even more capable. The ultimate aim is for them to completely take over the task of driving, and to that end we already have prototypes of such autonomous vehicles today.

The visual system of a human driver does not have a very high resolution, and they can only look in a forward direction. Though their visual field is slightly extended through the use of side and rearview mirrors, blind spots remain. A *self-driving car*, on the other hand, can have cameras with higher resolution in all directions and can also use sensors that a human does not have, such as GPS, ultrasound, or night vision, or it can be equipped with a special type of radar, called LIDAR, that uses a laser for measuring distance. A smart car can also access all sorts of extra information, such as the weather, much faster. An electronic driver has a much shorter reaction time.

Machine learning plays a significant role in self-driving cars that will result in both smoother driving, faster control, and greater fuel efficiency, but also in smart sensing, for example, by automatic recognition of pedestrians, cyclists, traffic signs, and so forth. Self-driving cars will be safer and faster. There are still problems though: lasers and cameras are not very effective in harsh weather conditions—when there is rain, fog, or snow—so technology should advance until we get smart cars that can run in all types of weather.

Self-driving cars and robot taxis are expected to take over driving in cities and on highways in the next decade, perhaps initially in designated lanes, and later on as part of the usual traffic. It also seems very likely that sometime in the next decade or so, cars and drones will fuse and we will have self-piloting flying cars, with their concomitant tasks that will be best handled by machine learning.

Machine learning has the basic advantage that a task does not need to be explicitly programmed but can be learned. Space will be the new frontier for machine learning as well. Future space missions will very likely be unmanned. Before, we needed to send humans because we did not have machines that were as smart and versatile, but nowadays we have capable robots. If there are no humans on board, the load will be lighter and simpler, and there is no need to bring the load back. If a robot is to boldly go where no one has gone before, it can only be a learning robot.

High-Performance Computation

With big and bigger data, we need storage systems that have higher capacity *and* faster access. Processing power will necessarily increase so that more data can be processed in a reasonable time. This implies the need for high-performance computer systems that can store a lot of data and do a lot of computation very quickly.

There are physical limits such as the speed of light and the size of the atom, which suggests a higher limit on the speed of transfer[1] and a lower limit on the size of the basic electronics. The obvious solution to this is parallel processing—if you have eight lines in parallel, you can send eight data items at the same time; and if you have eight processors, you can process those eight items simultaneously, in the time it takes to process a single one.

Nowadays parallel processing is routinely used in computer systems. We have powerful computers that contain thousands of processors running simultaneously. There are also multicore machines where a single computing element has multiple “cores” that can do simple computations simultaneously, implementing parallel processing in a single physical chip.

But high-performance computation is not just a hardware problem; we also need good software interfaces to distribute the computation and data of different applications efficiently over a very large number of processors and storage devices. Indeed, software and hardware for parallel and distributed computation for big data are important research areas in computer science and engineering today.

In machine learning, the parallelization of learning algorithms is becoming increasingly important. Models can be trained in parallel over different parts of the data on different computers and then these models can be merged. Another possibility is to distribute the processing of a single model over multiple processors. For example, with a deep neural network composed of thousands of units in multiple layers, different processors can execute different layers or subsets of layers, and streaming data can be processed much faster in a pipeline manner.

The *graphical processing unit* (GPU) was originally made for rapid processing and the transfer of images in graphical interfaces—for example, in video game consoles—but the type of parallel computation and transfer used for graphics has also made them suited for many machine learning tasks. Indeed, specialized software libraries are being developed for this purpose and GPUs are frequently used by researchers and practitioners effectively in various machine learning applications.

We are seeing a trend toward *cloud computing* in machine learning applications too, where instead of buying and maintaining the necessary hardware, people rent the use of offsite *data centers*. A data center is a physical site that houses a very large number of computing servers with many processors and ample storage. There are typically multiple data centers in physically different locations; they are all connected over a network, and the tasks are automatically distributed and migrated from one to the other, so that the load from different customers at different times and in different sizes is balanced. All of these requirements fuel significant research today.

One important use of the cloud is in extending the capability of smart devices, especially the mobile ones. These online, low-capacity devices can access the cloud from anywhere to exchange data or request computation that is too large or complex to do locally. Consider speech recognition on a smartphone. The phone captures the acoustic data, extracts the basic features, and sends them to the cloud. The actual recognition is done in the cloud and the result is sent back to the phone.

In computing, there are two parallel trends. One is in building general-purpose computers that can be programmed for different tasks and for different purposes, such as those used in servers in data centers. The other is to build specialized computing devices for particular tasks, packaged together with specialized input and output. The latter used to be called *embedded systems* but are nowadays called *cyber-physical systems*, to emphasize the fact that they work in the physical world with which they interact. A system may be composed of multiple units (some of which may be mobile,) and they are interconnected over a network—for example, a car, a plane, or a home may contain a multitude of such devices for different tasks. Making such systems smart—in other words, able to adapt to their particular environment, which includes the user—is an important research direction.

Data Mining

Though the most important, machine learning is only one step in a *data mining* application (Han and Kamber 2011). There is also the preparation of data beforehand and the interpretation of the results afterward.

Making data ready for mining involves several stages. First, from a large database with many fields, we select the parts that we are interested in and create a smaller database to work with. It may also be the case that the data comes from different databases, so we need to merge them. The level of detail may also be different—for instance, from an operational database we may extract daily sums and use those rather than the individual transactions. Raw data may contain errors and inconsistencies or parts of it may be missing, and those should be handled beforehand in a preprocessing stage.

After extraction, data is stored in a *data warehouse* on which we do our analysis. One type of data analysis is manual where we have a hypothesis—“people who buy beer also buy chips”—and check whether the data supports the hypothesis. The data now is in the form of a spreadsheet where the rows are the data instances—baskets—and the columns are the attributes—products. One way of conceptualizing the data is in the form of a *multidimensional data cube* whose dimensions are the attributes, and data analysis operations are defined as operations on the cube, such as slice, dice, and so on. Such manual analysis of the data as well as visualization of results is made easy by *online analytical processing* (OLAP) tools.

OLAP is restrictive in the sense that it is human-driven, and we can test only the hypotheses we can imagine. For example, in the context of basket analysis, we cannot find any relationship between distant pairs of products; such discoveries require a data-driven analysis, as is done by machine learning algorithms.

We can use any of the methods we discussed in previous chapters, for classification, regression, clustering, and so on, to build a model from the data. Typically, we divide our data into two as a training set and a validation set. We use the first part for training our model and then we measure its prediction accuracy on the validation set. By testing on instances not used for training, we want to estimate how well the trained model would do if used later on, in the real world. The validation set accuracy is one of our main criteria in accepting or rejecting the trained model.

Certain machine learning algorithms learn *black box models*. For example, with a neural network, given an input, the network calculates an output, but it is difficult to understand what happens in its intermediate layers. On the other hand, if-then rules as found by decision trees are interpretable, and such rules can be checked and evaluated by people who know the application (though they may not know machine learning). In many data mining scenarios—for example, in credit scoring—this process of knowledge extraction and model assessment by experts may be important and even necessary in validating the model trained from data.

Visualization tools may also help here. Actually, visualization is one of the best tools for data analysis, and sometimes just visualizing the data in a smart way is enough to understand the characteristics of a process that underlies a complicated data set, without any need for further complex and costly statistical processing; see Börner 2015 for examples.

As we have more data and more computing power, we can attempt more complicated data mining tasks that try to discover hidden relationships in more complex scenarios. Most data mining tasks nowadays work in a single domain using a single source of data. Especially interesting is the case where we have data from different sources in different modalities; mining such data and finding dependencies across sources and modalities is a promising research direction.

Data Privacy and Security

When we have a lot of data, its analysis can lead to valuable results, and historically, data collection and analysis have resulted in significant findings for humanity, in many domains from medicine to astronomy. The current widespread use of digital technology allows us to collect and analyze the data quickly and accurately, and in numerous new domains.

With more and detailed data, the critical point nowadays is *data privacy and security* (Horvitz and Mulligan 2015). How can we make sure that we collect and process data without infringing people's privacy concerns and that the data is not used for purposes beyond its original intention?

We expect individuals in a society to be aware of the advantages of data collection and analysis in domains such as health care and safety. And even in less critical domains such as retail, people always appreciate services and products tailored to their likes and preferences. Still, no one would like to feel that their private life is being pried into. Our smart devices, for example, should not turn into digital paparazzi recording the details of our lives and making them available without our knowledge or consent.

The basic requirements are that the user who generates the data should always know what and how much data is collected, what part of it is stored, whether the data will be analyzed for any purpose, and if so, what that purpose is. Companies should be completely transparent about the data they collect and analyze.

The owner of the data should always be informed during both data collection and use. Before any analysis, the data should be *sanitized* and all personal details hidden to make it anonymous. Anonymizing data is not a straightforward process. With human records, for instance, just removing unique identifiers such as the name or social security number is not enough; fields such as birth date, zip code, and so on provide partial clues and individuals can be identified by those who combine them (Sweeney 2002).

Data is becoming such a valuable raw resource that it behooves the collector of the data to take all the necessary steps for its safekeeping and to not share the data with someone else without the explicit consent of the data owner.

Individuals should have complete control over their data. They should always have the means to view what data of theirs has been collected; they should be able to ask for its correction or complete removal.

A recent line of research is in *privacy-preserving learning* algorithms. Let us say that we have parts of data from different sources (for example, different countries may have patients suffering from the same disease) and that they do not want to lend their data (detailed information about their citizens) to a central user to train a model with all the data combined. In such a case, one possibility is to share the data in a form that is sufficiently anonymized; another possibility is to train different models with the different parts and share the trained models, or combine the separately trained models.

