

Tue Herlau, Mikkel N. Schmidt and Morten Mørup

Introduction to Machine Learning and Data Mining

Lecture notes, Fall 2019, version 1.3

This document may not be redistributed. All rights belongs to
the authors and DTU.

September 26, 2019

Technical University of Denmark

Notation cheat sheet

	Matlab var.	Type	Size	Description
Regression	X	Numeric	$N \times M$	Data matrix: The rows correspond to N data objects, each of which contains M attributes.
	attributeNames	Cell array	$M \times 1$	Attribute names: Name (string) for each of the M attributes.
	N	Numeric	Scalar	Number of data objects.
	M	Numeric	Scalar	Number of attributes.
Classification	y	Numeric	$N \times 1$	Dependent variable (output): For each data object, y contains an output value that we wish to predict.
	y	Numeric	$N \times 1$	Class index: For each data object, y contains a class index, $y_n \in \{0, 1, \dots, C - 1\}$, where C is the total number of classes.
	classNames	Cell array	$C \times 1$	Class names: Name (string) for each of the C classes.
Cross-validation	C	Numeric	Scalar	Number of classes.
	All variables mentioned above appended with <code>_train</code> or <code>_test</code> represent the corresponding variable for the training or test set.			
	*.train	—	—	Training data.
	*.test	—	—	Test data.

This book attempts to give a concise introduction to machine-learning concepts. We believe this is best accomplished by clearly stating what a given method actually does as a sequence of mathematical operations, and use illustrations and text to provide an intuition. We will therefore make use of tools from linear algebra, probability theory and analysis to describe the methods, focusing on using as small a set of concepts as possible and strive towards maximal consistency.

In the following, vectors will be denoted by lower-case roman letters $\mathbf{x}, \mathbf{y}, \dots$ and matrices by bolder, upper case roman letters $\mathbf{A}, \mathbf{B}, \dots$. A superscript T denote the transpose. For instance

$$\mathbf{A} = \begin{bmatrix} -1 & 0 & 2 \\ 1 & 1 & -2 \end{bmatrix} \text{ and if } \mathbf{x} = \begin{bmatrix} -1 \\ 4 \\ 1 \end{bmatrix} \text{ then } \mathbf{x}^T = [-1 \ 4 \ 1].$$

The i th element of a vector is written as x_i and the i, j 'th element of a matrix as A_{ij} (and sometimes $A_{i,j}$ to avoid ambiguity). In the preceding example, $x_2 = 4$ and $A_{2,3} = -2$. During this course the observed data set, which we feed into our machine learning methods, will consist of N observations where each observation consist of a M dimensional vector. For instance if we have N observations $\mathbf{x}_1, \dots, \mathbf{x}_N$ then any given observation will consist of M numbers:

$$\mathbf{x} = [x_1 \dots, x_M]^T.$$

For convenience, we will often combine the observations into an $N \times M$ data matrix \mathbf{X}

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}$$

in which the i th row of \mathbf{X} corresponds to the row vector \mathbf{x}_i^T . We will use this notation for our data matrix and the *rows* of \mathbf{X} will correspond to N *observations* and the M *columns* of \mathbf{X} will correspond to M *attributes*. Often each of the observations \mathbf{x}_i will come with a *label* or *target* y_i corresponding to a feature of \mathbf{x}_i which we are interested in predicting. In this case we will collect the labels in a N -dimensional vector \mathbf{y} and the pair (\mathbf{X}, \mathbf{y}) will be all the data available for the machine learning method. A more comprehensive translation of the notation as used in this book and in the exercises can be found in the table on the previous page. Finally, the reader should be familiar with the big-sigma notation which allows us to conveniently write sums and products of multiple terms:

$$\sum_{i=1}^n f(i) = f(1) + f(2) + \dots + f(n-1) + f(n)$$

$$\prod_{i=1}^n f(i) = f(1) \times f(2) \times \dots \times f(n-1) \times f(n).$$

As an example, if $f(i) = i^2$ and $n = 4$ we have

$$\sum_{i=1}^4 f(i) = 1^2 + 2^2 + 3^2 + 4^2 = 30, \quad \prod_{i=1}^4 f(i) = 1^2 \times 2^2 \times 3^2 \times 4^2 = 576.$$

Course Reading Guide

It is our experience that when students have difficulties understanding a topic of this course, most often probability theory is the culprit. A reason for this is probability theory can be notationally challenging. For instance:

$$P(X = x|Y = y) \tag{0.1}$$

is, all being equal, a fairly unusual way to use an equality sign. Another reason is many students last encountered probability theory in conjunction with an introductory statistics course, where the main theorems are presented using the notation of stochastic variables and measure spaces. Especially when such a course has an applied focus, there is a tendency terms such as stochastic variables end up playing a mnemonic role; i.e. as a shorthand for which theorems or rules are supposed to be used in a given situation. This makes it difficult for students to map their notation onto probabilistic primitives such as events, in particular for multivariate distributions.

