Max Garzon · Ching-Chi Yang · Deepak Venugopal · Nirman Kumar · Kalidas Jana · Lih-Yuan Deng *Editors*

Dimensionality Reduction in Data Science

This book provides a practical and fairly comprehensive review of Data Science through the lens of dimensionality reduction, as well as hands-on techniques to tackle problems with data collected in the real world. State-of-the-art results and solutions from statistics, computer science and mathematics are explained from the point of view of a practitioner in any domain science, such as biology, cyber security, chemistry, sports science and many others. Quantitative and qualitative assessment methods are described to implement and validate the solutions back in the real world where the problems originated.

The ability to generate, gather and store volumes of data in the order of tera- and exo bytes daily has far outpaced our ability to derive useful information with available computational resources for many domains.

This book focuses on data science and problem definition, data cleansing, feature selection and extraction, statistical, geometric, information-theoretic, biomolecular and machine learning methods for dimensionality reduction of big datasets and problem solving, as well as a comparative assessment of solutions in a real-world setting.

This book targets professionals working within related fields with at least an undergraduate degree in any science area, particularly quantitative. Readers should be able to follow examples in this book that introduce each method or technique. These motivating examples are followed by precise definitions of the technical concepts required and presentation of the results in general situations. These concepts require a degree of abstraction that can be followed by re-interpreting concepts like in the original example(s). Finally, each section closes with solutions to the original problem(s) afforded by these techniques, perhaps in various ways to compare and contrast dis/advantages to other solutions.

ISBN 978-3-031-05370-2

9 783031 053702

▸ springer.com

Garzon · Yang · Venugopal · Kumar · Jana · Deng *Eds.*

Dimensionality Reduction in Data Science

Max Garzon · Ching-Chi Yang · Deepak Venugopal · Nirman Kumar · Kalidas Jana · Lih-Yuan Deng · *Editors*

Dimensionality Reduction in Data Science

Springer

Max Garzon
Ching-Chi Yang   Deepak Venugopal
Nirman Kumar Kalidas Jana Lih-Yuan Deng

Dimensionality Reduction in Data Science

– Professional Book –

March 22, 2022

Springer

Preface

Data science is about solving problems based on observations and data collected in the real world. Problems may range from the mundane to difficult scientific questions, for example, rating movies for recommendation systems, understanding the earning power of American taxpayers, increasing revenue for a business, spam, controlling the spread of misinformation through the internet, global warming, or the expansion of our universe. Our ability to generate, gather, and store volumes of data in the order of tera- and exo-bytes daily has far outpaced our ability to derive useful information from it in many fields, with available computational resources. The overarching goal of this book is to provide a practical and fairly complete, but not encyclopedic, review of Data Science (DS) through the lens of Dimensionality Reduction (DR).

The intended audience consists of professionals and/or students in any domain science who need to solve problems to answer questions about their domain based on data. Domain science is a fairly vague term that refers to a specialized area of human knowledge characterized by specific questions about a certain aspect of reality (like what is motion in *physics*, what are physical objects made of in *chemistry*, what is life in *biology*, and so forth.) In addition to the well-established sciences, they include just about any area where data can be recorded and analyzed to answer questions concerning the population of individuals or objects the data is about.

Data science presents a singular approach to problem solving when compared to more established sciences. Traditional sciences are motivated by understanding our world in order to survive and thrive. That requires a degree of analysis and theorizing to understand the specific phenomena involved and to enable predictive power. By contrast, with the advent of computer science and its abstractions into the information age (as embodied by the internet and web for example), tools have been created that can be used regardless of the specific domain. Once this threshold is crossed, then it is a natural next step from mathematics and statistics to synergistically combine them with the powerful computational tools developed by computer science to create a new science that is more than the sum of the parts, hence data science.

We have strived to leave our niche hats at the door and present an intuitive, integrative and synergistic approach that captures the best of the three worlds. That is the pervasive thread that readers will discover through examples and methods throughout the book. Sections begin with intuitive examples of a problem to be solved by the (perhaps new) concepts and results being described in the section. A professional with an undergraduate degree in any science, particularly quantitative, should be able to easily follow this part. These motivating examples are then followed by precise definitions of the technical concepts and presentation of the results in general situations. These require a degree of abstraction that can be followed by re-interpreting concepts like in the original example(s). Finally, each section closes with solutions to the original problem(s) afforded by these techniques, perhaps in various ways to compare and contrast dis/advantages of the various DR techniques based on quantitative and qualitative assessments back in the real world.

We are grateful to acknowledge support for this project from various sources. First, support from the University of Memphis CoRS (Communities of Research Scholars) program (through Deborah Hernandez) to start up research projects that facilitated initial interactions that eventually led to the interdisciplinary collaboration that produced this book, among other works, as well as access to the High Performance Cluster (HPC) used in development and testing of most results herein. Second, Lih-Yuan acknowledges support from the National Science Foundation under an award for The Learner Data Institute (NSF-1934745: The opinions, findings, and results are solely the authors' and do not reflect those of the funding agency.). Third, Max acknowledges faculty (Professor Luis Fernando Nino) and student support (especially Alfredo Bayuelo for TA support and help with cartoon design and rendering) at the National University of Colombia for their active feedback and project outcomes at the XI International Cathedra on *Data Science for Bioinformatics* in 2017, where several of the approaches, views, and some results included in this book were conceived.

Bon voyage!

Memphis, TN, USA
November 2021

Max Garzon
Ching-Chi Yang
Deepak Venugopal
Nirman Kumar
Kalidas Jana
Lih-Yuan Deng

Contents

Acronyms

General	
DS, DR, ML	Data Science, Dimensionality Reduction, Machine Learning
P	Probability (measure)
p, n	Dimensionality p, size n of a dataset
Data Science	
[**IrisC**]	Iris Flower Classification problem
[**MalC**]	Malware Classification problem
[**BioTC**]	Biotaxonomy Classification problem
[**SynC**]	Noisy Synoptic Data Classification problem
[**AstroP**]	Planetary Body Location Prediction problem
[**PhenP**]	Phenotypic Prediction problem
[**RossP**]	Rosette Area Prediction problem (of a plant)
[**LocP**]	Provenance of a Plant Prediction problem
MDS	Multi-Dimensional Scaling (classical or metric)
ISOMAP	Isometric Mapping
t-SNE	t-Stochastic neighbor embedding
RND	Random Scaling
MNIST	MNIST dataset
Statistical	
$\varOmega$	A population/sample space
D, **X**, **Y**	A sample or dataset from $\varOmega$
P	Probability (measure)
$P(B \mid A)$	Conditional Probability of event B given event A
X,Y, $\cdots$	Random variables (RVs, possibly with subindices)
E(X)	Expected value (mean) of a RV X
Var(X), $\sigma(X)$	Variance, standard deviation of a RV X
H	Shannon Entropy

N(0,1)	Standard Normal Distribution
N(μ, σ)	Normal Distribution with mean μ and standard deviation σ
EDA	Exploratory Data Analysis
PC, IC	Principal Component, Independent Component
PCA, ICA	Principal, Independent Component Analysis
KPCA	Kernel Principal Component Analysis
LDA,QDA	Linear, Quadratic Discriminant Analysis
GLM	Generalized Linear Model
MLR	MultiLinear Regression
BL	Bayesian Learning
SVD	Singular Value Decomposition
SVM	Support Vector Machine

Computational/Machine Learning

AP	Computational/Algorithmic problem
[**CharRC**]	Handwritten Character Recognition problem
[**TSP**]	Traveling Salesperson problem
$\mathbf{D}_m$	The DNA space of Watson-Crick paired oligomers (pmers) of length m
h, $h(x, y)$, $\|*, *\|$	Hybridization h-distance in $\mathbf{D}_m$
E, P^i	Equator, i^{th} Parallel in DNA space $\mathbf{D}_m$ (possibly with subindices m)
[**CWD**]	Noncrosshybridizing (nxh) basis/microarray Codeword Design problem
GNB	Gaussian Naive Bayes
MLR	MultiLinear Regression
MLP	MultiLayer Perceptron
FFN, NN, CNN	Feed-Foward, Neural, Convolutational Neural Network
SVM	Support Vector Machine
RF	Random Forest
kNN	k-Nearest Neighbors
NFL	No Free Lunch Theorem
R^2, AICS, BIC	Assessment metrics of statistical solutions: R-squared, Akaike's Information Criterion, Bayesian Information Criterion
$a, p, r, F1, RMSE, RE, s$	Assessment metrics of ML solutions: accuracy, precision, recall, F1-score, Root Mean Square Error, Relative Error, silhouette

Mathematical

$\approx, \propto, \ll, \gg$	Approximately equal to, proportional to, much smaller/greater than
$\alpha, \beta, \lambda, \nu, \sigma, x, y, \cdots$	Scalars (possibly with subindices or superindices)
$\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}, X, Y, \ldots$	Vectors (possibly with subindices or superindices)
$\mathbf{C}, \mathbf{X}, \mathbf{Y}, \mathbf{W}, \cdots$	Matrices (possibly with subindices)
N, Z	The set of natural numbers $\{0, 1, 2, \cdots\}$, integer $\{0, \pm 1, \pm 2, \cdots\}$ numbers
Q	The set of rational numbers $\{p/q : p, q \in \mathbf{Z}, q \neq 0\}$
R	The set of real numbers $(-\infty, +\infty)$
$\mathbf{D}_m$	The DNA space of Watson-Crick paired oligomers (pmers) of length m
$\lvert *, * \rvert; h, h(x, y), \lvert xy \rvert$	Distance; Hybridization h-distance in $\mathbf{D}_m$
NMF	Nonnegative Matrix Factorization

Chapter 1
What Is Data Science (DS)?

Max Garzon, Ching-Chi Yang, and Lih-Yuan Deng

Abstract Our ability to generate, gather, and store volumes of data (order of tera- and exo-bytes (10^{12}–10^{18} bytes) daily) has far outpaced our ability to derive useful information from it in many fields, with available computational resources. The theme of this book is a review of Data Science (DS) through the lens of Dimensionality reduction (DR). Data science is about solving problems based on observations of factors (referred to as *co-variates*, *predictors*, or just *features*) that may determine a solution. Typical kinds of problems are described, including classification, prediction, and clustering problems, as well as data collection methods.

1.1 Major Families of Data Science Problems

In this section, problems typically dealt with in data science are described and grouped into three major groups: **classification**, **prediction**, and **clustering**.

A *population* Ω is the entire group of individuals/objects of interest for a given problem, while a *sample* refers to a (relatively very small) subset $\mathbf{D} \subseteq \Omega$ of the population.

Data science problems can be regarded as computational problems in the sense that they call for certain types of inputs as data to a problem (as described in Sect. 1.3 below) and spell out an expectation in terms of the kind of result to be produced as a solution. The solution is usually some sort of model or program/code, ultimately to be translated into some sort of physical device to be run, typically a conventional computer. In computer science, these problems are referred to as *algorithmic problems* and are typically expressed in the forms of questions of a

M. Garzon (✉)
Computer Science, The University of Memphis, Memphis, TN, USA
e-mail: mgarzon@memphis.edu

C.-C. Yang · L.-Y. Deng
Mathematical Sciences, The University of Memphis, Memphis, TN, USA
e-mail: cyang3@memphis.edu; lihdeng@memphis.edu

M. Garzon et al. (eds.), *Dimensionality Reduction in Data Science*,
https://doi.org/10.1007/978-3-031-05371-9_1

similar kind being provided as inputs from a population of objects, as illustrated with the three major families described in the following subsections.

1.1.1 Classification Problems

One of the most common kind of problems is *classification*.

Example 1.1 Iris flower classification is a relatively simple example. The population of objects under consideration for inputs is a feature vector x giving the length and width of sepals and petals of an iris flower. The flower is to be placed in one of the three categories. These categories must be specified in advance for the problem and determine the type of classification being made into disjoint and exhaustive classes, i.e., they constitute a *partition* Π of the population. If the categories change, then we are dealing with a different problem since the answers will have to change across the inputs. The problem can be stated precisely as follows, with $\Pi_1 = \{$*Setosa, Versicolor, Virginica*$\}$. □

[IrisC] **IRIS FLOWER CLASSIFICATION** (Π_1)

INSTANCE:	A feature vector $x =$ (sepal length, sepal width, petal length, petal width) (in cms) describing an iris flower, e.g., (4.6, 3.1, 1.5, 0.2)
QUESTION:	Which kind of flower in Π_1 is x?

The brackets contain a mnemonic name **[IrisC]** used to refer to the problem in the sequel. A dataset for this problem is described in more detail in Sect. 11.5.

Example 1.2 Malware classification is a more complex example. The population of objects under consideration for inputs is a computer program/code that causes harm when run on a computer, usually referred to as *malware*. A sample from this population is discussed above in Sect. 1.1, where a given piece of malware is expected to be categorized into nine (9) classes or categories, as shown in Table 1.1. □

The problem can then be stated precisely as follows.

[MalC] **MALWARE CLASSIFICATION** (Π_2)

INSTANCE:	A piece of malware x
QUESTION:	Which category c in Π_2 does x belong in?

The Malware Classification Challenge (*arxiv.org/abs/1802.10135*, accessed October 2021) dataset was released in 2015 and is publicly available through *kaggle.com*. An available dataset consists of a set of 10,868 known malware files representing a mix of 9 different malware families, as summarized in Table 1.1. Each datapoint contains both the raw binary content of the malware file as well as metadata information extracted using the IDA disassembler tool, for a total of

Table 1.1 Microsoft's malware classification problem. The labels for the classes in the partition Π are in the first column

Labels	Class name	Type of malware	Count
1	Ramnit	Worm	1541
2	Lollipop	Adware	2478
3	Kelihos_ver3	Backdoor	2942
4	Vundo	Trojan	475
5	Simda	Backdoor	42
6	Tracur	TrojanDownloader	751
7	Kelihos_ver1	Backdoor	398
8	Obfuscator.ACY	Any kind of obfuscated malware	1228
9	Gatak	Backdoor	1013
			10,868

1804 raw features. The classification challenge is to correctly assign every piece of malware in the population to one of the nine (9) possible malware classes defined with the challenge. A dataset for **[MalC]** is described in more detail in Sect. 11.5.

Example 1.3 A third and more difficult example of a classification problem is species identification in biology for a given group of species (a taxon) T. The population of objects under consideration consists of all living organisms belonging to a species in T. The taxon T is the union of all specimens/individuals in the species in the second column of Table 1.2, according to a given biological taxonomy, while the partition labels Π are shown in the first column. □

For a data sample, representative genes were obtained from mitochondrial DNA (specifically, cytochrome Oxidase genes COI, COII, and COIII, CytB) of 249 organisms (as shown in the fourth column of Table 1.2) collected from biological genetic repositories (e.g., GenBank at *www.ncbi.nlm.nih.gov/genbank/*). A dataset for this problem is described in more detail in Sect. 11.5. The problem can then be stated precisely as follows for an arbitrary taxon T.

[BioTC] **BIOTAXONOMIC CLASSIFICATION**

INSTANCE: A DNA sequence representing a living organism x
QUESTION: What species in T does x belong in?

In summary, a *classification problem* is defined by a partition Π of a population Ω (into a number of *mutually disjoint classes exhaustive of* Ω) and the problem is to place an arbitrary element of the population into the right class. A solution to the problem is a model that will make a (hopefully correct) decision for every element from the population (*not just the sample*) as to which class it belongs to.

Table 1.2 Organisms in the sample data for **[BioTC]** of a biological taxon. The 21 classes/labels in the partition Π are in the first column

Label	T: genus species	Common name	Count
1	*Apis mellifera*	Western honey bee	4
2	*Arabidopsis thaliana*	Cress	5
3	*Bacillus subtilis*	Hay/grass bacillus	18
4	*Branchiostoma floridae*	Florida lancelet	18
5	*Caenorhabditis elegans*	Round worm	6
6	*Canis lupus*	Wolf	18
7	*Cavia porcellus*	Pork	4
8	*Danio rerio*	Zebra fish	9
9	*Drosophila melanogaster*	Fruit fly	18
10	*Gallus gallus*	Red junglefowl	18
11	*Heterocephalus glaber*	Naked mole rat	3
12	*Homo sapiens*	Human	18
13	*Macaca mulatta*	Rhesus macaque	8
14	*Mus musculus*	House mouse	18
15	*Neurospora crassa*	Red bread mold	5
16	*Oryza sativa*	Asian rice	12
17	*Pseudomonas fluorescens*	Infectious bacterium	6
18	*Rattus norvegicus*	Brown rat	18
19	*Rickettsia rickettsii*	Tick-born bacterium	18
20	*Saccharomyces cerevisiae*	Yeast	18
21	*Zea mays*	Corn/maize	7
			249

1.1.2 Prediction Problems

A second kind of problems is *prediction*.

Example 1.4 The population could be plants x (e.g., a specimen of *A. thaliana*) and the question of interest is to know the area of the rosette of the plant covered by the spread of its leaves at maturity. The problem can then be precisely defined as follows for a taxon of plants T.

[RossP] **ROSETTE AREA OF A PLANT** (T)

INSTANCE: a (long) DNA sequence x (over the alphabet $\{a, c, g, t\}$) representative of a plant in T?

QUESTION: What is the area of x's rosette when fully grown?

Note that there is a dependency (not necessarily functional since environmental factors appear to be involved, e.g., how well the plant was fed) that roughly determines the rosette area given the genome of the plant. A similar problem can be posed for the weight of a person as a function of their height. □

Example 1.5 Another example is the problem [**LocP**] of determining the origin of the plant for a taxon of plants T based on DNA sequence alone, as stated below. □

[LocP] **PROVENANCE OF A PLANT** (T)

INSTANCE:	a (long) DNA sequence x (over the alphabet $\{a, c, g, t\}$) describing a plant in T
QUESTION:	Where on Earth (latitude, longitude) was x grown?

In summary, a *prediction* problem is defined by a function f on a population that associates numerical values to every element in the population (usually hard to measure directly) and the problem is to find the value $f(x)$ of the function f for an arbitrary element x in the population Ω. A solution is a model to determine that value based on other features identifying the input elements x from Ω. Since this may be difficult, one may need to settle for just approximate values.

1.1.3 Clustering Problems

A third kind of common problem is *clustering*.

Example 1.6 When a typical US town grows to a certain size, it is desirable to set up some fire stations just in case a house or building catches fire. Naturally, when your house is on fire, you would like the station to be as close as possible. To conceptualize the problem, we can specify a number n of points c_i $(i = 1, \cdots, m)$ in a 2D Cartesian plane to designate the positions of the fire stations. Thus, any fire emergency at a house located at any given position x will be tended to by fire trucks at station c_{i_0} closest (at minimum distance) to it in distance, i.e., so that

$$i_0 = arg \min_i \|c_i - x\| .$$

So, if there are two stations c_1, c_2, locations get clustered in two clusters, defined by half-planes separated by the perpendicular bisector of the segment joining them. If there are three stations, the regions are like cake slices, as shown in Fig. 1.1a, with boundaries being the relevant fragments of the perpendicular bisectors of all three pairs of stations. More stations will complicate the partition of the 2D plane, as illustrated in Fig. 1.1b, but along the same ideas. This kind of diagram is called a *Voronoi diagram* and is defined by the fire station locations $c_1, \cdots, c_m$, which are generically referred to as *centroids*. This diagram is a solution to the problem of clustering the population into groups that are each served by the same fire station according to the ordinary Euclidean (ℓ_2) distance. □

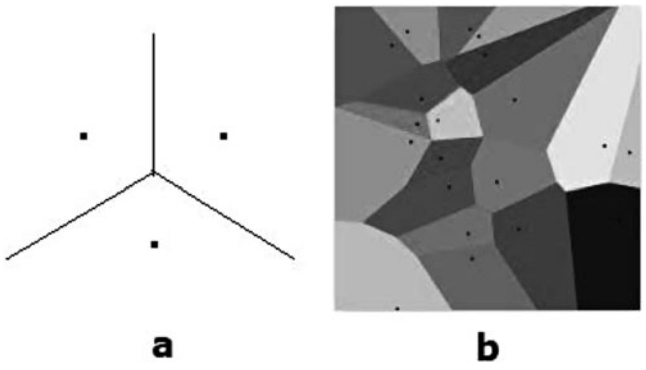

Fig. 1.1 Voronoi diagrams as solutions to the clustering problem defined by the Euclidean distance in the population of the 2D plane for (**a**) three points (e.g., fire stations) and (**b**) 20 points

In summary, a *clustering problem* is defined by a metric (usually a distance function, defined precisely below) function between elements in a population to capture degrees of (dis)similarity, and the problem is to produce a partition (*unlike a classification problem, where the partition is given*). A solution is to find a model that will produce a partition such that elements in any one cluster are more similar among each other than to elements in the other clusters.

It may be difficult to make a crisp distinction between these kinds of problems. For example, a partition Π in a given classification problem can be regarded as a function f on the same population assigning a numerical label in an enumeration of the classes in some order, so it would appear that every classification problem is a prediction problem (especially if a solution to the problem is used to label unknown inputs). Conversely, if the range of f in a prediction problem is small, one can regard it as a classification problem as well. Thus, we will speak of prediction problems only when the range of f is a fairly large set of (perhaps infinitely many) analog values. Therefore, typical classification problems usually involve a few categories in their partitions (in the order of at most tens). Likewise, once a solution to a clustering problem is obtained, one can use the clusters as a partition in a classification problem to predict the clusters for unknown elements in the population.

Other kinds of problem besides these three categories can certainly arise in data science. An example involves dynamical sequences of actions as inputs (e.g., a video of a pedestrian crossing a street) and questions concerning the next expected action (e.g., will the pedestrian run across or turn back on an upcoming car). There is no telling what kind of problem could arise in any domain science. However, these three main categories can always be used, at least in the first approximation to a given problem. For example, the answers to the pedestrian problem can range from a simple classification (forward/stop/back) to a prediction of the next step, to prediction of complex sequence of moves. (What sequence of actions will the pedestrian exactly take in the next 5 seconds?)

1.2 Data, Big Data, and Pre-processing

In this section, a characterization of what is understood by data and major techniques to conceptualize, visualize, and understand data are reviewed. Understanding data, particularly large datasets and big data, can prove to be extremely difficult, particularly when faced with acting on decisions based on data analytics discussed in later chapters to sell a solution of a particular problem to management. Further, the results of data analysis may impact an individual's health, or even large communities when applied at large scales in the real world, so understanding the scope and rationale for the results of the analyses is critical.

1.2.1 What Is Data?

A basic question that any book on data science must address is, what exactly is *data*? Many answers have been given to this question since the dawn of the information age in the 1940s. For example, well-known computer scientists (such as Peter Naur of BNF: Backus-Naur Normal Form fame) wanted to redefine computer science as *data science* back in the mid-1950s. So have well know statisticians (such as C.F. Jeff Wu of EM optimization and '*Statistics = Data Science*' fame in the 1970s), implying that data is about observations in the form of numbers, that it can be used to make inferences about entire populations and that there is a computational component to it. Like an iron age man on attempting to answer the question *What is iron?*, we can tell data when we see it but are very hard pressed when asked to give a rigorous noncircular definition of data. Plain numbers in abstract or computer programs to crunch them cannot really be data as it is understood today. We will thus adopt the following characterization.

> *Data* is an objective *recording* of one or several event(s) in the real world that is accessible at later times for perusal and analysis by at least one person.

Example 1.7 Thus, the following do *not* qualify as data: the plain occurrence of an event; a memory in someone's head as a witness of an event (unless an objective recording is made of the witness describing or reliving the event -humans update their memories as they remember them); the contents of a computer's RAM memory (unless it is being recorded); and so forth. On the other hand, the following can be considered data: website contents containing inaccurate facts and spreading misinformation; inaccurate scientific observations (due to faulty equipment or even malicious intent); fossils of dead or extinct animals; videos of a person (even if no longer living); and statistics collected through observations or machines and stored on paper or machine. Thus, data requires the so-called four Ws: *What, Who from, Where, and When.* □

It will be useful to make further distinctions with other related concepts, such as *information* and *knowledge* (*wisdom* is out of reach). In this book, data differs from information in that information requires, in addition, a purposeful observer, for example, trying to solve a (data science) problem, as discussed below in Sect. 1.3. Data becomes information if it can be used to extract a solution to the problem with some processing, as illustrated by many examples in the following chapters. (The more complex concept of knowledge will be re-visited in Sect. 10.2).

Data can assume a variety of formats, for example, numbers, text, DNA or protein sequences, voice recordings, images, x-rays, videos, browser histories, time stamps, and so forth. These recordings are considered to be *raw* or *primary* data. They usually require transformation and processing to enable the kind of analysis done in data science to solve a problem, as will be discussed in this section and later chapters.

The most common format for the simplest kind of data is a two-dimensional (henceforth, just 2D) *table* consisting of rows and columns, usually organized so that attributes of the data are placed in the columns and each specific observation (sometimes referred to as a *datum*) occupies an entry in the rows. Herein, they will be referred to as *features* and *records*, respectively. Therefore,

> a *record* is a p-dimensional (henceforth pD) vector of features values (datums, or its latin plural, *data*). A table can be regarded as an nD vector of pD vectors, the records. The *dimensionality* (*size*) of the data/sample is p (n, respectively.)

Mathematically, a table can be regarded as a sample and its columns can be regarded as random variables X, as described above in Sect. 1.1 (see Sect. 11.1 for statistical background concepts). Single tables are good enough for small datasets but a more complex dataset or problem may require many such tables to form a *data corpus* for the problem simply because there may be many very different kinds of observations and attributes about objects in the same population the corpus is all about. Data corpora raise the problem of properly organizing the data so that humans can comprehend it. A criterion or logic to *organize* a dataset or data corpus is usually called an *ontology* for the data.

Example 1.8 If the problem of interest is the earning power of American taxpayers, the underlying population is, of course, American citizens who must pay income taxes by law. An important table here would contain a taxPayerID, a fiscal year, an AGI (AdjustedGrossIncome), and possibly taxDue. Another table may include demographic information, including taxPayerID, taxPayerFullName, CurrentAddress, priorAddress, and bankAccountNr as features. A third table may contain information about itemizedDeductions for a given year, and so forth. Many other tables are probably necessary to keep a full history of a taxPayerID in the full data corpus. The ontology in this case is the grouping of the information by taxPayerID and into a number of categories (like tax data for the first table, demographic data for the second, itemized deductions for the third) for each fiscal year. Other ontologies are certainly possible (e.g., brute force, a single table with one record for each American tax payer, and hundreds or even thousands of attributes for each of them), although probably less desirable because it is less useful to work with and/or understand the data. □

On closer examination, a feature and its values (datums) in a table can be classified in various ways. For example, they could be qualitative or quantitative variables. A *qualitative* variable only observes values in a limited and fixed set of possible outcomes. If the list of outcomes can be sorted naturally, they are referred to as *ordinal*. Otherwise, the feature is called *nominal*. On the other hand, a *quantitative* variable observes values with a very large number of possibilities, usually the result of measuring something. They can be further separated into *discrete* or *continuous* variables. A discrete variable takes a value from a set of finite or countable numbers (indexable by natural numbers), while a continuous variable takes values from a set of uncountable numbers (like real numbers).

Example 1.9 (Qualitative or Quantitative) In order to distinguish qualitative and quantitative variables, a useful criterion is, *Can the values of the variable be added?* If a variable represents what is in a shopping bag, the bag may contain *oranges* and *apples* as answers. Since oranges and apples cannot be added (orange-apple makes no sense as a fruit), the list of items in the bag is qualitative data. However, if one is counting items in the bag, the data will be quantitative, say 2+3=5 items. □

Example 1.10 (Ordinal or Nominal) It is hard to objectively decide on a natural order for apples, oranges, and bananas (though their *names* can easily be sorted alphabetically). The variable typeOfFruit is nominal. On the other hand, a Likert scale for answering whether one likes a particular fruit is ordinal (e.g., *Like*, *Neutral*, *Dislike*). □

Example 1.11 (Discrete or Continuous) The criterion for distinguishing discrete or continuous variables can be, *Could a decimal number be an answer?* For example, if one is counting how many people are in a theater, a number such as 3.14 is nonsense. The number of people in a theater is discrete. However, if one considers the average number of people in a theater for one show over a period of one year, the number 3.14 could be an answer; this variable is continuous. □

1.2.2 Big Data

A similar problem to that in Example 1.8 arises with financial credit card transactions for a given bank or credit card company and many other problems for government and private organizations in every country. The data corpora can become really huge and they are usually referred to as *big data* (in the order of terabytes and larger). This fact raises the important issue of *understanding* datasets and data corpora. A good ontology will enable a human to grasp the large picture of what the data is all about (the four Ws) and make it possible to develop effective strategies to solve problems about the given population.

Big data is usually characterized by the so-called five V's:

- *Volume* (raw number of records/amount of data),
- *Variety* (how diverse is the type, nature, and format of the data),

- *Velocity* (speed of data generation),
- *Veracity* (trustworthiness/quality of captured data), and
- *Value* (insight and impact afforded by the data about the overall population).

Example 1.12 (Mobile Data with High Volume and Velocity) According to *ericsson.com*, total global mobile data traffic reached 49 exo-bytes (EB) per month at the end of 2020. It is expected to grow with the launch of the 5G internet. By comparison, the Iris dataset only contains 2.6×10^3 bytes, whereas a month of total global mobile data contains 4.9×10^{18} bytes. The data is generated extremely fast *www.ericsson.com/en/mobility-report/dataforecasts/mobile-traffic-forecast*, so this dataset exhibits very high volume and velocity. □

Example 1.13 (Data Variety) Long ago, data might have referred to a well-organized spreadsheet, e.g., .txt or .csv files. Variety refers to any data that could be mixedly generated by humans or by machines. For example, a Fitbit generates sensor data of a body by continuously monitoring the body movements. Browser history is generated by human activity. Twitch stream is a live video that records the show and chats between hosts and viewers. A data corpus containing these chats has high variety. □

Example 1.14 (Anonymity on the Internet) For freedom of speech, websites may offer online anonymity. However, it comes with the problem of data veracity. It is easy to lie and the information therein cannot be trusted. Moreover, fake information usually spreads faster than facts. □

Example 1.15 (Educational Data) Data value is a subtle concept. It refers to how useful the data is to answer research questions and turn it into business intelligence. In the case of educational data, students' progress/scores are stored from K-12th grades. If the data are just stored without further use or analysis, it offers little value. However, if one can utilize the data to further identify personalized key learning strategies for students, its value increases tremendously for the benefit of society.

□

Computational resources and techniques today are not able to process the extraordinary full volume of data being generated in so many fields. Because of the overwhelming growth of data collection, it can be challenging to extract useful information from big data with current computational resources. Choosing a good subset of the data as the training sample to build some suitable models is a critical issue. For proper big data modeling and analysis, it is common to use some statistical and machine learning methods for preliminary pre-processing in order to make big data amenable to analysis. It helps to reduce the computational time of the analysis from years to feasible.

1.2.3 Data Cleansing

Raw data, i.e., data that has not been processed and is not ready for analysis, requires *pre-processing* or *cleansing* (e.g., (re)formatting, organization, and selective extraction) to become data ready-to-use (sometimes referred to as *cooked data*). The process is also referred to as data *cleaning, scrubbing, wrangling*, and so forth. Cleansing can be manually done or automatically done by a machine for large datasets.

Example 1.16 (Handwritten Character Recognition) In handwritten character recognition, a trace of handwriting is processed (e.g., zip codes on ordinary mail envelopes in order to route the envelopes properly). Further details about this dataset can be found in Sect. 11.5. □

Example 1.17 (Malware Detection) Malware Classification (**[MalC]**) requires professional/domain knowledge to extract the information from an executable (.exe file) into cleanly labeled data suitable for analysis, where the rows are the pieces of observed malware and columns are the features extracted from the executable file. □

Example 1.18 (Stock Market) In the stock market, a trade can be made and recorded nowadays in approximately $\frac{1}{64}$ millionth of a second, i.e., at the rate of 64M per second. It is difficult to analyze this raw data because of its volume. It has to be organized and transformed for further analysis/prediction, e.g., by trading volume or opening/closing price in a day, to be of value. □

In Examples 1.16–1.18, raw data are not ready for answering research questions. Additional procedures are required to organize observations and their features.

Cleansing is the process of screening and/or transforming data to enable processing and analysis to derive value from it.

Needless to say, cleansing must be conducted prior to data analysis, although it is sometimes done later upon failure to get useful results from an analysis due to multiple reasons. The data may be *incomplete, noisy, inconsistent, duplicated, irrelevant*, or simply be *missing* values. The aim of cleansing and pre-processing is to remove noise, standardize the format, and retain useful information, so as to make sure the scripts for analyses that expect a common format will not crash due to formatting errors or missing values. This is a pre-condition to extracting value from the data analyses. There are several common ways to accomplish this goal.

1.2.3.1 Duplication

For example, if a minority opinion is surveyed repeatedly and a large number of duplicates are added to the dataset, the minority opinion might become the majority opinion. By removing duplication, one can avoid biased inferences and misleading conclusions.