Such a concern over data privacy and security should be an integral part of any data analysis scenario, and it should be resolved before any learning is done. Mining data is just like mining for gold—before you start any digging, you need to make sure that you have all the necessary permits. In the future, we may have data processing standards where every data set contains some metadata about this type of ownership and permission information; then, it may be required that any machine learning or data analysis software check for these and run only if the necessary stamps of approval are there.

Data Science

The advances and successes of machine learning methods on big data and the promise of more have prompted researchers and practitioners in the industry to name this endeavor as a new branch of science and engineering. There are still discussions about what this new field of *data science* covers, but it seems as if the major topics are machine learning, high-performance computing, and data privacy/security.

Of course, not all learning applications need a cloud, or a data center, or a cluster of computers. One should always be wary of hype and companies' sale strategies to invent new and fancier names under which to sell old products.

However, when there is a lot of data and the process involves a lot of computation, efficient implementation of machine learning solutions is an important matter.[2] Another integral part is the ethical and legal implications of data analysis and processing. For example, as we collect and analyze more and more data, our decisions in various domains will become more and more automated and data-driven, and we need to be concerned about the implications of such autonomous processes and the decisions they make.

It seems as if we will need many “data scientists” and “data engineers” in the future, because we see today that the importance of data and extracting information from data has been noticed in many domains. Such scenarios have characteristics that are drastically different than those of traditional statistics applications.

First, the data now is much bigger—consider all the transactions done at a supermarket chain. For each instance, we have thousands of attributes—consider a gene sequence. The data is not just numbers anymore; it consists of text, image, audio, video, ranks, frequencies, gene sequences, sensor arrays, click logs, lists of recommendations, and so on. Most of the time data does not obey the parametric assumptions, such as the bell-shaped Gaussian curve, that we use in statistics to make estimation easier. Instead, with the new data, we need to resort to more flexible nonparametric models whose complexity can adjust automatically to the complexity of the task underlying the data. All these requirements make machine learning more challenging than statistics as we used to know and practice it.

In education, this implies that we need to extend the courses on statistics to cover these additional needs, and teach more than the well-known but now insufficient, mostly univariate (having a single input attribute) parametric methods for estimation, hypothesis testing, and regression. It has also become necessary nowadays to teach the basics of high-performance computing, both the hardware and the software aspects, because in real-world applications how efficiently the data is stored and manipulated may be as critical as the prediction accuracy. A student of data science also needs to know the fundamentals of data privacy and security, and should be aware of the possible implications of data collection and analysis in ethics and law.

Machine Learning, Artificial Intelligence, and the Future

Machine learning is one way to achieve artificial intelligence. By training on a data set, or by repeated trials using reinforcement learning, we can have a computer program behaving so as to maximize a performance criterion, which in a certain context appears intelligent.

One important point is that intelligence is a vague term and its applicability to assess the performance of computer systems may be misleading. For example, evaluating computers on tasks that are difficult for humans, such as playing chess, is not a good idea for assessing their intelligence. Chess is a difficult task for humans because it requires deliberation and planning, whereas humans, just like other animals, have evolved to make very quick decisions using limited sensory data with limited computation. For a computer, it is much more difficult to recognize the face of its opponent than to play chess. Whether a computer can play chess better than the best human player is not a good indicator that computers are more intelligent, because human intelligence has not evolved for tasks like chess.

Researchers use game playing as a testing area in artificial intelligence because games are relatively easy to define with their formal rules and clearly specified criteria for winning and losing. There are a certain number of pieces or cards, and even if there is randomness its form is well defined: the dice should be fair and draws from the deck should be uniform. Attempts to the contrary are considered cheating behavior. In real life, all sorts of randomness occurs, and for its survival every species is slowly evolving to be a better cheater than the rest.

An important problem is what the performance criterion should be for a behavior to be considered intelligent (that is, how we measure intelligence) and whether there are tasks for which such a performance criterion is not evident. We have already discussed that ethical and legal concerns play a role in certain type of decisions.

As we have more and more computer systems that are trained from data and make autonomous decisions, we need to be concerned with relying so much on computers. One important requirement is the validation and verification of software systems—that is, making sure that they do what they should do and do not do what they should not do. This may be especially difficult for models trained from data, because training involves all sorts of randomness in data and optimization; this makes trained software less predictable than programmed software. Another concern is that models that learn the general behavior in the data may not make good decisions for underrepresented cases or outliers.

For example, there is an important risk in basing recommendations too much on past use and preferences. If a person only listens to songs similar to the ones they listened to and enjoyed before, or watches movies similar to those they watched and enjoyed before, or reads books similar to the books they read and enjoyed before, then there will be no new experience and that will be limiting, both for the person and for the company that is always eager to find new products to sell. So in any such recommendation scheme, there should also be some attempt at introducing some diversity.

A recent study (Bakshy, Messing, and Adamic 2015) has shown that a similar risk also exists for interactions on social media. If a person follows only those people they agree with and reads posts, messages, and news similar to the ones they have read in the past, they will be unaware of other people's opinions and that will limit their experience, as opposed to traditional news media outlets, such as newspapers or TV, that contain a relatively wider range of news and opinions.

When intelligence is embodied and the system takes physical actions, the correctness of the behavior becomes an even more critical issue and even human life may be at stake. The system does not need to be a drone with onboard weapons for this to be true; even an autonomous car becomes a weapon if it is driven badly. When such concerns come into play, the usual expected value or utilitarian approaches do not apply, as is discussed in the "trolley problems," a variant of which is as follows.

Let us say we are riding in an autonomous car when a child suddenly runs across the road. Assume the car is going so fast that it knows it cannot stop. But it can still steer, and it can steer to the right to avoid hitting the child. But let us say that the child's mother is standing to the right of the road. How should the autonomous car decide? Should it go ahead and hit the child, or steer right and hit the mom instead? How can we program such a decision? Or should the car instead steer left and drive off the cliff after calculating that your life is worth less than that of the child or the mother's?

The power that artificial intelligence promises is a concern for many researchers, and not surprisingly there is a call for regulation. In a recent interview (Bohannon 2015), Stuart Russell, a prominent researcher and coauthor of the leading textbook on artificial intelligence (Russell and Norvig 2009), says that unlimited intelligence may be as dangerous as unlimited energy and that uncontrolled artificial intelligence may be as dangerous as nuclear weapons. The challenge is to make sure that this new source of intelligence is used for good and not for bad, to increase the well-being of people, and for the benefit of humanity rather than to increase the profit of a few.

Some people jump to conclusions and fear that research on artificial intelligence may one day lead to metallic monsters that will rise to dominate us—electronic versions of the creation of Dr. Frankenstein. I doubt whether that will ever happen. But even today we have automatic systems that make decisions for us—some of which may be trained from data—in various applications from cars to trading. I believe we have more reason to fear the poorly programmed or poorly trained software than we do to dread the possibility of the dawn of superintelligent machines.

Closing Remarks

We have big data, but tomorrow's data will be bigger. Our sensors are getting cheaper and hence being used more widely and more precisely. Computers are getting bigger too, in terms of their computing power. We still seem to be far from the limits imposed by physics as researchers find new technologies and materials, such as the graphene, that promise to deliver more. New products can be designed and produced much faster using 3D printing technology and more of these products will need to be smart.

With more data and computation, our trained models can get more and more intelligent. Current deep networks are not deep enough; they can learn enough abstraction in some limited context to recognize handwritten digits or a subset of objects, but they are far from having the capability of our visual cortex to recognize a scene. We can learn some linguistic abstraction from large bodies of text but we are far from any real understanding of it—enough, for example, to answer questions about a short story. How our learning algorithms will scale up is an open question. That is, can we train a model that is as good as the visual cortex by adding more and more layers to a deep network and training it with more and more data? Can we get a model to translate from one language to another by having a very large model trained with a lot of data? The answer should be yes, because our brains are such models. But this scaling up may be increasingly difficult. Even though we are born with the specialized hardware, it still takes years of observing our environment before we utter our first sentence.

In vision, as we go from barcode to optical character readers to face recognizers, we define a sequence of increasingly complex tasks, each of which solves a need and each of which is a marketable product in its own time. More than scientific curiosity, it is this process of capitalization that fuels research and development. As our learning systems get more intelligent, they will find use in increasingly smarter products and services.

In the last half century, we have seen that as computers find new applications in our lives, they have also changed our lives to make computation easier. Similarly, as our devices get smarter, the environment in which we live, and our lives in it, will change. Each age uses its current technology, which defines an environment with its constraints, and these propel new inventions and new technologies. If we can go back two thousand years and somehow give Romans cell phone technology, I doubt that it would greatly enhance their quality of life, when they were still riding horses, that is, when the rest of their lives did not match up. The world when we will need human-level intelligence in machines will be a very different world.

When will we reach that level of intelligence and how much processing and training will be required are yet to be seen. Currently machine learning seems to be the most promising way to achieve it, so stay tuned.

NOTES

1 Why We Are Interested in Machine Learning

1. These use the ASCII code devised for the English alphabet and punctuation. The character-encoding schemes we use today cover the different alphabets of different languages.

2. It is not the computing power, storage capacity, or connectivity that by themselves produce added value, just as a higher population does not necessarily imply a larger workforce. The enormous number of smartphones in the developing countries does not translate to wealth.

3. A computer program is composed of an algorithm for the task and data structures for the digital representation of the processed information. The title of a seminal book on computer programming is just that: *Algorithms + Data Structures = Programs* (Wirth 1976).

4. Early scientists believed that the existence of such rules that explain the physical world is a sign of an ordered universe, which could only be due to a god. Observing nature and trying to fit rules to natural phenomena has an old history, starting in ancient Mesopotamia. Early on, pseudoscience could not be separated from science. In hindsight, the fact that the ancient people believed in astrology is not surprising: if there are regularities and rules about the movement of the sun and the moon, which can be used to predict eclipses for example, positing the existence of regularities and rules about the movement of human beings, which seem so petty in comparison, does not sound far-fetched.