To overcome these problems, both chapter 5 and chapter 6 will be concerned with probability theory. The idea is to provide a ground-up introduction to probability theory with a focus on distributions that can be represented using well-behaved density functions. We advice a reader to make absolutely sure he or she understands the definitions in the green boxes in these chapters.

The disadvantage of this approach is the amount of reading material for the first weeks may seem excessively long, and we will therefore use stars, i.e. \star , to signify a particular section (including subsections) is of less significance, perhaps because it recaps material from other courses (such as the introduction to linear algebra), or because it is technical in nature and is supposed to give a more in-depth idea of what is going on (c.f. section 5.5 and section 5.4.6).

We obviously advice a reader to do the assigned homework problems (see course website), but failing that, we *strongly* encourage a reader to *read* the homework problems to get an idea about what parts of the material is more likely to occur at the exam. The focus in the exam is to either be able to understand the material well enough to make common-sense inferences about how they apply in particular situations, or to concretely apply the methods/definitions to concrete situations. Note solutions to the homework problems are included at the end of this book.

Based on feedback in previous semesters, we have begun implementing colored boxes as an aid for the reader. These boxes are used as follows:

Method: Key definitions or summaries

Summarizing a method or particular relevant result. Should be fairly self-contained and relevant as a how-to resource. Make sure you understand the content.

Example: Illustration of how to do something

A small (concrete) example of how to calculate something, either because it is exam relevant, or to test how certain definitions are used in practice.

Technical note: A warning or derivation

Used to provide additional details which may be technical, confusing or simply a lot of work. Easily (and sometimes best) skipped.

Note the use of boxes is still work in progress and, as with all other aspects of this note, we will be very happy to get feedback on how to best make use of them.

Updates in version 1.1

- Added section 3.4.1 on interpretation of PCA components which will be useful for project 1.
- Added section 6.3.4 about the cumulative density function and its inverse. This material should be familiar from a statistics/probability class.

Updates in version 1.2

- Added chapter 11 on statistical evaluation and comparison of machine learning models
- Renamed chapter 16 to avoid confusion with chapter 11 (no other changes)
- Bayesian networks (section 13.3) is now optional reading and will not be part of the exam

Updates in version 1.3

- Equation (11.35) for McNemars confidence interval was expressed for the wrong coordinates and have been updated. A small comment was added to the text as an explanation.
- Added section 11.2.1 to provide a brief explanation of baseline models (useful for project 2)
- Aligned notation in eq. (11.40) with the subsequent notation ($\hat{s} \rightarrow \tilde{\sigma}$)

Contents

Notation cheat sheet	V
Course reading guide	VI

Part I Data: Types, Features and Visualization

1 Introduction	3
1.1 What is machine learning and data mining	3
1.1.1 Machine Learning	3
1.1.2 Data mining	4
1.1.3 Relationship to artificial intelligence	4
1.1.4 Relationship to other disciplines	4
1.1.5 Why should I care about machine learning.....	4
1.2 Machine learning tasks	5
1.2.1 Supervised learning	5
1.2.2 Unsupervised learning	7
1.2.3 Reinforcement learning.....	11
1.2.4 The machine-learning toolbox.....	12
1.3 Basic terminology	13
1.3.1 Models.....	14
1.3.2 A closer look at what a model does★	14
1.4 The machine learning workflow	17
2 Data and attribute types	19
2.1 What is a dataset?	19
2.1.1 Attributes	20
2.1.2 Attribute types	21
2.2 Data issues	22
2.3 The standard data format	23
2.4 Feature transformations	24
2.4.1 One-out-of-K coding	25