Example 1.19 (Online Sweepstakes) Online sweepstakes offer great prizes. If one enters multiple times, the case is duplicated. To be fair, the website keeper should remove the duplication or prevent a person from entering the sweepstakes twice. □

1.2.3.2 Fixing/Removing Errors

It is always recommended to re-check the source to obtain the correct values for the missing values. However, this is not always possible. Certain values cannot be recovered in many situations. One way to handle it is to change the erroneous datum to a missing one and handle it accordingly.

1.2.3.3 Missing Data

Missing data/value refers to a datum being unavailable or corrupted. Data can be missing due to many reasons. For example, in a survey, an individual chose not to report an answer; the datum was not observable/available; or the datum went missing during data (pre-)processing. *Missing value imputation* is one common technique to obtain a missing value. Some common imputation methods are:

1. *Mean imputation*
   The missing value x_{ij} for the variable X_j can be filled in as the mean, i.e., $\bar{X}_j$. This method is generally used for a continuous variable since the mean of a discrete variable or a categorical variable might not make sense or be appropriate.
2. *Interpolation and extrapolation* (for numerical data)
   The missing value can be imputed by prediction. While interpolation implies the prediction falls within the range of data points, extrapolation implies the prediction falls outside the range of data points. Basically, by connecting two close data points without missing values with a straight line, the missing values can be imputed as the value on the fitted line.
3. *Regression imputation*
   The missing value can also be imputed by a model for the data obtained later by analysis. One type of model is the regression type, which will be discussed in detail in Sect. 2.1.

1.2.3.4 Outliers

An *outlier* is a data point containing a datum that is out of the range of the other points in the feature. While the multivariate situation is considered, a distance measurement, e.g., Mahalanobis distance, will be used to disrobe how far the observation is from the bulk of the data. The goal of a whole research area called *outlier detection* (also *anomaly detection*) is to estimate this distance and identify the outliers. It can be used for detecting unusual observations, such as bank fraud.

Example 1.20 (Bank Fraud Detection) It is essential to detect fraudulent transactions not only for a personal account but also for money laundering. If an account usually has transactions for grocery shopping, the occurrence of a large transaction appears as an outlier. A bank will issue a hold on this type of activity until confirmed. How large is considered "large" requires fine-tuning. (While keeping its customers' assets safe is important, the bank will not want to issue a hold every time and ruin most honest users' experience.) □

1.2.3.5 Multicollinearity

Multicollinearity (also collinearity) refers to the case where one predictor variable in a multiple regression model (described in Sect. 2.1) can be linearly predicted from the others with high accuracy. This technique is similar to combined inter-extrapolation, but with more than one missing value. (Some consider dependencies of this type to be "dirty" and require cleansing.)

1.2.4 Data Visualization

One of the major problems with data, particularly big or complex data, is to make it *understandable* to humans. This is particularly important for humans to verify, interpret, and/or rationalize the results of any analytics to act on a solution to a problem as being consistent with the data that led to it. In Human–Computer Interaction (HCI), the problem is usually solved by reaching into the human's head and finding a metaphor to transform and present the data to enable his/her brainpower with an appropriate transformation into a more familiar situation.

Example 1.21 A typical example is the problem of navigation on the metro system in a big city. A computer system can only store a list of metro lines and the locations (latitude or longitude) of their stop stations. Presenting a time table of these stations to a human will not enable easy navigation to instruct her to move from point A to point B. The visualization in the form of a graph in various colors (the lines) roughly following the layout of the city and their points of intersections and stops along the lines is a familiar and effective solution because humans easily understand space, distances, and orientation to navigate the city. □

In statistics, data visualization refers to exploratory data analysis (EDA) as an approach to analyzing datasets to summarize their main characteristics, often using graphics about statistics of the data features. Based on the variable's type (qualitative/quantitative, discrete/continuous), different EDA methods can be used. Table 1.3 and Figs. 1.2, 1.3, 1.4, 1.5, and 1.6 illustrate various techniques with the Iris dataset.

Table 1.3 A stem-and-leaf plot of the sepal length for the Iris dataset

Key: 4|3 = 4.3 cm
4 | 3444
4 | 566667788888999999
5 | 0000000000111111111222234444444
5 | 5555555666666777777778888888999
6 | 00000011111122223333333334444444
6 | 5555566777777778889999
7 | 0122234
7 | 677779

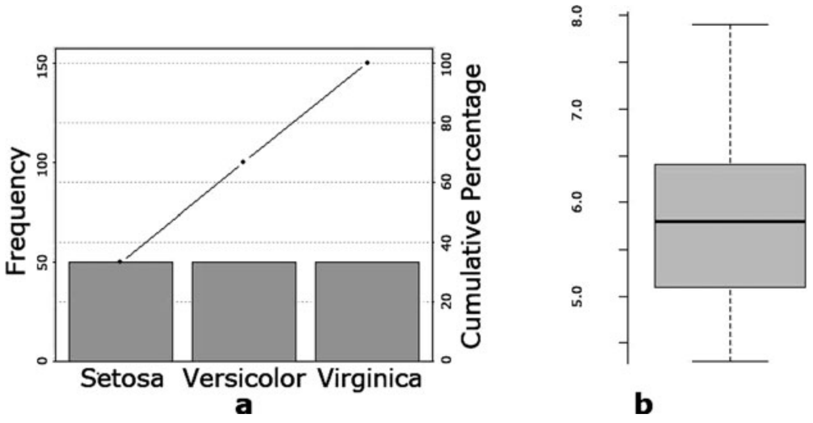

Fig. 1.2 (**a**) A Pareto chart with the count for the species and (**b**) a boxplot for the sepal length of the species in the species in the Iris dataset

Example 1.22 (Boxplot of a Continuous Variable) Boxplots are useful to represent the five number summary of a feature: the *minimum*, the *first quartile*, the *second quartile*, the *third quartile*, and the *maximal* of a variable. Figure 1.2b shows a boxplot of the sepal length for the Iris dataset.

Note that *quantiles* are points that divide the range of the observations in a sample variable into subranges (in a sequence) with equal probabilities. The most common quantiles are the 4-quantiles (are also called *quartiles*), the 10-quantiles (are also called *deciles*), and the 100-quantiles (are also called *percentiles*). □

Example 1.23 (Pareto Chart of a Variable) A *Pareto chart* is a bar graph. Figure 1.2a shows a Pareto chart of the three species in the Iris data sample: Virginica, Versicolor, and Setosa. The lengths of the bars are proportional to the frequency of each category and the curve presents their cumulative distribution. □

Example 1.24 (Histogram of a Continuous Variable) A *histogram* is an approximate representation of the distribution of numerical data. A histogram of the sepal length for the Iris dataset is shown in Fig. 1.3. □

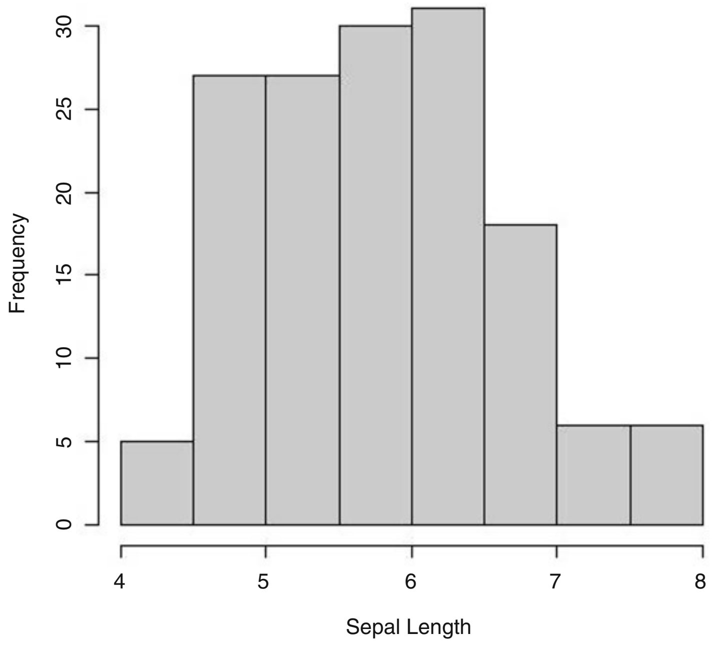

Fig. 1.3 A histogram of the sepal length in the Iris dataset

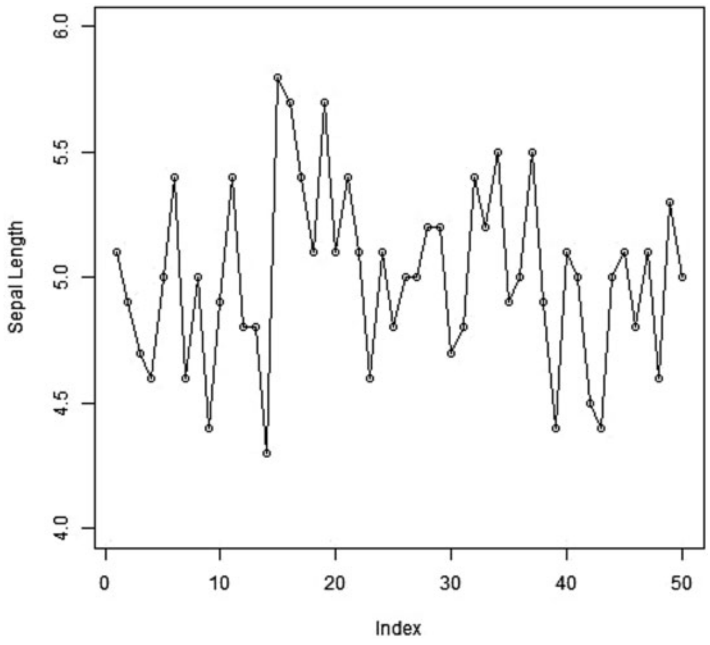

Fig. 1.4 A run chart of the sepal length in the Iris dataset

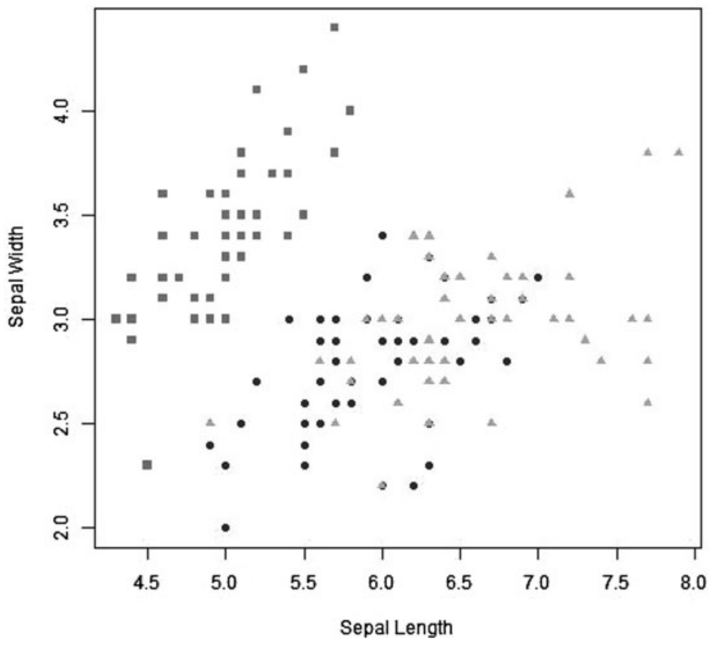

Fig. 1.5 A scatter plot of sepal length and width for the Iris dataset

Example 1.25 (Stem-and-Leaf Plot for a Quantitative Variable) A *stem-and-leaf plot* is a way of presenting a quantitative variable. The plot can also be treated as a table. For example, a stem-and-leaf plot of the sepal length in the Iris dataset is shown in Table 1.3. By studying a stem-and-leaf plot, one can achieve a visualization similar to the one obtained from a histogram. □

Example 1.26 (Run Chart)

To understand the dynamic change of a variable over time, a run chart is usually used. If the first 50 observations are recorded in a sequence over a time period of time for the Iris dataset, one may study whether the size of the flower will change over time. A run chart of the sepal length for the Iris dataset is shown in Fig. 1.4. □

Example 1.27 (Scatter Plot) It is common to have more than one variable (feature) in a dataset. A *scatter plot* represents an observation by a dot in the figure for two features (x-axis and y-axis). A scatter plot of the sepal length and sepal width in the Iris dataset is shown in Fig. 1.5. □

Example 1.28 (Multivariate Chart) A *multivariate chart* is used to combine different visualizations of multiple variables into one single chart to enable an overall comparison. A multivariate chart for the Iris dataset is shown in Fig. 1.6. □

Example 1.29 (Targeted Projection Pursuit) In big data, there are too many variables in the data corpus. It is hard to visualize all of them at once. *Targeted*

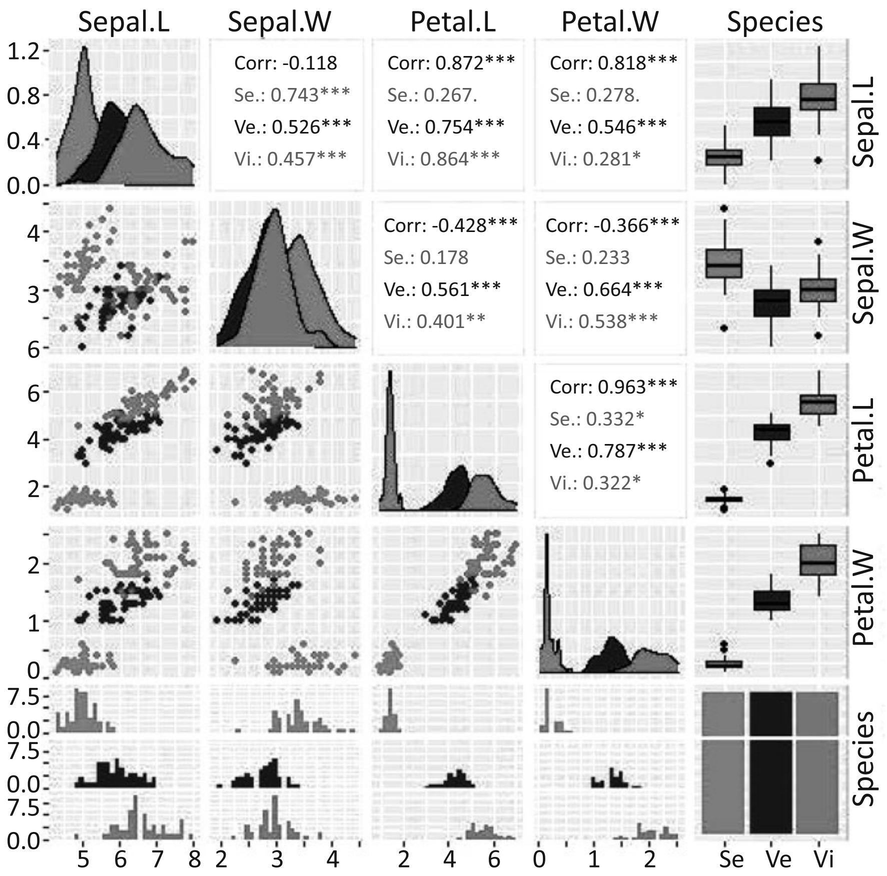

Fig. 1.6 A multivariate chart for the Iris dataset

projection pursuit projects the dataset into a lower dimensional space in order to get a clearer visualization. For example, if one is only interested in the combinations of the four Iris features, say

$$\text{SepalLength} = 0.4, \text{SepalWidth} = 0.1, \text{PetalLength} = 0.9, \text{PetalWidth} = 0.4\ ;$$

and

$$\text{SepalLength} = 0.7, \text{SepalWidth} = 0.7, \text{PetalLength} = 0.2, \text{PetalWidth} = 0.1\ ,$$

one can look at a scatter plot of the two targeted projection pursuit, shown in Fig. 1.7. It is clear that this might be a better visualization for a human to understand the difference/separation between the two species than Fig. 1.5. □

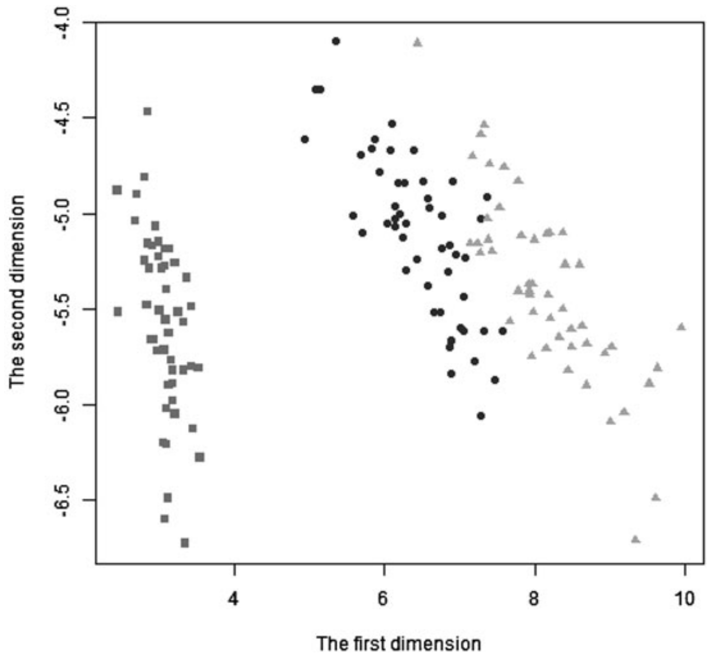

Fig. 1.7 A scatter plot of the targeted projection pursuit for the Iris dataset

In summary, various graphical visualization techniques can be used to visualize and help understand a dataset better, including but not limited to (in alphabetic order): *Boxplots, Histograms, Multivariate charts, Pareto charts, Run charts, Scatter plots, Stem-and-leaf plots*, and *Targeted projection pursuits*.

However, when dealing with big data, even these simple summarizing techniques might not perform well enough in providing a comprehensive picture. Other techniques, such as clustering and granularity re-scaling, may be useful, but they are applications of analytic techniques subject of the coming chapters, so they will be discussed in Chap. 9.

1.2.5 Data Understanding

Another way to understand data is through the use of analogies and abstractions, similar to the way life is understood in the form of specific organisms (my dog or cat), then abstractions into groups (like the biological class of all domestic cats or dogs), then even more abstract concepts (like mammals). Abstractions that are particularly pervasive and powerful among humans involve the concepts of *space* and *time*. They are fundamental to humans and much of intelligence has to do with that (even to the point that some influential philosophers like Kant have proposed that any concept has to be ultimately translated into such geometric concepts to really make sense to the human mind ([1] has more details). So, it is not surprising that mathematicians, physicists, and even cognitive scientists have developed conceptual tools to understand them. They are particularly useful in data science, so it is worthwhile to take a closer look.

The starting point is the mathematical notion of Cartesian coordinate systems familiar to most high school students. They are the usual Euclidean spaces used in calculus and analytic geometry. Geometry is about points, lines, planes, distances, and relationships between objects in terms of them (e.g., shape). (Although the so-called synthetic geometry does not make use of metric rulers to measure distance, as is the case of classical Euclidean geometry from high school, only analytic geometry and its generalizations will be of concern here.) Psychologists have discovered that humans use the concept of distance very loosely. For example, in a city, the distance to go from point A to point B is not the length of the straight line segment joining them because one cannot walk through buildings but has to follow the rectangular street layout of the city. Likewise, if one is flying across the continent between New York and San Francisco, the shortest straight route would require to dig a tunnel through the round Earth, so a pilot will rather take a course along a segment of a greatest circle in the air parallel to the surface of the Earth. Hence, mathematicians have concluded that a more precise concept is required; one that can be characterized as an *abstract concept of distance* in a given space **E** as follows.

A *distance* (or *metric*) is a function

$$d(*, *) : \mathbf{E} \times \mathbf{E} \longrightarrow \mathbf{R}$$

assigning a nonnegative real-number $d(x, y)$ to every pair of points $x, y \in \mathbf{E}$ so that three properties are satisfied for arbitrary points x, y, z:

- *Reflexivity*: $d(x, y) = 0$ if and only if $x = y$
  (different points must be at positive distance, not 0);
- *Symmetry*: $d(x, y) = d(y, x)$
  (distance is adirectional);
- *Triangle Inequality*: $d(x, z) \leq d(x, y) + d(y, z)$
  (it is always farther to make a stop at y on the way from x to z).

A set of points **E** endowed with a distance function is called a *metric space*. (To ease notation, $d(x, y)$ may be just denoted $|x, y|$ in the sequel.)

Mathematicians feel quite comfortable with it because they can reason about any such notion just like they would about ordinary distance in Cartesian geometry, as long as the three key properties above are satisfied for arbitrary points $x, y, z \in \mathbf{E}$. Likewise for ordinary people and pilots moving around cities on Earth.

Therefore, the city distance in a Euclidean plane between two points is fine, and different metrics in the Euclidean space can be used to endow it with a geometric structure for a variety of purposes. Real datasets that could be mapped into a metric space can be analyzed by leveraging its structural properties that would otherwise be invisible to us. This kind of approach will prove invaluable for dimensionality reduction in upcoming chapters.

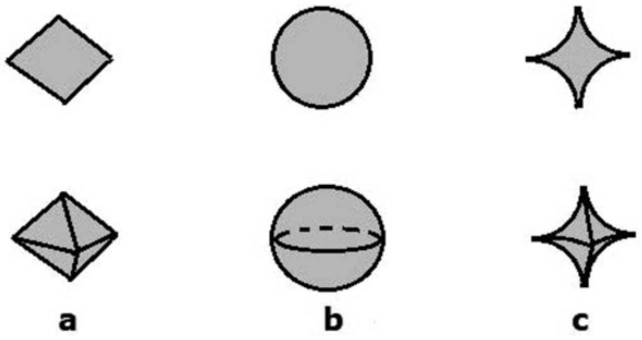

Fig. 1.8 Balls (sets of points within a given radius r from a center $\mathbf{c}$) in ℓ_p spaces for (**a**) $p = 1$; (**b**) $p = 2$; (**c**) $p = 3$ in the Euclidean 2D plane (circles, *top*) and 3D space (spheres, *bottom*). The balls can be closed $B[\mathbf{c}, r]$ or open $B(\mathbf{c}, r)$, depending on whether they include the points at distance exactly r (on their boundary) or not, respectively

Example 1.30 In a Cartesian 2D plane, the distance between two points $x = (x_1, x_2)$ and $y = (y_1, y_2)$ given by

$$\ell_1 : |x, y| = |x_1 - y_1| + |x_2 - y_2|$$

is called the *Manhattan distance* and is the ideal definition of a city distance where unit squares stand in for city blocks. The reader may notice that this distance has been obtained by substituting the exponent $p = 2$ in the ordinary Euclidean distance in Cartesian geometry by $p = 1$. The three properties of a metric still hold, although we need to adjust our intuition a little. For example, in the ordinary distance, the disk of radius 1 centered at the origin (i.e., the set of points within distance 1 from the origin) looks like an ordinary circle, whereas in the Manhattan distance, the same concept turns into a diamond since an equation $|(x, y), \mathbf{0}| = |x| + |y| = 1$ defines line segments joining the four points $(\pm 1, \pm 1|$, as shown in Fig. 1.8. □

Example 1.31 The Manhattan distance is sometimes called the ℓ_1 distance because it is in fact just one in a family of distance functions $\{\ell_p\}_p$ indexed by positive integers $p > 0$ and given by

$$\ell_p : \ |x, y|_p = [\ \Sigma_{1 \le i \le n} \ |x_i - y_i|^p \ ]^{1/p}.$$

The particular case $p = 2$ is actually identical to the ordinary Euclidean distance in Cartesian geometry (the square root of the sum of squares of the coordinate differences) and $p = 1$ gives the Manhattan distance. □

These abstractions enable familiar concepts in a human mind to understand and reason other apparently unrelated concepts (like the concept of a circle and a diamond being abstractly as similar as cats and dogs are both mammals), as will become more and more evident in the following chapters.

1.3 Populations and Data Sampling

This section describes how to cope with the fact that populations are usually large and inaccessible in the whole. Even if they became available, computational constraints to find solutions usually prevent full use of them anyway, despite the fact that what is needed are models that *answer questions for arbitrary and unknown elements of the population!*

Solutions to problems require either (rarely met) assumptions on the target population or lots of data to train models that may help answer the questions. Since collecting a full or even a significant sample (about a subset) from the population is commonly materially infeasible, sampling techniques are crucial to collect unbiased and representative data. A good sample will contain enough information to represent the key properties/characteristics of the population necessary to solve the problem for the entire population (not just the sample). This is easier said than done because

- Data collection is subject to available data and usually requires measurements with available technology;
- Data acquisition is very costly in terms of effort and time;
- Data is ephemeral and constantly requires updates (even for historical data as a result of new research!).

Therefore, sampling a population for good data requires careful planning and carefully chosen methods.

1.3.1 Sampling

Example 1.32 (Political Poll/Economic Survey) One may want to make a prediction about the result of the next presidential election. A single opinion is hardly a good prediction, but obviously, it is impossible to survey all individuals or make a proper and random selection from all citizens of the USA. With a proper sampling, a total sample size of 1500–2000 would be sufficient to achieve, say 95% confidence about the prediction of the outcome of the election in a population of over 200M eligible voters. □

Example 1.33 (Biomedical Study) Methylation of cytosine residues of cytosine-phosphate-guanine dinucleotides (CpGs) is one of the important DNA-based biomarkers associated with aging and/or some diseases. For any individual, there are many CpG sites (out of 28,000,000 CpGs with over 680,000 “useable” CpGs) that can be collected with all the CpGs values normalized to the range from 0 to 1. Because of the high cost of the data collection, only a few thousands of individuals can be sampled. The data can be used to build an “epigenetic clock” to predict epigenetic age, or to find key biomarkers associated with some specific diseases. □

For these examples, a “good” random subset (sample) should include similar characteristic of key variables (e.g., gender, race, age, socio-economic status). However, there are major differences between these two examples in terms of the goals of the study. The first example is typical in *survey sampling*, where the main interest is to estimate the population characteristic (e.g., election result), whereas the second example is typical in data science, where the major goal is to build a predictive model between response (e.g., epigenetic age or specific disease status) and the input features (e.g., CpG sites). In particular, the major goal of this book is to describe techniques to reduce the number of variables (factors or features) necessary to build a predictive model that yields results comparable with or even better than a model obtained from many more features.

In general, feeding low quality data to a solution will produce, unsurprisingly, a low quality solution (also known as *Garbage in? Garbage out!*). A good sample should exhibit four characteristic properties (R^2IBS):

- *Relevant*
- *Representative*
- *Informative*
- *Barely Sufficient*

Traditional sampling techniques were guided by statistical methods and usually assumed that the population is finite. Sampling techniques can be classified according to several criteria. One of them is whether a probabilistic model is considered in choosing the datapoints. Probabilistic sampling selects elements assuming a uniform distribution, e.g., every element of the population gets an equal chance to be selected in the sample. Others include the types of sampling summarized in Table 1.4

Usually, a combination of stratified sampling or cluster sampling and simple random sampling is used. The key advantage of probabilistic sampling is that it usually produces good results, e.g., meaningful statistical inferences on the population based on the sample collected.

Table 1.4 Probabilistic sampling methods for data acquisition

Sampling	How
Simple random sampling	Select points randomly assuming a uniform distribution.
Stratified sampling	Divide the population into strata (e.g., gender, regions) and do simple sampling in each stratum.
Systematic sampling	Index the population, select a first datapoint and then select every kth datapoint after until a certain sample size is achieved.
Cluster sampling	Divide the population into a few groups (e.g., family, genus), then sample each group.
Multistage sampling	If the population cannot be indexed, divide the population into stages (e.g., states or counties), then sample each stage.

Table 1.5 Nonprobabilistic sampling methods

Sampling	How
Convenience sampling	Select datapoints that happen to be most accessible to the researcher.
Purposive (judgement) sampling	Select datapoints that appear most useful for the problem.
Quota sampling	Nonprobabilistic version of stratified sampling.

When no probability distribution is available for the population, the sampling techniques include those in Table 1.5. However, convenience sampling is unlikely to yield a representative sample, so it cannot produce generalizable results.

1.3.2 Training, Testing, and Validation

Most of the data collected in the field of data science are ad-hoc “observational” and so lack proper consideration of the methods used for sampling. Thus data collection is usually done by nonprobabilistic sampling. As pointed out above, the main goal of data science is to build a good predictive model for the response/target variable based on input variables/features in the data. On the other hand, in survey applications, other random sampling schemes play a major role for inference/estimation of population characteristics (e.g., mean or proportion) of interest. Therefore, in statistical applications, it is common to treat the observed big data as the “population” of interest and use probabilistic sampling techniques to select a so-called *training sample* from this “population” to build a model and evaluate the model accuracy on another selected *testing sample*. In this case, one can apply various random sampling strategies (e.g., stratified random sampling) to choose “good” training and testing sets for representative samples. This is a key element to build a better model without the problem of over/under model fitting. Furthermore, one can also choose yet a third dataset for the purpose of “validation” which is commonly used for the purpose of model selection to choose from among competing candidate model solutions. Such a sample is called a *validation sample*.

Since random sampling schemes are used in choosing various samples (training, testing, and validation), the model built and its performance are expected to exhibit random variations/fluctuations, especially for data of small/moderate sample sizes. To obtain a more reliable performance measure, the entire procedure can be repeated a few times to combine these results by taking simple averages or using other advanced ensemble methods. With the decreasing computational cost and increasing power of parallel processing, this may become standard practice in the future.

Finally, random sampling is also used in *cross-validation* (CV) methods used to confirm a model’s performance with random multiple “folds” (subsamples) of the training data against the remaining data points in the sample. This is clearly different from the validation sample because it is separate (probably disjoint) from the training sample. CV has its origin in the *Leave-one-out* procedure used to build and test a model leaving out one observation at a time (clearly infeasible when the data size is moderately large or huge). *Stratified cross-validation* (SCV) is another variation of CV where the data is split into folds, with some stratification, on several subpopulations (e.g., gender or some rare event cases), so that each fold is representative of the whole dataset. Using stratified random sampling for some complex datasets can help to maintain the same proportion of different classes in each fold.

On the other hand, in machine learning and data science, as pointed out before, models are expected to have predictive power independently of the sample data used to obtain them. The population is then assumed to be unknown and possibly infinite and a sample is selected once and for all, then split into disjoint subsets as needed (including training, testing, and/or cross-validation). (Sect. 1.4 has further details.)

1.4 Overview and Scope

The overarching goal of this book is to provide a practical and fairly complete, but not encyclopedic, review of Data Science (DS) through the lens of Dimensionality reduction (DR). It approaches data science from the standpoint of applications and problem solving, while also providing prerequisite unifying theoretical foundations and case studies. We have strived to capture the greatest value for professionals seeking to solve problems in their domains of interest, including representative sample datasets and readily available tools to produce tested solutions of high quality.

The intended target audience consists of professionals in any domain science where data science can help in solving problems and answering questions. Domain science is a fairly vague technical term that refers to a specialized area of human knowledge (the domain) characterized by specific questions about a certain aspect of reality (like *what is motion* in physics, *what are physical objects made of* in chemistry, *what is life* in biology, and so forth). In addition to the well-established sciences (physics, chemistry, biology, and their subdomains), they include just about any area where data can be recorded and analyzed to answer questions concerning the individuals or objects the data is about.

Data science presents a singular approach to problem solving when compared to the more established sciences. Traditional sciences are motivated by pressing problems for people to survive and thrive in the world. That requires a degree of understanding of the phenomena involved that enables predictive power. With the advent of computer science and its abstractions into the information age (as embodied by the internet and web, for example), tools were created that can be used regardless of the specific domain. Once this threshold is crossed, then it is natural to group and develop methods and platforms to do this kind of generic science, hence data science. It is a natural next step from mathematics and statistics to *synergistically combine them with the powerful computational tools developed* *in computer science to create a new science that is more than the sum of the parts*. The basic background concepts required from these building blocks are summarized in Chap. 11. We have strived to leave our niche hats at the door and present a integrative and synergistic approach that captures the best of the three worlds. That is the pervasive thread that readers will discover through examples and methods throughout the book.

1.4.1 Prerequisites and Layout

The content of the book is presented using the same template approach in every section. Sections begin with an intuitive example of a problem to be solved by the concepts being introduced in that section. A professional with an undergraduate degree in any quantitative science should be able to follow this part. We have assumed that the reader is familiar with basic undergraduate mathematics (multivariable calculus and linear algebra), including matrix algebra. Likewise, we have assumed that the reader is familiar with the basic concepts in statistics and probability, including sample spaces, probability distributions, random variables, and the main results associated with them. Nevertheless, a refresher summary is given in the Appendices in Sects. 11.1 and 11.2. Since readers may be less familiar with basic computational background, a summary is likewise made of basic concepts in computer science as well in Sects. 11.3 and 11.4

These motivating examples in a section are then followed by precise definitions of the technical concepts and presentation of the results in general situations. That requires a degree of abstraction that can be followed by re-interpreting the general terms like in the original example(s). Finally, each section closes with solutions to the original problem afforded by these techniques, perhaps in various ways to compare and contrast advantages and disadvantages of the various DR techniques based on quantitative and qualitative assessments of the solution(s) in the real world.