2 Machine Learning, Statistics, and Data Analytics

1. https://en.wikipedia.org/wiki/Depreciation.

2. For an excellent history of artificial intelligence, see Nilsson 2009.

3. See Sandel 2012 for some real-life scenarios where decision making based on expected value, or expected utility, may not be the best way. *Pascal's wager* is another example of the application of expected value calculation in an area where it should not be applied.

3 Pattern Recognition

1. Here, we are talking about *optical* character recognition where the input is an image; there is also *pen-based* character recognition where the writing is done on a touch-sensitive pad. In such a case, the input is not an image but a sequence of the (x,y) coordinates of the stylus tip, while the character is written on the touch-sensitive surface.
2. Let us say F represents the flu and N represents a runny nose. Using Bayes' rule, we can write:

$$P(F|N)=P(N|F)P(F)/P(N),$$

where $P(N|F)$ is the conditional probability that a patient who is known to have the flu has a runny nose. $P(F)$ is the probability that a patient has the flu, regardless of whether they have a runny nose or not, and $P(N)$ is the probability that a patient has a runny nose, regardless of whether they have the flu or not.

3. It is interesting that in many science fiction movies, though the robots may be very advanced in terms of vision, speech recognition, and autonomous movement, they still continue to speak in an emotionless, "robotic" voice.
4. Bayesian estimation uses Bayes' rule in probability theory (which we saw before) named after Thomas Bayes (1702–1761) who was a Presbyterian minister. The assumption of a prior that exists before and underlies the observable data should have come naturally with the job.

7 Where Do We Go from Here?

1. The speed of light is approximately 300,000 km/sec, so it takes approximately 3.33 milliseconds to traverse 1,000 km—distance to a data center. This is not actually such a small number with electronic devices. The connection is never direct and there are always delays due to intermediate routing devices; and remember that to get a response, we need to send a query first, so we need to double the time.
2. For more, see *Frontiers in Massive Data Analysis* (Washington, DC: National Academies Press, 2013).

GLOSSARY

Anonymization
Removal or hiding of information such that the source cannot be uniquely identified. It is not as straightforward as one would think.

Artificial intelligence
Programming computers to do things, which, if done by humans, would be said to require "intelligence." It is a human-centric and ambiguous term: calling computers "artificially intelligent" is like calling driving "artificial running."

Association rules
If-then rules associating two or more items in *basket analysis*. For example, "People who buy diapers frequently also buy beer."

Autoencoder network
A type of *neural network* that is trained to reconstruct its input at its output. Because there are fewer intermediary hidden units than inputs, the network is forced to learn a short compressed representation at the hidden units, which can be interpreted as a process of abstraction.

Backpropagation
A learning algorithm for artificial neural networks used for *supervised learning*, where connection weights are iteratively updated to decrease the approximation error at the output units.

Bag of words
A method for document representation where we preselect a lexicon of N words and we represent each document by a list of length N where element i is 1 if word i exists in the document and is 0 otherwise.

Basket analysis
A basket is a set of items purchased together (for example, in a supermarket). Basket analysis is finding items frequently occurring in the same basket. Such dependencies between items are represented by *association rules*.

Bayes' rule
One of the pillars of probability theory where for two or more random variables that are not independent, we write conditional probability in one direction in terms of the conditional probability in the other direction:

$$P(B|A) = P(A|B)P(B)/P(A).$$

It is used, for example, in diagnosis where we are given $P(A|B)$ and B is the cause of A. Calculating $P(B|A)$ allows a diagnostics—that is, the calculation of the probability of the cause B given the symptoms A.

Bayesian estimation
A method for parameter estimation where we use not only the sample, but also the prior information about the unknown parameters given by a *prior distribution*. This is combined with the information in the data to calculate a *posterior distribution* using Bayes' rule.

Bayesian network
See *graphical model*.

Bioinformatics
Computational methods, including those that use machine learning, for analyzing and processing biological data.

Biometrics
Recognition or authentication of people using their physiological characteristics (for example, face, fingerprint) and behavioral characteristics (for example, signature, gait).

Character recognition
Recognizing printed or handwritten text. In optical recognition, the input is visual and is sensed by a camera or scanner. In pen-based recognition, the writing

is done on a touch-sensitive surface and the input is a temporal sequence of coordinates of the pen tip.

Class
A set of instances having the same identity. For example, 'A' and '*A*' belong to the same class. In machine learning, for each class we learn a *discriminant* from the set of its examples.

Classification
Assignment of a given instance to one of a set of *classes*.

Cloud computing
A recent paradigm in computing where data and computation are not local but handled in some remote off-site data center. Typically there are many such data centers, and the tasks of different users are distributed over them in a way invisible to the user. This was previously called grid computing.

Clustering
Grouping of similar instances into clusters. This is an *unsupervised learning* method because the instances that form a cluster are found based on their similarity to each other, as opposed to a *classification* task where the supervisor assigns instances to classes by explicitly labeling them.

Connectionism
A neural network approach in cognitive science where mind is modeled as the operation of a network of many simple processing units running in parallel. Also known as *parallel distributed processing*.

Cyber-physical systems
Computational elements directly interacting with the physical world. Some may be mobile. They may be organized as a network to handle the task in a collaborative manner. Also known as *embedded systems*.

Data analysis
Computational methods for extracting information from large amounts of data. *Data mining* uses machine learning and is more data-driven; *OLAP* is more user-driven.

Data mining
Machine learning and statistical methods for extracting information from large amounts of data. For example, in *basket analysis*, by analyzing large number of transactions, we find *association rules*.

Data science
A recently proposed field in computer science and engineering composed of machine learning, high performance computing, and data privacy/security. Data science is proposed to handle in a systematic way the "big data" problems that face us today in many different scenarios.

Data warehouse
A subset of data selected, extracted, and organized for a specific data analysis task. The original data may be very detailed and may lie in several different operational databases. The warehouse merges and summarizes them. The warehouse is read-only; it is used to get a high-level overview of the process that underlies the data either through *OLAP* and visualization tools, or by *data mining* software.

Database
Software for storing and processing digitally represented information efficiently.

Decision tree
A hierarchical model composed of decision nodes and leaves. The decision tree works fast, and it can be converted to a set of *if-then rules*, and as such allows *knowledge extraction*.

Deep learning
Methods that are used to train models with several levels of abstraction from the raw input to the output. For example, in visual recognition, the lowest level

is an image composed of pixels. In layers as we go up, a deep learner combines them to form strokes and edges of different orientations, which can then be combined to detect longer lines, arcs, corners, and junctions, which in turn can be combined to form rectangles, circles, and so on. The units of each layer may be thought of as a set of primitives at a different level of abstraction.

Dimensionality reduction
Methods for decreasing the number of input attributes. In an application, some of the inputs may not be informative, and some may correspond to different ways of giving the same information. Reducing the number of inputs also reduces the complexity of the learned *model* and makes training easier. See *feature selection* and *feature extraction*.

Discriminant
A rule that defines the conditions for an instance to be an element of a *class* and as such separates them from instances of other classes.

Document categorization
Classification of text documents, generally based on the words that occur in the text (for example, using *bag of words* representation). For instance, news documents can be classified as politics, arts, sports, and so on; emails can be classified as spam versus not-spam.

Embedded systems
See *cyber-physical systems*.

Face recognition
Recognizing people's identities from their face images captured by a camera.

Feature extraction
As a method for *dimensionality reduction*, several original inputs are combined to define new, more informative features.

Feature selection
A method that discards the uninformative features and keeps only those that are informative; it is another method for *dimensionality reduction*.

Generalization
How well a model trained on a training set performs on new data unseen during training. This is at the core of machine learning. In an exam, a teacher asks questions that are different from the exercises she has solved while teaching the course, and a student's performance is measured on her performance on these new questions. A student that can solve only the questions that the instructor has solved in class is not good enough.

Generative model
A model defined in such a way so as to represent the way we believe the data has been generated. We think of hidden causes that generate the data and also of higher-level hidden causes. Slippery roads may cause accidents, and rain may have caused roads to be slippery.

Graphical model
A model representing dependencies between probabilistic concepts. Each node is a concept with a different truth degree and a connection between nodes represents a conditional dependency. If I know that the rain causes my grass to get wet, I define one node for rain and one node for wet grass, and I put a directed connection from the rain node to the node for wet grass. Probabilistic inference on such networks may be implemented as efficient graph algorithms. Such networks are a visual representation and this helps understanding. Also known as a *Bayesian network*—one rule of inference used in such networks is *Bayes' rule*.

High-performance computing
To handle the big data problems we have today in reasonable time, we need powerful computing systems, both for storage and calculation. The field of high-performance computing includes work along these directions; one approach is *cloud computing*.

If-then rules
Decision rules written in the form of "IF antecedent THEN consequent." The antecedent is a logical condition and if holds true for the input, the action in the consequent is carried out. In *supervised learning*, the consequent corresponds to choosing a certain output. A rule base is composed of many if-then rules. A model that can be written as a set of if-then rules is easy to understand and hence rule bases allow *knowledge extraction*.

Ill-posed problem
A problem where the data is not sufficient to find a unique solution. Fitting a model to data is an ill-posed problem, and we need to make additional assumptions to get a unique model; such assumptions are called the inductive bias of a learning algorithm.

Knowledge extraction
In some applications, notably in data mining, after training a model, we would like to be able to understand what the model has learned; this can be used for validating the model by people who are experts in that application, and it also helps to understand the process that generated the data. Some models are "black box" in that they are not easy to understand; some models—for example, linear models and decision trees—are interpretable and allow extracting knowledge from a trained model.