2.4.2	Binarizing/thresholding	26
Problems	27
3	Principal Component Analysis	29
3.1	Projections and subspaces★	29
3.1.1	Subspaces	30
3.1.2	Projection onto a subspace	31
3.2	Principal Component Analysis	33
3.3	Singular Value Decomposition and PCA	38
3.3.1	The PCA algorithm	39
3.3.2	Variance explained by the PCA	40
3.4	Applications of principal component analysis	41
3.4.1	Example 1: Interpreting PCA components	43
3.4.2	Example 2: A high-dimensional example	44
3.4.3	Uses of PCA	46
Problems	49
4	Summary statistics and measures of similarity	53
4.1	Attribute statistics	53
4.1.1	Covariance and Correlation	55
4.2	Term-document matrix	56
4.3	Measures of distance	57
4.3.1	The Mahalanobis Distance	58
4.4	Measures of similarity	59
Problems	62
5	Discrete probabilities and information	63
5.1	Probability basics	64
5.1.1	A primer on binary propositions★	65
5.1.2	Probabilities and plausibility	65
5.1.3	Basic rules of probability	67
5.1.4	Marginalization and Bayes' theorem	67
5.1.5	Mutually exclusive events	69
5.1.6	Equally likely events	70
5.2	Discrete data and stochastic variables	74
5.2.1	Example: Bayes theorem and the cars dataset	75
5.2.2	Generating random numbers★	77
5.2.3	Expectations, mean and variance	78
5.3	Independence and conditional independence	79
5.4	The Bernoulli, categorical and binomial distributions	80
5.4.1	The Bernoulli distribution	80
5.4.2	The categorical distribution	81
5.4.3	Parameter transformations	82
5.4.4	Repeated events	82
5.4.5	A learning principle: Maximum likelihood	83
5.4.6	The binomial distribution★	85
5.5	Information Theory★	85

	Contents	XI
5.5.1 Measuring information	86	
5.5.2 Entropy	88	
5.5.3 Mutual information	89	
5.5.4 Normalized mutual information	90	
6 Densities and models	93	
6.1 Probability densities	93	
6.1.1 Multiple continuous parameters	94	
6.2 Expectations, mean and variance	97	
6.3 Examples of densities	98	
6.3.1 The normal and multivariate normal distribution	99	
6.3.2 Diagonal covariance	101	
6.3.3 The Beta distribution	102	
6.3.4 The cumulative density function	103	
6.3.5 The central limit theorem★	104	
6.4 Bayesian probabilities and machine learning	107	
6.4.1 Choosing the prior	109	
6.5 Bayesian learning in general	109	
Problems	113	
7 Data Visualization	115	
7.1 Basic plotting	115	
7.2 What sets apart a good plot?	121	
7.3 Visualizing the machine-learning workflow★	123	
7.3.1 Visualizations to understand loss	123	
7.3.2 Use visualizations to understand mistakes	124	
7.3.3 Visualization to debug methods	125	
7.3.4 Use visualization for an overview	126	
7.3.5 Illustration of baseline and ceiling performance	129	
7.3.6 Visualizing learning curves	130	
Problems	132	

Part II Supervised learning

8	Introduction to classification and regression	137
8.1	Linear models	137
8.1.1	Training the linear regression model	139
8.2	Logistic Regression	143
8.2.1	The confusion matrix	145
8.3	The general linear model★	146
Problems		149
9	Tree-based methods	151
9.1	Classification trees	152
9.1.1	Impurity measures and purity gains	152
9.1.2	Controlling tree complexity	156
9.2	Regression trees	158
Problems		161
10	Overfitting and cross-validation	163
10.1	Cross-validation	163
10.1.1	A simple example, linear regression	163
10.1.2	The basic setup for cross-validation	165
10.1.3	Cross-validation for quantifying generalization	168
10.1.4	Cross-validation for model selection	170
10.1.5	Two-layer cross-validation	170
10.2	Sequential feature selection	173
10.2.1	Forward Selection	175
10.2.2	Backward Selection	177
10.3	Cross validation of time-series data★	177
10.3.1	The setup	177
10.3.2	Cross-validation	181
10.3.3	Two-layer cross-validation	182
10.4	Visualizing learning curves★	182
10.4.1	The setup	182
Problems		185
11	Performance evaluation	189
11.1	Statistical testing for machine learning	189
11.2	Statistical primer★	191
11.2.1	Baselines	195
11.3	Setup I: the training set is fixed	196
11.3.1	Translating to a statistical test	196
11.3.2	First task: Evaluation of a single classifier	197
11.3.3	Comparing two classifiers	200
11.3.4	Estimating the generalization error of a regression model	203
11.3.5	Comparing two regression models	206