1.4.2 Data Science Methodology

Solving a data science problem typically requires several steps:

1. *Define the problem precisely.*
   A good definition requires a clear distinction between the *WHAT* are one is trying to do, versus *HOW* one is going to actually find a solution. The definition should first make sense in the real world to common people who know nothing about DS but have a *goal* to achieve. One can then say this definition is more like a *business* definition. It should answer fundamental leading questions such as: *WHAT needs to be changed? WHAT is the desired outcome?* Figure 1.9 illustrates the point. An analogy to taking a trip is most appropriate. It is common to hear people say,

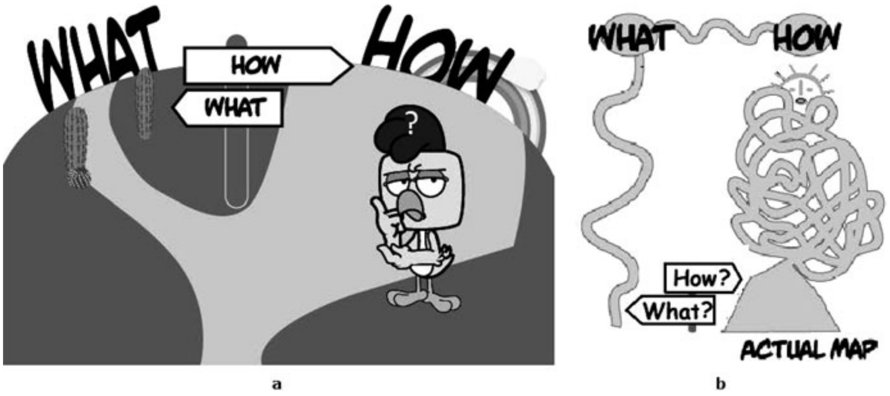

Fig. 1.9 (**a**) Defining a data science problem appears easy, but (**b**) is a difficult task, because it has to be distinguished from HOW to solve the problem. WHAT goal to achieve is a destination that has to be decided first because it determines practically everything else and makes decisions easier

let us just get the data first, then we will see what we can do. That amounts to saying, let us get in the car and start driving, we will decide on a *destination* later. A moment's reflection will make it obvious that this is nonsense really. Knowing where one wants to go will lead her some place with some nonzero probability (even if it is the wrong place!). Not knowing where to go should be expected to prove disastrous. An objectively defined and clear *destination* is paramount, possibly including a criterion of quality for a solution to be good enough and acceptable.

Examples of proper ways to define a problem have been given in earlier chapters and more examples are shown in Sect. 11.4. The easiest way is to identify the problem as one of the standard problems in Data Science (Classification? Prediction? Clustering?) or reformulate the problem into a related problem of this kind that might help.

2. *Name a destination.*

   Naming a destination determines virtually everything else (HOW to get there, what to pack to wear, and so forth). In Data Science, it will help decide what kind of data needs to be gathered or obtained, what kind of solutions could be tried. There are no preset recipes (contrary to traditional sciences), just a set of tools to try. What is amazing is that this approach usually lands one with some good enough (although perhaps not perfect) solution.

3. *Select an evaluation metric.*

   As lord Thompson (the inventor of the Celsius temperature scale) said, one cannot really know something unless one can quantify it. The next step is how to decide if a potential solution is good or not. A clear definition of the problem will make it fairly obvious (they are described further below in this section). Since the problem definition spelling a business goal to achieve was clearly defined in the first step, the definition should provide a criterion whether the solution is good

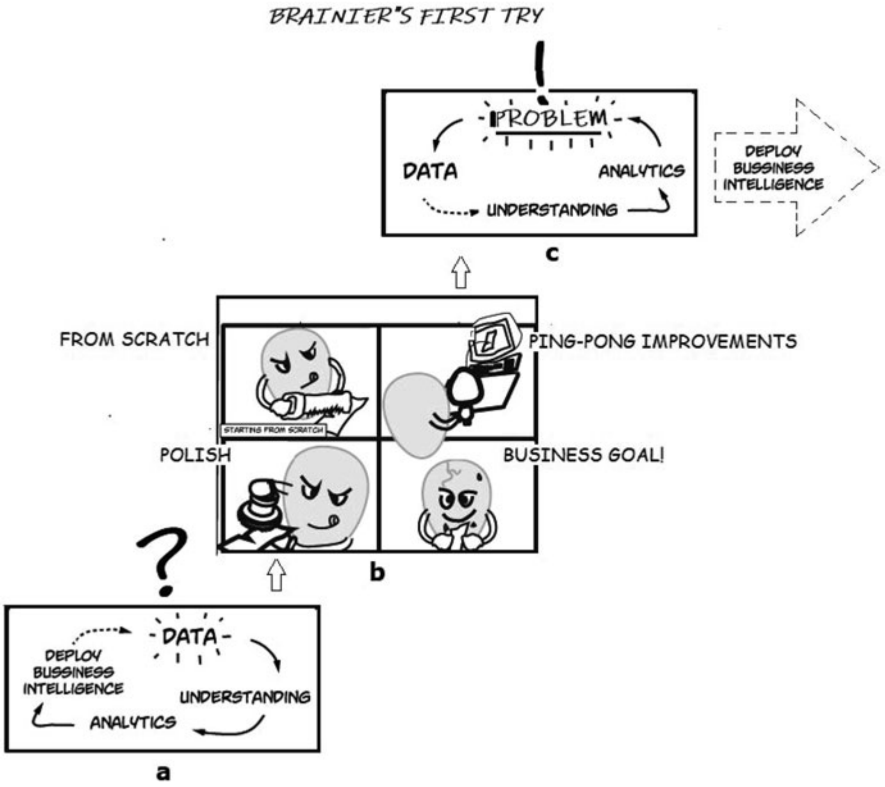

Fig. 1.10 Data science appears to have the (**a**) all-essential data as the core concept. However, upon reflection, (**b**) it is really about problem identification and problem solving because (**c**) the problem being solved dictates not only what and how much data is appropriate but also whether we have gone enough around the loop to be able to deploy a solution for it that is viable in the real world

enough or not (not whether it is optimal). If not, one may need to try to go through this loop again, as shown in Fig. 1.10.

4. *Business Intelligence*
   Finally, an acceptable solution requires re-insertion and implementation back in the real world. That will be the real test whether the solution really makes sense, i.e., whether the problem has been solved or not. If not, one needs to go through the process all over again, with appropriate refinements for a better chance.

1.4.3 Scope of the Book

This book is not meant to be an encyclopedia of data science that includes every method or technique known to humans. Data science is too young to even tell where it is headed to allow that. Deeper questions concerning related matters are touched upon in an exploratory manner in Chap. 10, but only to give the reader a take-home epiphany as to what data science and/or dimensionality reduction mean back in the real world. Humans usually refer to that as *experience* and *knowledge*. They are simply heuristics, because they are just rules of thumb that can easily fail with some practical problem faced by a professional in his/her own domain. With this caveat, readers can use it as a guide to choose and adapt the methods presented here to tackle their own personal challenges.

Reference

1. Broad, C. D. (1978). *Kant: An introduction*. Cambridge: Cambridge University Press.

Chapter 2
Solutions to Data Science Problems

Deepak Venugopal, Lih-Yuan Deng, and Max Garzon

Abstract This chapter presents a review of statistical and machine learning models to tackle data science problems, arguably the most popular approaches. Both supervised and unsupervised algorithms are described along with practical considerations when using these methods. Empirical results on exemplar datasets are also presented where applicable to illustrate the application of these methods to real-world problems.

2.1 Conventional Statistical Solutions

To study whether a feature can help in solving DS problems, most statistical models formulate the problems into probabilities of mass/density functions in various perspectives. By modeling the probabilities of getting the situations of interest, practitioners can select important features, describe the relationship among variables, make an inference, and classify, predict, or cluster future events.

This section presents a summary of various classical statistical solutions that can be used to solve a problem by building a model or reduce the dimension of the feature space.

2.1.1 Linear Multiple Regression Model: Continuous Response

A number of classical statistical methods can be used in many cases to provide a better foundation for finding solution models and improve their predictions.

D. Venugopal (✉) · M. Garzon
Computer Science, The University of Memphis, Memphis, TN, USA
e-mail: dvngopal@memphis.edu; mgarzon@memphis.edu

L.-Y. Deng
Mathematical Sciences, The University of Memphis, Memphis, TN, USA
e-mail: lihdeng@memphis.edu

M. Garzon et al. (eds.), *Dimensionality Reduction in Data Science*,
https://doi.org/10.1007/978-3-031-05371-9_2

In the notation of Chap. 1, given n data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, where $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})$ is a covariate vector for the ith observation of predictor features and y_i is the target response, it is common to represent the dataset in matrix form

$$\mathbf{X} = \begin{bmatrix} x_{11} \ldots x_{1p} \\ x_{21} \ldots x_{2p} \\ . \\ . \\ x_{n1} \ldots x_{np} \end{bmatrix} = [X_1\, X_2 \cdots X_p] \quad \mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{bmatrix},$$

where the covariate matrix can be viewed as n row vectors $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ of dimension p, or p column vectors of dimension n $(X_1, X_2, \ldots, X_p)$ and $\mathbf{Y}$ is the response column vector of dimension n, $(y_1, y_2, \ldots, y_n)'$.

If the response column vector $\mathbf{Y}$ is continuous, it is common to assume at first that a solution model is given by

$$\mathbf{Y} = f(\mathbf{X}) + \boldsymbol{\epsilon} = f(X_1, X_2, \ldots, X_p) + \boldsymbol{\epsilon},$$

where $f(\cdot)$ is a function to be estimated and $\boldsymbol{\epsilon}$ is a random variable (Sect. 11.1 gives some probabilistic and statistical background) for the error in the estimation. Clearly, the general linear model is a special case with a linear function $f(\cdot)$

$$\mathbf{Y} = \sum_{j=0}^{p} \beta_j X_j + \boldsymbol{\epsilon}$$

and $X_0 = 1$.

When the dimensionality of the data (the number of columns p of $\mathbf{X}$) is large, one may consider some variable selection procedure to find a reduced model with fewer significant variables, or one may use leading principal dimensions from PCA (principal component analysis, described in Sect. 4.1). Popular *feature selection* and *subset selection* methods are:

1. Best subset
2. Stepwise selection: forward and backwards

A common selection criterion for assessing quality of fit of a model is the classical R^2 or the root mean-squared error (RMSE), given by

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

and

$$\text{RMSE} = \sqrt{\text{MSE}}, \quad \text{MSE} = \frac{\text{SSE}}{n-p}.$$

Clearly, R^2 or RMSE for assessing quality of fit is very suitable because they tend to choose models with more predictors.

Hence, a better quality criterion is needed that takes into account the size of the model since small models are preferred that still fit well, even if one has to sacrifice a small amount of "goodness-of-fit" for a smaller model. Three more criteria are available, namely AIC, BIC, and Adjusted R^2.

2.1.1.1 Akaike Information Criterion (AIC)

Another measure of the quality of a model is the AIC, defined as

$$\text{AIC} = n \log\left(\frac{\text{RSS}}{n}\right) + 2p,$$

where $\text{RSS} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ and $\hat{\boldsymbol{\beta}}$ and $\hat{\sigma}^2$ and p is the number of β parameters in the model. The term $2p$ is called a penalty component of the AIC because it is large when p is large, while the aim is to find a small AIC (small RSSD and p).

Thus, a good model will strike a good balance between the conflicting goals of fitting well and using a small number of parameters. The smaller the AIC, the better.

2.1.1.2 Bayesian Information Criterion (BIC)

The Bayesian Information Criterion BIC is similar to the AIC but has a larger penalty:

$$\text{BIC} = n \log\left(\frac{\text{RSS}}{n}\right) + \log(n)p.$$

BIC also quantifies the trade-off between a model that fits well and the number of model parameters, although, for a reasonably large sample size, it generally picks a smaller model than AIC. As with AIC, the model with the smallest BIC is selected.

The penalty for BIC is $\log(n)p$ rather than the AIC's penalty of $2p$. Therefore, for any dataset where $\log(n) > 2$, the BIC penalty will be larger than the AIC penalty, and thus BIC will be likely to lead to a smaller model.

2.1.1.3 Adjusted *R*-Squared

It is common to make an adjustment to the popular R^2 as

$$R_a^2 = 1 - \frac{\text{SSE}/(n-p)}{\text{SST}/(n-1)} = 1 - \left(\frac{n-1}{n-p}\right)(1 - R^2).$$

Unlike R^2 that can never become smaller with added predictors, this adjusted R^2 effectively penalizes for additional predictors and can decrease with added predictors. As with R^2, larger is still better for the adjusted R^2.

2.1.2 Logistic Regression: Categorical Response

If the response column vector $\mathbf{Y}$ is binary vector of 0s and 1s, it is common to consider a generalized linear model of the form

$$g(E(\mathbf{Y}|\mathbf{X})) = g(\boldsymbol{\mu}) = f(\mathbf{X}) + \boldsymbol{\epsilon} = f(X_1, X_2, \ldots, X_p) + \boldsymbol{\epsilon},$$

where $g(\cdot)$ is a link function to be chosen and $f(\cdot)$ is a response function. Several common choices can be considered:

1. **Logistic regression** (LR)
   It is a very popular model for binary response with the logit link function, $g(\boldsymbol{\mu}) = \ln[\boldsymbol{\mu}/\,(1-\boldsymbol{\mu})]$, and

$$g(\boldsymbol{\mu}) = \ln\left[\frac{\boldsymbol{\mu}}{1-\boldsymbol{\mu}}\right] = \sum_{j=0}^{p} \beta_j X_j + \boldsymbol{\epsilon}.$$

2. **Generalized additive model** (GAM) [16]
   The GAM can be considered an extension of LR without assuming $f(\cdot)$ is a linear response function since it replaces $\beta_j X_j$ with a general smooth function $s_j(X_j)$

$$g(\boldsymbol{\mu}) = \sum_{j=0}^{p} s_j(X_j) + \boldsymbol{\epsilon}.$$

   Typically, GAM is generally used for nonlinear regression problems with continuous response, but it can also be used to build a binary response classifier. Hastie [16] first proposed a generalized additive model (GAM). The GAM can be adapted to different situations, as generalized linear regression can be used when different link functions are utilized. While there is no limit on the choice of the link function $g(\boldsymbol{\mu})$, the "logit" link is commonly used for a binary classifier.

2.1.3 Variable Selection and Model Building

High-dimensional statistical problems are quite common in various fields of science, and variable selection is important in statistical learning and scientific discovery.

The standard procedure of best subset selection methods is based on AIC or BIC as variable selection criteria are suitable only for a moderate number of input variables. For high-dimensional data, these procedures can be very computationally expensive. Procedures using penalized likelihood methods have been successfully developed recently to deal with high-dimensional datasets. In addition to variable selection, these methods can also be used in estimating their effect in high-dimensional statistical inference. (Further discussion is given in Chap. 8.)

Model building to solve a problem is a process of finding the appropriate relationship between response and input variables. Such a relationship could be either a simple linear relationship or a complicated nonlinear relationship. In addition to the first-order linear model, it is possible to consider other models such as polynomial regression models or generalized additive models (GAMs). One of the major problems for building models for high-dimensional data is the problem of multi-collinearity among many input variables. Furthermore, building GAM could be infeasible for a large number of highly correlated input variables. In such cases, one can consider dimension reduction techniques such as principal component analysis (PCA), to be discussed in Chap. 4.

2.1.4 Generalized Linear Model (GLM)

A popular model that includes both continuous and discrete models is the generalized linear model (GLM). The popular logistic regression model is a special case. There are three components in the GLM, namely, a random component, a systematic component, and a link function component.

Random Component
The random component of a GLM characterizes the distribution of the response variable $y_i, i = 1, 2, \ldots, n$ with a general form of the exponential family of distributions (described in Sect. 11.1 for probability background). It is given by

$$f(y_i; \theta_i) = a(\theta_i) b(y_i) \exp[y_i Q(\theta_i)],$$

where $a(\cdot)$ and $Q(\cdot)$ are functions of θ_i and $b(\cdot)$ is a function of y_i.

Systematic Component
The systematic component of a GLM specifies a linear relationship between a transformed parameter (via a link function below) and its input variables. Specifically, for each y_i and p predictor variables x_{ij}, $j = 1, \ldots, p$, the parameter η_i is a linear function of x_{ij} of the form

$$\eta_i = \sum_j \beta_j x_{ij}, \quad i = 1, \ldots, n .$$

Link Function Component
The link function component of a GLM specifies the transformation, say $g\cdot$ (monotone, differentiable), on the mean response of Y_i, $\mu_i = E(Y_i)$, so that

$$\eta_i = g(\mu_i)\,,$$

that is,

$$g(\mu_i) = \sum_j \beta_j x_{ij}, \quad i = 1, \ldots, n\,.$$

Special cases of the link function are illustrated next.

Example 2.1 (Linear Model/Regression Model) The usual linear model/regression model is included in the GLM formulation with

$$y_i = \sum_j \beta_j x_{ij} + \epsilon_i,$$

where ϵ_i's are i.i.d. $\epsilon_i \sim N(0, \sigma^2)$. The link function $g(\mu) = \mu$ could be the identity link function.

In particular, the canonical link is a link function that transforms the mean μ to the natural parameter

$$g(\mu_i) = Q(\theta_i) = \sum_j \beta_j x_{ij}\,.$$

□

Example 2.2 (Logit Models for Binary Data) For binary response data $Y \sim B(1, P)$, its probability density function (pdf) can be written as

$$f(y; P) = P^y(1-P)^{1-y} = (1-P)\exp\left(y \log \frac{P}{1-P}\right),$$

where:

- $a(P) = 1 - P, b(y) = 1, Q(P) = \log \frac{P}{1-P}$.
- The natural parameter is $\log \frac{P}{1-P}$ log odds or logit of P.
- GLM is also called the *logit model.* □

Example 2.3 (Poisson Loglinear Model) When the response Y represents data counts, it is common to assume a Poisson distribution (described in Sect. 11.1) in the Poisson Loglinear Model $Y \sim Poisson(\mu)$, with probability distribution given by

$$f(y; \mu) = \frac{e^{-\mu}\mu^y}{y!} = e^{-\mu}(1/y!)\exp(y \log \mu), y = 0, 1, 2, \ldots,$$

Table 2.1 Generalized Linear Models (GLMs)

Random	Link	Systematic	Model
Normal	Identity	Continuous	Regression
Binomial	Logit	Mixed	Logistic regression
Poisson	Log	Mixed	Loglinear

Table 2.2 Dataset with three features to predict whether a person likes a restaurant or not

Price	Fast	On-campus	Likes
High	Yes	Yes	No
Low	Yes	Yes	Yes
High	Yes	No	No
High	No	Yes	No
Low	Yes	No	Yes

where:

- $a(\mu) = e^{-\mu}$, $b(y) = (1/y!)$, $Q(\mu) = \log \mu$.
- The natural parameter is $\log \mu$, and the canonical link function is $\eta = \log \mu$.

$$\log \mu_i = \sum_j \beta_j x_{ij}, \quad i = 1, \ldots, n .$$

□

A summary of the various kinds of GLMs is shown in Table 2.1.

2.1.5 Decision Trees

Decision trees are models that use a tree structure to solve a classification problem. Specifically, the internal nodes in the tree represent questions about the values of features in the data, and the leaf nodes represent classes. Each feature is branched (or conditioned) on different possible values for that feature. Given a dataset as in Table 2.2, an example decision tree is shown in Fig. 2.1. To classify a data instance, starting with the root node, each node is a decision point where one of the branches is selected based on the feature value in that instance. The leaf nodes in the decision tree (nodes with no children) correspond to class labels. For instance, for the example tree shown in Fig. 2.1, to classify whether a patron likes a restaurant or not, if for a specific data instance, Price = "high," Fast = "no," and On-campus = "no," the decision tree will make decisions to follow branches corresponding to the feature values from root to leaf and output a class Likes = "No".

Decision trees are highly versatile models since they can implement any Boolean function. Specifically, given the truth table for any Boolean function, each path in the decision tree can encode one row in the truth table of the Boolean function. However, the size of a decision tree can be large when learning a complex classifier. That is, a large number of nodes may be required to express the classifier function.

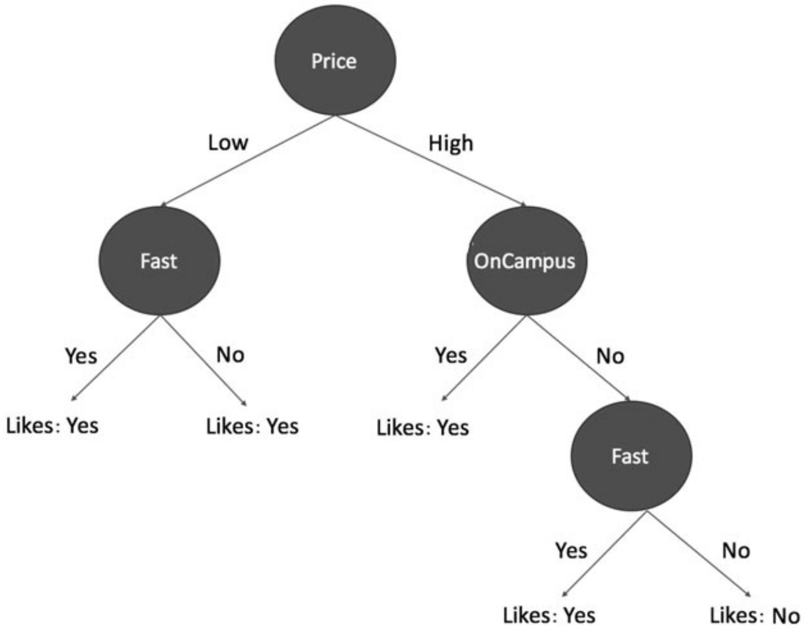

Fig. 2.1 A decision tree (DT) classifier for the restaurant dataset in Table 2.2

The main goal in learning a decision tree from data is to learn a *compact* tree that has as few nodes as possible. Several heuristics used to construct such trees are described next.

Decision tree learning (sometimes called *induction*) incrementally grows the decision tree given a training dataset. Typically, a greedy approach is used for tree induction, where in each step, a feature is selected as a node in the decision tree. The selected feature essentially splits the data into different branches based on the possible values that the feature can take on. Therefore, in each step, the main computational task is to select a "good" feature, i.e., a feature that acts as the best classifier for the data. For example, among the three features in for dataset given in Table 2.2, if the feature Price always determines whether one likes a restaurant or not, then this feature encodes the most useful information required for classifying the data and should be considered ahead of the other features while learning the decision tree. To select the best feature in each step of decision tree learning, a splitting criterion is used to score the utility of each feature.

While there are several different possible heuristics that can be used as a splitting criterion, one of the most widely used decision tree learning algorithms, ID3 [22], and its variants use conditional entropy in the splitting criteria. Entropy is a measure of uncertainty or randomness in a sample, to be defined precisely in Sect. 6.1. For example, given a dataset where all the instances belong to the same class, the entropy of this dataset is 0 since there is no randomness. At the other extreme, consider two possible classes where half of the dataset belongs to one class and the other half belongs to another class; in this case, the entropy is maximum. The conditional entropy for a feature is a measure of randomness with respect to the class label after splitting or conditioning the data according to each value of the feature. Thus, in the example in Table 2.2, the conditional entropy for Price is 0 since for each value of Price, the data samples have the same class label. On the other hand, for the feature On-Campus, for each of its values, the data samples are equally distributed among the two class labels. Therefore, the conditional entropy for the On-Campus feature is large. In other words, if $H(D)$ stands for the entropy of data D given c classes in D, the entropy of D is given by

$$H(D) = \sum_{i=1}^{c} -P_i \ \log_2 P_i \ ,$$

where P_i is the proportion of D belonging to class i. If the splitting/conditioning of D is performed using feature $\mathbf{x}_i$, the conditional entropy after this split is given by

$$H(\, D \,|\, \mathbf{x}_i \,) = \sum_{v \in Values(\mathbf{x}_i)} \frac{|D_v|}{|D|} H(D_v),$$

where $Values(\mathbf{x}_i)$ is the set of possible values for feature $\mathbf{x}_i$, $D_v \subseteq D$ where feature $\mathbf{x}_i$ has the value v. Thus, the conditional entropy for $\mathbf{x}_i$ is computed by computing the entropy values for subsets of the data specific to each value of $\mathbf{x}_i$ and then computing a weighted average of these values. It is easy to see that if a feature splits the data such that for each value of the feature, the class labels are uniform (for example, the feature Price in the aforementioned example), then the conditional entropy is equal to 0. Thus, a feature with smaller conditional entropy can classify the data more effectively than a feature with larger conditional entropy.

The decision tree algorithm proceeds as follows. In each step, the feature with the smallest conditional entropy is selected. The selected feature is then used to partition the data D, into splits $D_1 \ldots D_k$, where each split of the data corresponds to a specific value of that feature. This process is repeated recursively for $D_1 \ldots D_k$.

Some practical considerations of decision tree learning include:

- To compute the conditional entropy, the features are assumed to be discrete. Continuous features are thresholded into discrete values to approximate the conditional entropy.
- Entropy computations are computationally expensive. Therefore, a common problem in decision trees is that scaling them to big data is expensive. A simple solution to this is to compute the conditional entropy values from a smaller sample of the data.
- A significant advantage of decision trees is that they are perhaps among the most interpretable machine learning models. That is, a human user can understand the output of the decision tree, and for each classification that the decision tree

makes, a trace of the sequence of decisions that led to that classification can easily serve as a causal chain of reasoning for the label assignment. Therefore, decision trees are among the leading models in applications where interpretability is important such as healthcare [5] and business analytics [19].

2.1.6 Bayesian Learning

The Bayes theorem is among the most fundamental theorems in probability theory (Sect. 11.1 defines background concepts in probability and statistics and gives a precise statement). It can be applied for classification by assigning the most probable class value given a set of features. Specifically,

$$C' = \arg\max_{C} P(C|\mathbf{x}),$$

where $P(C|\mathbf{x})$ is the probability of instance $\mathbf{x}$ having class label C. Using Bayes theorem, one can rewrite the above results in the following classifier:

$$C' = \arg\max_{C} P(\mathbf{x}|C)\frac{P(C)}{P(\mathbf{x})},$$

where $P(C)$ is the probability of the class C and $P(\mathbf{x}|C)$ is the probability of observing features $\mathbf{x}$ given that the class is C. The denominator on the right-hand side of the above equation is immaterial to compute the maximum, so the classifier can be re-written as

$$C' = \arg\max_{C} P(\mathbf{x}|C)P(C).$$

Given training data, estimating $P(C)$ is quite straightforward. $P(C)$ is estimated by computing the percentage of instances in the data with class value equal to C. Specifically, if the total number of training instances is n and among these, the number of instances with class label C is n_c, then the estimated probability is $P(C) = \frac{n_c}{n}$. On the other hand, estimating $P(\mathbf{x}|C)$ is much harder. This probability is the percentage of instances that have feature values exactly equal to $\mathbf{x}$. To compute this, the data needs to contain sufficient instances of any configuration of feature values, which may not happen in practice. For example, if there are 10 binary features (0/1), there are $2^{10} = 1024$ possible configurations of feature values given a class label, and the training data would need to contain sufficient instances to estimate all 2^{10} probabilities. As the number of features increase, the number of probabilities needed to find the classifier grows exponentially.

To scale up Bayesian learning, the *Naive Bayes* classifier makes a simplifying assumption that the features are conditionally independent given the class. This means that the joint probability (defined in Sect. 11.1) over the features given a class can be expressed as a product of probabilities over each feature as follows:

$$P(C|\mathbf{x}) = \frac{P(\mathbf{x}|C)P(C)}{P(x)} \propto P(\mathbf{x}|C)P(C) \tag{2.1}$$

$$\propto P(x_1, x_2, \ldots, x_K|C)P(C) = \prod_{i=1}^{K} P(x_i|C)P(C)\,,$$

where $P(x_i|C)$ is the probability of a single feature value x_i given class C. In this case, considering the previous example, 10 features only require 10 probabilities corresponding to each class instead of 2^{10} probabilities, an exponential reduction that allows the classifier to scale up to a large number of features.

Learning the Naive Bayes classifier is straightforward where the probability of a feature value given the class is simply the proportion of training examples of the class where the feature takes that specific value. For example, if the classification task is to identify spam/nonspam emails and one of the features is the word "free," where "free" occurs in 10% of the nonspam emails and 50% of the spam emails in our training data, independently of other features (based on an assumption of conditional independence), then $P(\text{“}free\text{”}|Spam) = 0.5$ and $P(\text{“}free\text{”}|NonSpam) = 0.1$. In general, if the total number of training instances of class C is n_c and among these, feature x_i has value v in n_v instances, the estimated conditional probability is given by

$$P(x_i = v|C) = \frac{n_v}{n_c}. \tag{2.2}$$

Some practical considerations in using the Naive Bayes classifier include:

- If a feature x_i is continuous, then $P(x_i = v|C)$ can no longer be estimated by counting the proportion of training examples with $x_i = v$. In this case, a variant of Naive Bayes called Gaussian Naive Bayes is used to estimate the conditional probability. Specifically, each continuous feature has a conditional probability that is assumed to be Gaussian, and the Gaussian parameters are learned from the data.
- A probability estimate of 0 for the conditional probability corresponding to a single feature makes the entire product of conditional probabilities in Eq. (2.1) equal to 0. This happens when a particular feature value may be absent in the training data for a given class. A widely used approach to correct zero probabilities is called the *Laplace correction*. The idea is to assume that the feature value occurred a constant number of times in the training data (even though it may not have). The Laplace corrected value in Eq. (2.2) is equal to $\frac{n_v+r}{n_c+r|x_i|}$, where r is a constant and $|x_i|$ is the number of values feature x_i can take on.
- The Naive Bayes classifier makes the assumption of conditional independence even though this assumption is generally false, i.e., in most cases features are not

conditionally independent given the class. However, the Naive Bayes classifier works quite well and in some cases (such as in text classification), where it achieves highly competitive accuracy compared to much more sophisticated models. (A formal analysis on the reasons for its good classification performance despite its assumptions is provided in [8].)

- The Naive Bayes model can be easily interpreted since the conditional probabilities of features can be compared with each other to understand the relative importance of features in the model.

2.2 Machine Learning Solutions: Supervised

Machine learning (ML) methods are arguably one of the most widely used approaches to solve data science problems. A machine learning model can be viewed as a program that improves itself with experience [20]. The next two sections provide a summary review of most commonly used machine learning algorithms. The descriptions focus on practical aspects of these solutions related to solving data science problems, with appropriate references to other sources for a more comprehensive treatment of the technique(s).

ML algorithms can be roughly classified into two major categories, *supervised* and *unsupervised*. Supervised algorithms require that the data contain a designated target (response) feature with the expected answers to the instances of the problem at hand, in addition to a given set of predictor features in a data point, whereas unsupervised methods just require the predictor feature vectors describing the data point.

This section gives an overview of supervised learning algorithms, and the next is concerned with unsupervised learning methods. To ground the abstract concepts, the classification problem for the problem [**CharRC**] of handwritten character recognition and the MNIST dataset (Sect. 11.3 has more details) is used to illustrate the methods throughout this section.

Example 2.4 The classification problem of handwritten digit recognition [**CharRC**] (described in Sects. 1.1 and 11.4) calls for a category label of digits 0, 1, . . . , 9 for a given input image (presumably a handwritten digit), as illustrated in Fig. 2.2. Each image is a 28x28 grayscale image of a single digit from actual handwritten ZIP codes. Due to personal variations in writing, naturally the same digit can appear differently in different instances, and while humans can perceive these variations quite easily, coming up with an automated program to do the same is quite challenging. To improve itself automatically and learn, a machine learning algorithm must have a performance measure to decide how to improve. A good choice here can be the accuracy with which the program identifies the handwritten digits over a dataset. Much like how humans become better at a task with practice, to obtain improvement in performance, some training helps a program gain experience in solving the task. This training is provided to the program through data. The key

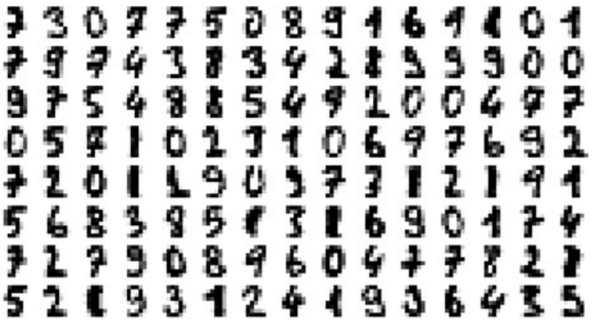
Fig. 2.2 A sample of images from the MNIST digits dataset for the problem of handwritten character recognition [**CharRC**]

requirement of learning (as opposed to memorization) is that the program not only performs well on data that it has been trained on, but it should also work on new data it has never seen before. In other words, given an unseen image of a handwritten digit, the program should be able to successfully identify the digit, even though that specific image is new to the program. Thus, the program does not simply memorize the data but learns some deeper patterns in the data to make useful inferences about new data. □

These kinds of algorithms are examples of what is known as *supervised learning*. Specifically, in supervised learning, the algorithm is trained with data where each data point includes a label for a specific discrete class giving the correct answer in the problem. In general, the goal of any supervised learning algorithm is to refine successive possible solutions based on data instances and their labels in such a way that a new data instance that it has previously not seen during training is very likely to elicit a correct classification.