Latent semantic analysis
A learning method where the aim is to find a small set of hidden (latent) variables that represent the dependencies in a large sample of observed data. Such hidden variables may correspond to abstract (for example, semantic) concepts. For example, each news article can be said to include a number of "topics," and although this topic information is not given explicitly in a supervised way in the data, we can learn them from data such that each topic is defined by a particular set of words and each news article is defined by a particular set of topics.

Model
A template formalizing the relationship between an input and an output. Its structure is fixed but it also has parameters that are modifiable; the parameters are adjusted so that the same model with different parameters can be trained on different data to implement different relationships in different tasks.

Natural language processing
Computer methods used to process human language, also called computational linguistics.

Nearest-neighbor methods
Models where we interpret an instance in terms of the most similar training instances. They use the most basic assumption: similar inputs have similar outputs. They are also called instance-, memory-, or case-based methods.

Neural network
A model composed of a network of simple processing units called neurons and connections between neurons called synapses. Each synapse has a direction and a weight, and the weight defines the effect of the neuron before on the neuron after.

Nonparametric methods
Statistical methods that do not make strong assumptions about the properties of the data. Hence they are more flexible, but they may need more data to sufficiently constrain them.

Occam's razor
A philosophical heuristic that advises us to prefer simple explanations to complicated ones.

Online analytical processing (OLAP)
Data analysis software used to extract information from a *data warehouse*. OLAP is user-driven, in the sense that the user thinks of some hypotheses about the process and using OLAP tools checks whether the data supports those hypotheses. Machine learning is more data-driven in the sense that automatic data analysis may find dependencies not previously thought by users.

Outlier detection
An outlier, anomaly, or novelty is an instance that is very different from other instances in the sample. In certain applications such as fraud detection, we are interested in the outliers that are the exceptions to the general rules.

Parallel distributed processing
A computational paradigm where the task is divided into smaller concurrent tasks, each of which can be run on a different processor. By using more processors, the overall computation time is reduced.

Parametric methods
Statistical methods that make strong assumptions about data. The advantage is that if the assumption holds, they are very efficient in terms of computation and data; the risk is that those assumptions do not always hold.

Pattern recognition
A pattern is a particular configuration of data; for example, 'A' is a composition of three strokes. Pattern recognition is the detection of such patterns.

Perceptron
A perceptron is a type of a *neural network* organized into layers where each layer receives connections from units in the previous layer and feeds its output to the units of the layer that follow.

Prior distribution
The distribution of possible values that an unknown parameter can take *before* looking at the data. For example, before estimating the average weight of high school students, we may have a prior belief that it will be between 100 and 200 pounds. Such information is especially useful if we have little data.

Posterior distribution
The distribution of possible values that an unknown parameter can take *after* looking at the data. Bayes' rule allows us to combine the *prior distribution* and the data to calculate the posterior distribution.

Q learning
A reinforcement learning method based on *temporal difference learning*, where the goodness values of actions in states are stored in a table (or function), frequently denoted by *Q*.

Ranking
This is a task somewhat similar to regression, but we care only whether the outputs are in the correct order. For example, for two movies *A* and *B*, if the user enjoyed *A* more than *B*, we want the score estimate to be higher for *A* than for *B*. There are no absolute score values as we have in regression, but only a constraint on their relative values.

Regression
Estimating a numeric value for a given instance. For example, estimating the price of a used car given the attributes of the car is a regression problem.

Reinforcement learning
It is also known as learning with a critic. The agent takes a sequence of actions and receives a reward/penalty only at the very end, with no feedback during the intermediate actions. Using this limited information, the agent should learn to generate the actions to maximize the reward in later trials. For example, in chess, we do a set of moves, and at the very end, we win or lose the game; so we need to figure out what the actions that led us to this result were and correspondingly credit them.

Sample
A set of observed data. In statistics, we make a difference between a population and a sample. Let us say we want to do a study on obesity in high school students. The population is all the high school students, and we cannot possibly observe the weights of all. Instead, we choose a random subset of, for example, 1,000 students and observe their weights. Those 1,000 values are our sample. We analyze the sample to make inferences about the population. Any value we calculate from the sample is a statistic. For example, the average of the weights of the 1,000 students in the sample is a statistic and is an estimator for the mean of the population.

Smart device
A device that has its sensed data represented digitally and is doing some computation on this data. The device may be mobile and it may be online; that is, it has the ability to exchange data with other smart devices, computers, or the cloud.

Speech recognition
Recognizing uttered sentences from acoustic information captured by a microphone.

Supervised learning
A type of machine learning where the model learns to generate the correct output for any input. The model is trained with data prepared by a supervisor who can provide the desired output for a given input. *Classification* and *regression* are examples of supervised learning.

Temporal difference learning
A set of methods for *reinforcement learning* where learning is done by backing up the goodness of the current action to the one that immediately precedes it. An example is the *Q learning* algorithm.

Validation
Testing the *generalization* performance of a trained model by testing it on data unseen during training. Typically in machine learning, we leave some of our data out as validation data, and after training we test it on this left out data. This validation accuracy is an estimator for how well the model is expected to perform if used later on in real life.

Web scraping
Software that automatically surfs the web and extracts information from web pages.

REFERENCES

Ballard, D. H. 1997. *An Introduction to Natural Computation*. Cambridge, MA: MIT Press.

Bakshy, E., S. Messing, and L. A. Adamic. 2015. “Exposure to Ideologically Diverse News and Opinion on Facebook.” *Science* 348:1130–1132.

Bohannon, J. 2015. “Fears of an AI Pioneer.” *Science* 349:252.

Börner, K. 2015. *Atlas of Knowledge: Anyone Can Map*. Cambridge, MA: MIT Press.

Blei, D. 2012. “Probabilistic Topic Models.” *Communications of the ACM* 55:77–84.

Buchanan, B. G., and E. H. Shortliffe. 1984. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, MA: Addison Wesley.

Feldman, J. A., and D. H. Ballard. 1982. “Connectionist Models and Their Properties.” *Cognitive Science* 6:205–254.

Fukushima, K. 1980. “Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.” *Biological Cybernetics* 36:93–202.

Han, J., and M. Kamber. 2011. *Data Mining: Concepts and Techniques*. 3rd ed. San Francisco, CA: Morgan Kaufmann.

Hebb, D. O. 1949. *The Organization of Behavior*. New York: Wiley & Sons.

Hirschberg, J., and C. D. Manning. 2015. “Advances in Natural Language Processing.” *Science* 349:261–266.

Horvitz, E., and D. Mulligan. 2015. “Data, Privacy, and the Greater Good.” *Science* 349:253–255.

Hubel, D. H. 1995 *Eye, Brain, and Vision*. 2nd ed. New York: W. H. Freeman. http://hubel.med.harvard.edu/index.html.

Jordan, M. I., and T. M. Mitchell. 2015. “Machine Learning: Trends, Perspectives, and Prospects.” *Science* 349:255–260.

Koller, D., and N. Friedman. 2009. *Probabilistic Graphical Models*. Cambridge, MA: MIT Press.

LeCun, Y., Y. Bengio, and G. Hinton. 2015. “Deep Learning.” *Nature* 521:436–444.

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. “Backpropagation Applied to Handwritten Zip Code Recognition.” *Neural Computation* 1:541–551.

Liu, T.-Y. 2011. *Learning to Rank for Information Retrieval*. Heidelberg: Springer.

Marr, D. 1982. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Cambridge, MA: MIT Press.

McCulloch, W., and W. Pitts. 1943. “A Logical Calculus of the Ideas Immanent in Nervous Activity.” *Bulletin of Mathematical Biophysics* 5:115–133.

Minsky, M. L., and S. A. Papert. 1969. *Perceptrons*. Cambridge, MA: MIT Press.

Mitchell, T. 1997. *Machine Learning*. New York: McGraw Hill.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. G. Bellemare, A. Graves, et al. 2015. “Human-Level Control through Deep Reinforcement Learning.” *Nature* 518:529–533.

Nilsson, N. J. 2009. *The Quest for Artificial Intelligence: History of Ideas and Achievements*. Cambridge, UK: Cambridge University Press.

Orbanz, P., and Y. W. Teh. 2010. “Bayesian Nonparametric Models.” In *Encyclopedia of Machine Learning*. New York: Springer.

Rosenblatt, F. 1962. *Principles of Neurodynamics*. Washington, DC: Spartan Books.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1986. “Learning Representations by Back-Propagating Errors.” *Nature* 323:533–536.

Rumelhart, D. E., and J. L. McClelland and the PDP Research Group. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press.

Russell, S., and P. Norvig. 2009. *Artificial Intelligence: A Modern Approach*. 3rd ed. Upper Saddle River, NJ: Prentice Hall.

Sandel, M. 2012. *What Money Can't Buy: The Moral Limits of Markets*. New York: Farrar, Straus and Giroux.

Schmidhuber, J. 2015. "Deep Learning in Neural Networks: An Overview." *Neural Networks* 61:85–117.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, et al. 2016. "Mastering the Game of Go with Deep Neural Networks and Tree Search." *Nature* 529: 484–489.

Sutton, R. S., and A. G. Barto. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Sweeney, L. 2002. "K-Anonymity: A Model for Protecting Privacy." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10:557–570.

Tesauro, G. 1995. "Temporal Difference Learning and TD-Gammon." *Communications of the ACM* 38:58–68.

Thrun, S., W. Burgard, and D. Fox. 2005. *Probabilistic Robotics*. Cambridge, MA: MIT Press.

Valiant, L. 1984. "A Theory of the Learnable." *Communications of the ACM* 27:1134–1142.