11.4	Setup II: The training set is random★	208
11.4.1	A problem	209
11.4.2	The correlation heuristic	209
12	Nearest neighbor methods	213
12.1	K-nearest neighbour classification	213
12.1.1	A Bayesian view of the KNN classifier★	214
12.2	K-nearest neighbour regression	217
12.2.1	Higher-order KNN regression★	218
12.3	Cross-validation and nearest-neighbour methods	218
	Problems	221
13	Bayesian methods	223
13.1	Discriminative and generative modeling	223
13.1.1	Bayes classifier	224
13.2	Naïve-Bayes classifier	225
13.2.1	Naïve-Bayes for non-binary data and robust estimation	227
13.3	Bayesian networks★	229
13.3.1	A brief comment on causality	232
	Problems	234
14	Regularization and the bias-variance decomposition	237
14.1	Least squares regularization	237
14.1.1	The effect of regularization	239
14.2	Bias-variance decomposition	242
	Problems	247
15	Neural Networks	249
15.1	The feedforward neural network	249
15.1.1	Artificial neural networks	249
15.1.2	The forward pass in details	250
15.2	Training neural networks	253
15.2.1	Gradient Descent★	254
15.3	Neural networks for classification	257
15.3.1	Neural networks for binary classification	257
15.3.2	Neural networks for multi-class classification	258
15.3.3	Multinomial regression	259
15.3.4	Flexibility and cross-validation	260
15.4	Advanced topics★	260
15.4.1	Mini-batching	260
15.4.2	Convolutional neural networks	261
15.4.3	Autoencoders	262
15.4.4	Recurrent neural networks	262
15.4.5	Serious neural network modelling	263
	Problems	264

XIV Contents

16	Class imbalance	267
16.1	Dealing with class imbalance	267
16.1.1	Resampling	268
16.1.2	Penalization	268
16.2	Area-under-curve (AUC)	270
16.2.1	The confusion matrix and thresholding	271
	Problems	275
17	Ensemble methods	277
17.1	Introduction to ensemble methods	277
17.2	Bagging	279
17.3	Random Forests	282
17.4	Boosting	283
17.4.1	AdaBoost	283
17.4.2	Properties of the AdaBoost algorithm \star	286
	Problems	288

Part III Unsupervised learning

18 Distance-based clustering techniques	291
18.1 Types of clusters	291
18.1.1 The distance-based cluster types	291
18.1.2 More elaborate cluster types	292
18.2 K -means clustering	292
18.2.1 A closer look at the K -means algorithm	295
18.2.2 Practical issues with the K -means algorithm	296
18.3 Hierarchical agglomerative clustering	297
18.3.1 Selecting linkage function	299
18.4 Comparing partitions	302
18.4.1 Rand index	306
18.4.2 Jaccard similarity	308
18.4.3 Comparing partitions using normalized mutual information	309
Problems	312
19 Mixture models for unsupervised clustering	315
19.1 The Gaussian mixture model	315
19.2 The EM algorithm	318
19.2.1 Why the EM algorithm works [★]	321
19.2.2 Some problems with the EM algorithm	323
19.2.3 Selecting K for the GMM using Cross-validation	324
Problems	326
20 Density estimation	329
20.1 The kernel density estimator	329
20.1.1 Selecting the kernel width λ	330
20.2 Average relative density	332
Problems	336
21 Association rule learning	339
21.1 Basic concepts	339
21.1.1 Itemsets and association rules	340
21.1.2 Support	341
21.1.3 Confidence	341
21.2 The Apriori algorithm	342
21.2.1 An example of the Apriori algorithm	343
21.3 Using the Apriori algorithm to find itemsets with high confidence	345
21.4 Some limitations	346
Problems	347
Solutions	349

XVI Contents

A Mathematical Notation	365
Elementary notation	366
Linear Algebra	366
Analysis	367
Probability Theory	369
References	371

Part I

Data: Types, Features and Visualization

1

Introduction

In this chapter, we will try to define machine learning and data mining, as well as give a high-level grouping of the various machine-learning methods. Understanding the different machine learning tasks is essential in order to determine which method is suitable in a given situation.

1.1 What is machine learning and data mining

How can we build intelligent machines? More than 65 years ago Alan Turing made this question the subject of his famous essay “*Computing machinery and intelligence*” [Turing, 1950]. Alan Turing suggested that when we phrase the question in this manner, we unavoidably get bogged down in the definition of the word “intelligence”. Instead, he proposed we should rather consider a different question: Can we construct a machine that can do the same things a human can do? This may ultimately be as hard to answer as the first question, but at least we don’t have to begin our efforts by defining intelligence. A second part of Turing’s essay discuss *how* we might build such a human-imitating machine. Turing proposed that instead of writing a computer program that behaves like a human from scratch, we should build a machine which initially cannot do a great many things but which can *learn from past experience*. For instance, if we wished to construct a machine which translates from English to French, we should instead construct a machine which is able to learn how to translate by observing examples of translated sentences in both languages, much like how a child acquires language.