Formally, supervised learning is defined as follows. Given n data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, where $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip}), y_i$ are vectors representing features for the ith observation and its corresponding class, the supervised learning algorithm is to return a function $f(\cdot)$ defined on the population for the problem so that $y_i = f(\mathbf{x}_i)$.

Several algorithms for supervised learning are reviewed in the remainder of this section.

2.2.1 k-Nearest Neighbors (kNN)

Example 2.5 The key idea in the k-nearest neighbors classifier is illustrated in Fig. 2.3. If each image in the MNIST dataset can be represented as a single point in a Euclidean space, an algorithm can classify a new image by simply using the majority class among all its neighbors within certain radius in this space. This idea is simple and intuitively appealing, but will it produce good results as a learning algorithm? How can the neighborhood be chosen? □

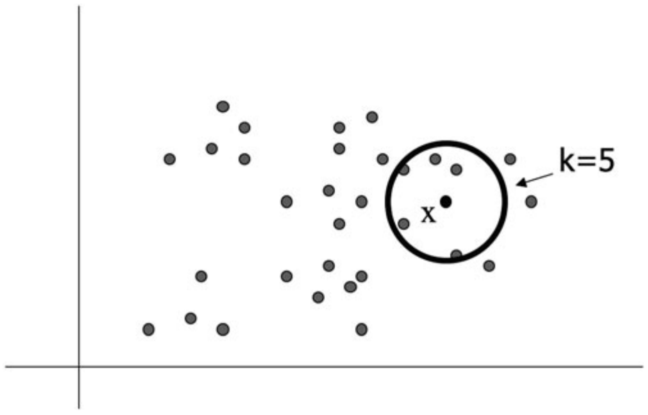

Fig. 2.3 The key idea in the k-nearest neighbors algorithm on the MNIST dataset. Data points belong to two classes (e.g., digits in the MNIST data), and X represents a new point to be classified based on the majority class of $k = 5$ of its nearest neighbors

The **kNN** (k-nearest neighbors) algorithm is a type of *instance-based* learning. In contrast to other solutions such as neural networks or decision trees, instance-based learning methods do not learn a function for classification from training data. Instead, they simply store the training data instances. When asked to classify a new instance, they retrieve similar instances in the training data to help classify this new instance. Thus, the function f is not defined in abstract, but in the context of every instance, making this approach more flexible.

The **kNN** algorithm assumes that each data instance is a point in a pD metric space, where P is the number of features in the data points. Given a new point $\mathbf{x}_j$ to classify, it computes k-nearest neighbors to this point among all the points in the training data and classifies the new instances as the same class as the majority of the k neighbors. Typically, the neighbors are computed using the ordinary Euclidean distance ℓ_2 (defined in Sect. 1.2), although other distance metrics are not uncommon. Naturally, an odd number is chosen as the value of k to guarantee a majority among the class labels in the neighborhood. **kNN** assigns $\mathbf{x}_j$ the same class label as the one that occurs in majority of the k-nearest neighbors in its neighborhood of $\mathbf{x}_k$.

kNN implicitly assumes that all neighbors are equally important. In a variant of k-nearest neighbors, called *distance-weighted* k-nearest neighbors, neighbors to a data point are weighted inversely by their distance to that point, thus giving greater importance to closer neighbors. Further, the Euclidean distance computation is also affected by the scale of each feature. For example, if one of the features is the annual income and the other is credit card rating, naturally the scale of annual income is much larger than credit card rating and dominates the distance computation. Therefore, typically, the features are standardized using Z-scores (defined in Sect. 11.1) before applying the **kNN** classifier (each feature value is subtracted with the mean value for that feature and divided by the standard deviation value for that feature).

Table 2.3 Results for MNIST varying the neighborhood in **kNN**

k	Precision	Recall	F1-score
1	0.982	0.982	0.982
3	0.986	0.986	0.986
5	0.993	0.993	0.993

Example 2.6 (k-Nearest Neighbors for MNIST) In the **kNN** solution to the handwritten digits classification problem [**CharRC**], each pixel in the image represents a single dimension in Euclidean space for a **kNN** classifier. Table 2.3 shows the precision, recall, and F1-score (defined precisely below in Sect. 2.4) on a test dataset (chosen to be 25% of the data) for varying k when **kNN** is applied to this problem. As seen here, increasing k can smooth some of the local irregularities in the labels and improves performance. □

Important practical considerations in applying **kNN** include:

- **kNN** is a computationally expensive classifier since it needs to store/index all the training data. In contrast, classifiers such as decision trees or neural networks learn a model from the training data and do not need to use it again after trained for classification. Thus, although training a classifier is time-consuming, classifying a new instance is easy after training. On the other hand, **kNN** has all the overhead in classifying a every new instance since they need to search over the training data to find the neighborhood of that instance. Finding neighborhoods for a point in higher-dimensional space (when the number of features is large) is a search problem that becomes exponentially harder with the size of the data. Typically, specialized data structures (such as KD trees) are used to quickly find neighborhoods for a data point. Even with these data structures, however, the scalability of **kNN** is quite limited when compared to other methods.
- In most applications, it is generally the case that some features will be more relevant than others. As the number of irrelevant features increases, the distances computed in **kNN** become dominated by these features. This fact is generally termed the curse of dimensionality (discussed further in Sect. 10.3). While most machine learning methods perform poorly in the presence of irrelevant features, **kNN** are sensitive to the curse of dimensionality and typically perform much worse than other approaches with a large number of features.
- Since **kNN** assumes that data points live in a common Euclidean space, all the features in the data must be real-valued. While it is possible to have features that are discrete/categorical, such features need to be embedded in a Euclidean space using an additional step to enable the k-nearest neighbors algorithm.

2.2.2 Ensemble Methods

Ensemble methods refer to a general approach where various classifiers are combined together for better performance.

Example 2.7 A natural approach in medical diagnosis is to use several expert opinions rather than relying on a single doctor's opinion. An illustration of ensemble methods is shown in Fig. 2.4. Although each classifier separately may make a certain degree of error in the classification, the combined classifier that uses the output of all classifiers can correct these errors based on the collective outputs.

Two widely used approaches to combine classifiers include *Bagging* (*Bootstrap Aggregation*) [4] and *Boosting* [25]. These methods try to improve generalization performance of a classifier. Specifically, the ability of a classifier to generalize to data points not used in training depends upon both the bias and variance of the classifier. Intuitively, *bias* refers to the ability of the classifier to fit the training data, i.e., a low-bias classifier gives accurate results on the training data. *Variance* refers to the flexibility of a classifier, i.e., a high-variance classifier changes significantly even with a slight change in the training data. The *bias–variance trade-off* in machine learning states that a classifier with low bias usually has high variance and vice versa. Ensemble models reduce generalization error by reducing the bias and/or variance in the classifier.

In the basic Bagging approach, a (sub)sample of the given training data is drawn (with replacement), and a classifier f_i is learned using this sample. Repeating the process k times produces k classifiers $f_1 \ldots f_k$. Given a new data point $\mathbf{x}$ to classify, these classifiers produce k labels $f_1(\mathbf{x}) \ldots f_k(\mathbf{x})$, and the final class for x is decided based on the majority.

Bagging does not have an effect on the bias of a classifier. To reduce bias, Boosting methods are used on a classifier. Boosting methods combine multiple *weak* classifiers into a combined classifier. Weak classifiers are those that may

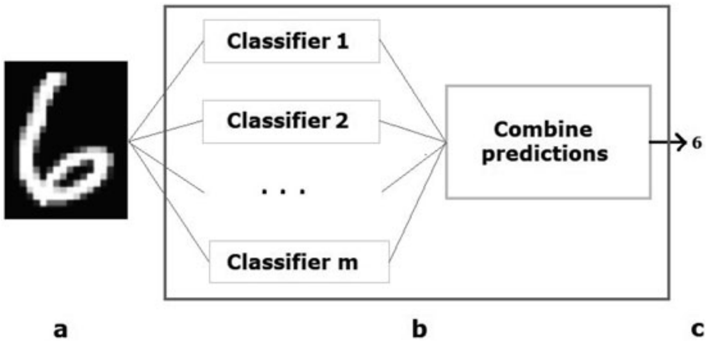

Fig. 2.4 Ensemble classifiers combine the labels from several other classifiers (e.g., by majority) on **a** a common given input (*left*) to produce a label **c** (*far right*) for the same input

not fit the training data completely. That is, they have high bias and are simple classifiers. The idea in boosting is to add several such simple classifiers to make a combined classifier that has low bias. The most popular version of Boosting is the AdaBoosting algorithm, a binary classifier with outputs of $+/-$. AdaBoosting is an additive model that sequentially adds classifiers so that the classifier added in iteration k is likely to correct the errors made by the classifier on the training data in iteration $k-1$. To do this, AdaBoosting maintains a weighted training dataset, where weights roughly correspond to the importance of choosing the instance to train the next classifier in the ensemble. Specifically, in each iteration of Adaboosting, the samples are re-weighted based on the errors made on the training data in the previous iteration. A classifier is learned in each iteration from a sample of the training data where the sampling is based on the weights, i.e., larger weighted instances are more likely to be chosen in the training sample. If $f_1 \ldots f_k$ are k classifiers learned in k iterations, they can be combined into a single classifier as follows:

$$f(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m f_m(x)\right), \tag{2.3}$$

where each $f_m(x)$ is assumed to output a 1 or -1 value (assuming a binary classification) and α_m is a real-valued weight that encodes the importance of the mth classifier in the ensemble. The importance of the classifier added to the ensemble in this iteration is given by

$$\alpha_m = \frac{1 - err_m}{err_m},$$

where err_m is the error in the mth iteration. Thus, a smaller value of err_m implies that the classifier in the mth iteration has a larger weight in the overall ensemble. The training instances are re-weighed using an exponential re-weighting. Specifically, for all the correctly classified instances, their previous weight is multiplied by $\exp^{-\alpha_m}$ (which reduces the weight), and for the wrongly classified instances, their previous weight is multiplied by $\exp^{\alpha_m}$ (which increases their weight). It can be shown that this approach indeed reduces bias.

Example 2.8 (Ensemble Models for MNIST) One can combine several decision trees (defined in Sect. 2.1) to get an ensemble model for MNIST classification. Table 2.4 shows the precision, recall, and F1-score (described in Sect. 2.4) on a test dataset (chosen to be 25% of the data) for increasing the number of decision trees that are bagged together. Table 2.5 shows the same scores as the number of decision trees in the AdaBoosting ensemble is increased. Boosting the number of classifiers significantly improves performance since it reduces bias in the model, as compared to bagging that only reduces variance. □

Table 2.4 Performance of various ensemble solutions for the [**CharRC**] problem on the MNIST dataset increasing the number (N) of decision trees in the Bagging ensemble

N	Precision	Recall	F1-score
10	0.94	0.94	0.94
25	0.95	0.95	0.95
50	0.95	0.95	0.95
100	0.96	0.96	0.96

Table 2.5 Performance of various ensemble solutions for the [**CharRC**] problem on the MNIST dataset increasing the number of decision trees in the AdaBoosting ensemble (N)

N	Precision	Recall	F1-score
10	0.65	0.60	0.60
25	0.81	0.78	0.79
75	0.87	0.81	0.83
100	0.92	0.98	0.90

Practical considerations in the use of ensemble methods include the following:

- Bagging typically works better with unstable classifiers [4] such as decision trees and neural networks. In such classifiers, small changes to the training data affect the learned model significantly. A specialized form of Bagging is implemented in a random forest [4] (an ensemble of decision trees). Random forests are extremely powerful classifiers. A study based on more than 150 standard classification benchmarks showed that random forests outperform most other classifiers in a majority of benchmarks [9].
- Though Adaboosting was originally designed for weak classifiers, in practice, it can be used with all classifiers. Decision trees work especially well with boosting methods. A newer boosting approach called gradient boosting [10] is not only more flexible than the original Adaboosting approach, but it is highly scalable. An open source implementation of Boosted trees is available with the XGBoost library.

2.2.3 Support Vector Machines (SVMs)

Example 2.9 In an attempt to separate the data points from two classes shown in Fig. 2.5 by a line, the best line separator is a line that maximizes its distance from the data points in either class. The idea in support vector machine (SVM) classifiers is to try to learn such a separator. In some cases, a linear separator may be too restrictive, and therefore, using the idea of kernels (Sect. 4.1 gives more details on kernels), the SVM can learn more complex shapes of separators, as illustrated in Figs. 2.5 and 2.6.

Support vector machines (SVMs) are binary classifiers that learn a function such that the distance between the decision boundary of this function and data instances

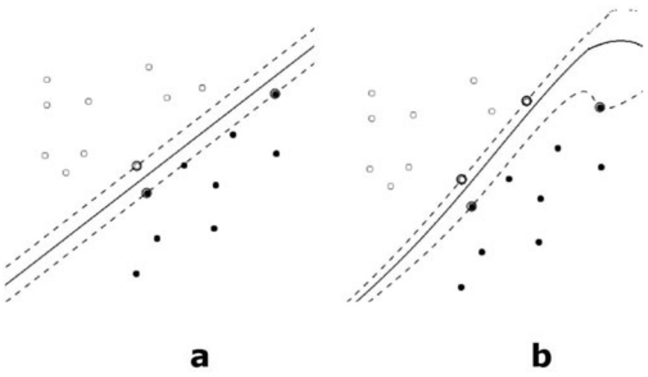

Fig. 2.5 (**a**) Linear SVM and (**b**) polynomial kernel decision boundary that separates data points of two classes (filled and empty circles). The encircled points are the support vectors, i.e., data points closest to the boundary

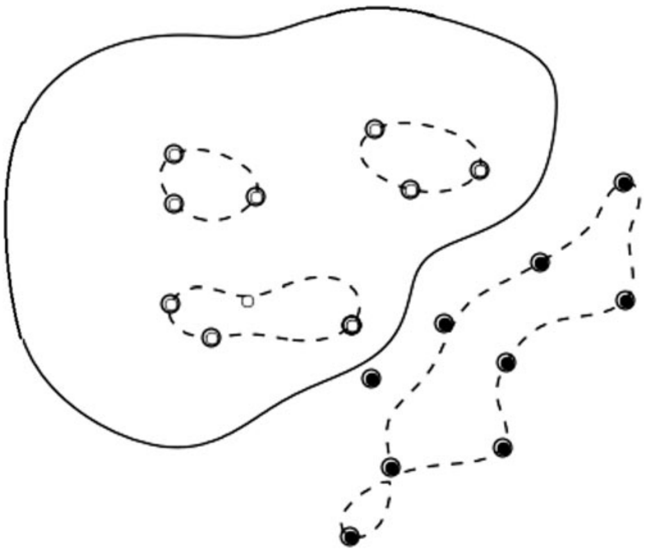

Fig. 2.6 RBF kernel SVM decision boundary that separates data points of two classes (filled and empty circles). The encircled points illustrate the support vectors, i.e., data points closest to the boundary

from either class is maximized. For example, if the training data contains 2 classes and the data corresponding to these classes can be separated by a linear decision boundary, the line may be very close to data points from either class, and the classifier is more likely to make mistakes on new data points that were not in the training data. On the other hand, if the distance between the line and the training data points on either side of the line is large, the classifier is more likely to assign the correct label on new data points. SVMs learn this type of decision boundary through a technique called max-margin optimization.

It turns out that learning a decision boundary in an SVM is fairly complex since it requires quadratic optimization. The idea behind this optimization procedure is to add *Lagrange coefficients* α_i corresponding to each data point. The optimization method solves a dual problem that computes the optimal values for these coefficients, from which the parameters for the classifier decision boundary can be derived. The objective function of the dual problem is

$$\max_{\boldsymbol{\alpha}_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j), \tag{2.4}$$

where $\boldsymbol{\alpha} = \alpha_1 \ldots \alpha_n$ are the Lagrange coefficients, $\mathbf{x}_i \cdot \mathbf{x}_j$ is the dot product between the instances $\mathbf{x}_i$ and $\mathbf{x}_j$ in the training data, and $y_i y_j$ is the product of their class labels (assumed to be either 1 or -1). Importantly, in most cases, only a few Lagrange coefficients have nonzero values. These correspond to the data points that are those closest to the decision boundary, the so-called *support vectors*. The parameter vector $\mathbf{w}$ defining the decision boundary can be derived from the support vectors as follows:

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i,$$

where $\alpha_1 \ldots \alpha_l$ are the support vectors. (Full technical details can be found in [7].)

The decision surface of SVMs is constrained to be linear when using the above max-margin optimization. That is, if the training data has two features, the decision surface is a line, for three features a plane and for n features, a $(n-1)D$ hyperplane. To learn nonlinear decision boundaries, the data can be pre-processed using *kernel* functions to enable SVMs. The idea in a kernel function is to implicitly add features (or dimensions) that are transformations of some selected features in the given data. Learning a linear decision boundary in this increased feature space is equivalent to learning a nonlinear decision boundary in the feature space of the original data. To learn an SVM using a kernel, the dot product between features, $\mathbf{x}_i \cdot \mathbf{x}_j$ in formula (2.4), is replaced by a general kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$. There are two commonly used kernel functions, the polynomial kernel and the radial basis (RBF) kernel. The polynomial kernel combines features using a polynomial function, while the RBF kernel uses a Gaussian function.

Example 2.10 Table 2.6 shows the precision, recall, and F1-score (defined in Sect. 2.4) on a test dataset (chosen to be 25% of the data) for various types of kernels. The performance of SVMs with the RBF kernel yields the best performance in this case. □

Table 2.6 Performance of SVMs on the handwritten digit classification problem [**CharRC**] using different kernels to combine the pixels in the image to form higher-order features in pre-processing

Kernel	Precision	Recall	F1-score
Linear	0.967	0.967	0.967
Polynomial	0.967	0.953	0.956
RBF	0.982	0.982	0.982

Some practical considerations in using SVMs include:

- Practical SVM implementations generally perform a soft-margin optimization that contains a tunable hyper-parameter (typically called *cost*). Tuning this parameter allows misclassifications by the SVM in the training data when the training data cannot be separated by the decision boundary.
- SVMs are not easily explainable especially when using nonlinear decision boundaries. The effect of a single feature cannot easily be mapped to the output since features are combined with each other.

2.2.4 Neural Networks (NNs)

How do humans learn to recognize handwritten decimal digits, such as those in MNIST images? Most likely, the neurons in our brains are specialized in recognizing specific low-level features (such as line segments in certain orientations) and then combining them together into more complex patterns (such as curves) resulting in a full image, as discovered by Nobel prize winners, neurophysiologists D. Hubel and T. Wiesel in visual processing in cats' brains (*braintour.harvard.edu/archives/portfolio-items/hubel-and-wiesel*). A neural network (NN) is an artificial model that tries to emulate this process.

Example 2.11 Figure 2.7 illustrates a neural network. It typically consists of many processing units (akin to neurons) that are connected to each other by synaptic-like connections, each of which plays a tiny part in processing the image. The network is usually organized in layers, where previous layers pass on their outputs to neurons in a higher layer, which combines them to discriminate increasingly complex patterns in the input features. The final layer (usually a single neuron) gives an output that labels the actual classification of the original input vector. □

Neural networks were originally inspired by the working of the mammalian brain. Specifically, just like the human brain performs complex tasks using signals from an inter-connected network of neurons, neural networks learn functions through a distributed network of computational units. Each individual unit, referred to as a (artificial) *neuron*, is characterized by a certain number of *activation states* and performs a mathematical operation based on inputs given to it to produce an output that is forwarded to other units in the network, as specified by a

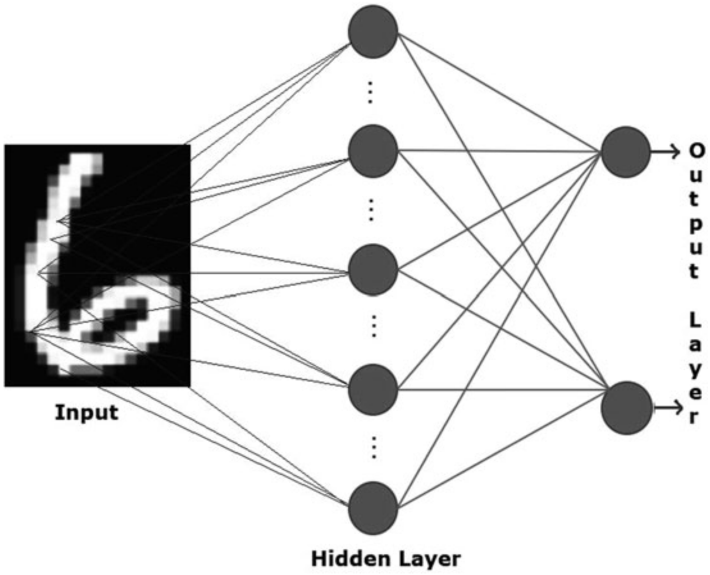

Fig. 2.7 A feedforward neural network (FNN) architecture consists of layers of neurons, including an input layer and an output layer. Inputs are fed to an input layer (e.g., pixels in an image, *left*); the network cascades these signals through any neuron in successive hidden layers (*middle*) refining feature extraction as it goes; finally, neurons in an output layer produce(s) a response in the output layer (e.g., digit 6 coded in binary (*right*))

directed graph called the *architecture* of the neural network. The most widely used architecture in neural networks is the *feedforward* architecture, where the units are arranged in layers and units in one layer are only connected to units in the layer next to it in the network. Each unit i in the neural network performs a seemingly insignificant but nonlinear operation that transforms inputs given to unit i into a single output based on weighted evidence net_i of all incoming units using a characteristic *activation function*, i.e., it determines how the individual unit in the neural network changes its activation. Specifically, each unit i has weights w_{ij} from other neurons j into it corresponding to each of its inputs. The inputs are multiplied by these weights, and the activation function is applied to this net input (the summation of the weighted outputs from other incoming neurons $x_j(t)$ at time t), i.e.,

$$A_i(t) = \sigma_i(net_i) = \sigma_j \sum_i w_{ij} x_j(t) \ ,$$

where w_i is the weight for the ith input unit. The activation function σ_i on $net_i = \sum_i x_i w_i$ computes a single output for time $t + 1$. There are many possible types of networks depending on the type of activations and activation functions used to produce the output. Most learning neural networks use *sigmoidal* units with the activation function given by

$$\sigma(u) = \frac{1}{1 + e^{-u}},$$

where $u = net_i = \sum_j w_{ij} x_j$. Several other types of activations may be used, including functions such as $tanh$, $RelU$, and *Softmax*. A *neural net* (NN) is thus a complex structure consisting of an architecture, a matrix of synaptic weights, and activation sets and functions for each neuron (usually a sigmoidal common to all). Changing any of them, particularly, the weights in training, will change the network because its responses to the same inputs will change accordingly.

Given a data point input, a FNN classifies the input by propagating the outputs of the units from layer to layer. This is called *forward propagation*, where the input features are clamped on the units of the *input layer*. The outputs from the input layer are then forwarded to the units in the next layer (at the ends of the edges pointing right in the architecture graph in Fig. 2.7). The neurons in layers that are not connected to the inputs or output neurons are called *hidden neurons*. They may in turn be organized into one or more hidden layers in the neural network. The neurons feeding signals to no other neurons form the final *output layer*.

What confers upon FNNs their powerful and versatile learning ability is the *backpropagation algorithm*. It is the most well-known learning algorithm that usually works on most varieties of data science problems. For example for a classification problem, the learning here is to produce the weights for all units in the neural network such that the output answers given by the neural network match the true classes in the data points in the training subset of data points as closely as possible. Specifically, if $y_1 \ldots y_n$ are the labels for the n instances in the training data and $o_1, \ldots, o_n$ represents the outputs of the neural network (given the weights for the units in the neural network), backpropagation performs the following optimization:

$$\mathbf{W_0} = \arg\min_{w_{ij}} \frac{1}{2} \sum_{k=1}^{n} (y_k - o_k)^2, \tag{2.5}$$

where $\mathbf{W} = [w_{ij}]$ represents the weight matrix among units j, i in the neural network. Note that Eq. (2.5) assumes a single unit in the output layer for simplicity, i.e., the neural network outputs a single value for each input instance. (In general, the neural network can have multiple output neurons, but the same procedure is applied to all of them.)

Backpropagation (BackProp) is a typical gradient descent algorithm (as described in Sect. 11.3) if the activation function is differentiable (e.g., of sigmoidal type). It solves the optimization problem for the objective function in Eq. (2.5). In gradient descent, starting with a random initialization of weights, for each input instance $\mathbf{x}_k$ in the training data, weight updates are roughly proportional to their effect on the output o_k the network generates for that instance. Bigger weights tend to have a bigger effect on the outputs, and so more guilt is assigned to them for a wrong answer, i.e., they change more. Intuitively, for an input instance, if the neural network correctly classifies this instance, then the weights remain unchanged. But for an incorrect classification, BackProp modifies the weights so that if the same instance is encountered by the neural network again, the output of the modified network will be closer to the true label for that instance. The weight update for the output layer is slightly different from the weight update for the hidden layers. Assuming that all the units are sigmoidal units, the equation for updating the weight (w_{kj}) from unit j in the output layer is

$$\Delta w_{kj} = \eta(y_k - o_k)o_k(1 - o_k)I_j,$$

where η is a small constant (the *learning rate*), o_k is the output of the neural network for the k-th instance in the training data (thus, the difference $(y_k - o_k)$ is the error from the true class label incurred by the output of the neural network for the kth training data instance), and I_j is the value coming into the jth input of the output unit. Note that if $(y_k - o_k) = 0$, then the weights remain unchanged, i.e., the neural network accurately estimates the label for the kth instance with its current weights. Recursively, for a *hidden unit* j now, its weight update depends upon the units that it is connected to in the layer immediately above it. Intuitively, if $downCone(j)$ refers to all the units whose net input includes the output of j, the unit can influence the output of the overall neural network only through downstream units in $DownCone(j)$. The weight updates can reflect this influence as

$$\delta_j = o_j(1 - o_j) \sum_{k \in DownCone(j)} \delta_k w_{kj},$$

where o_j is the output of j and w_{kj} is the weight of the synaptic connection between j and one of its downstream units k. A larger weight implies a greater influence that j has through this downstream units. The update for hidden unit jth's input weights is then

$$\Delta w_{ji} = \eta \delta_j I_{ji},$$

where η is a small constant called the *learning rate* and I_{ji} is the value coming into the ith input for the jth hidden unit. In practice, the learning rate η can be finetuned to an optimal value based on experimental results. A very large value might cause the weights to fluctuate rapidly and never converge, while a very small value might result in slow convergence of the weights. Further, in practice, the weights may only be updated with a net change once for every *batch* of inputs in an epoch to reduce the running time of learning. A run of backpropagation where all the input data has been processed is termed an *epoch*. In real-world data, a neural network may require hundreds of epochs before the weights converge to an acceptable solution. (A full derivation and more details of the BackProp algorithm can be found in [20].)

Table 2.7 The performance of a neural network for **[CharRC]** improves with the size of the hidden layer since they are learning higher-level features from the pixels. Performance peaks out at 20 nodes for this choice of training data

N	Precision	Recall	F1-score
5	0.880	0.880	0.880
10	0.940	0.940	0.940
15	0.964	0.964	0.964
20	0.971	0.971	0.971
25	0.971	0.971	0.971

Example 2.12 (Neural Networks for MNIST) One can apply neural networks to the [**CharRC**] problem by considering each pixel as a feature in the input layer. Each pixel is then connected to all nodes in one hidden layer. The hidden layer neurons learn more complex, higher-level features of digit representations by combining pixels from the images into abstract features. Finally, the hidden layer is connected to the output layer to produce a class for the input image. The performance of neural networks for different hidden layer architectures is shown in Table 2.7 using precision, recall, and F1-score (defined below in Sect. 2.4) on a test dataset (chosen to be 25% of the data) as the number of nodes in the hidden layer progressively increases. The performance improves as the number of nodes in the hidden layer increases since it is learning higher-level features from the pixels. Once all the useful information has been represented, the performance peaks out at a certain point (20 nodes in this example), and adding neurons to the hidden layer does not contribute any significant performance improvements. □

Practical considerations for neural network learning include:

- Neural networks are perhaps the most powerful and versatile among machine learning algorithms. A famous theorem discovered independently by several groups shows that neural networks are universal function approximators for continuous functions [11, 18], even after unbounded iteration as dynamical systems [12], i.e., a neural network, even with a single hidden layer, always exists to compute any continuous function on a given bounded region to any given arbitrary degree of accuracy. However, this does not always mean that neural networks produce the best approximations because large neural networks could require an infeasibly large training dataset or training time for backpropagation to learn appropriate weights.
- Overfitting is a common problem associated with neural networks. Overfitting occurs when the neural network fits the training data accurately (memorizes the data) but fails to generalize to data not used in training. One way to avoid overfitting is to regularize the neural network by forcing it to learn smaller valued weights by placing a penalty on large weights in Eq. (2.5). Recently, other approaches have been developed to avoid overfitting, such as *dropout*. The main idea in dropout is to force the neural network to learn simpler functions by removing units from the network at random.

- When the number of hidden layers is large (more than 2), the neural network is typically called a *deep network* [14]. Deep networks have obtained the state-of-the-art results in several applications such as image understanding, natural language understanding, and game playing. One of the issues with applying backpropagation to deep networks is that the weight updates tend to become 0 as the number of hidden layers increase, a problem referred to as the *vanishing gradient* problem. Therefore, variants of backpropagation algorithms have been developed to learn deep networks [17]. Further, since backpropagation and its variants use several matrix operations, they can be efficiently implemented in specialized hardware called GPUs to obtain significant improvements in the speed of large deep learning networks.
- Another problem with neural networks is that it is hard to explain/interpret results from the neural network, sometimes referred to as the *credit assignment* problem. Specifically, the hidden layers in the neural network transform the original features in the data into a representation that acts as a blackbox and makes it hard to understand or rationalize why it is producing the results it does. Further, since the final output from a neural network is based on a series of such transformations layer after layer, particularly in a deep network with several hidden layers, the network appears to be a black box harder to *interpret* by a human observer trying to make sense of the responses it is putting out. Interpreting the results in neural networks continues to be a highly active area of research that is vital to their use in real-world applications (Sect. 10.4 further discusses this matter).

2.3 Machine Learning Solutions: Unsupervised

Labels in the features in a dataset are extremely useful to find solutions to a problem using supervised learning methods, as described in the previous section. Naturally, answers to specific instances of a problem are not always available in the real world. For example, in a clustering problem, a label for a data point requires solving the problem holistically for *all* data points before it can be given, so it is impossible to include it in a dataset that one may hope to use to obtain a solution. Another family of methods in the area of unsupervised machine learning become useful to address such problems. Thus, the inputs to the algorithm are simply the data points without the labels. The only recourse left for unsupervised learning is to aim to discover regularities in the form of *hidden patterns* in the data in order to solve a problem. The goal of this section is to describe some of these methods, including the most popular ones, k-Means and Gaussian mixture models.

Example 2.13 By removing the labels in the Iris dataset (described in Sects. 1.1 and 11.5), a clustering problem arises. Ideally, the clustering would place instances of the same variety of Iris flower within the same cluster. To make the illustrations easy to visualize, just two features (the sepal width and sepal length) are considered in the discussion in this section, although the methods can work with the full dataset in higher dimensions just as well. □

There are two main types of clustering:

- *Hard clustering* is ideal clustering, i.e., it requires a partition of the data points so that each instance belongs to exactly one part (or cluster).
- *Soft clustering* allows a fuzzy partition of the data points, where an instance is assigned to belong in more than one cluster (i.e., the parts in the partition overlap) or even perhaps no cluster at all (i.e., the parts do not exhaust the whole feature space), or both.