Vapnik, V. 1998. *Statistical Learning Theory*. New York: Wiley.

Winston, P. H. 1975. "Learning Structural Descriptions from Examples." In *The Psychology of Computer Vision*, ed. P. H. Winston, 157–209. New York: McGraw-Hill.

Wirth, N. 1976. *Algorithms + Data Structures = Programs*. Upper Saddle River, NJ: Prentice Hall.

Zadeh, L. A. 1965. "Fuzzy Sets." *Information and Control* 8:338–353.

FURTHER READINGS

Duda, R. O., P. E. Hart, and D. G. Stork. 2001. *Pattern Classification*. 2nd ed. New York: Wiley.

Feldman, J. A. 2006. *From Molecule to Metaphor: A Neural Theory of Language*. Cambridge, MA: MIT Press.

Hastie, T., R. Tibshirani, and J. Friedman. 2011. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer.

Kohonen, T. 1995. *Self-Organizing Maps*. Berlin: Springer.

Manning, C. D., and H. Schütze. 1999. *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

Murphy, K. 2012. *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: MIT Press.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge, UK: Cambridge University Press.

Witten, I. H., and E. Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd ed. San Francisco, CA: Morgan Kaufmann.

INDEX

Accuracy, 9, 14, 25–26, 99, 104, 155, 161
Acoustics, 7, 67–68, 71, 153
Actions
  agents and, 128, 131, 135
  algorithms and, 134, 136–138
  class and, 129–130
  classification and, 129
  credit assignment problem and, 127–128
  environment and, 126–129, 131–132, 135, 148, 153
  estimation and, 129–135
  expected reward and, 131
  expected value and, 133
  exploitation and, 132
  exploration and, 129, 132
  games and, 125–127, 131–132, 136, 139
  hidden factors and, 131
  information and, 132, 138
  internal value and, 128
  K-armed bandit and, 128–133
  knowledge base and, 50–51
  prediction and, 131–132, 135
  probability and, 135
  reinforcement learning and, 125–132, 136–139, 161
  representation and, 138
  rewards and, 127–135
  robots and, 126–127, 131–135
  sequence of, 127–128, 134
  supervised learning and, 127, 129–130, 136–137
  temporal difference learning and, 133–135
Activation value, 86–87
Active learning, 79–80
Activity recognition, 14, 71
Adamic, L. A., 164
Adaptation, xii, 17–18, 28, 66, 91, 148, 153
Added value, 169n2
Affective computing, 66
Agents, 128, 131, 135
Algorithms, xiv
  actions and, 134, 136–138
  association rules and, 118–120
  backpropagation, 92, 99
  clusters and, 116–122
  computer programs and, 169n3
  credit scoring, 45–50, 75, 156
  deep learning and, 104, 108
  discriminatory features and, 24
  expert systems and, 23
  future issues and, 142, 147–148, 151, 155, 158, 166
  human brain and, 19–20
  inductive bias and, 42
  learning, 15, 24–27, 38–40, 42, 45, 64, 71, 73–74, 79–80, 88–91, 104, 108, 116, 121, 136–137, 147–148, 155, 158, 166
  modifiable parameters and, 24
  nearest-neighbor, 60, 77

Algorithms (cont.)
  neural networks and, 88–92, 99
  number sorting, 17
  parameter adjustment and, 24–25
  pattern recognition and, 60, 64, 71, 73–74, 77, 79–80
  privacy-preserving, 158
  programs and, 169n3
  representation and, 20–22
  as sequence of instructions, 16
  smart, 15
  symbol manipulation and, 21
  templates for, 24–25, 36, 56, 114, 117
Alignment, 114
Alphabets, 57, 69, 169
AlphaGo program, 138
Anna Karenina (Tolstoy), 72
Anomalies, 72
Anonymization, 158
Artificial intelligence (AI)
  adaptation and, xii, 17–18, 28, 66, 91, 148, 153
  computer science and, 19
  deep learning and, 17–20, 23, 27, 51, 80, 161–162, 165
  domination of humans by, 165–166
  Fifth Generation Computers Systems Project and, 51
  future issues and, 161–162, 165
  human brain and, 19
  learning algorithms and, 27–28
  necessity of machine learning for, 17–20
  parallel processing and, 51
  pattern recognition and, 23, 80
  performance criterion and, 162–163
  Science article on, xiii
  self-driving cars and, 149–150, 164–165
  statistics and, 28
ASCII code, 169n1
Association rules, 118–120
Autoencoder neural networks, 76, 102–103, 107–108
Autonomous agents, xiv, 149, 160, 163–167, 170n3

Backpropagation, 92, 99
Bag of words, 69–70, 116, 122
Bakshy, E., 164
Ballard, D. H., 95, 136
Bandwidth, 144
Barcodes, 55–56, 167
Barto, A. G., 127
Basket analysis, 154–155
Bayesian estimation, 63–64, 83–84, 170n4
Bayesian network, 63–64
Bayes' rule, 63–64, 170nn2,4
Bengio, Y., 106
Big data, 15, 104, 108, 150–151, 159–161, 166
Binary representation, 2, 21
Bioinformatics, 114
Biology, 15, 19, 96, 114, 117
Biometrics, 67
  face recognition and, 23–24, 65–66, 76, 167
  fingerprints and, 66, 71
  iris and, 66
  palm and, 66
  photos and, 66
  signatures and, 62, 66
  spoofing, 66
Black box models, 155–156

Blei, D., 122
Bohannon, J., 165
Buchanan, B. G., 51
Burgard, W., 135

Captchas, 58
CDs, 144
Character recognition, 56, 61, 65, 68, 115, 170n1
Class
  actions and, 129–130
  clusters and, 115
  data analytics and, 46–50, 53
  discriminants and, 27, 47–50, 75, 78
  expected values and, 52–54
  invariance and, 61–62, 65
  multiple groups and, 115
  negative, 53
  pattern recognition and, 56–57, 62, 70, 80, 82, 84
  positive, 53
  shape representation and, 47f
  supervised learning and, 46
Classification, 155
  actions and, 129
  clusters and, 116–117
  credit scoring, 45–50
  deep learning and, 104
  discriminant analysis and, 27, 75
  document categorization and, 68
  K-armed bandit and, 128–132
  kernel function and, 116
  pattern recognition and, 68, 76, 81–82
  regression and, 76
Classifiers, 48, 75, 82, 115
Cloud computing, 143–144, 152–153, 159
Clusters
  algorithms and, 116–122
  alignment and, 114
  association rules and, 118–120
  character recognition and, 115
  class and, 115
  classification and, 116–117
  clustering and, 112–117, 155
  complexity and, 121
  compression and, 112–113
  computers and, 114
  customer relationship management (CRM) and, 112
  customer segmentation and, 35, 112–113
  data structures and, 117
  document categorization and, 122
  document clustering and, 114
  estimation and, 119–120, 123
  exploratory data analysis and, 115
  finding groups in data and, 111–118
  generative models and, 119, 123
  hidden factors and, 119–123
  hierarchical clustering and, 117
  information and, 112–113, 116–117, 119
  input attributes and, 116
  kernel function and, 116
  latent semantic indexing and, 122–123
  matrix decomposition and, 120–121
  mixture models and, 112
  motifs and, 114–115
  phylogeny and, 117
  prediction and, 115, 118–119, 122

Clusters (cont.)
  recommendation systems and, 118–123
  regression and, 116
  representation and, 116, 120
  samples and, 115
  sparse data and, 121–122
  spectral clustering and, 116
  speech recognition and, 115–118
  statistics and, 116
  support vector machine and, 116
  unsupervised learning and, 111–112, 114, 117
  Web pages and, 116
Cobol, 142
Cognitive science, 19, 28, 95–96, 99
Coin tossing, 32–36, 53–54, 82–83
Complexity, 43–44, 161
  clusters and, 121
  neural networks and, 100, 102
  pattern recognition and, 59, 73–77, 79, 84
Compression, 45, 112–113
Computational learning theory, 80
Computational theory, 20–22, 85
Computers
  advancements in hardware and, 141–143
  algorithms and, 16 (*see also* Algorithims)
  binary representation and, 2, 21
  bit sequences and, 2–3
  building better, 19
  centers of, 2, 8, 143
  cloud computing and, 143–144, 152–153, 159
  clusters and, 114
  data analytics and, 27
  data exchange and, 5–6
  dataquake and, x, 10–16
  deep learning and, 104
  diagnostics and, 51–54, 62–64, 72, 74
  expense of early, 1
  expert systems and, 23, 50–52
  Fifth Generation Computer Systems Project and, 51
  future issues and, 141–143, 150–151, 153, 159–163, 166–169
  graphical user interfaces (GUIs) and, 4
  grid computing and, 144
  hardware and, 18–22, 28, 97–98, 141–142, 151–152, 161, 167
  high-performance computation and, 150–153, 159–163
  human brain and, 19–20
  Internet and, 6–7, 9, 15
  lowering costs of, 2
  miniaturization technology and, 3–4
  mobile computing and, 7–9
  neural networks and, 28, 96, 99
  online information and, 6–11, 66, 90–91, 118, 145, 147–148, 153
  pattern recognition and, 58, 66, 77
  personal, 4–5, 8, 10, 142–143
  processors and, 3–4, 19–20, 28, 96–98, 108, 141–143, 151–152
  programming and, 4, 16–17, 24–25, 28, 50, 98, 142, 169n3
  reapproachment with people and, 4–5
  rule extraction and, 14
  smartphones, 7–8, 14, 71, 139, 145, 153, 169
  social data and, 10

  storage, 3–5, 90, 143–144, 150–152, 169n2 (*see also* Databases)
  ubiquitous computing and, 9
  understanding, 20–22
  World Wide Web and, 6
Computer science, 19, 23, 27–28, 77, 99, 114, 151
Connectionism, 95–96
Connectivity, 5–7, 20, 106, 144, 169
Convolutional neural networks, 101, 104, 106–107, 136, 138
Credit assignment problem, 127–128
Credit scoring, 45–50, 75, 156
Customer relationship management (CRM), 112. *See also* Clusters
Customer segmentation, 35, 112–113
Cyber-physical systems, 153

Data analytics
  adding smartness and, 147–148
  anonymization and, 158
  class and, 46–50, 53
  clusters and, 111–123
  compression and, 45, 112–113
  computers and, 27
  credit scoring and, 45–50, 75, 156
  dimensionality reduction and, 73–76
  discriminants and, 27, 47–50, 75, 78
  estimation and, 27, 29–40, 46, 58–60, 63, 77–84, 89–90, 119–120, 123, 129–135, 155, 160–161, 170n4
  experiments and, 26–27, 100
  exploratory data analysis and, 115
  future issues and, 154–156, 159–160
  Gaussian curve and, 160
  general models and, 35–36
  hidden causes and, 60–63, 120
  hidden units and, 100–104, 106
  inductive bias and, 42
  input representation and, 30, 116
  latent causes and, 60–63
  latent semantic indexing and, 122
  learning a sequence and, 43–45
  levels of analysis and, 20–21, 85
  models and, 35–36 (*see also* Models)
  OLAP, 155
  outliers and, 72–74, 112, 163
  pattern recognition and, 60
  prediction and, 13–17 (*see also* Prediction)
  price of used car and, 29–32, 34, 40, 59, 81, 89
  privacy and, 66, 156–159, 161
  randomness and, 32–35 (*see also* Randomness)
  regression and, 38 (*see also* Regression)
  sanitization and, 158
  supervised learning and, 38–42, 46, 89, 111–117, 127, 129–130, 136–137
  training sets and, 31, 39–40, 89–90, 155
  validation sets and, 155
  visualization and, 155–156
Databases, 17
  advancements in hardware and, 142–143
  central, 11
  emergence of, 3

Databases (cont.)
  exabyte, 143
  Go game and, 138
  knowledge discovery in, 28
  languages and, 147
  natural language processing and, 108
  ontologies and, 108
  petabyte, 143
  predefined, 108
  preparation for mining of, 154
  storage and, 3–5, 11, 138, 143
  terabyte, 143
Data centers, 108, 143, 152–153, 159
Data mining, 14, 154–156, 159
Dataquake, x, 10–16
Data science, xv, 15, 159–161
Data warehouses, 154–155
Decision making, 169
  actions and, 126–127
  agents and, 128, 131, 135
  diagnostics and, 51–54, 62–64, 72, 74
  expected values and, 52–54
  expert systems and, 23, 50–52
  false negatives and, 53–54
  false positives and, 53–54
  human brain and, 162
  knowledge base and, 50–51
  logic and, 19, 50–51, 91–92
  pattern recognition and, 73
  self-driving cars and, 149–150, 164–165
Decision trees, 77–79, 155
Deep learning
  abstract levels and, 106
  algorithms and, 104, 108
  artificial intelligence (AI) and, 17–20, 23, 27, 51, 80, 161–162, 165
  classification and, 104
  computer vision and, 104
  error and, 106
  face recognition and, 106
  future issues and, 166–167
  hidden layers and, 104, 106–109
  information and, 106
  less manual interference and, 107–108
  machine translation and, 70, 109
  neural networks and, 104–109
  representation and, 104, 108–109
  rule extraction and, 14
  translation and, 109
Deep Q Network, 136, 138
Deoxyribonucleic acid (DNA), 114
Deterministic systems, 32, 48, 129
Diagnostics, 51–54, 62–64, 72, 74
Digital communication, ix-x, 6
Digital subscriptions, 144
Dimensionality reduction, 73–76, 102
Discriminants, 27, 47–50, 75, 78
Document categorization, 68, 122
Document clustering, 114
Drones, 150
DVDs, 144

E-books, 144–145
Email, 6, 16, 68
Embedded systems, 153
Emergent properties, 96
Engineering
  artificial intelligence and, 19
  constraints and, 55–56
  efficiency and, 55–56

  future issues and, 142, 151, 159–160
  neural networks and, 86, 99
  pattern recognition and, 27–28, 55
  probability theory and, 32
  reverse, 22
  trade-offs in, 55–56
Environment, 126–129, 131–132, 135, 148, 153
Error, 44, 53, 56, 72–73, 80, 90, 106, 117, 154
Estimation
  actions and, 129–135
  Bayesian, 63–64, 83–84, 170n4
  coin tossing and, 32–36, 53–54, 82–83
  data analytics and, 27, 29–40, 46, 58–60, 63, 77–84, 89–90, 119–120, 123, 129–135, 155, 160–161, 170n4
  expected value and, 52–54
  future issues and, 160–161
  Gaussian curve and, 160
  generalization and, 39–42, 57, 121
  granularity and, 58–60
  inference and, 27, 50–51, 62–64, 83, 148
  internal reward, 132
  intervals and, 32, 44, 80
  learning a sequence and, 43–45
  learning clusters and, 119–120, 123
  MAP, 83
  MCMC, 83–84
  medical diagnostics and, 52–54
  neural networks and, 89–90
  parametric, 58–59, 77
  pattern recognition and, 58–60, 63, 77–84
  prediction and, 13–17 (*see also* Prediction)
  price of used car and, 29–32, 34, 40, 59, 81, 89
  prior belief and, 82–84
Europe, 51
Exabyte, 143
Expected values, 52–54, 133, 169
Experiments, 26–27, 100
Expert systems, 23, 50–52
Exploitation, 73, 112, 132
Exploration, 115, 129, 132

Face recognition, 23–25, 65–66, 76, 106, 167
False negatives, 53–54
False positives, 53–54
Feature extraction, 74–76, 102, 104, 108, 118
Feature selection, 74–76
Feedback, 36, 125–128, 132, 180
Feldman, J. A., 95
Fibonacci sequence, 43–44
Fifth Generation Computers Systems Project, 51
Fingerprints, 66, 71
Flying machines, 85–86
Fonts, 56–57, 61
Forgeries, 66
Fortran, 142
Fox, D., 135
Fraud, 72
Friedman, N., 64
Fukushima, K., 101
Fuzzy logic, 51

Games, 4, 34, 125–127, 131–132, 136, 139, 152, 162
Gaussian curve, 160
Generalization, 39–42, 57, 121
Generative models, 60–64, 119, 123
Global models, 58
Go, 138, 139
GPS, 149
Grandmother cell theory, 96
Granularity, 58–60, 75, 117
Graphical models, 63–64
Graphical processing unit (GPU), 152
Graphical user interfaces (GUIs), 4
Grid computing, 144

Hackers, 72
Han, J., 154
Hardware, 18–22, 28, 97–98, 141–142, 151–152, 161, 167
Hebb, D. O., 89
Hebbian learning, 89, 96
Hidden causes, 60–63, 120
Hidden factors
  actions and, 131
  clusters and, 119–123
  pattern recognition and, 60–63, 65
Hidden units, 100–104, 106
Hierarchical clustering, 117
Hierarchical cone, 102, 104–105
High-performance computing, 150–153, 159–163
Hinton, G. E., 92, 106
Hirschberg, J., 68
Horvitz, E., 157
Hubel, D. H., 100
Human brain
  artificial intelligence (AI) and, 19
  cognitive science and, 19, 28, 95–96, 99
  computational theory and, 20–22, 85
  computer science and, 19–20
  decision making and, 162
  grandmother cell theory and, 96
  hardware implementation and, 20, 22
  learning and, 17–22
  levels of analysis and, 20–21, 85
  neural networks and, 85–86, 96, 109
  neurons and, 19–22
  parallel processing and, 19–20
  representation and, 20–22
  sensory systems and, 42, 145–146
  synapses and, 20, 86–87, 92
  understanding workings of, 85–86
Hypercomplex cells, 100
Hyperparameters, 44, 49, 138

If-then rules, 49–50, 77–78, 155
Ill-posed problem, 42
Image recognition, 14, 99
Inductive bias, 42
Inference, 27, 50–51, 62–64, 83, 119, 148
Information
  accuracy and, 9, 14, 25–26, 99, 104, 155, 161
  actions and, 132, 138
  binary representation and, 2, 21 (*see also* Representation)
  contextual, 93
  customer, 45–50
  deep learning and, 106

  - diagnostics and, 51–54, 62–64, 72, 74
  - email and, 6, 16, 68
  - errors and, 44, 53, 56, 72, 80, 90, 106, 117, 154
  - future issues and, 142, 145, 147–149, 158–160, 169
  - learning clusters and, 112–113, 116–117, 119
  - levels of analysis and, 20–21, 85
  - medical, 52–54
  - neural networks and, 93, 103
  - online, 6–11, 66, 90–91, 118, 145, 147–148, 153
  - pattern recognition and, 55, 59, 64, 67–69, 71, 73–76, 80
  - processing of, 3, 19–20, 145, 169
  - storage of, 5 (*see also* Databases)
- Infotainment, 148
- Input representation, 30, 116
- Internal reward, 132–133
- Internal value, 128
- Internet, 6–7, 9, 15
- Intervals, 32, 44, 80
- Invariance, 61–62, 65

- Japan, 51
- Jordan, M. I., 141

- Kamber, M., 154
- Kanizsa triangle, 42
- K-armed bandit, 128–132
- Kernel function, 116
- Keystroke, 66
- Knowledge base, 50–51
- Knowledge discovery in databases, 28
- Knowledge extraction, 26, 50, 74, 156
- Koller, D., 64