2.3.1 Hard Clustering

Example 2.14 The most intuitive way of assigning data points to clusters follows up on the idea of a Voronoi diagram (defined in Sect. 1.1), i.e., to select a number of *centroids* and assign the points to the nearest centroids. The problem is that there are no centroids to begin with. Even if some initial centroids could be selected at random, how can one ascertain that the clustering is of good enough quality? One could choose as centroids the midpoints (i.e., the component-wise average of the points) in a cluster and so require a Euclidean space where these operations can be performed. Figure 2.8 shows such a clustering for the Iris dataset. In this case, the assumption is that data points that are close to each other in terms of the Euclidean distance are similar to each other and should be placed in the same cluster and vice versa. It thus appears necessary to have some criterion to assess the quality of a clustering to decide the assignment is good enough or refine it. □

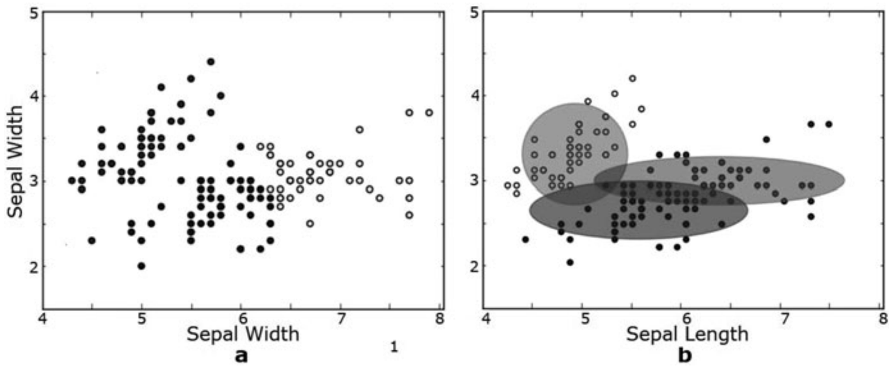

Fig. 2.8 Clustering of the Iris data. The three optimal clusters are indicated by gray shades. (**a**) k-Means clustering with three clusters based on sepal features; (**b**) Soft clustering by Gaussian mixture clustering

The most popular and elegant algorithm (due to its simplicity) for hard clustering is the *k-Means* algorithm for cases where the measure of similarity defining the problem is a Euclidean distance and a desired number of clusters k are given. Specifically, if the input is the usual data matrix $[\mathbf{x}_1 \ \ldots \ \mathbf{x}_n]$, where each data instance is a point in a Euclidean space (i.e., all its features are real numbers), k-Means starts with a random initialization of k cluster centroids and iteratively performs the following two steps aimed to improve performance until convergence:

1. Given the current centroids, assign each data point to the cluster determined by its closest cluster centroid.
2. Given assignments of all points to their clusters, update the cluster centroids to the mean of all its data points.

Convergence is achieved when the centroids no longer change. Figure 2.8 shows the results of k-Means applied to the Iris dataset.

To assess the quality of the solution thus provided, consider the functional Δ given by

$$\Delta = \min_{\{\mathbf{c}_j\}} \left\{ \sum_{i=1}^{n} |\mathbf{x}_i, \mathbf{c}_{j_i}|_2 \right\}$$

for the ℓ_2 distance, where $\mathbf{c}_{i_j}$ represents the cluster to which $\mathbf{x}_i$ is assigned, $\mathbf{c}_j$ is the cluster center of a cluster j, and $|*, *|_2$ is the ℓ_2 distance. Δ is a measure of the *within cluster variation* for a choice of k centroids $\{\mathbf{c}_j\}$. It has been shown [3] that k-Means converges to a local minimum of this aggregated measure of how close the data points are to their centroids across all clusters, so k-Means does solve an optimization problem. However, convergence is not guaranteed, and when it does converge, the clusters may not be guaranteed to be optimal across all possible choices of centroids.

Inter-cluster variance minimization is re-assuring, but the value itself tells little about the quality of the clustering. One could look at the results and assess them visually, but that becomes impossible in higher dimensions, where the challenging problems lie. Other metrics can be used, such as *silhouette* (defined below in Sect. 2.4). A larger score indicates better clustering of similar images compared to smaller scores. The range of silhouette scores is the interval [0, 1].

Example 2.15 (k-Means on the MNIST Dataset) Each pixel is considered a feature, and similarity is based on the pixel RGB (or grayscale) values. Table 2.8 shows silhouette scores for various k, the number of clusters. As seen in this table, as we approach the true number of clusters (here 10), the silhouette score progressively increases indicating that k-Means is detecting true patterns in the images corresponding to each digit and is therefore able to cluster them optimally. □

Table 2.8 Performance of k-Means (*second column*) and Gaussian Mixtures Models (GMMs) (*third column*) on the MNIST dataset for various numbers of clusters (k). As k nears the optimal value 10, the silhouette score progressively increases to an optimal value of quality for k-Means (still far from the ideal) and then increases further for GMMs

k	Silhouette k-means	Silhouette GMM
1	0.116	0.100
3	0.118	0.080
5	0.137	0.040
8	0.157	0.060
10	0.159	0.167

Some practical considerations when using k-Means clustering include:

- The clustering generated using k-Means depends upon the initial cluster centroids. More sophisticated initialization methods than random initialization are used in algorithms such as k-Means ++ [1].
- k-Means works well when the optimal clusters are *globular*, i.e., are roughly spherical-shaped. Likewise, k-Means works well when the clusters have similar sizes and densities, i.e., have roughly the same number of points in each cluster with a similar spread of points.
- With nonglobular shapes, k-Means tends to separate clusters poorly. For nonspherical clusters, a variant called kernel k-Means can be used where kernel functions are applied to increase dimensionality of the data before clustering. Alternatively, if the clusters are diamond-shaped, a different metric such as ℓ_1 (defined in Chap. 1 and illustrated in Fig. 1.8) could be used.

2.3.2 *Soft Clustering*

Example 2.16 In clustering the Iris dataset, if a certain flower data point's features put it into two different clusters, instead of forcing that instance into a single cluster, one can determine how likely is an instance to belong to a cluster. The right panel of Fig. 2.8 shows the clusters formed using a Gaussian Mixture Model (GMM) for the Iris data. The three shaded regions illustrate the shape of the Gaussians, and so these distributions will overlap, as shown. For every flower in the dataset, one can now determine how likely it is to be part of the 3 clusters, and therefore, it is perfectly feasible that a flower that lies at the intersection of two clusters is equally likely to be a member of either cluster. □

When the optimal clusters are uncertain, i.e., each data point can belong to multiple clusters, instead of randomly choosing a cluster for a data point, probabilistic methods can be used. The most well-known approach for probabilistic clustering is the *Gaussian Mixture Model* (GMM)-based clustering. GMMs are probability distributions using multiple Gaussians with some designated means. In the case of clustering, each cluster is represented by a Gaussian in the mixture. The expectation–maximization (EM) algorithm (described in Sect. 11.3.2) is used to learn the parameters of the GMM. Specifically, if there are k clusters in the data, a GMM distribution is defined as

$$P(x|\Theta) = \sum_{i=1}^{k} \alpha_i P_i(x|\theta_i),$$

where $P_i(x|\theta_i)$ is the ith Gaussian probability distribution and α_i is its weight. The ith Gaussian is defined by mean and co-variance matrix parameters $\theta_i = (\mu_i, \Sigma_i)$, i.e., its probability density function is

$$P(\mathbf{x}|\theta_k) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} \exp^{-\frac{1}{2}(\mathbf{x}-\mu_k)^\intercal \Sigma^{-1}(\mathbf{x}-\mu_k)},$$

where p is the number of features in $\mathbf{x}$. The parameters θ_i can be learned using the EM algorithm. EM starts by assigning random values for the Gaussian parameters in the mixture and iteratively updates them until they converge by repeatedly performing the following two steps:

- *Expectation step*
  The probability of each data point is computed for each of the Gaussians in the mixture using the current Gaussian clusters. This step performs the soft clustering.
- *Maximization step*
  The Gaussians are re-parameterized based on the probabilities of the data points computed in the expectation step using a method called *max-likelihood estimation* (also described in Sect. 11.3.2).

Expectation–maximization is repeated until all the parameters for all the Gaussian distributions have converged.

Example 2.17 (GMMs for MNIST) Table 2.8 also shows the silhouette scores as the number of Gaussians (N) in the GMM varies. As seen in this table, the results are less intuitive than k-Means since increasing the Gaussians initially reduces the silhouette score that seems counterintuitive. One reason for this could be that since GMMs perform soft clustering, for similar looking digits (e.g., 1 and 7), the probability of that digit belonging to several classes is somewhat similar, and thus it is hard to make a distinction as to which class it truly belongs to. However, the silhouette score increases indicating that as the sufficient number of Gaussians in the GMM (in this case 10 digits) is reached, each Gaussian is modeling images corresponding to a single digit. □

Some practical considerations when using Gaussian mixtures models include:

- GMMs are very expressive and can represent a wide range of distributions. In fact, they are considered as universal approximators, i.e., they can represent any type of distribution [27].
- The EM algorithm is not guaranteed to find the optimal clusters.
- While the basic GMM clustering algorithm pre-specifies the number of clusters (i.e., it fixes the number of Gaussian distributions in the mixture), there are more advanced variants of nonparametric Gaussian Mixture Models that infer the optimal number of clusters in the data [6].
- A well-known application of GMM clustering is in topic modeling where the task is to infer the number and topics in a text document [2].

2.4 Controls, Evaluation, and Assessment

Two tasks are critical in solving a well-defined problem in data science, namely, the methods to find solutions and the evaluation and assessment of their quality [21]. This section describes methods and metrics to quantify and evaluate the quality of solutions. Methods include training, testing, and cross-validation. Metrics include accuracy, precision (specificity), recall (sensitivity), and clustering metrics. Such methods are required not only in the training phase of the solution or model but more importantly in the testing and cross-validation phases of a solution development cycle. In addition, an assurance that the solution will work well in the production environment is also desirable.

Example 2.18 The Netflix movie recommendation problem can be regarded as a classification problem. A good movie recommendation system should provide a user with a list containing movies that s/he is most likely to like. Therefore, a solution not only needs to identify whether the user is going to like (with varying degree of preference) a movie, but also predict the corresponding ratings so that the system can figure out the top 5 or so to recommend the user. Accuracy does not appear to be an appropriate metric to quantify performance since there are no objective labels associated with a rating, and they are viewer-dependent. What would be an appropriate metric to use? Should it use all the data points or only some of them to get a sense how the solution will perform on unknown future data points? □

2.4.1 Evaluation Methods

Generally, a dataset is split into three subsets, i.e., a training set, a testing set, and a validation set (already mentioned in Sect. 1.3). A *training* dataset is a set of data points used to train a model by fitting a set of relevant parameters. At the end of the training phase, a fitted model is obtained that can then be assessed using a *testing* dataset, i.e., a set of data points left out during the training phase, to determine the model's quality. Finally, a *validation dataset* is a set of instances used to tune the hyper-parameters for the production environment [23]. The distinction between the testing and the validation phase is less sharp since both require data points left out during the training phase. However, a distinction can be made [23] by defining a validation set as a set of points used to tune the parameters of a classifier (for example, the number of hidden units in a neural network), whereas the testing set is used to assess the performance of a fully specified classifier. Several quantitative metrics used to measure the performance of a ML model in each phase are summarized in Table 2.10. For low-scale applications, just training and testing sets are used to zoom into a solution.

A more robust (hence very popular) approach is k-fold *cross-validation* ($k \geq 2$). In this approach, the dataset is split into, say $k = 5$ parts, and a model is then trained using four partitions and tested on the remaining partition, recording the performance score on the latter partition. The process is repeated s number of times (16 is a common number to establish a significance for a sample of small size), and the average of these cross-validation scores is used as a metric to assess the quality of the model.

2.4.2 Metrics for Assessment

In terms of specific scores for a given dataset, there are a number of choices, each appropriate for various kinds of problems and cases. They are summarized in Table 2.10. In the simplest case of a classification problem, the performance of a solution (called a *classifier*) can be measured in various ways. The simplest one is well known.

The *accuracy* of a classifier M on a dataset $\mathbf{X}$ is the ratio $hits/misses$, where a hit ($miss$) is data point $\mathbf{x}$ for which M's answer does (does not, respectively) agree with the label in $\mathbf{x}$.

A binary classifier M for a classification problem with two categories T and F (so that $\Omega = T \cup F$, say T being the class of true interest) may *miss* in two different ways: by placing the data point from F into T (a so-called *false positive (FP)* , or vice versa (a so-called *false negative (FN)*) and ditto for F. A hit will place elements of T in T, i.e., the *True Positives (TP)* and *True Negatives* obtained from M satisfy $|T| = |TP| + |FN|$. The accuracy would then be $a = (|TP| + |TN|)/(|T| + |F|)$. If the classifier is not binary, hits and misses are calculated for elements in a class of interest T by considering F to be the union of the remaining classes as a binary classification.

Accuracy can be very misleading in lopsided classification problems where one class is disproportionally large compared to the others. A lazy classifier can simply place all elements in the large class and be assured high accuracy. In many situations, especially in clinical settings, where T is healthy patients and F is patients with some disease, the category of more interest is really F despite the fact it may be a relatively small class. In such a case, there are three other choices, depending on which of the two categories is more important, T or F.

The *recall* (also called *sensitivity*) of a classifier M on a dataset $\mathbf{X}$ is the ratio $TP/(TP + FN)$, i.e., the proportion of correct classifications out of the total number of *positive* classifications made (only considering the class T, which includes the FNs for the classifier).

The *precision* of a classifier M on a dataset $\mathbf{X}$ is the ratio $TP/(TP + FP)$, i.e., the proportion of correct classifications out of the total number of *positive* classifications made (only considering the class F, which includes the FPs for the classifier).

The *specificity* of a classifier M on a dataset $\mathbf{X}$ is the ratio $TN/(TN + FP)$, i.e., the proportion of correct classifications out of the total number of *negative* classifications made (only considering the class F, which includes the FPs for the classifier).

The *F1-score* of a classifier M is the geometric mean

$$F1 = \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = \text{precision} * \text{recall}/(\text{precision} + \text{recall}).$$

These quantities are usually displayed together in a so-called *confusion matrix* (no doubt, to avoid confusion!), as illustrated in Table 2.9.

For prediction problems where the target (or loss) function f takes on numerical values, the error is measured by some ℓ_d (usually $d = 1$ or 2) distance between the predicted value and the observed value (or label), either on average or relative to the true value (Table 2.10).

The *root mean-squared error (RMSE)* is the average ℓ_2 distance between the model value and the true value, averaged across all data points, i.e.,

$$\text{RMSE} = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(\mathbf{y}_i - \mathbf{x}_i)^2 \,/\, n}\,.$$

The *relative error (RE)* is the average ℓ_1 distance (absolute value) between the model value and the true value scaled to the true value, i.e.,

$$\text{RE} = \frac{1}{n}\sum_{i=1}^{n} |\text{Observed}_i - \text{Predicted}_i| \,/\, |\text{Observed}_i|\,.$$

Table 2.9 Confusion matrix for a classifier solution to a classification problem in relation to a category T and the remaining categories F (The brackets are actual values for a given problem and dataset)

	Predicted	
Observed	True	False
True (in T)	[TP]	[FN]
False (not in T)	[FP]	[TN]

Table 2.10 Common quantitative metrics to assess the quality of a data science solution M [13, 15, 23, 24, 26] for a dataset $\mathbf{D}$ of size n and data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$

Metric	Definition
Classification	
Accuracy	$a = (TP + TN)/(TP + TN + FP + FN)$
Recall (sensitivity)	$r = TP/(TP + FN)$
Precision	$p = TP/(TP + FP)$
Specificity	$p = TN/(TN + FP)$
F1-score	$F1 = pr/(p + r)$
Prediction	
Root mean-squared error (RMSE)	$\text{RMSE} = \sqrt{\sum_{i=1}^{n} (\text{Observed}_i - \text{Predicted}_i)^2/n}$
Relative error (RE)	$\text{RE} = \frac{1}{n} \sum_{i=1}^{n} \lvert\text{Observed}_i - \text{Predicted}_i\rvert / \lvert\text{Observed}_i\rvert$
Clustering $\mathbf{D} = C_1 \cup \ldots \cup C_k$	
SSD	Total sum of squared distances of $\mathbf{x}_i$s from their centroids $\bar{\mathbf{x}}_i$: $\text{SSD} = \sum_{i=1}^{n} \text{SSD}_i(M) = \sum_{i=1}^{n} d(\mathbf{x}_i, \bar{\mathbf{x}}_i)^2$
k-Means—Elbow Method	Choose k that causes a sharp turn in SSD
Silhouette value of a point	$s(i) = (b(i) - a(i))/\max\{a(i), b(i)\}$, where $a(i) = \frac{1}{(\lvert C_{k_i}\rvert - 1)} \sum_{\mathbf{x}_j \in C_{k_i}, \mathbf{x}_j \neq \mathbf{x}_i} d(\mathbf{x}_i, \mathbf{x}_j)$, $b(i) = \min_{j \neq k_i} \frac{1}{\lvert C_j\rvert} \sum_{\mathbf{u} \in C_j} d(\mathbf{x}_i, \mathbf{u})$, C_{k_i} is the cluster given by M for $\mathbf{x}_i$ and $d(\mathbf{x}_i, \mathbf{u})$ is the distance between data points $\mathbf{x}_i$ and $\mathbf{u}$
Silhouette of a clustering	s is the average of the $s(i)$ values of all points $\mathbf{x}_i$ in $\mathbf{D}$
Silhouette score for k	s_k for k clusters is the average of all such s's
Silhouette coefficient	Of $\mathbf{D}$ is the maximum silhouette score s_k over all $1 \leq k \leq n$
Regression	
R-squared	$R^2 = 1 - (\text{RSS}/\text{TSS})$, where RSS is the sum of squares of residuals and TSS is the total sum of squares
Akaike's Information Criterion (AIC)	$\text{AIC} = 2k - 2\log(L)$, where k is the number of estimated parameters in the model M and L is the maximum value of the likelihood function for M
Bayesian Information Criterion (BIC)	$\text{BIC} = k\log(n) - 2\log(L)$, where L is the maximized value of the likelihood function of the model M and k is the number of parameters estimated by M

For clustering problems, there are several metrics to assess the quality of a solution. They focus on measuring the density of the clusters (average intra-cluster distance), as well as the separation between clusters (average inter-cluster distance), on the average. A popular combined measure is the silhouette.

The *silhouette value* of a point $\mathbf{x}_i$ in a clustering of a dataset $\mathbf{D} = C_1 \cup \cdots \cup C_k$ is given by

$$Silhouette \quad s(i) = (b(i) - a(i))/\max\{a(i), b(i)\},$$

where $a(i)$ is the average *cohesion* value of a data point $\mathbf{x}_i$ in its cluster C_{k_i} to all other data points in the same cluster, measured by $a(i) = 0$ if $C_{k_i} = \{\mathbf{x}_i\}$, else by

$$a(i) = \frac{1}{(|C_{k_i}| - 1)} \sum_{\mathbf{x}_j \in C_{k_i}, \mathbf{x}_j \neq \mathbf{x}_i} d(\mathbf{x}_i, \mathbf{x}_j),$$

$b(i)$ is the minimum *separation* of the data point $\mathbf{x}_i$ with all data points in the closest cluster measured by

$$b(i) = \min_{j \neq k_i} \frac{1}{|C_j|} \sum_{\mathbf{u} \in C_j} d(\mathbf{x}_i, \mathbf{u}),$$

and $d(\mathbf{x}_i, \mathbf{u})$ is the distance between data points $\mathbf{x}_i$ and $\mathbf{u}$.
The *silhouette* for the clustering solution is the average of the silhouettes $s(i)$ of its data points $\mathbf{x}_i$. The *silhouette score* s_k for k clusters is the average of the silhouettes s for all clustering with k clusters. The *silhouette coefficient* for a dataset is the maximum of the silhouettes scores s_k across all possible values $1 \leq k \leq n$ (Table 2.11).

Once a performance metric is selected and computed on a test dataset, an important question arises: *is this score good enough for a quality solution?* In Example 2.18, if a recommender system got an average RE of 50%, can it be deployed in a market? To make such a decision, a more thorough analysis should be made considering several solutions, including competitors' performance. One approach is to compare it to published performance scores for similar problems as a baseline and decide accordingly.

Another approach involves running an *experimental control*. This approach is common for decision problems whose solutions have a large impact on a person

Table 2.11 Use of assessment metrics in typical problems in data science (PwP: Possible with Pre-processing)

Metrics	Classification	Clustering	Prediction
Accuracy	Ok if supervised	N/A (no labels)	PwP
Recall (Sensitivity)	Ok if supervised	N/A (no labels)	PwP
Precision	Ok if supervised	N/A (no labels)	PwP
Specificity	Ok if supervised	N/A (no labels)	PwP
F1-score	Ok if supervised	N/A (no labels)	PwP
Silhouette	PwP	Yes	PwP
R-squared	PwP	N/A (no response)	Yes
Akaike's Information Criterion (AIC)	PwP	N/A (no response)	Yes
Bayesian Information Criterion (BIC)	PwP	N/A (no response)	Yes
Root mean-squared error (RMSE)	PwP	N/A (no response)	Yes
Relative error (RE)	PwP	N/A (no response)	Yes

or population. The simplest case is the well-known *placebo control*, where the effectiveness of a drug may depend on subjective self-reporting that may be misleading (the well-known placebo effect). A control requires changing a critical feature predictor (provide a placebo instead of the actual drug) and comparing the results with the complementary case where the predictor has the opposite value (e.g., the actual drug is administered), while *keeping everything else unchanged.* The difference in scores will be considered significant if the difference between the scores is larger than the standard error of the scores in the positive case (e.g., actual drug taken).

In summary, the decision whether a particular score is good enough really depends on the definition of the business problem being tackled (as defined in Sect. 1.4). Without that definition, it is impossible to make a decision. With it, there is still room for argument in terms of the impact of the solution or decision being implemented. A solution to be deployed into a market for production requires careful consideration of financial and/or other implications. A movie recommendation system that is going to be used by a handful of people would tolerate a medium score. If the number of viewers ranges in the millions, the impact may require much higher scores.

References

1. Arthur, D., & Vassilvitskii, S. (2007). k-Means++: The advantages of careful seeding. In *SODA '07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 1027–1035).
2. Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research, 3*, 993–1022.
3. Bottou, L., & Bengio, Y. (1995). Convergence properties of the k-means algorithms. In G. Tesauro, D. Touretzky, & T. Leen (Eds.), *Advances in neural information processing systems* (Vol. 7). MIT Press.
4. Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32.
5. Chen, C., Liu, Y., & Peng, L. (2019). How to develop machine learning models for healthcare. *Nature Materials, 18*, 410–414.
6. Christopher, M. (2006). *Pattern recognition and machine learning*. Springer.
7. Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other Kernel-based learning methods*. Cambridge University Press.
8. Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning, 29*(2), 103–130.
9. Fernández-Delgado, M., Cernadas, E., Barro, S., & Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research, 15*(1), 3133–3181.
10. Friedman, J. B. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics, 29*, 1189–1232.
11. Funahashi, K. I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks, 2*(3), 183–192.
12. Garzon, M., & Botelho, F. (1999). Dynamical approximation by recurrent neural networks. *Neurocomputing, 29*(1), 25–46.

13. Glantz, S. A., Slinker, B. K., & Neilands, T. B. (1990). *Primer of applied regression and analysis of variance*. McGraw-Hill Inc.
14. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.
15. Gideon, S., et al. (1978). Estimating the dimension of a model. *The Annals of Statistics, 6*(2), 461–464.
16. Hastie, T. J., & Tibshirani, R. J. (1986). Generalized additive models. *Statistical Science, 43*(3), 297–310.
17. Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*, 1527–1554.
18. Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2*(5), 359–366.
19. Kelleher, J. D., MacNamee, B., & D'Arcy, A. (2015). *Fundamentals of machine learning for predictive data analytics: Algorithms, worked examples, and case studies*. MIT Press.
20. Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.
21. Mount, J., & Zumel, N. (2019). *Practical data science with R*. Simon & Schuster.
22. Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81–106.
23. Ripley, B. D. (2007). *Pattern recognition and neural networks*. Cambridge University Press.
24. Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics, 20*, 53–65.
25. Schapire, R. E. (2013). Explaining AdaBoost. In *Empirical inference* (pp. 37–52). Springer.
26. Taddy, M. (2019). *Business data science: Combining machine learning and economics to optimize, automate, and accelerate business decisions*. McGraw Hill Professional.
27. Titterington, D. M., Smith, A. F. M., & Makov, U. E. (1985). *Statistical analysis of finite mixture distributions*. New York: Wiley.

Chapter 11
Appendices

Max Garzon, Lih-Yuan Deng, Nirman Kumar, Deepak Venugopal, Kalidas Jana, and Ching-Chi Yang

Abstract This chapter presents a summary review of prerequisite concepts from statistics, mathematics and computer science, although readers are expected to have a nodding familiarity with most of them. It also provides some background on a number of computational problems and data sets used in the book or particularly useful for data science and dimensionality reduction; as well as a review of computing environments and platforms that could be used as a playground to run and test the methods and solutions described in this book. The aim is to provide a refresher of what they are and point to sources in the literature where they could be studied in more detail, if needed.

Basic concepts such as "element," "set," "subset," "power set" of a given universe or populations of objects Ω and operations among them (such as "union," "intersection," "complementation," and such) and "functions" among them will be assumed to be familiar to the reader so as to require no definition beyond their ordinary intuitive meaning from elementary high school or college mathematics. For example, the *indicator* function χ of a subset $E \subset \Omega$ is a function with domain Ω into the real numbers $\mathbf{R}$, given by $\chi_E(s) = 1$ if $s \in E$ and $\chi_E(s) = 0$ in the complement of E.

M. Garzon (✉) · N. Kumar · D. Venugopal
Computer Science, The University of Memphis, Memphis, TN, USA
e-mail: mgarzon@memphis.edu; nkumar8@memphis.edu; dvngopal@memphis.edu

L.-Y. Deng · C.-C. Yang
Mathematical Sciences, The University of Memphis, Memphis, TN, USA
e-mail: lihdeng@memphis.edu; cyang3@memphis.edu

K. Jana
Fogelman College of Business, Memphis, TN, USA
e-mail: kjana@memphis.edu

M. Garzon et al. (eds.), *Dimensionality Reduction in Data Science*,
https://doi.org/10.1007/978-3-031-05371-9_11

11.1 Statistics and Probability Background

The subject matter of *statistics* is *understanding and processing data at the level of aggregates and populations through observations of their individual objects.* The subject matter of *probability* is *random phenomena and their inherent uncertainty*. A cornerstone concept for both is that of a random *experiment*, i.e., any process in the world that can be repeated any number of times and produces one of a certain number of observations (or *outcomes*) every time, such as tossing a coin, rolling a die, rolling a pair of dice, the total rainfall amounts at home on a given day, the life length of the only light bulb in the kitchen, or the time between consecutive emissions of atomic particles from this sample of a radioactive material. The set of single outcomes (called *simple* or *elementary*) *events* of such an experiment is called the *sample space* Ω of the experiment. For example, for the toss of the same coin twice, the sample space is (Table 11.1)

$$\Omega = \{(H, H), (H, T), (T, H), (T, T)\}.$$

Probabilistic analysis proceeds by assigning a measure of *likelihood* or *uncertainty* to these elementary events and their combination into more complicated compound events (all referred to as just *events*), like parity ({2, 4, 6}) in the case of the roll of a die. These measures are usually assigned by frequency of occurrence in a large number of repetitions of the experiment, but they can be arbitrarily assigned as long certain rules of common sense are respected. Such a structure is called a probability space.

A *probability space* consists of a sample space Ω and a function P on Ω that assigns a real value $P(s)$ to every simple event $s \in \Omega$ and by extension, a probability value to composite events E according to

$$P(E) = \sum_{s \in E} P(s),$$

so that $P(\varnothing) = 0$ and $\mathrm{P}(\Omega) = 1$.

In particular, if an event $A = A_1 \cup \cdots \cup A_k$ can be partitioned into a *disjoint* set of parts A_i, then $P(A) = P(A_1 \cup \cdots \cup A_k) = \sum_i P(A_i)$, whereas only $P(A) = P(A_1 \cup \cdots \cup A_k) \leq \sum_i P(A_i)$ is true if the subsets have overlaps. If Ω is finite, this procedure assigns a probability to every subset of Ω. For infinite sets in the continuum, the assignment is more involved (with sums of an infinite number of

Table 11.1 Sample space and possible values of a random variable X that counts heads in the random experiment of tossing a coin twice

Simple event s	First toss	Second toss	$X(s)$
(H, H)	1	1	2
(H, T)	1	0	1
(T, H)	0	1	1
(T, T)	0	0	0

terms), but P usually boils down to using measures such as length (1D intervals), areas (2D regions), volumes (in 3D spaces) and perhaps hypervolumes (in higher dimensional spaces), and follow the same basic rules.

Probability spaces can be classified as *discrete* or *continuous* depending on whether the sample space is a finite or countable set, or whether it is larger in cardinality, say the continuum of a line segment in 1D Euclidean space, where the underlying random experiment could be taking a random point from the unit interval [0, 1].

In a probability spaces, the *conditional probability* of an event B given another event A is

$$P(\,B \mid A\,) = P(B \cap A)/P(A)$$

Two events A, B in a probability space are *stochastically independent* if the outcome of the random experiment being in A does not affect the probability of it being in B, i.e., if $P(\,B \mid A\,) = P(B)$, or equivalently, $P(A \cap B) = P(A)P(B)$.

A probability function P is a special case of a more general concept of random variable as observations made on the outcomes of a random experiment.

A *random variable* (RV) is a real-valued function $X : \Omega \to \mathbf{R}$ defined over the probability space (Ω, P), i.e., for each $s \in \Omega$, $X(s)$ is a (properly defined) real value. A (discrete) *stochastic process* is a sequence of random variables $\{X_t\}_{t \in \mathbf{N}}$ defined on a common probability space. A *sample* of a probability space (Ω, P) are the values of a run of the stochastic process $(X_{t \le n})$ for some finite value $n \in \mathbf{N}$ selected according to the same probability model P (so the variables are *independent and identically distributed (i.i.d.)* .)

Example 11.1 The number of heads X when tossing two coins (all possible values of X are 0, 1, or 2), and a value between 1 and 6 for the roll of a die are properly defined RVs, but the name or age of the person who tossed the coin(s) is not since the coin(s) tosses do not determine it. □

RVs can also be classified as *discrete* or *continuous* depending on whether the set of values taken on is a countable set (like a subset of the natural numbers $\mathbf{N}$, as with counting heads on coin tosses) or not (like the continuum of values in the real numbers $\mathbf{R}$, as with some measurement of a continuous quantity, like length, weight or time.)

Any random variable X defines certain important events for given values a of one observation. For example, the event denoted $[X = a]$, $(X = a)$, or simply $X = a$, consists of the elementary events that produce a specific observation with value a. The probabilities of these events define an important *distribution* associated with X.

The *probability density function* (pdf), also called *mass function* (pmf), of X is the function f_X (or just f if X is clear from the context) given by

$$f(x) = P(X = x), \quad x \in \mathbf{R}\,.$$

For continuous random variables that can take on infinitely many values in $\mathbf{R}$ corresponding to points on an interval, the values could be nonzero despite the fact that there is an infinite number of them and all their probabilities must still add up to 1. Therefore, this definition still makes sense for continuous RVs X as well, but the calculations get more involved and it is not really required for the purpose of this book.

Tables 11.2 and 11.3 show characteristic properties and facts about commonly used discrete and continuous probability distributions, respectively.

In statistics and data science, understanding a RV may be difficult because of the various values it may take in a variety of situations (say, the salaries of American citizens today.) A human mind makes better sense of a few values, so statisticians are challenged with the data science task of making RVs accessible to human minds. For example, what *single value to choose as most representative* of the values taken on by the RV? How spread are the values of the variable across the population Ω?

The *expected value* (or *mean*) $\mu = E(X)$ of a RV X is the weighted sum of the values of X using as weights the probabilities P_k of the events $[X = x_k]$, i.e., for discrete X,

$$E(X) = \sum_{x \in \mathbf{R}} x f(x) = \sum_k x_k \, P(X = x_k) = \sum_k P_k \, x_k \, ,$$

where the x_ks range over all the values taken on by X; and for continuous X,

$$E(X) = \int_{x \in \mathbf{R}} x f(x) \, dx$$

Table 11.2 Common discrete RVs, probability distributions and their characterizing parameters (pdf $f(x)$, mean μ and variance σ^2)

Distribution $X \sim$ parameters Range of X	pdf	Mean	$\boldsymbol{\sigma^2}$
Standard uniform $X \sim U(1, \ldots, m)$	$\frac{1}{m}$	$(m+1)/2$	$\frac{m^2-1}{12}$
$\{1, \cdots, m\}$			
General uniform $X \sim U(A, \ldots, B)$ $\{A, A+1, \cdots, B\}$	$\frac{1}{B-A+1}$	$(A+B)/2$	$\frac{(B-A+1)^2-1}{12}$
Bernoulli $X \sim Ber(P)$	$P^x(1-P)^{n-x}$	P	$P(1-P)$
$\{0, 1\}$			
Binomial $X \sim B(n, P)$	$\binom{n}{x} P^x (1-P)^{n-x}$	nP	$nP(1-P)$
$\{0, 1, 2, \cdots, n\}$			
Poisson $X \sim Poisson(\lambda)$	$\frac{\lambda^x e^{-\lambda}}{x!}$	λ	λ
$\mathbf{N} = \{0, 1, \cdots\}$			
Geometric $X \sim Geo(P)$	$P(1-P)^x$	$\frac{1-P}{P}$	$\frac{1-P}{P^2}$
$\mathbf{N} = \{0, 1, \cdots\}$			
Negative binomial $X \sim NB(r, P)$	$\binom{x+r-1}{r-1} P^r (1-P)^x$	$\frac{r(1-P)}{P}$	$\frac{r(1-P)}{P^2}$
$\mathbf{N} = \{0, 1, \cdots\}$			
Hypergeometric $X \sim HG(n, N, S)$	$\frac{\binom{S}{x}\binom{N-S}{n-x}}{\binom{N}{n}}$	$n\frac{S}{N}$	$n\frac{S}{N}\frac{(N-S)}{N}\frac{N}{(N-1)}$
$\{L, \cdots, U\}$	where $L = \max(0, n+S-N)$	and	$U = \min(n, S)$

Table 11.3 Common continuous RVs, probability distributions and their characterizing parameters (pdf $f(x)$, mean μ and Variance σ^2)

Distribution $X \sim$ parameters	pdf	Mean	$\boldsymbol{\sigma^2}$
Range of X			
Standard uniform $X \sim U([0,1])$	$f(x) = 1$	1	$\frac{1}{12}$
Interval $[0, 1]$			
General uniform $X \sim U([A, B])$	$f(x) = \frac{1}{B-A}$	1	$\frac{B-A}{12}$
Interval $[A, B]$			
Standard normal $X \sim N(0, 1)$	$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$	0	1
$\mathbf{R} = (-\infty, +\infty)$			
General normal $X \sim N(\sigma, \mu)$	$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/(2\sigma^2)}$	μ	σ^2
$\mathbf{R} = (-\infty, +\infty)$			
Standard logistic $X \sim Logis(0, 1)$	$f(x) = \frac{e^{-x}}{(1+e^{-x})^2}$	0	$\frac{\pi^2}{3}$
$\mathbf{R} = (-\infty, +\infty)$			
General logistic $X \sim Logis(\mu, \beta)$	$f(x) = \frac{1}{\beta} \frac{e^{-(x-\mu)/\beta}}{(1+e^{-(x-\mu)/\beta})^2}$	μ	$\frac{\pi^2}{3\beta^2}$
$\mathbf{R} = (-\infty, +\infty)$			
Standard exponential: $X \sim Exp(1)$	$f(x) = e^{-x}$	1	1
$[0, +\infty)$			
General exponential: $X \sim Exp(\lambda)$	$f(x) = \lambda e^{-\lambda x}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
$[0, +\infty)$			
Double exponential: $X \sim DExp(\lambda)$	$f(x) = \frac{\lambda}{2} e^{-\lambda \lvert x \rvert}$	0	$\frac{2}{\lambda^2}$
$\mathbf{R} = (-\infty, +\infty)$			
Gamma: $X \sim Gamma(\alpha, \beta)$	$f(x) = \frac{1}{\Gamma(\alpha)\beta^\alpha} x^{\alpha-1} e^{-x/\beta}$	$\alpha\beta$	$\alpha\beta^2$
$[0, +\infty)$			
Beta: $X \sim Beta(\alpha, \beta)$	$f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$	$\alpha\beta$	$\alpha\beta^2$
$[0, 1]$			
$\chi^2 : X \sim \chi^2(\nu/2, 2)$	$f(x) = \frac{1}{\Gamma(v/2)2^{v/2}} x^{v/2-1} e^{-x/2}$	ν	2ν
$[0, +\infty)$			
Student t: $X \sim t(\nu/2, 2)$	$f(x) = \frac{\Gamma v+1}{\Gamma v \sqrt{v\pi}} \left(1 + \frac{x^2}{v}\right)^{-\frac{v+1}{2}}$	$0\ (v > 1)$	$\frac{v}{v-2}$
$\mathbf{R} = (-\infty, +\infty)$			$(v > 2)$

because the infinitely many terms in the sum force an integral.
The *variance* of X is

$$Var(X) = E(X - E(X)) = \sum_{x \in \mathbf{R}} (x-\mu)^2 f(x) \quad \text{or} \quad \int_{x \in \mathbf{R}} (x - E(X))^2 f(x) dx$$

i.e., the mean of the RV $Y = X - E(X)$. The *standard deviation* (*std*) of X is

$$\sigma = \sqrt{Var(X)}\ .$$

These parameters $\mu = E(X)$ and $\sigma(X)$ provide useful statistics for humans to get an intuitive understanding of a RV X. Besides having single numbers summarizing a dataset, they are useful in many other ways. For example, the features in a given dataset **X** may be scaled different for different features and common scales may be desirable. One can *center* the data by subtracting from each feature X_i its mean μ_i, so that the mean of the features are then all common and equal to 0. Further, one can also rescale the centered values by their standard deviation σ, i.e., replace X_is by $(X_i - \mu_i)/\sigma_i$. This is the *standard normalization* of the new features to mean 0 and std 1 and are called the *Z-scores*. They exhibit essentially the same statistical properties of the original data **X**, but may facilitate other methods (e.g., Principal Component Analysis PCA with heterogeneous data.)

The concept of random variable can be extended to that of a *random vector* **X** in dimension pD in the standard way, by adding more components in a Cartesian way. Likewise for the mean, variance and standard deviation of a random vector **X**.

For a 2D discrete random vector $\mathbf{X} = (X, Y)$, the *joint pdf* of X is given by $f_{X,Y}(x, y) = P(X = x, Y = y)$ and its marginal pdf's are

$$f_X(x) = P(X = x) = \sum_y P(X = x, Y = y) = \sum_y f_{X,Y}(x, y)\ , \quad \text{and}$$

$$f_Y(y) = P(Y = y) = \sum_x P(X = x, Y = y) = \sum_x f_{X,Y}(x, y).$$

11.1.1 Commonly Used Discrete Distributions

Table 11.2 shows common discrete probability distributions and typical applications.

Discrete Uniform Distribution
$X \sim U(1, 2, ..., m)$ with pdf $f(x) = \frac{1}{m}, \quad x = 1, 2, \cdots, m\ (m \geq 1)$.

One random experiment for this distribution is to toss a fair die once and observe for X the number shown. In this case, $X \sim U(1, 2, ..., 6)$. Another experiment for such a distribution is to choose a random digit and observe for X the digit chosen. In this case, $X \sim U(0, 1, 2, ..., 9)$.

1. $E(X) = \frac{m+1}{2}$.
2. $Var(X) = \frac{m^2-1}{12}$.

In the general case, choosing an integer at random in the interval $[A, B]$ for X gives $X \sim U(A, A+1, ..., B)$ with pdf $f(x) = \frac{1}{C}, \quad x = A, A+1, \cdots B, \quad \text{where } C = (B - A + 1)$.

1. $E(X) = \frac{A+B}{2}$.
2. $Var(X) = \frac{C^2-1}{12}$, where $C = (B - A + 1)$.

Binomial Distribution
$X \sim B(n, p)$ with pdf $f(x) = \binom{n}{x} P^x (1-P)^{n-x} = \frac{n!}{x!(n-x)!} P^x (1-P)^{n-x}, \quad x = 0, 1, \cdots, n.$

One experiment for such a distribution is to toss a fair coin 10 times and observe for X the number of successes (say, heads.) In this case, $X \sim B(10, 0.5)$.

1. $E(X) = nP$.
2. $Var(X) = nP(1-P)$.

There are conditions on the random experiment to yield a RV with this distribution:

- the experiment consists of n identical and independent trials;
- each trial is Bernoulli, i.e., it results in one of two outcomes, success or failure;
- from trial to trial, the probability of success on a trial is P and remains constant. The probability of failure is $1 - P$.

Poisson Distribution
$X \sim Poisson(\lambda)$ with pdf $f(x) = \frac{\lambda^x e^{-\lambda}}{x!}, \quad x = 0, 1, 2, \cdots$

One experiment for such a distribution is to observe for X the number of traffic accident at a certain location during a certain time interval (e.g., rush hour.)

1. $E(X) = \lambda$.
2. $Var(X) = \lambda$.

Geometric Distribution
$X \sim Geo(P)$ with pdf $f(x) = P(1-P)^x, \quad x = 0, 1, 2, \cdots$

One experiment for such distribution is to observe for X the number of (repeated) failures in a Bernoulli trial needed to obtain the first success.

1. $E(X) = \frac{1-P}{P}$.
2. $Var(X) = \frac{1-P}{P^2}$.

Negative Binomial Distribution
$X \sim NB(r, P)$ with pdf

$$f(x) = \binom{x+r-1}{r-1} P^r (1-P)^x, \quad x = 0, 1, 2, \cdots$$

One experiment for a such distribution is to observe for X the number of failures in a Bernoulli trial until the r th success is observed.

1. $E(X) = r\frac{1-P}{P}$.
2. $Var(X) = r\frac{1-P}{P^2}$.

Hypergeometric Distribution
A RV X has a hypergeometric distribution if and only if its pdf is

$$f(x) = \frac{\binom{S}{x}\binom{N-S}{n-x}}{\binom{N}{n}},$$

1. $E(X) = n\frac{S}{N}$.
2. $Var(X) = n\frac{S}{N}\frac{N-S}{N}\frac{N}{N-1}$.

One experiment for such distribution is to draw balls from an urn and for X to observe the number of balls drawn marked as "Success" in choosing n balls without replacement from an urn containing N balls, with s of them marked as "Success" and the remaining $N - s$ balls marked as "Failure."

11.1.2 Commonly Used Continuous Distributions

Table 11.3 shows commonly arising continuous probability distributions and their characterizing parameters.

Uniform Distribution
$X \sim U(A, B)$ with pdf $f(x) = \frac{1}{B-A}, \quad A \leq x \leq B.$

1. $E(X) = \frac{A+B}{2}$.
2. $Var(X) = \frac{(B-A)^2}{12}$.

The special case with $A = 0$, $B = 1$ yields the standard uniform continuous distribution $X \sim U(0, 1)$ in the interval $[0, a]$ with pdf

$$f(x) = 1, \quad 0 \leq x \leq 1 .$$

1. $E(X) = \frac{1}{2}$.
2. $Var(X) = \frac{1}{12}$.

Normal Distribution
$X \sim N(\mu, \sigma^2)$ with pdf $f(x) = \frac{1}{\sqrt{2\pi}\sigma}e^{-(x-\mu)^2/(2\sigma^2)}$.

1. $E(X) = \mu$.
2. $Var(X) = \sigma^2$.

If $X \sim N(\mu, \sigma^2)$, then $Y = aX + b \sim N(a\mu + b, a^2\sigma^2)$.

The standard normal distribution is $Z \sim N(0, 1)$ with pdf $f(z) = \frac{1}{\sqrt{2\pi}}e^{-z^2/2}$.

1. $E(Z) = 0$
2. $Var(Z) = 1$

Logistic Distribution
$X \sim Logistic(\mu, \beta)$ with pdf $f(x) = \frac{1}{\beta}\frac{e^{-(x-\mu)/\beta}}{(1+e^{-(x-\mu)/\beta})^2}$.

1. $E(X) = 0$.
2. $Var(X) = \frac{\pi^2}{3\beta^2}$.

The standard logistic distribution is $Logistic(0, 1)$ with $\mu = 0$ and $\beta = 1$, i.e., its pdf is $f(x) = \frac{e^x}{(1+e^x)^2}$, or equivalently, $f(x) = \frac{e^{-x}}{(1+e^{-x})^2}$, $-\infty < x < \infty$..

This is an important distribution because the popular logistic regression is based on this distribution.

Exponential Distribution
$X \sim Exp(\lambda)$ with pdf $f(x) = \lambda e^{-\lambda x}\ , \quad x \geq 0$.

1. $E(X) = \frac{1}{\lambda}$.
2. $Var(X) = \frac{1}{\lambda^2}$.

The standard exponential distribution is $Z = \lambda X$, with pdf $f(z) = e^{-z}\ , \quad z \geq 0$.

1. $E(Z) = 1$.
2. $Var(X) = 1$.

Double Exponential Distribution
$X \sim DExp(\lambda)$ with pdf $f(x) = \frac{\lambda}{2} e^{-\lambda |x|}$.

1. $E(X) = 0$.
2. $Var(X) = \frac{2}{\lambda^2}$.

Gamma Distribution
$X \sim Gamma(\alpha, \beta)$ with pdf $f(x) = \frac{1}{\Gamma(\alpha)\beta^\alpha} x^{\alpha-1} e^{-x/\beta}, \quad 0 \leq x < \infty$.

1. $E(X) = \alpha\beta$.
2. $Var(X) = \alpha\beta^2$.

Beta Distribution
$X \sim Beta(\alpha, \beta)$ with pdf $f(x) = \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}$,

where $0 < x < 1$ and $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ is a complete beta function.

1. $E(X) = \alpha\beta$.
2. $Var(X) = \alpha\beta^2$.

χ^2 Distribution
$X \sim \chi^2(v)$ if and only if $X \sim Gamma(v/2, 2)$.

1. $E(X) = v$.
2. $Var(X) = 2v$.

When $v = 2$, $X \sim \chi^2(2)$ if and only if $X \sim Exp(2)$.
If $U \sim U(0, 1)$, then $X = -2\ln(U) \sim \chi^2(2)$.
If $Z \sim N(0, 1)$, then $X = Z^2 \sim \chi^2(1)$.

Student's t Distribution
$X \sim t(v)$ with pdf $f(x) = \frac{\Gamma\left(\frac{v+1}{2}\right)}{\Gamma(\frac{v}{2})\sqrt{v\pi}} \left(1 + \frac{x^2}{v}\right)^{-\frac{v+1}{2}}, \quad -\infty < x < \infty\,.$

1. $E(X) = 0$ (where $v > 1$.)
2. $Var(X) = \frac{v}{v-2}$ (where $v > 2$.)

***F* Distribution**
Y with pdf $Y = \frac{X_1/v_1}{X_2/v_2}$, where $X_1 \sim \chi^2(v_1)$ is independent of $X_2 \sim \chi^2(v_2)\,.$

1. $E(X) = \frac{v_2}{v_2-2}$ (where $v_2 > 2$.)
2. $Var(X) = 2\left(\frac{v_2}{v_2-2}\right)^2 \frac{v_1+v_2-2}{v_1(v_2-4)}$ (where $v_2 > 4$.)

The F distribution is commonly used in ANOVA tests.
Connection with the t distribution: if $X \sim t(v)$, then $Y = X^2 \sim F(1, v)\,.$
Connection with the beta distribution: if $Y \sim F(v_1, v_2)$, then $Y = \frac{v_2}{v_1}\frac{X}{1-X}$, with $X \sim Beta(v_1/2, v_2/2)\,.$

Weibull Distribution
X with pdf $f(x) = \frac{\alpha}{\beta}x^{\alpha-1}e^{-x^\alpha/\beta}, \quad 0 \le x < \infty\,.$

Cauchy Distribution
X with pdf $f(x) = \frac{1}{\pi b\left(1+(\frac{x-a}{b})^2.\right)}$

1. $E(X)$ does not exist.
2. $Var(X)$ does not exist.

Multivariate Normal Distribution
$X \sim N_p(\boldsymbol{\mu}, \Sigma)$ with pdf $f(\mathbf{x}) = \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{(\mathbf{x}-\boldsymbol{\mu})'\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})}{2}\right)$
If $\mathbf{X} \sim N_p(\boldsymbol{\mu}, \Sigma)$, then $\mathbf{Y} = \mathbf{AX} + \mathbf{b} \sim N_p(\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{A}\Sigma\mathbf{A}')$
If $\mathbf{Z} = \Sigma^{-1/2}(\mathbf{X} - \boldsymbol{\mu}) \sim N_p(\mathbf{0}, \mathbf{I})$, then $\mathbf{Z} \sim N_p(\mathbf{0}, \mathbf{I})$ and

$$\mathbf{X} = \boldsymbol{\mu} + \Sigma^{1/2}\mathbf{Z} \sim N_p(\boldsymbol{\mu}, \Sigma)\,.$$

The marginal distribution and conditional distribution are normal distributions.

The well-known property that $\left(\frac{X-\mu}{\sigma}\right)^2 \sim \chi^2_1$, where $X \sim N(\mu, \sigma^2)$, can be extended to

$$(\mathbf{X} - \boldsymbol{\mu})'\Sigma^{-1}(\mathbf{X} - \boldsymbol{\mu}) \sim \chi^2_p$$

Trinomial Distribution (Extension of the Binomial Distribution)
If there are three possible outcomes for a trial, where X is the count for the first category with probability p_x, Y is the count for the second category with probability p_y and n is the total number of trials, $Z = n - X - Y$ is the count for the third

category with probability $1 - p_x - p_y$. This random vector $\mathbf{X} = (X, Y)$ follows a trinomial distribution with the probability density function

$$P(X = x, Y = y) = \frac{n!}{x!y!(n - x - y)!} P_x^x P_y^y (1 - P_x - P_y)^{n-x-y},$$

where $x \geq 0$, $y \geq 0$ and $x + y \leq n$ This is an extension of the binomial distribution where the marginal distributions of X, Y, and $X + Y$ are $B(n, p_x)$, $B(n, p_y)$, $B(n, p_x + p_y)$, respectively.

It can be further extended to multinomial distribution with more categories.

11.1.3 Major Results in Probability and Statistics

Bayes' Theorem
If $A, B \subset \Omega$ are events in a probability space with $P(A)P(B) \neq 0$, then

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

More generally, if $\Omega = B_1 \cup B_2 \cup \cdots \cup B_m$ is a partition of the sample space Ω (hence $B_i \cap B_j = \varnothing$ for $i \neq j$), with $P(B_i) \neq 0$, then

$$P(\,B_i \,|\, A\,) = P(A \,|\, B_i)\; P(B_i)/[\Sigma_i P(B_i)P(\,A \,|\, B_i\,)]\,.$$

For the next three results, let $X_i, i = 1, 2, \cdots, n$ be a random sample of size n from a general distribution with mean μ and variance σ^2 and let $\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$ be the mean of the initial subsample of size n.

Central Limit Theorem
If the sample size n is large, regardless of the given distribution of the X_is, the random variable $\bar{X}$ is approximately normally distributed

$$\bar{X} \sim N(\mu, \sigma^2/n), \qquad \frac{\bar{X} - \mu}{\sigma/\sqrt{n}} \sim N(0, 1).$$

That is, the limiting distribution of $W_n = \frac{\bar{X}-\mu}{\sigma/\sqrt{n}}$ will converge to $N(0, 1)$ as $n \to \infty$.

Chebyshev's Inequality
Regardless of the distribution of the X_is, for every $\epsilon > 0$,

$$P(\,|\,\bar{X} - \mu\,| \geq \epsilon\,) \leq \frac{\sigma^2}{n\epsilon^2}\,.$$

Law of Large Numbers
Regardless of the distribution P and the value of $\epsilon > 0$, the distribution of $\bar{X}$ is concentrated around its mean μ, i.e.,

$$P(\,|\,\bar{X}_n - \mu\,| > \epsilon\,) \to 0,$$

for all $n > N_\epsilon$, for some N_ϵ sufficiently large.

11.2 Linear Algebra Background

This section summarizes important facts in linear algebra and matrices that are useful to understand their role and implications for data science.

11.2.1 Fields, Vector Spaces and Subspaces

Concepts such as numbers of various kinds (natural, integer, rational, real and perhaps complex) are usually part of the modern high school curriculum. The concept of a *vector* occurs in courses like physics and chemistry as a quantity that has associated with it not only a real number as a length but also a direction, such as force, velocity and acceleration, usually represented by arrows to signify magnitude (the length) and direction (the tip of the arrow).

These concepts are generalized in Cartesian geometry by abstraction into *scalars* and *vectors* in higher dimensions beyond 3D using the same intuitions (just like distances are in Sect. 1.1) and even into more abstract objects, still called *scalars* and *vectors* by analogy. Mathematicians have produced a systematic exploration of these concepts in the subfield of linear algebra, by distilling out the main concepts necessary for working with vectors and coordinate systems in a more general and effective fashion. The key rules to require are the usual properties of real and complex numbers for the scalars and vectors for vectors spaces. The advantage of this abstract setup is that the thinking only has to be done once by mathematicians, but conclusions can be applied to any set of objects by us all, just as long as they satisfy some key rules, as described next. They can be reasoned with in just the same way one would with lengths and arrows pointing in some direction, even though they may not be them literally.

A *field* F is a set of objects endowed with two operations addition ($+$) and multiplication ($\cdot$) between pairs of them that satisfy the usual properties of their likes with real numbers in $\mathbf{R}$:, i.e., for all elements $a, b, c \in F$:

- Both operations are *associative* ($(a+b)+c = a+(b+c)$), *commutative* ($a+b = b+a$ and $a \cdot c = c \cdot a$), have *neutral elements* (like 0 for addition and 1 for multiplication) and *inverses* (denoted $-a$ for addition and b^{-1} for multiplication, except for 0, so that $a + 0 = a$, $a + (-a) = 0$ and $b \cdot b^{-1} = 1$ for $b \neq 0$.)

- Multiplication distributes over addition, i.e.,

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)\,.$$

(The $\cdot$ for multiplication is, as usual, dropped to ease notation.)

Example 11.2 The set $\mathbf{R}$ of real numbers is a field with the usual operations of addition and multiplication. So are the sets of complex and of rational numbers. Even the set with just two elements $\mathbf{B} = \{0, 1\}$ (the Boolean set) is a field under addition modulo 2 and multiplication as usual, which makes it the smallest field possible. It is even interesting to note that the hours in a clock of only 5 (or 3 or 7, or any prime number of) hours, i.e., the set $\mathbf{Z}_5 = \{0, 1, 2, 3, 4\}$ is also a field if the operations are performed as usual with integers, but taking care of subtracting or adding 5 every time a result goes out to bring it back to this range 0-4 (e.g., $2+3 = 0$ and $2 \cdot 3 = 6 - 5 = 1$ so 2,3 are both additive and multiplicative inverse of one another. These latter are examples of finite fields (know as *Galois fields*, in honor of their discoverer, the French mathematician E. Galois [9] in the 1800s.) □

The concept of vector space requires a scalar field F. In this book, as in common applications, the field of scalars is always the real $\mathbf{R}$ or complex numbers $\mathbf{C}$.

A *(linear) vector space V over a field F* (with elements called *scalars*) is a nonempty set of objects called *vectors* endowed with two operations of addition (+) and multiplication by scalars from F (indicated by just concatenation) that satisfy the usual properties of their likes with real numbers in $\mathbf{R}$ and vectors in Cartesian spaces, i.e., for all elements $\lambda, \mu \in F$ and $u, v, w \in V$:

- addition is *associative* ($(u + v) + w = u + (v + wc)$), *commutative* ($u + v = v + u$), has a *neutral element* (like 0 for addition) and *inverses* (denoted $-u$) so that $u + (-u) = u - u = \mathbf{0}$;
- Multiplication by a scalar is also *associative* and *distributes over* vector addition, i.e.,

$$\lambda(\mu u) = (\lambda\mu)u \quad \text{and} \quad \lambda(u + v) = \lambda u + \lambda v\,.$$

- Scalar multiplication by the multiplicative identity scalar leaves vectors unchanged, i.e., $1v = v$ for all $v \in V$.

Example 11.3 If $d > 0$ is an integer and F is any field, the Cartesian product $V = F^d = F \times \cdots \times F$ (d times) is a vector space over F with vector addition and multiplication defined componentwise. (To ease the notation, a column vector is written as the transpose (x') of a row vector $x = (x_1, \cdots, x_d)$.) The familiar vector space $\mathbf{R}^d$ (like the 2D plane and 3D spaces) are thus recovered as particular cases with the real number field. □

A subset W of a vector space V is a *subspace* if it satisfies the properties required by the definition of vector spaces with the operations inherited from the container space V, i.e., W is closed under addition, taking additive inverses and multiplying its elements by arbitrary scalars.

A subspace is never empty, as it must contain at least the neutral element $\mathbf{0}$ under addition.

Example 11.4 If $F = \mathbf{B}$, the smallest field (in fact, the Galois field $\mathbf{Z}_2$), the vector space of dimension d over $\mathbf{B}$ is called the (Boolean) *hypercube* of dimension d and consists of binary vectors of length/dimension d. It plays an important role in Shannon's theory of communication and error-detecting and error-correcting codes for robust communication through a noisy channel. □

11.2.2 Linear Independence, Bases and Dimension

The following concepts make sense in an arbitrary vector space V over a field F (which can be thought of as $\mathbf{R}$ for the purposes of this book.)

A *linear combination* in V is a vector obtained as a sum of scalar products

$$\lambda_1 v_1 + \lambda_2 v_2 + \ldots + \lambda_n v_n$$

of some finite subset of elements $v_1, v_2, \ldots, v_d \in V$ with scalars $\lambda_1, \lambda_2, \ldots, \lambda_n \in F$, The set of all possible linear combinations of vectors $v_1, v_2, \ldots, v_n \in S$ is the (linear) *span* of a subset $S \subseteq V$ and it is denoted $\mathscr{L}(S)$.
The subset S is *linearly independent* if no element in it is a linear combination of any other elements in S. It is called *linearly dependent* otherwise.

Example 11.5 The vectors $u = (1, 2, 3)'$, $v = (2, -1, 4)'$ and $w = (0, -5, -2)'$ are linearly dependent since $2u - v + w = 0$, i.e., $v = 2u + w$. A set containing the zero vector $\mathbf{0}$ is always linearly dependent as $\mathbf{0}$ can always be obtained from others as a linear combination with scalar coefficients 0. □

Example 11.6 The vectors $u = (1, 1, 1)'$ and $v = (1, 0, -1)'$ are linearly independent because neither one is a linear combination (in this case, that means a scalar multiple) of the other: $\lambda u = (\lambda, \lambda, \lambda) = v = (1, 0, -1)$ implies that $\lambda = 1 = -1 = -\lambda$, which is nonsense (except perhaps in the Galois field $\{0, 1\}$.) If a linear combination is 0, e.g., $\lambda u + \mu v = 0$ for some $\lambda, \mu \in \mathbf{R}$, then $(\lambda + \mu, \lambda, \lambda - \mu)' = 0$ and this would mean that both λ and μ are the scalar 0, the trivial linear combination producing the vector $\mathbf{0}$. □

It is evident that $\mathscr{L}(S)$ is always a (linear) subspace of V. For a vector space V, it is of much interest to find a smallest subset B that generates the *full* space by doing linear combinations with scalars from F.

A (linear) *basis* of a vector space V is a subset of vectors B with the smallest number of elements that spans the space V, i.e., $\mathscr{L}(B) = V$. The (linear) *dimension* of V is the cardinality (number of elements) of such a basis.

Such a set has to be linearly independent since a vector that is a linear combination of others can always be excluded to obtain a smaller such set with the same linear span. Naturally, a vector space V that has a finite basis B is called *finite dimensional*. Not all vector spaces are finite dimensional (for example, the space consisting of one-way infinite sequences $(x_1, x_2, \cdots)$ of real numbers with componentwise operations. Such spaces will be of no concern in this book since they are out of reach for conventional computers, unless finite dimensional approximations are used.)

Example 11.7 The vector space $\mathbf{R}^d$ is finite dimensional since it has a basis $B = \{e_1, e_2, \ldots, e_d\}$ where $e_i = (0, \ldots, 1, \ldots, 0)'$ has 0 coordinate everywhere except at the i^{th} position, where $e_i = 1$. It is easy to verify that every vector $v = (v_1, v_2, \ldots, v_d)'$ is a linear combination of the basis elements $v = v_1 e_1 + v_2 e_2 + \ldots + v_d e_d$. This basis is called the *standard basis*, so the dimension of $\mathbf{R}^d$ is $dim(\mathbf{R}^d) = d$. □

A subspace always has dimension no larger than that of the entire host space V, although this is not entirely obvious from the definitions above.

A subspace W of a finite dimensional vector space of finite dimension is never larger than $dim(V)$, i.e. $dim(W) \leq dim(V)$.

Example 11.8 Given a basis B of a vector space V, every vector must be expressible as a linear combination of the elements of B. The coefficients in this combination are *unique*, because two different combinations being equal would lead to a combination for **0** (by transposing terms to just one side and re-grouping) with some nonzero coefficient(s), which would in turn lead to one vector being expressible as a linear combination of the others, nonsense since elements in B are linearly independent. These coefficients can be put together into a vector $(\lambda_1, \lambda_2, \cdots, \lambda_d)$ to obtain a Cartesian object over the scalar field so one can get rid of the abstract nature of the objects in the original vector spaces V and handle them as usual business in $\mathbf{R}^d$ for a vector space of dimension d, or in F^d for some scalar field F. □

11.2.3 Linear Transformations and Matrices

In a linear space, as the basis elements go, so follows the rest of the space, as illustrated by the following facts.

A *linear transformation* (or *mapping*) $T : V \rightarrow W$ between two vector spaces over the same field F is a function that preserves linear combinations, i.e, for every pair of scalars $\lambda, \mu \in F$ and of vectors $u, v \in V$,

$$T(\lambda u + \mu v) = \lambda T(u) + \mu T(v)\,.$$

Example 11.9 In the 2D Euclidean space, any rotation counterclockwise (say 45°) is a linear transformation. A perpendicular projection onto a 1D line (such as the 45° line) is also a linear map. However, a translation by a fixed nonzero vector v_0 like $(1, 1)$ given by $T(u) = u + v_0$ is not since $T(u + v) = u + v + v_0 \neq (u + v_0) + (v + v_0) = (u + v) + 2v_0$. (The latter are called *affine* transformations.) □

There are several interesting subspaces associated with a linear map.

The *kernel* (or *null space*) K of a linear map $T : V \to W$ consists of all vectors in V that collapse to the $\mathbf{0}$ of W under T, i.e.,

$$K = \{u \in V | T(u) = \mathbf{0}\} = T^{-1}(\mathbf{0}) .$$

The *range* of T is the set $T(V) = \{T(u) : u \in V\}$, i.e., the set of all images $T(u)$ obtained by ranging u through all elements of V.

The action of any linear map is entirely determined by its mapping of the elements in any basis of V. Linear maps can be represented by yet a richer concept of a number over a given field F.

A *matrix* A (or $A_{d\times k}$ if dimensions are important) over a field F is a rectangular array of dk elements of F (d rows and k columns.) The matrix A *associated to a linear map* T for given bases B, B' of V, W respectively, has as i^{th} column (out of dim(V) $= d$ columns) of dimension $k = dim(W)$ consisting of the coefficients of the image of the i^{th} element b_i in B with respect the basis B' ; i.e., $A = [T(b_1)', \ldots, T(b_d)']$.

Such a matrix representation for the linear transformation T is *not unique* as it depends on the choice of bases B, B', although it is of course unique if the bases are fixed. Once B, B' are fixed, it can serve as a full representation of T. For example, one advantage is that the image of an arbitrary element $u \in V$ can be obtained as

$$T(u) = Au' ,$$

i.e., just as the *matrix product* of A and the column vector u' with coefficients the components of u with respect to the basis B. Thus, just like a linear function in one variable is given by $f(x) = mx$, where m is the slope of the line representing its graph, so is the concept of matrix A an abstraction of the concept of slope to a generalized vector space. In particular, one can regard a given dataset (a 2D table) as a matrix A defining some linear transformation and whose decomposition(s) below provide some high-level analysis of the dataset (examples are found in Sects. 2.1 and 9.3.)

11.2.4 Eigenvalues and Spectral Decomposition

A lot of other essential information about T is hidden inside A.

An *eigenvalue* of A (and so of T) is a scalar λ such that $T(u) = \lambda u$ for some nonnull vector u, i.e., T only stretches (if $|\lambda| > 1$) or shrinks (if $|\lambda| < 1$) u by a factor of λ (perhaps reversing orientation if $\lambda < 0$) without changing its direction Such a vector u is called an *eigenvector* of T for the eigenvalue λ.

Therefore, if T happens to have a set of linearly independent eigenvectors that form a basis of V, understanding the effect of the transformation T becomes evident.

Example 11.10 If the eigenvectors are e_1 and e_2 in the 2D Euclidean plane and T has two eigenvalues $\lambda_1 = 2$ and $\lambda_2 = -1$, the vector $v = (3, 4)$ decomposes as

$V = 3e_1 + 4e_2$, and then the effect of T on v is $T(v) = 3T(e_1) + 4T(e_2) = 3(2e_1) + (-1)e_2) = 6e_1 - 4e_2$, i.e., to stretch the e_1 component by a factor of 2 and the e_2 component by a factor of (-1) (reversing orientation) without stretching since $|\lambda| = |-1| = 1$ and combine the results back to get $T(v) = (6, -4)$. If the eigenvectors happen to be rotated with respect to the standard basis, changing the basis to a basis of eigenvectors will simplify the expression of T into a just a diagonal matrix (where the entries are zero (0) off the main diagonal (entries $A_{ij} = 0$ where $i \neq j$.) □

The eigenvectors of a linear transformation $T\colon V \to V$ within a space V can be computer manually by solving a system of equations determined by $T(u) = \lambda u$, for each eigenvalue λ root of the polynomial equation of degree d vanishing the determinant

$$|I - \lambda A| = 0\,,$$

one equation for each component, where I is the identity matrix of the same dimension. Alternatively, they can be computed (perhaps only approximately) by using software libraries in a computational platform (some are described in Sect. 11.6.)

Spectral Decomposition
If $A = A_{d\times d}$ has d different eigenvalues $\lambda_1, \ldots, \lambda_d$ and their eigenvectors are linearly independent, then A has a factorization (the spectral decomposition *of A) of the form*

$$A = Q\Lambda Q^{-1}\,,$$

where Q is an invertible matrix consisting of the eigenvectors of A (as columns) and Λ is a diagonal matrix with the eigenvalues in the main diagonal (i.e., all entries off the diagonal are 0, hence denoted just by $\Lambda = \mathbf{D}(\lambda_1, \ldots, \lambda_d)$, or just $(\lambda_1, \ldots, \lambda_d)$.)