- Language models, 68
- Latent causes, 60–63, 120
- Latent semantic analysis, 122
- Learning programs, 23–25, 57, 65, 70, 128
- LeCun, Y. B., 101, 106
- Levels of analysis, 20–21, 85
- LIDAR, 149
- Linear discriminant analysis, 75
- Linear models, 37–38, 46, 49, 79, 87, 91
- Linguistics, 28, 68, 166
- Lisp, 51
- Liu, T.-Y., 81
- Local representation, 96
- Logic, 19, 50–51, 91–92
- LPs, 144

- Machine learning, xiv
  - adaptation and, xii, 17–18, 28, 66, 91, 148, 153
  - advancements in hardware and, 141–143
  - artificial intelligence (AI) and, 17–20, 23, 27, 51, 80, 161–162, 165
  - bit sequences and, 2–3
  - compression and, 45, 112–113
  - computational theory and, 20–22, 80, 85
  - computers and, 1–2 (*see also* Computers)
  - credit scoring and, 45–50, 75, 156
  - data mining and, 14, 154–156, 159

Machine learning (cont.)
  data science and, 159–161
  environment and, 126–132, 135, 148, 153
  expected reward and, 131
  expected values and, 52–54
  expert systems and, 50–52
  face recognition and, 23, 25, 65, 76, 106, 167
  future issues and, 166–168
  games and, 125–127, 131–132, 136, 139
  generalization and, 39–42, 57, 121
  hardware advancements and, 141–142
  if-then rules and, 49–50, 77–78, 155
  intelligence and, xi-xii
  K-armed bandit and, 128–133
  logic and, 19, 50–51, 91–92
  memorization and, 4, 39
  modifiable parameters and, 24
  parameter adjustment and, 24–25
  power of digital technology and, 1–9
  prediction and, 13–17, 25–26 (*see also* Prediction)
  price of used car and, 29–32, 34, 40, 59, 81, 89
  probably approximately correct learning and, 80
  programs for, 16–17, 23–25, 57, 65, 70, 128
  recommendation systems and, 81, 145, 160, 163–164
  reinforcement learning and, 125–132, 136–139, 161
  rewards and, 127–135
  self-driving cars and, 149–150
  templates and, 24–25, 36, 56, 114, 117
  temporal difference learning and, 133–135
Machine reading, 147
Manning, C. D., 68
Markov chain Monte Carlo (MCMC) sampling, 83–84
Marr, D., 20
Matrix decomposition, 120–121
Maximum a posteriori (MAP) estimator, 83
McClelland, J. L., 95
McCulloch, W., 86
Memorization, 4, 39
Memory
  advancements in hardware and, 141–143
  compression and, 45, 112–113
  human brain and, 18, 20
  neural networks and, 86, 92–93, 98
  learning vs. programming and, 16
  pattern recognition and, 74, 77
  prediction and, 26
  storage and, 3–5, 90, 143–144, 150–152, 169n2
Messing, S., 164
Microprocessors, 3, 141–142
Minsky, M. L., 91
Mitchell, T., 42, 141
Mixture models, 112
Mnih, V., 136
Mobile computing, 7–9
Models
  attribute weight and, 37
  black box, 155
  clustering, 111–112, 119–120, 123

combining multiple, 71–73
complexity of, 43–44, 59, 73–77, 79, 84, 102, 121, 161
connectionist, 95–96
constructing simple, 14
data fitting and, 26–27
dimensionality reduction and, 73–76
experiments and, 26–27, 100
future issues and, 151–152, 155–156, 158, 161, 163, 166–167
generative, 60–64, 119, 123
global, 58
hidden causes and, 60–63
hyper parameters and, 44, 49, 138
if-then rules and, 49–50, 77–78, 155
inductive bias and, 42
input representation and, 30, 116
invariance and, 61–62, 65
language, 68
learning a general, 35–36
linear, 37–38, 46, 49, 79, 87, 91
matching granularity and, 58–60
mixture, 112
multiview, 71
neural networks and, 19, 28, 86–87, 90–91, 95–96
nonparametric, 77, 79, 84, 161
optimizing a performance criterion and, 25
outliers and, 72–74, 112, 163
parameters of, 24–25, 37–39, 42, 44, 49, 58–59, 63, 82–84, 98, 101, 107, 111, 138
pattern recognition and, 58–64, 68, 71–74, 79–82, 84
perceptron, 86–87, 91–95, 99
performance of, 36, 38
prediction and, 13–17, 25–26 (*see also* Prediction)
probability theory and, 32, 51–52, 170n4
ranking and, 81–82
recommendation, 81, 118–123
samples and, 33
selection of, 36–38
statistics and, 33 (*see also* Statistics)
supervised learning and, 38–42, 46, 89, 111–117, 127, 129–130, 136–137
templates and, 24–25, 36, 56, 114, 117
useful approximation and, 13–14
Motifs, 114–115
Mulligan, D., 157
Multidimensional data cube, 154–155
Multiple instruction, multiple data (MIMD) machines, 97
Multiview models, 71
Music, 4, 6, 8, 144
Mutations, 117
MYCIN, 51

Natural language processing, 64, 68–70, 108–109
Nearest-neighbor methods, 60, 77
Near misses, 80
Neocognitrons, 101
Neural instruction, multiple data (NIMD) machines, 98
Neural networks, xiv
activation value and, 86–87
algorithms and, 88–92, 99
artificial, 85–88

Neural networks (cont.)
  autoencoder, 76, 102–103, 107–108
  backpropagation, 92, 99
  black box models and, 155
  complex cells and, 100
  computers and, 28, 96, 99
  connectionism and, 95–96
  convolutional, 101, 104, 106–107, 136, 138
  deep learning and, 104–109
  Deep Q Network and, 136–137
  dependencies and, 100–101
  dimensionality reduction and, 102
  emergent properties and, 96
  error and, 90, 106
  estimation and, 89–90
  feature extractors and, 102, 104, 108, 118
  feedforward, 92
  grandmother cell theory and, 96
  Hebbian learning and, 89, 96
  hidden units and, 100–104, 106
  hierarchical representations and, 99–103
  hierchical cone and, 102, 104–105
  human brain and, 85–86, 96, 109
  image recognition and, 14, 99
  increased research in, 28
  information and, 93, 103
  interconnected processing units and, 19
  layered architecture of, 87, 92, 99–108, 152
  logic gates and, 91–92
  MIMD machines and, 97
  models and, 19, 28, 86–87, 90–91, 95–96
  neocognitrons and, 101
  NIMD machines and, 98
  parallel processing and, 19–20, 28, 51, 86–87, 95–98, 143, 151–153
  prediction and, 108
  recurrent, 92–94
  regression and, 90
  representation and, 96, 99, 102
  samples and, 89, 106
  short-term memory and, 92–93
  SIMD machines and, 97–98
  simple cells and, 100
  simulation studies and, 19
  software and, 96–97
  state of, 94
  synapses and, 20, 86–87, 92
  training, 89–92, 95, 99, 106–109
NeuroGammon program, 136
Neurons
  activation value and, 86–87
  cognitive science and, 19, 28, 95–96, 99
  context and, 92–93
  firing of, 87, 92
  Hebbian learning and, 89, 96
  hidden units and, 100–104, 106
  human brain and, 19–22
  input units and, 89, 93, 101
  layers of, 87, 92, 99–108
  output, 86–87, 95
  perceptrons and, 86–87, 91–95, 99
  synapses and, 20, 86–87, 92
Neuroscience, 19, 28
Nobel Prize, 100
Nonparametric methods, 77, 79, 84, 161
Norvig, P., 165
Novelty detection, 73

Occam's razor, 43
Online analytical processing (OLAP), 155
Online information, 6–11, 66, 90–91, 118, 145, 147–148, 153
Ontologies, 108
Operating systems, 142
Optical character recognition (OCR), 56, 61, 65, 68, 115, 167, 170n1
Optimizing a performance criterion, 25
Orbanz, P., 84
OR gate, 91–92
Outliers, 72–74, 112, 163

Papert, S. A., 91
Parallel distributed processing (PDP), 95
Parallel processing
  artificial intelligence (AI) and, 51
  connectionism and, 95–96
  graphical processing unit (GPU) and, 152
  neural networks and, 86–87, 95–98, 143, 151–153
Parametric methods
  estimation and, 58–59, 77
  Gaussian curve and, 160
  pattern recognition and, 58–59, 84
  univariate attributes and, 161
Passwords, 66
Pattern recognition
  active learning and, 79–80
  activity recognition and, 14, 71
  affective computing and, 66
  algorithms and, 60, 64, 71, 73–74, 77, 79–80
  alphabets and, 57, 69, 169
  artificial intelligence (AI) and, 23, 80
  auto-encoder networks and, 76
  barcodes and, 55–56, 167
  Bayesian estimation and, 63–64, 83–84, 170n4
  Bayes' rule and, 63–64, 170nn2,4
  biometrics and, 62, 65–67, 71, 76, 167
  captchas and, 58
  character recognition and, 56, 61, 65, 68, 115, 170n1
  class and, 56–57, 62, 70, 80, 82, 84
  classification and, 68, 76, 81–82
  combining multiple models and, 71–73
  computers and, 58, 66, 77
  data analytics and, 60
  decision trees and, 77–79
  dimensionality reduction and, 73–76
  estimation and, 58–60, 63, 77–84
  expert systems and, 23
  face recognition and, 23–25, 65–66, 76, 106, 167
  fonts and, 56–57, 61
  fraud and, 72
  generative models and, 60–64
  hidden factors and, 60–63, 65
  image recognition and, 14, 99
  information and, 55, 59, 64, 67–69, 71, 73–76, 80
  invariance and, 61–62, 65
  learning to read and, 55–58
  matching model granularity and, 58–60, 75
  models and, 58–64, 68, 71–74, 79–82, 84