Example 11.11 For the transformation in Example 11.10, $\Lambda = \mathbf{D}(2, -1)$ and the columns of Q are $u_1 = (1, 0)'$ and $u_1 = (0, 1)'$, i.e., $Q = I$ is the identity matrix (which it its own inverse) and A is its own spectral decomposition $A = \Lambda$. If T is a rotation as in Example 11.9, then the matrix of T with respect to the standard basis is $A = [s(1, 1)' \quad s(-1, 1)']$, where $s = \sin(45°) = \cos(45°) = \sqrt{2}/2$ is a shrink factor to get the rotation of the standard vector $(1, 1)'$ to have length 1, i.e., $s = \sqrt{2}/2$. The eigenvalues of A are the solution to the quadratic equation

$$|\ I - \lambda A\ | = \ \ [(1, 0)'(0, 1)' - \lambda A\ | = \lambda^2 - s\lambda + (1 + s^2)\,,$$

which are not real, but have a complex imaginary component, so A is not diagonalizable. □

All symmetric matrices have real eigenvalues (possibly with so-called *multiplicities* when two eigenvalues are equal, as in the solution of $(\lambda - 1)^2 = 0$, in which case the eigenspace generated by the eigenvectors of such an eigenvalue have dimension larger than 1.) Nonetheless, in the more general case of a nondiagonalizable matrix, a similar decomposition is possible if one is willing to substitute the entries in the spectral decomposition by *square blocks* submatrices (of dimension equal to the dimension of the eigensubspaces corresponding to such eigenvalues of multiplicity larger than 1.)

In Euclidean spaces, every matrix A has a Jordan Canonical Form (*or* Jordan decomposition) *as a product*

$$A = P \Lambda Q \ ,$$

where Λ is a block superdiagonal matrix with eigenvalues in the tridiagonal band and possibly some ones (1s) in the first superdiagonal (entries λ_{ij} with $j = i + 1$.)

In the most general case where the range space W of T has different dimension from V, a decomposition is still possible, but the matrices may no longer be square, e.g., Λ may be a truncated diagonal matrix and P, Q must have the appropriate dimensions for the matrix products to make sense (in a product ΛQ, the number of columns of Λ must match the number of rows of Q.) This factorization is called the *Singular Value Decomposition (SVD)* of A. This decomposition has found many interesting applications in data science, for example in Latent Semantic Analysis (LSA), where the number of documents (data points) $n = d$ is different from the number of distinct words (features) p appearing in all the documents. (More details can be found in Sects. 4.3 and 10.2.)

Finally, Euclidean and Boolean spaces are yet richer in structure. The length of the projection of a vector u onto another v can be computed using the so-called *dot product* (also *scalar product*), given by

$$\gamma = u \cdot v = \sum_{i=1}^{d} u_i \, v_i = \|u\| \|v\| \cos(\alpha)$$

(sometimes simply denoted by the juxtaposition uv .) The projection of u onto v is then γv (and also γu of v onto u.) So, vectors orthogonal to one another have dot product 0 and a matrix A is *orthogonal* exactly when every pair of its column vectors are orthogonal. They represent linear transformations obtained by rotating the standards basis vectors $\mathbf{e}_i$ to other vectors (the columns in A) so as to preserve their orthogonality between any pair of them. In particular, the *length* of a vector (distance of the tip from the origin $\mathbf{0}$) is then given by $\|u\| = \sqrt{u \cdot u}$. This structure applies as well to Boolean hypercubes, which are metric spaces with the resulting Hamming distance between u, v given by $Ham(u, v) = \sum_{i=1}^{d} u_i \oplus v_i$, the sum of the XOR (exclusive *or*) of their components.

Further details can be found in any standard textbook in linear algebra, e.g., [20].

11.3 Computer Science Background

This section summarizes important facts in computer science that are useful to understand its role and implications for data science, as we well as its scope and limitations.

11.3.1 Computational Science and Complexity

The working definition of data science is problem solving to extract information about questions concerning a population from samples of it. To really make sense of this definition, the question of *What exactly is information?* would need to be addressed. That is precisely the subject matter of computer science and so it plays a central role in any science, data science in particular.

The subject matter of computer science would appear to go along the lines of *the science concerned with the study of computers*. This definition turns out to be inadequate for a variety of reasons. Among many others, there is a difference to be made between information and its embodiment, usually called *data*, as well as between information processing and *data communication* (which is what Shannon's concept of information aims to answer.) Second, there is no other science like it (like car science, or microscope science, or even human science; but there is computer science!) Third, a case can also be made that there are different kinds of information, such as *knowledge* and even *wisdom*. The funny thing is, humans and living organisms alike are all remarkable users of information, yet no one can define precisely what it is. Thus, defining information is a far more difficult challenge that has remained and will probably remain unsolved for a good time to come [5].

Another way to define computer science would be along the lines of *the study of information and its processing*. But alas, in addition to answering "What is *information*?", this definition would require an answer to the question "What is a *processor*." Therefore, it is not surprising that the digital age has been born more directly concerned with data processing *devices*, particularly in their specific electronic implementation, i.e., the conventional computers of our time.

Yet another way to characterize a science is by defining the scope or aspect of the world it is concerned with. Like data science, the scope of computer science can then be regarded as concerning computational problems in the sense that they all call for certain types of inputs as data (as described in Sect. 1.3) to a problem and spell out an expectation in terms of the kind of result to be produced as an answer.

> An *algorithmic problem* (AP) is a list of questions, each fully expressible as a finite string over a fixed finite alphabet of characters, with an answer determined by the question and also expressible as a finite string over the same alphabet.

APs include all data science problems dealt with in this book, but can be much more general in terms of the questions to be answered and the solutions provided.

Example 11.12 Primality testing is one of the most representative examples. The alphabet is decimal (another problem could be in binary) and questions are follows.

[REC(Primes)] **PRIMALITY TESTING**

INSTANCE: A decimal string x
QUESTION: Does x represent a prime integer (no proper factorization)?

There is an infinite number of questions (one for each x, e.g., $x = 19$ or 32), but for each there is a unique answer (e.g., *Yes*, *No* respectively.)

On the other hand, the outcome of a given toss of a specific coin (heads or tails) is not an AP because first, it is probably impossible to fully specify a coin with a finite string containing full information to determine it as a physical object different from other coins, but more importantly, the answer to a question is not unique and fully determined by the coin (even for the same coin) in advance. □

APs come in three flavors, *recognition problems*, *generation problems* (how to produce all and only the strings in a given language, like Python 7.9, from some specification of it) and *compaction problems* (how to produce finite strings fully describing potentially infinite sets such as a certain natural or computer programming language.) Recognition problems have proved challenging enough that most of what is understood by the foundations of computer science today is concerned with them. Primality testing [**REC(Primes)**] is a typical example. A recognition problem is specified by a set of strings (called a formal *language*) L over some finite alphabet Σ as follows.

[REC(L)] **MEMBERSHIP**(L)

INSTANCE: A string x over alphabet Σ
QUESTION: Does x belong to L ("Yes" or "No")?

Of course, many other types of APs are of interest in their own right in practice. Common problems are [**SORTING**] and [**SEARCHING**]. A particular type is particularly relevant to data science because they are about optimization of the type involved in machine learning and data science algorithms.

[TSP] [**TRAVELING SALES PERSON**]

INSTANCE: A weighted finite graph G consisting of vertices V and (undirected) edges joining them, weighted by nonnegative integer numbers
QUESTION: What is a tour of the graph visiting all nodes with the shortest total sum of (the) weight(s) among all possible tours?

[SAT] [**SATISFIABILITY**]

INSTANCE: A Boolean formula $\varphi(x_1, \cdots, x_k)$ of k Boolean variables
QUESTION: Is φ satisfiable, i.e., does it evaluate true for any one assignment of truth values of its variables $x_1, \cdots, x_k$?

For example, the formula $\varphi(x, y) = (x \, and \, y)$ is satisfiable (if both x, y are assigned the value *True*), but $\varphi(x, y) = (x \, and \, \bar{x})$ is not ($\bar{x}$ is the Boolean negation of x.)

[**MORT**(d)] [**MORTALITY**](d)

INSTANCE:	A finite set S of matrices of size $d \times d$ with integer entries
QUESTION:	Is the set S mortal?

Here, a set of matrices is *mortal* if some product of elements from S (in some order and possibly with repetitions of the choices) multiply out to the **0** (null) matrix. Although with numbers ($d = 1$) this is only possible if some element in S is 0 itself, as generalized numbers, products of two nonzero square matrices of size 2×2 may actually be the null matrix (like, two diagonal matrices with a single 1 in their diagonal at different positions.)

What constitutes an acceptable solution of an arbitrary AP *in general*? Intuitively, one would expect some kind of device that answers the questions in the AP, i.e.,

[S1] the device must be able to read any of the questions x in the problem as an input, work on it for a *finite* period of time and eventually stop (halt) the processing, after
[S2] returning the *true answer* to question x; in addition,
[S3] the same device must do so systematically for every question x in the AP without any further modification, after its design is complete.

A *solution* to an AP is usually some sort of model or program/code, ultimately to be translated into some sort of physical device to be run, typically a conventional computer, but possibly very different (e.g., a human brain or a quantum computer) that satisfies conditions [S1]–[S3] above.

What constitutes an acceptable specification of a device (which may quickly turn into wishful thinking for a magical solution) is beyond the scope of this book. (Alan Turing's proposal in the foundational paper that marks the birth of computer science [21] in 1936 is considered to be the simplest and rigorous standard.) Nevertheless, one can get an understanding of what is now called a *Turing machine* with an intuitive equivalent, the concept of an *algorithm*. Computer scientists generally agree that the following thesis is the appropriate characterization of what constitutes an acceptable solution to an AP, arrived at after searches lasting over half a century of computer science by computer scientists all over the world for a sound, objective and appropriate definition of "computer" and discovering that many other possibilities are just Turing machines in disguise. That includes concepts such as algorithms, computer programs, mechanical procedures, automatic procedures, automatable procedures and many others.

The Church–Turing Thesis
Every algorithm solving an algorithmic problem can be specified and implemented as a Turing machine.

In a nutshell, an algorithmic problem is solvable if some device exists (which is not quite to say it will be easy for humans to find) implementing some algorithm that receives input data encoded as strings, crunches symbols with a *fixed program but unbounded storage and running time*, and returns the answer encoded in a finite string expressing the answer to *each* and *all* question(s) in the AP, correctly and systematically. Solving an AP requires either a full and rigorous description of a device that must ideally satisfy the three conditions [S1]–[S3], or, alternatively, some argument that such a device could not possibly exist (i.e., that the problem is *unsolvable*), independently of our state of knowledge today or in the future.

In particular, strictly speaking, a machine learning algorithm is such a procedure, but usually the problem is only vaguely specified in advance. It produces a device (such as a neural net, decision tree or the like) to answer some set of questions that may not be exactly the ones originally intended, or produce answers that may not be correct, or fail to provide answers to some of the intended questions but not to all of them. They have to be algorithms nonetheless, executable on a physical device, either practically built (such as a conventional digital computer) or at least an ideal device that could possibly be built (such as a Turing machine.)

Much of computer science is devoted to elucidating the solvability status of APs in general, either in theory or in practice. Turing [21] demonstrated that there exist many *unsolvable* problems beyond the reach of conventional computers. For example, **MORT**(d) is algorithmically unsolvable, according to the definition above, i.e., no algorithm can ever be created or discovered by any genius that will answer the question of mortality systematically for all sets of integer matrices of size 3×3. In fact, most algorithmic problems are actually unsolvable (even ideally) and hence, *it is necessary to relax the conditions* [S1]-[S3] *on a solution and accept imperfect solutions that err or fail to produce exact and true answers* in a systematic fashion (systematicity is probably harder to give up since it requires intensive labor in changing devices or programs as the questions change.)

Within the realm of solvable problems, there is a further distinction that is very material for computer science and data science in particular. The definition of solution to an AP does not impose any restrictions on how long it will take for the device to produce the answers to the questions. In practice, this can become a huge issue, if the amount of resources (typically *time* or *memory*) taken by the solution device is too large, as illustrated in Table 11.4.

Table 11.4 Small and large growth rates of a running time for the brute force solution of the **[SAT]** problem

n	1000nlog n	100n^2	n$^{log\, n}$	...	$2^{n/3}$	2^n
20	.09 sec	.04 sec	4 sec	...	.0001 sec	1 sec.
50	.3 sec	.25 sec	1.1 hr	...	.1 sec	35 years
100	.6 sec	1 sec	220 days	...	2.7 hr	10^{10} cent.
200	1.5 sec	4 sec	125 cent.	...	3×10^4 cent.	?
1000	10 sec	2 min	?	...	?!	??!!

Thus, for example, to determine whether a formula containing $n = 100$ symbols (a small size in typical applications of the algorithm) is satisfiable or not would take

$$2^{100} sec/10^9 \times 3600 \times 24 \times 365 \times 100 \geq 10^8 \text{ centuries}.$$

This is obviously unacceptable since any solution device would probably take longer than the age of the universe. The trouble is, however, that this algorithmic problem **[SAT]** of satisfiability of a Boolean formula (defined above) is not just an intellectual curiosity of interest only to logicans and mathematicians. The hard fact of the matter is, *literally hundreds of problems of immediate economic importance in such diverse fields* as scheduling, combinatorial optimization (e.g., linear programming), secure secret communication (e.g., factoring and primality testing) and even national security can all be shown to be just as hard as (if not harder than) **SAT**, including **TSP**. These considerations explain, at least in part, why one of the most important problems in computing today is precisely to find new computational strategies that do overcome the complexity barrier posed by this combinatorial explosion.

As an area of computer science, *computational complexity* aims to quantify and analyze the efficiency of solutions to algorithmic problems, and hence their *inherent difficulty* if they are to be solved by any device that qualifies as a computer. To understand the basic paradigm, it is necessary to realize that the various instances of an algorithmic problem will pose various degrees of difficulty to a solution device. The situation is in contrast with other definitions of efficiency in physics and chemistry, where the efficiency of a procedure is defined by some sort of percentage measuring the amount of useful output compared to the amount of resource given. For example, in thermodynamics one measures the efficiency of a thermal engine as, say, 80% to indicate that the engine in capable of devoting eighty percent of the given amount of energy in useful work output, while spending 20% in doing its internal operations. The efficiency of a chemical reaction to produce alcohol, for example, can likewise be given as some percentage of the amount of raw materials fed to the reaction to produce it. The yield of an investment is usually given by a percentage as well.

Such a simple minded scheme cannot be used to quantify the efficiency of an algorithmic/computational problem simply because there are usually many questions to gauge the performance on; moreover, each question may take more or less work to answer. (e.g., it is much harder to figure out whether a longer number 7583 is prime integer than to decide whether 5 is prime.) An example is probably already familiar to the reader. The solutions to [**SORTING**], namely sorting algorithms, require performing a number of operations. The input size can be taken to be, for example, the number of items to be sorted, or the number of bits necessary to describe the unsorted instance file. In a comparison-based sort, it is to be expected that the bulk of the effort by a solution algorithm will be spent on performing comparisons between the various items in the input instance. Thus, a reasonable choice of resource to quantify to perform the analysis is comparisons, although it is certainly not the only one. With these measurements, the **worst-case** efficiency (or complexity) of `QUICKSORT` , is $O(n \log n)$. Surprisingly, the **average case** efficiency of `QUICKSORT` is $O(n \log n)$ as well.

In general, the measure of input size n for an arbitrary algorithmic problem in Turing computation is naturally the length on the string x describing the instance. The prime resources used in computational studies of complexity is based on the mode of operation of the solutions, which ultimately go back to Turing machines (by virtue of the Church–Turing thesis.) They are *time* (as measured by the number of elementary steps or instructions executed by the algorithms to return the answers) and *space* (the amount of memory used by the machine in the course of calculating the answer to a given instance.) Since the problem usually has many questions given by strings of variable length n, this time or memory spent becomes a function t_n or s_n of n. Since counting can become very difficult, one is only interested in the order of growth $O(t_n)$ or $O(s_n)$ of these functions for a given solution.

Once the complexity of a **particular solution** is defined, one can then make precise the complexity of an algorithmic **problem** as the lowest possible complexity achieved among all of the many possible solutions, either for time or space.

Given a bound $\{f_n\}_n$ on the amount of resource ρ available, the *complexity class* $\rho(f_n)$ consists of all algorithmic problems that admit (deterministic) solutions with complexity $O(f_n)$ units of the given resource ρ.

Thus, in the running example, **[SORTING]**, belongs to the class **TIME**$(n \log n)$ (thanks to the `QUICKSORT` algorithm), but it also belongs to the class **DTIME**(n^2), as does **[MINIMAL SPANNING TREE]**. **[SORTING]** does not belong, however, to the class **DTIME**(n) since `QUICKSORT` is an optimal general-purpose sorting algorithm.

Based on these and other considerations to become evident below, it has been proposed that the class of algorithmic problems that admit solutions that take at most a **polynomial** amount of resource $O(n^d)$ in the size n of the input, should be considered the class of *feasibly solvable* problems. The precise definition is as follows.

An algorithmic problem is *feasible* if there exists a solution whose complexity is a polynomial function in $O(n^d)$, for some integer constant $d \geq 1$. The class of *tractable/feasible* problems is the class of algorithmic problems that can be solved in polynomial time, i.e.,

$$\mathbf{P} = \bigcup_{d \geq 0} \mathbf{DTIME}\,(n^d)\,.$$

*Example 11.13 (***REC**(Primes)*)* is a feasible problem solvable in time $O(n^3)$ [1], a fact that took over 2300 years to establish. (The ancient Greeks were aware of the problem since they discovered there are infinitely many prime numbers.) □

However, the feasibility of the related solvable problem [**FACTORIZATION**] of integers into prime numbers remains a challenge. Moreover, the status of most *optimization* problems of interest in data science, such as **[TSP]** or it numerous equivalents, remains unknown, i.e., some unbeknownst clever algorithm (like for primality testing) may exist that puts them in **P**. Or, perhaps they do not belong in

P at all. No one knows and some computer scientists claim humans may not have the tools to figure the answer out yet. In fact, there are outstanding prizes (like $1M dollars) for anyone with a valid proof for an answer to this vexing question for over half a century now. More details about Turing computation and complexity can be found in any textbook in theoretical computer science, e.g., [11].

Nevertheless, there is some good news concerning this class of problems (which most computer scientists believe is as good it it is going to get.) When confronted with an optimization problem (such as the shortest route in the **[TSP]** problem), *no one can stop us or any dumb machine from randomly guessing an ordering of the cities for a tour* and figure out its cost by simply adding up the costs of the edges as given for that tour. A lucky person will get a tour after a few trials that can hardly be improved with further trials. As dumb luck may have it, that may well be the optimal tour that cannot be improved at all by any other means, or at least be just *good enough* for a practical solution. Most other optimization problems exhibit similar properties, so they can be collected in a class of their own, the class **NP** (**N** stands for "nondeterministic" because their solution can only be found by guesswork, but they can be verified deterministically in polynomial time, although it may be impossible to find them out without guessing them nondeterministically.) The question of whether the two classes of algorithmic problems are identical, i.e., whether **NP** =?**P**, has remained an outstanding open problem in computer science for over half a century now, and it is recognized today as perhaps the most profound and consequential question for many sciences, data science included.

11.3.2 Machine Learning

Machine learning (ML) is commonly understood [18] as the design of algorithms/programs that improve their performance with "experience," i.e., accessing and processing data about a problem. In general, ML faces three major issues:

- *Problem representation*
  Humans have to represent the problem as a data science problem (as discussed in Sect. 11.4) as well as gathering and cleansing appropriate data (possibly including dimensionality reduction) to solve the problem. Successful data extraction often requires deep understanding of the domain of interest the problem comes from.
- *Optimization*
  ML solutions usually formulate learning tasks as an optimization problem. Typical methods used to solve these optimization problems include gradient descent, Lagrange multipliers and Maximum Likelihood.
- *Evaluation*
  Evaluating Machine learning solutions requires quantification of how well a solution works on new datasets, a nontrivial task (described in the previous subsection.) It is quite possible that the same procedure can yield very different results based on different evaluation methods.

Machine Learning (ML) can thus be better characterized as a collection of optimization procedures for learning algorithms. The most common types of techniques used to implement Machine learning solutions are described next.

Gradient Descent
This is one of the most frequently used tools in ML. The most common supervised learning algorithm, Backpropagation (described in Sect. 2.2) used in training neural networks, is a gradient descent algorithm.

Example 11.14 Gradient descent is used to find the minimum (or maximum) of a given multivariable function $f(x_1 \dots x_n)$, i.e.,

$$\min_{x_1,\dots,x_n} f(x_1, \dots x_d)$$

For such a function, the direction (of the many inputs x_i) of fastest change from a point $\mathbf{x}$ (the steepest slope in the container space $(d+1)$D space of the graph of the function) is indicated by a vector called the *gradient* whose components are the partial derivatives with respect to each of the independent variables in the function evaluated at the point $\mathbf{x}$, i.e.,

$$\nabla f(x_1, \dots x_n) = \left[\frac{\partial f}{\partial x_1}(\mathbf{x}), \dots, \frac{\partial f}{\partial x_n}(\mathbf{x}) \right]$$

Naturally, this direction changes with the specific point $\mathbf{x}$. To minimize f, one can start $(t = 0)$ at a nearby point $\mathbf{x}^0$ and take a small step in the direction that is opposite to the gradient direction, i.e., that of $-\nabla f(\mathbf{x}^0)$. To update the candidate optimal location $\mathbf{x}^{(t)}$ for a minimum, the x_i in iteration $t > 0$ is given by

$$x_i^{(t)} = x_i^{(t-1)} - \epsilon \frac{\partial f}{\partial x_i}(\mathbf{x}^{(t-1)})$$

where ϵ is a small positive constant. A choice of a very small ϵ may take a long time to converge, while a large ϵ may jump over the optimal location for a minimum value of f. For convex functions (with a single global minimum), gradient descent is guaranteed to find the location where f attains its minimum value, the optimal solution. □

Max-Likelihood Estimation
Max-Likelihood Learning, also called Max-Likelihood Estimation (MLE), is a general method for learning probabilistic models. MLE learns the parameters of a distribution from observations. Given independent and identically distributed (i.i.d) observations, $x_1 \dots x_n$, MLE aims to optimize the function

$$\max_{\theta_1,\theta_2,\dots,\theta_k} P(x_1 \dots x_n \mid \theta_1, \theta_2, \dots, \theta_k)$$

where $\theta_1, \theta_2, \ldots, \theta_k$ represent the parameters of the distribution. To estimate these parameters, one can use a gradient ascent procedure (analogous to the gradient descent procedure above, moving in the direction of $\nabla P(\Theta^0)$.)

Lagrange Multipliers
This method is common in multivariable calculus to optimize functions with several variables, usually when there are constraints on the variables.

Example 11.15 Support Vector Machines (SVMs) employ this technique of optimization. In order to maximize a function $f(x_1, x_2)$ subject to a constraint between x_1 and x_2 that $g(x_1, x_2) \geq 0$, one considers a multiplier λ (called a *Lagrange multiplier*) and reformulate the problem as an optimization problem with an extra variable λ to maximize

$$L(x_1, x_2, \lambda) = f(x_1, x_2) + \lambda g(x_1, x_2)\,.$$

To solve the optimization problem in terms of the variables x_1, x_2 and λ, the Karush–Kuhn–Tucker conditions (KKT conditions) specify that one can solve the optimization problem in terms of three constraints.

$$g(x_1, x_2) \geq 0$$

$$\lambda \geq 0$$

$$\lambda g(x_1, x_2) = 0$$

Thus, whenever $\lambda \neq 0$, $g(x_1, x_2) = 0$. In SVMs, points corresponding to nonzero Lagrange multipliers are referred to as support vectors. Intuitively, they represent the boundary of the function. The stationary points at $\lambda \neq 0$ turn out to be the points at the boundary of the function, i.e., where $g(x_1, x_2) = 0$. □

Gaussian Learning
Gaussian distributions (described in Appendix 11.1) are widely applied in unsupervised learning, including Gaussian Mixture Models and autoencoders. For instance, a single variable Gaussian distribution is defined by 2 parameters, the mean μ ($-\infty < \mu < \infty$) and variance ($\sigma^2 \geq 0$). The distribution is defined as follows.

$$N(X \mid \mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

$$E(X) = \mu\,,$$

$$Var(X) = \sigma^2\,,$$

$$Mode(X) = \mu\,.$$

In a multivariable situation of dD vectors ($d > 1$), the Gaussian parameters include a d-dimensional mean vector and $d \times d$ covariance matrix that is both symmetric and positive-definite (described in Appendices 11.1 and 11.2.)

11.4 Typical Data Science Problems

Data science problems are defined as computational algorithmic problems and are assigned a mnemonic name in brackets (e.g., [**IrisC**]) used to refer to the problem elsewhere. The last character indicates the type of problem they are (**C**lassification, **P**rediction or **Clustering**.) The population of inputs is described in the INSTANCEs of the problem, together with an example of such. The problems are usually solved using certain samples described in the next Appendix 11.5, where more details about the population can be gathered.

[AstroP] **PLANETARY BODY LOCATION**

INSTANCE: A celestial body x in our solar system (e.g., Mars or Saturn), a location on Earth (latitude, longitude) and a time stamp t

QUESTION: What is the location (azimuth and elevation) in the sky where x will be found from the given location at time t?

[IrisC] **IRIS FLOWER CLASSIFICATION** ({ Setosa, Versicolor, Virginica })

INSTANCE: A feature vector $x =$ (sepal length, sepal width, petal length, petal width) (in cms) describing an iris flower, e.g., (4.6, 3.1, 1.5, 0.2)

QUESTION: Which kind of flower is x?

[NoisC] **NOISY CLASSIFICATION**

INSTANCE: A synthetic dataset D where some (derived) features have been obtained by certain combinations (linear or statistical) from the remaining hidden (primitive) features

QUESTION: Which are the derived and the primitive features in D?

[CharRC] **CHARACTER RECOGNITION** ({0, 1, 2, 3, 4, 5, 6, 7, 8, 9})

INSTANCE: A 2D picture of a handwritten decimal numerical digit x

QUESTION: Which digit is it?

[MalC] **MALWARE CLASSIFICATION** (Π)

INSTANCE: A piece of malware x (program)

QUESTION: Which category c in partition Π does x belong in?

The categories in the partition Π could be those listed in Table 11.13 in Appendix 11.4.

[BodyFP] **HUMAN BODY FAT ESTIMATION**

INSTANCE:	A feature vector with a person x's body measurements including weight, height, age and others[1]
QUESTION:	What is x's body fat percent of the total weight?

The full set of features is listed in Table 11.9 in Sect. 11.4.

[BioTC] **BIOTAXONOMIC CLASSIFICATION** (T)

INSTANCE:	A (long) DNA sequence (over the alphabet $\{a, c, g, t\}$) from a living organism x
QUESTION:	What species in taxon T does x belong in?

The features representing an organism could be, for example, mitochondrial genes COI, COII, COIII, and CytB from the organism's genome. Other choices give rise to different problems, as shown in the corresponding dataset in Sect. 11.5. A most complex problem will arise if the full genome is used to represent an instance of a biological organism.

[PhenoP] **PHENOTYPIC PREDICTION** (T, F)

INSTANCE:	A (long) DNA sequence x (over the alphabet $\{a, c, g, t\}$) from a living organism in T?
QUESTION:	What's the quantitative measurement of x's phenotypic feature F when fully grown?

Examples of phenotypic features are the area of the cephalic apotome (head top), its peculiar spot pattern, the body color of its thorax/abdomen, or the area of the postgenal cleft (mandible) in a black fly larva; Or, it could be the rosette dry mass (leaf weight), or the life span of a specimen of *A. thaliana* when fully grown (end of the reproductive cycle to adulthood.)

[RossP] **ROSETTE AREA OF A PLANT** (T)

INSTANCE:	A (long) DNA sequence x (over the alphabet $\{a, c, g, t\}$) from a plant in T?
QUESTION:	What's the area of x's rosette when fully grown?

[LocP] **PROVENANCE OF A PLANT** (T)

INSTANCE:	A (long) DNA sequence x (over the alphabet $\{a, c, g, t\}$) describing a plant in T?
QUESTION:	Where on Earth (latitude, longitude) was x grown?

[CWD] **CODEWORD DESIGN**

INSTANCE:	A positive integer m and a threshold $\tau > 0$
QUESTION:	What is a largest set B of single DNA strands of length m that do not cross hybridize to themselves or to their complements (nxh set) under stringency τ according to the h-distance , i.e., $\lvert xy \rvert > \tau$ for all $x, y \in B$?

11.5 A Sample of Common and Big Datasets

This section summarizes datasets that are used throughout the book to illustrate methods and solutions for a variety of dimensionality reduction methods and problem solving, or that could be used in other applications. (They are organized by domain area and are listed in alphabetic order.)

Astronomy
This dataset is perhaps one the oldest and most comprehensive and accurate set of observations (for his time) of the skies with the naked eye, compiled over a period of 40+ years by the Danish astronomer Tycho Brahe and his assistants with the naked eye (telescopes were not invented yet) in the late 1500s. His student Johannes Kepler used the data to synthesize our current model of the solar system that reproduces the data based in the knowledge expressed in the three well-known physical Kepler's laws (Table 11.5).

Biology
The Iris Flower dataset was used by Fisher to test his ideas about Discriminant Analysis [7]. It is perhaps one of the oldest and well-known dataset in the statistics, pattern recognition and data science literature. It contains 3 classes of 50 instances each, where each class refers to a type of iris plant, as illustrated in Tables 11.6 and 11.7.

The Body Fat dataset contains estimates of the percentage of body fat for 252 men determined by underwater weighing and various body circumference measurements. (Accurate measurement of body fat is inconvenient/costly and it is desirable to have alternative methods of estimating body fat based on more easily available measurements.) The set was assembled and published by [12] (Tables 11.8, 11.9, and 11.10).

Table 11.5 Tycho Brahe's dataset

Population	Celestial bodies
Dimensionality (features)	Hundreds
Class distribution	N/A
Size (Nr of data points)	Tens of thousands
Sample data point	Positions of Mars and comets in the celestial sky
Origin	[3]

Table 11.6 The Iris flower dataset

Population	Iris flowers
Dimensionality (features)	4
Class distribution	50 50 50
Size (nr of data points)	150
Sample data point	(4.6, 3.1, 1.5, 0.2)
Origin	[7]

Table 11.7 Iris dataset (lengths are given in *cm*)

ID	Sepal length	Sepal width	Petal length	Petal width	Label (species)
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
...	...	...	...	...	...
50	5.0	3.3	1.4	0.2	Iris-setosa
51	7.0	3.2	4.7	1.4	Iris-versicolor
52	6.4	3.2	4.5	1.5	Iris-versicolor
...	...	...	...	...	...
100	5.7	2.8	4.1	1.3	Iris-versicolor
101	6.3	3.3	6.0	2.5	Iris-virginica
102	5.8	2.7	5.1	1.9	Iris-virginica
...	...	...	...	...	...
150	5.9	3.0	5.1	1.8	Iris-virginica

Table 11.8 Human body fat dataset

Population	Humans
Dimensionality (features)	4
Size (Nr of data points)	252
Sample Data Point	(154.25, 67.25, $\cdots$, 17.10, 23, 12.30)
Origin	[12]

Table 11.9 Examples of 16 features in the body fat dataset of 252 men (columns.) (If unspecified, circumferences are given in *cm*)

	Person IDs					
Feature	1	2	3	4	...	252
Weight (lbs)	154.25	173.25	154.00	184.75	...	207.5
Height (in)	67.75	72.25	66.25	72.25	...	70
BMI(= $\frac{W}{H}$ in Kg/m)	23.70	23.40	24.70	24.90	...	29.80
Neck	36.20	38.50	34.00	37.40	...	40.8
Chest	93.10	93.60	95.80	101.80	...	112.4
Abdomen	85.20	83.00	87.90	86.40	...	108.5
Hip	94.5	98.70	99.20	101.20	...	107.1
Thigh	59.00	58.70	59.60	60.10	...	59.3
Knee	37.30	37.30	38.90	37.30	...	42.2
Ankle	21.90	23.40	24.00	22.80	...	24.6
Bicep	32.00	30.50	28.80	32.40	...	33.7
Forearm	27.40	28.90	25.20	29.40	...	30
Wrist	17.10	18.20	16.60	18.20	...	20.9
Age (years)	23	22	22	26	...	74
BodyFat (lbs)	12.30	6.10	25.30	10.40	...	31.9

Table 11.10 The biotaxonomy dataset

Population	Organisms in 21 species across the biome
Dimensionality (features)	500-10,000
Class distribution	(Table 11.11)
Size (data points)	249
Typical data point	(COI, COII, COIII, CytB) (DNA sequences)
Origin	[16]

Table 11.11 Organisms in the sample data for species classification of a biological taxon. The 21 classes/labels in the partition are in the first column

Label	T: Genus species	Common name	Count
1	*Apis mellifera*	Western honey bee	4
2	*Arabidopsis thaliana*	Thale cress	5
3	*Bacillus subtilis*	Hay/grass bacillus	18
4	*Branchiostoma floridae*	Florida lancelet	18
5	*Caenorhabditis elegans*	Round worm	6
6	*Canis lupus*	Wolf	18
7	*Cavia porcellus*	Pork	4
8	*Danio rerio*	Zebra fish	9
9	*Drosophila melanogaster*	Fruit fly	18
10	*Gallus gallus*	Red junglefowl	18
11	*Heterocephalus glaber*	Naked mole rat	3
12	*Homo sapiens*	Human	18
13	*Macaca mulatta*	Rhesus macaque	8
14	*Mus musculus*	House mouse	18
15	*Neurospora crassa*	Red bread mold	5
16	*Oryza sativa*	Asian rice	12
17	*Pseudomonas fluorescens*	Infectious bacterium	6
18	*Rattus norvegicus*	Brown rat	18
19	*Rickettsia rickettsii*	Tick-born bacterium	18
20	*Saccharomyces cerevisiae*	Yeast	18
21	*Zea mays*	Corn/maize	7
			249

The Biotaxonomy dataset was obtained from 249 organisms collected from genetic repositories (e.g., GenBank at www.ncbi.nlm.nih.gov/genbank/) and is illustrated in Tables 11.11 and 11.12.