Pattern recognition (cont.)
  natural language processing and, 64, 68–70
  novelty detection and, 73
  optical character recognition (OCR) and, 56, 61, 65, 68, 115, 167, 170n1
  outlier detection and, 72–74
  parametric methods and, 58–59, 84
  phonemes and, 67, 101
  prediction and, 61, 71, 77, 79
  prior belief and, 82–84
  probability and, 63–64, 80, 82
  randomness and, 24, 44, 57, 71, 79
  ranking and, 81–82
  regression and, 58, 76, 81–82
  representation and, 55, 63, 69–70
  samples and, 57, 59, 70, 72, 83
  signatures and, 62, 66
  speech recognition and, 67–68, 71
  statistics and, 58, 63, 73
  supervised learning and, 111–117
  template matching and, 56
  translation and, 70
PDP Research Group, 95
Perceptrons, 86–87, 91–95, 99
Performance
  criterion for, 25, 38–39, 162–163
  high-performance computation and, 150–153, 159–163
  models and, 36, 38
  optimizing, 25, 71, 75
  pattern recognition and, 71, 75
Personal computers, 4–5, 8, 10, 142–143
Petabyte, 143
Phonemes, 67, 101
Photographs, 2–4, 23, 66
Phylogeny, 117
Physics, 15, 28, 166
Pitts, W., 86
Politics, 68, 70, 114, 147
Posterior distribution, 83
Prediction
  accuracy and, 14, 25–26
  actions and, 131–132, 135
  clusters and, 115, 118–119, 122
  credit scoring and, 45–50
  diagnostics and, 51–54, 62–64, 72, 74
  expected value and, 52–54
  false negatives and, 53–54
  false positives and, 53–54
  future issues and, 141, 155, 161, 163, 169
  information analysis and, 13–14, 31–35, 38–45, 49
  learning a sequence and, 43–45
  learning vs. programming and, 16–17
  memory and, 26
  neural networks and, 108
  pattern recognition and, 61, 71, 77, 79
  statistics and, 33 (*see also* Statistics)
Primitives, 67, 101, 103, 112
Prior belief, 82–84
Privacy, 66, 156–159, 161
Probability
  actions and, 135
  coin tossing and, 32–36, 53–54, 82–83
  deterministic systems and, 32, 48, 129
  false negatives and, 53–54

  false positives and, 53–54
  inference and, 27, 50–51, 62–64, 83, 148
  intervals and, 32, 44, 80
  pattern recognition and, 63–64, 80, 82
  prior belief and, 82–84
  randomness and, xi, 13, 24 (*see also* Randomness)
  risk and, 45–50, 52, 117, 163–164
  uncertainty and, 32, 34, 51–52, 74, 80, 83, 131, 135
Probably approximately correct learning, 80
Processors, 3–4, 19–20, 28, 96–98, 108, 141–143, 151–152
Programming
  algorithms and, 169n3
  Cobol and, 142
  computers and, 4, 16–17, 24–25, 28, 50, 98, 142, 169n3
  expert systems and, 50–52
  Fortran and, 142
Prolog, 51
Psychology, 28, 95

Q learning, 134

Randomness, xi
  actions and, 129, 131–133, 136
  clusters and, 113, 119
  coin tossing and, 32–36, 53–54, 82–83
  credit scoring and, 45–50
  customer behavior and, 13
  deep learning and, 197
  games and, 162
  optimization and, 163
  pattern recognition and, 24, 44, 57, 71, 79
  probability theory and, 32–36
Ranking, 81–82
Recommendation systems, 81, 118–123, 145, 160, 163–164
Recurrent networks, 92–94
Regression
  credit scoring, 45–50
  learning clusters and, 116
  neural networks and, 90
  pattern recognition and, 58, 76, 81–82
  price of used car and, 29–32, 34, 40, 59, 81, 89
  statistics and, 38, 46, 58, 76, 81–82, 90, 116, 155, 161
Regularization, 84
Reinforcement learning, 161
  agents and, 128, 131, 135
  credit assignment problem and, 127–128
  environment and, 126–132, 135
  expected reward and, 131
  expected value and, 133
  exploitation and, 132
  exploration and, 129, 132
  feedback and, 125–128, 132
  games and, 125–127, 131–132, 136, 139
  internal value and, 128
  K-armed bandit and, 128–132
  learning with a critic and, 127
  rewards and, 127–135
  robots and, 126–127, 131–135
  TD-Gammon program and, 136
Representation
  actions and, 138
  algorithms and, 20–22

Representation (cont.)
  bag of words, 69–70, 116, 122
  deep learning and, 104, 108–109
  digital technology and, 3, 6
  distributed, 96
  hierarchical, 99–103
  input, 30, 116
  kernel function and, 116
  learning clusters and, 116, 120
  local, 96
  natural, 55
  neural networks and, 96, 99–103
  pattern recognition and, 55, 63, 69–70
  visual, 63
Retail, x, 157
Rewards, 127–135
Ribonucleic acid (RNA), 114
Risk, 45–50, 52, 117, 163–164
Robots, x
  actions and, 126–127, 131–135
  autonomous agents and, xiv, 149, 160, 163–167, 170n3
  future issues and, 149–150
  reinforcement learning and, 126–127, 131–135
  self-driving cars and, 149–150
  speech synthesizers and, 67
Rosenblatt, F., 86
Rumelhart, D. E., 92, 95
Russell, S., 165

Safety, 66
Samples
  learning clusters and, 115
  Markov chain Monte Carlo (MCMC), 83–84
  neural networks and, 89, 106
  pattern recognition and, 57, 59, 70, 72, 83
  spam and, 16–17
  statistics and, 16–17, 27, 33, 47, 57, 59, 70, 72, 83, 89, 106, 115
Sandel, M., 169n3
Sanitization, 158
Schmidhuber, J., 106
Science journal, xiii
Search engines, 82, 144, 147
Security, 66, 156–159, 161
Self-driving cars, 149–150, 164–165
Semi-parametric estimation, 58–59
Sentiment, 147
Shortliffe, E. H., 51
Short-term memory, 92–93
Signatures, 62, 66
Silver, D. A., 138
Simple cells, 100
Single instruction, multiple data (SIMD) machines and, 97–98
Smart cars, 148–149
Smart devices, x, 9, 71, 142–144, 148, 153, 157
Smartphones, 7–8, 14, 71, 139, 145, 153, 169
Social data, 9–10
Social media, 10, 15, 69, 147, 164
Software
  adaptation by, ix
  advancements in, 141–142
  distributed computing and, 151
  hardware constraints and, 18
  high-performance computing and, 161
  language limitations and, 141–142
  neural networks and, 96–97
  parallel computing and, 151
  personal applications and, 4

  poorly trained, 166
  privacy and, 159
  specialized libraries and, 152
  validation/verification of, 163
Spam, 16, 68–69
Sparse data, 121–122
Spectral clustering, 116
Speech recognition, 19, 28, 67–68, 71, 115, 117, 153, 170n3
Speed of light, 1, 6, 151, 170n1
Spoofing, 66
Statistics
  artificial intelligence (AI) and, 28
  big data and, 160–161
  coin tossing and, 32–36, 53–54, 82–83
  discriminants and, 27, 47–50, 75, 78
  estimation and, 27 (*see also* Estimation)
  inference and, 27, 50–51, 62–64, 83, 148
  learning clusters and, 116
  mixture models and, 112
  nonparametric models and, 77, 79, 84, 161
  pattern recognition and, 58, 63, 73
  price of used car and, 29–32, 34, 40, 59, 81, 89
  processing costs and, 156
  regression and, 38, 46, 58, 76, 81–82, 90, 116, 155, 161
  samples and, 16–17, 27, 33, 47, 57, 59, 70, 72, 83, 89, 106, 115
Storage, 3–5, 90, 143–144, 150–152, 169n2
Streaming data, 90, 144–145, 148, 152
Supervised learning
  actions and, 127, 129–130, 136–137
  models and, 38–42, 46, 89, 111–117, 127, 129–130, 136–137
  pattern recognition and, 111–117
  regression and, 38 (*see also* Regression)
Support vector machine, 116
Sutton, R. S., 127
Sweeney, L., 158
Synapses, 20, 86–87, 92

TD-Gammon program, 136
Teh, Y. W., 84
Templates, 24–25, 36, 56, 114, 117
Temporal difference (TD) learning, 133–136
Terabyte, 143
Tesauro, G., 136
3D printing technology, 166
Thrun, S., 135
Tolstoy, L., 72
Training sets, 31, 39–40, 89–90, 155
Translation, xii, 68, 70, 95, 109, 114, 139, 145, 167
Trending topics, 69–70, 147
TV, x, 8, 24, 144, 164

Ubiquitous computing, 9
Uncertainty, 32, 34, 51–52, 74, 80, 83, 131, 135
Universal approximators, 99
Unsupervised learning, 111–112, 114, 117

Valiant, L., 80
Validation, 155–156, 163
Vapnik, V., 116

Video, 6, 16, 66–67, 118, 144, 147–148, 160
Video games, 136, 152
Videotapes, 144
Vision, 19, 100, 104, 108, 149, 166–167, 170n3
Visual cortex, 74, 100, 166–167
VLSI (very-large-scale integration), 28

Web scraping, 147
Wiesel, T., 100
Williams, R. J., 92
World Wide Web, 6

XOR gate, 91–92, 99

Zadeh, L. A., 51

ETHEM ALPAYDIN is Professor in the Department of Computer Engineering at Bogaziçi University, Istanbul. He is the author of the widely used textbook *Introduction to Machine Learning*, now in its third edition (MIT Press).