Cybersecurity

The Malware Classification Challenge (*arxiv.org/abs/1802.10135*) [2, 19] dataset was released in 2015 and is publicly available through *kaggle.com*. An available dataset consists of a set of 10,868 known malware files representing a mix of nine (9) different malware families, as summarized in Table 11.13, e.g., "ramnit." *Code.exe* is a computer executable binary file obtained from compiling the source malware

Table 11.12 Microsoft's Malware classification dataset

Population	Malware
Dimensionality (features)	1804
Class distribution	1541 2478 2942 475 42 751 398 1228 1013
Size (data points)	10,868
Sample data point	(*Code.exe*, . . ., Vundo)
Origin	*kaggle.com*

Table 11.13 Microsoft's Malware classification problem. The labels for the classes in the partition are in the first column

Labels	Class name	Type of Malware	*Count*
1	Ramnit	Worm	1541
2	Lollipop	Adware	2478
3	Kelihos_ver3	Backdoor	2942
4	Vundo	Trojan	475
5	Simda	Backdoor	42
6	Tracur	TrojanDownloader	751
7	Kelihos_ver1	Backdoor	398
8	Obfuscator.ACY	Any kind of obfuscated malware	1228
9	Gatak	Backdoor	1013
			—–
			10, 868

(headers removed.) Each datapoint contains both the raw binary content of the malware file as well as metadata information extracted using the IDA disassembler tool. These features include source code snippets, assembly command frequencies, registers used and their frequency, data section of the code, key words in the disassembly code, as well as a number of other extracted features (e.g., entropies of the various features.) The classification challenge was to correctly assign every piece of malware to one of the nine (9) categories.

Finance

The Adult dataset was extracted from the 1944 Census Bureau database found at the source by Barry Becker to predict whether a person makes over 50, 000 a year (Tables 11.14 and 11.15).

The Audit dataset was used to support research by the Ministry of Electronics and Information Technology (MEITY) of the government of India. The goal was to help auditors classify whether a firm is fraudulent or not on the basis of historical and current risk factors (Table 11.16).
The Bank Marketing dataset was designed by a Portuguese banking institution to direct marketing campaigns on the phone to decide whether a client will subscribe (yes/no) to a term deposit (Table 11.17).

Table 11.14 Adult dataset

Population	Census income
Dimensionality (features)	15
Class distribution	23.93% 76.07%
Size (nr of data points)	48,842
Sample data point	(53, Confidential, 234, 721, 11th, 7, Married-civ-spouse, Handlers -cleaners, Husband, Black, Male, 0, 0, 40, United-States, $\leq 50K$)
Origin	*www.census.gov/ftp/pub/DES/www/welcome.html*

Table 11.15 Audit dataset

Population	Fraudulent firms
Dimensionality (features)	27
Class distribution	114 77 82 70 47 95 1 4 5 3 1 41 37 200
Size (nr of data points)	776
Sample data point	(3.89, 6, 0, 0.2, 0, 10.8, 0.6, 6.48, 10.8, 6, 0.6, 3.6, 11.75, 0.6, 7.05, 2, 0.2, 0.4, 0, 0.2, 0, 4.4, 17.53, 0.4, 0.5, 3.506, 1)
Origin	*archive.ics.uci.edu/ml/machine-learning-databases/00475/*

Table 11.16 Bank marketing dataset

Population	Bank clients
Dimensionality (features)	17
Class distribution	N/A
Size (nr of data points)	45,211
Sample data point	(47, blue-collar, married, N/A, no, 1506, yes, no, N/A, 5, may, 92, 1, -1, 0, N/A, no)
Origin	*archive.ics.uci.edu/ml/machine-learning-databases/00222/*

Table 11.17 Taiwanese bankruptcy dataset

Population	Customer default payers
Dimensionality (features)	24
Class distribution	Unknown
Size (nr of data points)	30,001
Sample data point	(50, 000, 2, 2, 1, 37, 0, 0, 0, 0, 0, 0, 46, 990, 48, 233, 49, 291, 49, 291, 28, 314, 28, 959, 29, 547, 2000, 2019, 1200, 1100, 1069, 1000, 0)
Origin	*archive.ics.uci.edu/ml/machine-learning-databases/00350/*

The Taiwanese Bankruptcy dataset was used to evaluate the predictive accuracy of six data mining probability models of default based on labeled customer data (Yes/No) on default payments in Taiwan.

Image Processing

The MNIST dataset [15] consists of gray scale images of handwritten digits. The digits were handwritten by members of the US Census Bureau and high school students. The digit in each image is size-normalized and centered. The dataset is fairly large. The MNIST dataset is a popular dataset and a benchmark in the image processing, machine learning and neural network literature. One of its major real world applications was in the postal service to automatically scan and infer handwritten zip codes. The pixels are binary images, i.e., the pixels are either on or off (0 or 1.) (Tables 11.18 and 11.19).

The ImageNet dataset [14] is the largest hand-labeled image dataset currently available. The latest dataset was published for the ImageNet Large Scale Visual Recognition Challenge 2017 (ILSVRC17) for the problem of object detection in and classification of images, i.e., to identify and find the location of objects familiar to humans in images in a bounding box. The dataset was inspired to create a standard benchmark (North Star) for computer vision research, is organized according to the WordNet ontology (nouns only) and has been expanded to include 1000 categories for object localization and 30 fully labeled categorized for object detection in videos. It is also publicly available through *kaggle.com*.

Leisure

The Old Faithful Geyser dataset was singled out by Azzalin and Bowman [4] and is available as a standard set in the package R or at *www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat*. Eruptions have been clustered into 3 categories (Short, Medium, Long) (Tables 11.20 and 11.21).

Table 11.18 MNIST dataset

Population	28×28D handwritten images of decimal digits
Dimensionality (features)	784
Class distribution	0: 5923, 1: 6742, 2: 5958, 3: 6131, 4: 5842, 5: 5421, 6: 5918, 7: 6265, 8: 5851, 9: 5949
Size (nr of data points)	60,000
Sample data point(s)	(Figs. 2.2 and 5.6)
Origin	[15]

Table 11.19 ImageNet dataset

Population	Images of objects familiar to humans
Dimensionality (features)	181,503
Class distribution	200 object categories;
	456,567 images of 478,807 objects in a training set;
	40,152 images in a testing set;
	20,121 images of 55,502 objects in a validation set
Size (nr of data points)	516,840
Sample data point	N/A
Origin	*image-net.org*

Table 11.20 Old faithful Geyser dataset

Population	Eruptions of old faithful (Yellowstone national park)
Dimensionality (features)	2
Class distribution	N/A
Size (nr of data points)	272
Sample data point	(3.600, 79) =(Duration, wait time to next eruption) (mins)
Origin	[7]

Table 11.21 Netflix movie rating dataset

Population	Netflix movies
Dimensionality (features)	4
Class distribution	N/A
Size (nr of data points)	2,817,131
Sample data point	Quadruplets <user, movie, date of rating, rating>
Origin	*www.kaggle.com/laowingkin/netflix-movie-recommendation*

Table 11.22 The Ames housing dataset

Population	Residential properties in Ames, Iowa
Dimensionality (features)	82
Class distribution	N/A
Size (nr of data points)	2930
Sample data point	82D vectors describing property/houses
Origin	Assessor's office,*jse.amstat.org/v19n3/decock/DataDocumentation.txt*

Th Movie Rating dataset was provided by Netflix to support an open competition to develop a movie recommendation system in 2006. The dataset is publicly available through *kaggle.com*. The movie rating files contain ratings (1 to 5 *s) from 480,189 randomly chosen, anonymous Netflix customers (not associated with Netflix nor in certain blocked countries) over 17,770 movie titles. The data were collected between October, 1998 and December, 2005 and reflect the distribution of all ratings received during this period. The average user rated 2000 movies and each movie was rated by over 5000 users on the average, with wide variation (some movies received only 3 ratings and one user rated over 17,000 movies.) Although no user information is provided, the dataset has been has been criticized by privacy advocates for leaking customer information, although it was supposed to have been constructed to preserve customer privacy (Table 11.22).
The Ames Housing dataset contains information from the Ames Assessor's Office about the assessed values for individual residential properties sold in Ames, Iowa from 2006 to 2010 (Fig. 11.1). The data has 82 columns which include 23 nominal, 23 ordinal, 14 discrete, and 20 continuous variables (and 2 additional observation identifiers).

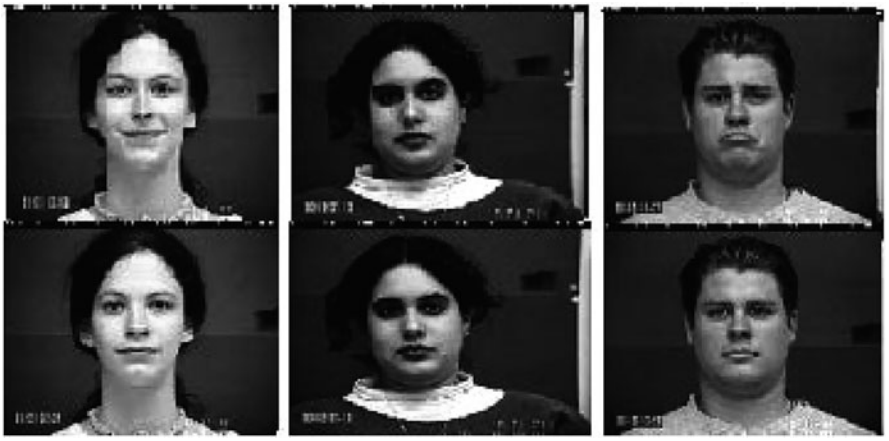

Fig. 11.1 Facial expressions (positive: *left*; neutral: *middle*; negative: *right)* by trained actors in the Cohn-Kanada dataset

Table 11.23 The Cohn-Kanade dataset

Population	Humans
Dimensionality (features)	64
Size (Nr of data points)	500 from 100 actors
Sample Data Point	(Fig. 11.1)
Origin	[13]

Psychology

The Cohn-Kanade dataset consists of approximately 500 image sequences from 100 subjects ranging in age from 18 to 30 years (65% are female; 15% are African–American and 3% are Asian or Latino.) Each begins from a neutral or nearly neutral face. For each, an experimenter described and modeled the target display. Six are based on descriptions of prototypic emotions (i.e., joy, surprise, anger, fear, disgust, and sadness.) These six tasks and mouth opening in the absence of other action units were annotated by certified FACS coders [6]. Seventeen percent (17%) of the data were comparison annotated. Inter-observer agreement was quantified with the kappa coefficient, which is the proportion of agreement above what would be expected to occur by chance [22] [8], with an average observer agreement of 0.86. Image sequences from neutral to target display were digitized into 640 $\times$ 480 or 640 $\times$ 490 pixel arrays with 8-bit precision for grayscale values. (Further details can be found at *://www.ri.cmu.edu/project/cohn-kanade-au-coded-facial-expressiondatabase/* (Table 11.23).

Table 11.24 US DOE campus safety and security dataset

Population	Title IV College Campuses in the US
Dimensionality (features)	Hundreds
Class distribution	(Variable by year)
Size (data points)	Nearly 11,000 campuses
Sample data point	N/A
Origin	US Department of Education at *ope.ed.gov/campussafety*

Table 11.25 Synthetic dataset SYN12

Population	12D unit cube in Euclidean space
Dimensionality (features)	12
Class distribution	12 (6 primitive and 5 derived features, plus one label)
Size (nr of data points)	1000
Sample data point	12D numerical vectors in the unit hypercube
Origin	[16]

Table 11.26 Synthetic dataset SYN23

Population	23D unit cube in Euclidean space
Dimensionality (features)	12
Class distribution	23 (12 primitive and 10 derived features, plus one label)
Size (nr of data points)	500
Sample data point	23D numerical vectors in the unit hypercube
Origin	[16]

Safety in College Campuses
The Department of Education (DOE) of the United States collects and disseminates the yearly crime statistics for all title IV colleges and universities in the United States. Major crime categories include arrests, criminal offenses, disciplinary actions, fire statistics, hate crimes, VAWA offenses, and sexual misconduct (Table 11.24).

Synthetic Data
These synthetic datasets are fully described in the reference above [16] where the sets originated. They afford perfect prior knowledge of *feature dependencies* that can remain hidden from the DR methods. The effectiveness of DR methods can then be assessed by how well they discover these hidden but most relevant and independent features, from a full set that includes other confounding features derived from the few primitive ones (Tables 11.25 and 11.26).

The primitive features were designed using the method (API) described in [10] and publicly available in Python at *sklearn.datasets.make_classification*. The second data set was generated likewise but halving the number of all parameters involved in generating features in the second dataset SYN23, so just the second and largest set is described. The dataset was generated in two phases. First, the sklearn API was used to generate the primitive features in a 12D hypercube with sides of length $1.6 = 2 * 0.8$ and assign about the same number of points to each of the eight classes. Each class is composed of several Gaussian clusters located near corners of the 12D hypercube. For each class, the informative features are drawn independently from the standard normal distribution $N(0, 1)$. Four more features X_{13}, X_{14}, X_{15} and X_{16} were generated as linear combinations (with random coefficients) of some primitive features, as provided in the API. In the second phase, 6 more predictors were generated as follows. Features X_{17} and X_{18} were drawn as a repeat of two randomly selected primitive features/columns. X_{19} was generated as the sum of squares of two features selected randomly as

$$X_{19} = X_i^2 + X_j^2 \quad \text{for} \quad \text{some} \quad \text{fixed} \ \ 0 \leq i, j \leq 12\,.$$

X_{20} was obtained from the values of a linear regression model fitted using two randomly selected features X_i and X_j as predictors with the *Label* as response. The value for X_{20} was the deviation between the pair of values in *Label* and the regression prediction. X_{21} was obtained the same way but with X_i^2 and X_j as predictors from another random selection of predictors X_i and X_j. Finally, X_{22} was generated as the outcome of the natural *logarithm* of a randomly selected but uniform predictor X_i (a transformation that does not change entropy.) Again, different types of predictors were used as described above.

In general, synthetic data has a number of uses in data science. First, as illustrated with datasets SYN12 and SYN23, they may increase prior knowledge about the data and allow for much greater control on the part of the analyst to test, validate and assess data analyses. Second, they may alleviate a number of concerns making data available to the analyst, including *privacy* (e.g., with health records), *anonymity* (legal compliance with, e.g., HIPPA regulations), *intellectual property*, *value* (data is nowadays considered to be the most valuable asset in an organization) and *accessibility*. considered to be probably the major roadblick to data analytics.) Where any of these issues is a concern, real datasets can be cleansed and/or transformed into roughly equivalent synthetic datasets that preserve the critical trends and structure of the real data while being completely "fake," thus affording results of the analyses that nonetheless apply to the original data and problems. Third, using the DR techniques discussed in Chap. 8, they can be converted into much more informative datasets while being smaller in size and easier to crunch and analyze. This can be very useful in large datasets where a pilot feasibility study would inform a decision whether the results will be worth a large and costly effort and even whether a project is feasible. It is therefore not surprising that most data analytics platforms (described in Appendix 11.6) offer facilities to generate synthetic data for specific problems and/or solutions (e.g., classification, clustering, regression.)

Text Processing

WordNet is a lexical database of semantic relations between abstract language concepts (so-called *synsets* of cognitive synonyms) in a given natural language (more than 200 now), including synonyms, hyponyms, and meronyms. WordNet has been used for many purposes in information systems, including word-sense disambiguation, information retrieval, automatic text classification and text summarization, machine translation and even automatic crossword puzzle generation. WordNet was originally a lexical database for the English language created by researchers at Princeton led by George Miller as part of the NLTK corpus (Table 11.27). It is regularly updated by the Princeton WordNet group, whose aim is to ensure that there is an up-to-date version of high quality available for the English language. It is available at *https://CRAN.R-project.org/package=wordnet*.

Other Data Collections

Many other datasets are constantly being generated or updated, so it is impossible to account for them all. A selection of other sources that may come in handy currently are shown in Table 11.28, particularly for big data sets.

Table 11.27 WordNet

Population	Distinct synsets (representing abstract semantic concepts)
Dimensionality (features)	Labeled edges between synsets indicating lexical relationships
	(like antonym, meronym, part-of-speech) in the WordNet tree
Class distribution	N/A
Size (nr of data points)	117,000
Sample data point(s)	(leg, meronym, chair), (arm-chair, hyponym, chair),
	(love, antonym, hate)
Origin	[17]

Table 11.28 Other available big dataset collections

Name	Source	Description
Digital sky survey	*sdss.org*	The Sloan Digital Sky Survey has collected and organized outer space astronomical observations (order of terabytes daily) that has produced the most detailed 3D maps of the Universe ever made
EARTHDATA	*https://earthdata.nasa.gov/*	NASA's Common Metadata Repository (CMR) is a high-performance, high quality, continuously evolving metadata system that catalogs all data and service metadata records for NASA's Earth Observing System Data and Information System (EOSDIS)
Marine Geoscience Data System (MGDS)	*www.re3data.org/repository/r3d100010273*	Data repository that provides free public access to marine geophysical data products and datasets relevant to the formation and evolution of the seafloor and sub-seafloor

(continued)

Table 11.28 (continued)

Name	Source	Description
GenBank (NIH)	*www.ncbi.nlm.nih.gov/genbank/*	US National Institute of Health genetic sequence data repository, an annotated collection of all publicly available DNA sequences
IEEE dataPort	*ieee-dataport.org/datasets*	Over 3000 datasets in 25+ categories of topics, including biomedicine, COVID-19, ecology, environment, power and energy, social science and transportation
NOAA Data Catalog	*https://data.noaa.gov/dataset/*	US National Oceanic and Atmospheric Administration's datasets, including oceans, currents, glaciers, temperature and barometric pressure
UCI Machine Learning Repository	*https://archive-beta.ics.uci.edu/*	600 datasets about a variety of topics, from flowers, wine, diabetes and heart disease, to auction verification, student academic success and ImageNet, to air quality, energy efficiency, the ozone. The datasets are organized by various criteria (e.g., subject, problems/tasks) and properly documented (original design and problems.) There is a facilitate to donate datasets
Visual Datasets	*https://public.roboflow.com/*	About 40 datasets consistings of images on topics including mushrooms, aerial maritime drones, wildfire smoke, pistols, and self-driving cars

11.6 Computing Platforms

This section provides summaries of computing platforms for data science and basic tips to get started with the software. More detailed tutorials can be found readily available online.

11.6.1 The Environment R

R is a popular interactive environment for computational programming and statistical analysis. It is also considered to be a programming language. R can be traced back to S, a language and package developed at Bell Laboratories of AT&T by John Chambers and others (in 1976) for statistical analysis that was originally free but eventually became a commercial package, S-PLUS. In 1991, R was created by

Ross Ihaka and Robert Gentleman in the Department of Statistics at the University of Auckland and it become available to the public in 1993. Since S-Plus and R are implementations of S, both products can execute the same functions and most common statistical functions without modification.

R is freely available for download from a comprehensive R archive network, CRAN at *cloud.r-project.org*. CRAN is composed of a set of mirror servers distributed around the world. There is an extension, RStudio that offers a user interface in an integrated development environment that can be downloaded from *www.rstudio.com/download*.

R has several software packages (few system supplied and many user-contributed) available to implement various supervised and unsupervised statistical and machine learning algorithms and evaluation methods. A list of the important software libraries in R is shown in Table 11.29. The packages can be downloaded

Table 11.29 Popular software packages in R

Package::FunName()	Description
`stats::lm()`	fits linear models
`stats::glm()`	fits generalized linear models (GLMs)
`glmnet::glmnet()`	fits a GLM with LASSO or ElasticNet regularization
`base::svd()`	returns the singular value decomposition of a rectangular matrix
`stat::princomp()`	returns the principal components of the given data matrix in an object of class princomp (via spectral decomposition)
`stat::prcomp()`	returns the principal components of the given data matrix in an object of class prcomp (via singular value decomposition-SVD)
`fastICA::fastICA()`	performs Independent Component Analysis (ICA)
`kernlab::kpac()`	performs kernel PCA (KPCA)
`dimRed::`	various dimensionality reduction methods implemented in R (e.g., PCA, KPCA and nonlinear methods **MDS** , **ISOMAP** and *t*-**SNE**)
`e1071::svm()`	fits a support vector machine-SVM for general regression and classification
`randomForest::randomForest()`	fits a random forests for classification and regression
`mlogit::mlogit()`	fits a multinomial logit model for estimation and classification (possibly with alternative-specific and/or individual-specific variables)
`tree::tree()`	fits a classification or regression tree
`class::knn()`	fits a k-Nearest Neighbor classifier
`MASS::lda()`	performs a linear discriminant analysis
`MASS::qda()`	performs a quadratic discriminant analysis
`infotheo::entropy()`	returns the values of the Shannon entropy(ies) of the given data
`infotheo::condentropy()`	returns the conditional entropy of the given random variables

from *cloud.r-project.org*, along with tutorials and demonstrations for the packages. Some handy commands to get a quick start with the methods discussed in this book are shown in Table 11.30.

Table 11.30 Handy commands in R for dimensionality reduction in data science

Command	Action	Notes
`help`	provide documentation/help with any topic	`help(help)` shows more help with how to use `help`
`y<-eigen(A)`	assigns dataframe `y` the spectral decomposition of matrix `A`	`y$val` are the eigenvalues; `y$vectors` are the eigenvectors
`y<-svd(A)`	assigns dataframe `y` the SVD of matrix `A`	`y$d` is the vector of singular values of A
		`y$u` is a matrix with the left singular vectors in the columns
		`y$v` is a matrix with the right singular vectors in the columns
`PC <- prcomp(X)`	assigns dataframe `PC` the principal components of data `X`	`PC$rotation` contains the rotated data (in the PC axes)
`PC <- prcomp(X, retx = TRUE, center = TRUE, scale. = FALSE tol = NULL, rank = NULL, ....)`	`retx$d = TRUE` iff rotated values are returned; `center/scale=TRUE` iff the data X is centered/scaled to Z-scores (advisable in general, but *scale* cannot be used if there are zero or constant variables); `subset` is an optional vector used to select rows from data X;	defaults for `prcomp`; components are omitted if their standard deviations are less than or equal to `tol` times the standard deviation of the first component; `rank` can be used instead to select only the max rank of *X* number of components;
`PC <- princomp(X)`	similar to `prcomp` for PCs, but `princomp` is a generic function with "formula" and "default" methods	`PC$rotation` contains the rotated data (in the PCs axes)
`predict(object, newdata, ...)`	predicts values for the `newdata` using the SVD of `object`	
`entropy(X, method="emp")`	computes the entropy using estimator emp	*X* must be discrete; the option `equalfreq` can be used to discretize where necessary)
`condentropy(X, Y=NULL, method="emp")`	computes the conditional entropy $H(X \mid Y)$ using estimator method emp	as with `entropy`

11.6.2 Python Environments

A general list of software packages useful in data science is shown in Table 11.31.

The *scikit-learn* (sklearn) module in Python is a high-level module that includes all libraries required for pre-processing data, implementing supervised and unsupervised machine learning algorithms and computing evaluation metrics. A list of the important libraries within sklearn is shown in Table 11.32. The packages can be downloaded from *scikit-learn.org*, where tutorials and demonstrations how to use the packages are also available.

Tables 11.33 and 11.34 show software libraries and commonly used commands in Python (sklearn) for dimensionality reduction and supervised/unsupervised Machine learning methods.

Table 11.31 Data science software in python

Name	Description	Link
Scikit-Learn	Python module for data science	*scikit-learn.org*
WEKA	Machine Learning software in Java	*www.cs.waikato.ac.nz/ml/weka/*
R	Several packages related to data science	*cran.r-project.org/*
Matlab	Statistics and Machine Learning Toolbox	*www.mathworks.com/products/statistics.html*
TensorFlow	Deep Learning Libraries	*www.tensorflow.org/*
PyTorch	Deep Learning Libraries	*pytorch.org/*
Keras	Python Deep Learning APIs	*keras.io/*
Spark MLlib	Machine Learning for Big Data	*spark.apache.org/mllib/*
Google Vertex AI	Machine Learning on the Cloud	*cloud.google.com*

Table 11.32 Scikit-Learn Software packages in python

Name	Description
sklearn.preprocessing	Feature scaling, imputation of missing values
sklearn.decomposition	Dimensionality reduction (PCA, KernelPCA, Nonnegative Matrix Factorization, Singular Value Decomposition)
sklearn.manifold	Nonlinear Dimensionality reduction (**ISOMAP**, t-**SNE** and **MDS**)
sklearn.feature_selection.RFE	Wrapper methods for feature selection
sklearn.model_selection	Model selection using cross-validation and other approaches
sklearn.model_selection.metrics	Metrics for supervised learning (e.g., precision, recall, and F1-score) and unsupervised learning (e.g., silhouette score)
sklearn.neighbors	k-Nearest Neighbors
sklearn.naive_bayes	Naive Bayes Classifier
sklearn.tree	Decision Trees
sklearn.ensemble	Bagging, Boosting and other ensemble methods
sklearn.neural_network	Neural Networks
sklearn.svm	Support Vector Machines
sklearn.cluster	Clustering algorithms
sklearn.mixture	Gaussian Mixture Models

Table 11.33 Handy commands in sklearn for dimensionality reduction in data science

Command	Action	Notes
`feature_selection.RFE (estimator, n_features_to_select)`	Creates an RFE feature selection object	The base classifier `estimator` selects `n_features_to_select`
`manifold.Isomap(n_neighbors, n_components)`	Creates an object used to perform **ISOMAP** based dimensionality reduction	uses `n_neighbors` for each point and `n_components` coordinates for the manifold
`manifold.LocallyLinear Embedding(n_neighbors, n_components)`	Creates an object used to perform LocallyLinearEmbedding based dimensionality reduction	uses `n_neighbors` neighbors for each point and `n_components` coordinates for the manifold
`manifold.MDS(n_components)`	Creates an object used to perform MDS based dimensionality reduction	immerses the dissimilarities in `n_components` dimensions
`manifold.TSNE (n_components,perplexity)`	Creates an object used to perform *t*-SNE based dimensionality reduction	uses `perplexity` nearest neighbors to embed into `n_components` in manifold learning algorithms
`manifold.Spectral Embedding (n_components)`	Creates an object used to perform Spectral embedding for nonlinear dimensionality reduction	projects into a subspace with `n_components`
`decomposition.PCA (n_components)`	Creates an object used to perform PCA	keeps `n_components`
`decomposition.KernelPCA (n_components,kernel)`	Creates an object used to perform KernelPCA	keeps `n_components` after using a `kernel` type
`decomposition.Truncated SVD(n_components)`	Creates an object used to perform dimensionality reduction using truncated SVD	keeps `n_components` in the output data
`decomposition.non_negative_ factorization(X)`	Creates an object used to perform NMF	`X` is the input data
`random_projection. GaussianRandomProjection (n_components)`	Creates an object used to perform dimensionality reduction through random projections	projects onto `n_components` in the target space

Table 11.34 Handy commands in sklearn for supervised and unsupervised Machine learning methods and evaluation

Command	Action	Notes
`metrics.roc_auc_score (y_true, y_score)`	Computes the Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores	`y_true` are true labels `y_score` are predicted
`metrics.f1_score(y_true, y_pred)`	Computes the f1-score of the predicted labels	`y_true` are true labels `y_pred` are predicted
`metrics.mean_squared_error(y_true, y_pred)`	Computes the Mean squared error regression loss	`y_true` are true values `y_pred` are predicted
`metrics.silhouette_score(X, labels)`	Computes the mean Silhouette Coefficient of all samples	`X` is an array of pairwise distances between samples; `labels` are their cluster labels
`model_selection. cross_val_score (estimator, X, y, cv)`	Performs cross-validation	`estimator` is the classifier to be evaluated, `X` is the data, `y` is the set of labels and `cv` is the number of folds
`tree.DecisionTree Classifier(criterion)`	Creates a decision tree classifier	`criterion` is the splitting criterion
`naive_bayes.MultinomialNB(alpha)`	Creates a Naive Bayes Classifier	`alpha` is the Laplacian smoothing parameter
`neighbors.KNeighbors Classifier(n_neighbor)`	Creates a Nearest Neighbor Classifier	considers `n_neighbor` neighbors
`ensemble.Bagging Classifier (base_estimator, n_estimators)`	Creates a Bagging Ensemble	`base_estimator` uses `n_estimators` in the ensemble
`ensemble.AdaBoost Classifier (base_estimator, n_estimators)`	Creates an Adaboost classifier	`base_estimator` is the base classifier in the ensemble and `n_estimators` is the number of classifiers in the ensemble
`svm.SVC(C, kernel)`	Creates an SVM classifier	`C` is the regularization parameter with a `kernel` type
`neural_network.MLPClassifier (hidden_layer_sizes, activation)`	Creates a neural network classifier	`hidden_layer_sizes` is the number of neurons in the hidden layers and `activation` is the activation function
`cluster.KMeans(n_clusters)`	Creates a K-Means clustering object	generates `n_clusters` clusters
`mixture.Gaussian Mixture(n_components)`	Creates a Gaussian Mixture Model	uses `n_components` in the mixture

References

1. Agrawal, M., Kayal, N., & Saxena, N. (2004). Primes is in **P**. *Annals of Mathematics, 160*(2), 781–793.
2. Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., & Giacinto, G. (2016). Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy* (pp. 183–194)
3. Anonymous (2019). *Tychonis Brahe Dani Opera Omnia*. London: Forgotten Books.
4. Azzalini, A., & Bowman, A. W. (1990). A look at some data on the old faithful geyser. *Applied Statistics, 39*, 357–365.
5. Devlin, D. (1991). *Logic and information*. Cambridge: Cambridge University Press.
6. Ekman, P. (2008). An argument for basic emotions. In *Basic emotions, rationality, and folk theory* (pp. 169–200). New York: Springer.
7. Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics, 7*(2), 179–188 (1936)
8. Fleiss, J. L. (1981). *Statistical Methods for Rates and Proportions*. London: Wiley.
9. Galois, E. (1830). Sur la theorie des nombres. *Bulletin des Sciences Mathematiques, XIII*, 428.
10. Guyon, I. (2003). Design of experiments of the nips 2003 variable selection benchmark. In *NIPS 2003 Workshop on Feature Extraction and Feature Selection* (Vol. 253).
11. Hopcroft, J., & Ullman, J. (2000). *Introduction to Automata Theory Languages and Computation*. Cambridge: Cambridge University Press.
12. Johnson, R. W. (1996). Fitting percentage of body fat to simple body measurements. *Journal of Statistics Education, 4*(1), 265–266.
13. Kanade, T., Cohn, J. F., & Tian, Y. (2000). Comprehensive database for facial expression analysis. In *Proceedings of the 4th IEEE Conference on Automatic Face and Gesture Recognition* (pp. 46–53)
14. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing Systems* (Vol. 25). Red Hook: Curran Associates, Inc.
15. LeCun, Y. (2010). MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist
16. Mainali, S., Garzon, M., Venugopal, D., Jana, K., Yang, C.-C., Kumar, N., Bowman, D., & Deng, L.-Y. (2021). An information-theoretic approach to dimensionality reduction in data science. *International Journal of Data Science and Analytics, 12*, 1–19.
17. Miller, G. (1995). Wordnet: A lexical database for english. *Communications of the ACM, 38*, 39–41
18. Mitchell, T. M. (1997). *Machine Learning*. (McGraw-Hill, New York, 1997)
19. Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., & Ahmadi, M. (2018). Microsoft malware classification challenge (2018). CoRR abs/1802.10135
20. Strang, G. (2016). *Introduction to Linear Algebra*. Wellesley: Wellesley-Cambridge Press.
21. Turing, A. M. (1936). On computable numbers with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society, 42*(2), 230–265. A correction, ibid 43: 544–546, 1936.
22. Yang, X., Garzon, M. H., & Nolen, M. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement, 20*, 37–46.