1.1.1 Machine Learning

Machine learning is the implementation of Turing’s idea: The study of algorithms which can learn to do interesting things. The learning is based on observed data, whether from a spreadsheet, a sensor attached to a robot or human instructions. The goal of machine learning is therefore to use past experience to learn how to accomplish a task in such a way this learned ability generalizes to future situations of the same type, and we will simply refer to this process as *learning*. The focus of machine learning is on automatic and general methods. In other words, the goal is to learn *as much as possible* with *as little as possible* human intervention, preferably none at all.

1.1.2 Data mining

Data mining refers to the discovery of patterns or relationships in data and translating these into a useful structure. The datasets are usually considered to be very large, possibly undergoing change and too complicated for any human to sit down and understand them. For instance, an insurance company may continuously collect information about its customers relating to their spending habits and life situation. Making predictions about future events in the customers' life, or finding similar groups of customers, are tasks ideally suited for machine learning. In the following sections we will nearly exclusively discuss "machine learning methods" and a student may wonder what happened to data mining; this is partly to simplify the vocabulary and the student should keep in mind the methods are suited for various data mining tasks.

1.1.3 Relationship to artificial intelligence

Artificial intelligence is the construction of intelligent, thinking machines. Machine learning is an important subarea of artificial intelligence in that it is nearly unthinkable to have an intelligent system that cannot learn from past experience. Furthermore, it is arguably true that machine learning is the research area where most progress towards truly intelligent artificial systems is currently being made.

1.1.4 Relationship to other disciplines

Machine learning draws on a number of disciplines including most importantly mathematics, computer science and statistics. A basic knowledge of these subjects is required to get started, and specialization in machine learning will require good knowledge in at least one (though not necessarily all) of these subjects. However, this course will focus on the underlying machine-learning concepts to make the course material as accessible as possible for non-expert students.

It is worth mentioning that biology has an important role in machine learning and researchers are inspired both by evolution as an information-creating principle (this is known as *evolutionary computing*) as well as how the human brain processes and store information. The latter is known as (artificial) *neural computing* or *artificial neural networks*.

Furthermore, machine learning is a very broad subject which caters to very different types of researchers. At the same conference there will be mathematicians who present theoretical results (such as a mathematical analysis of a particular algorithm), very practically oriented computer scientists who have implemented a neural network on a low-power smartphone, a neuroscientist who uses machine learning to analyse brain data and a biologist who works with cancer genetics just to mention a few examples.

1.1.5 Why should I care about machine learning

We might not notice, but machine learning is becoming more and more pervasive in our society these years. Today, a person can use automatically trained speech recognition to order a product on an online shop which he learned about in an advertisement, which was specifically tailored to him by a machine-learning recommender system, and pay with a credit card that is automatically checked for fraud. All these steps involve machine learning; however it is just the tip of the iceberg. Artificial intelligence systems for self-driving cars can accomplish many transportation tasks, computers can

learn to automatically play video games better than humans and beat the grandmaster of Jeopardy. They can correctly recognize if an image contains a rock or an armadillo and learn to translate sentences from two languages with no expert input or initial knowledge of grammar. These are just a few examples of things that can be accomplished today!

We might still think these tasks, impressive and useful as they may be, have little to do with us in our professional lives. However, since the machine learning methods are general, the same algorithms that can classify observations into 20 000 categories can be used to solve much simpler data analysis problems we might encounter in our every-day life. To give an example, suppose Susi is an electrical engineer who is in charge of maintaining a hundred wind turbines. The wind turbines already register a lot of data (wind speed, amount of electricity generated, vibrations, etc.), and Susi notices that if the vibrations for a wind-turbine exceed a certain threshold even if the wind is not very strong, the turbine is likely to become faulty in the near future. Accordingly, she writes a small program: If vibrations exceed level x on a day where the wind is no greater than y , call in a technician to check the turbine. By putting this simple program in place, the downtime of the windfarm is reduced by 10%. However even in this simple case, Susi is faced with important choices: What should x and y be? Are there other things that are relevant to determine if the wind turbine should be monitored? If she comes up with another rule, how does she prove it is better (or worse)? Will this rule be suitable for the land-based windfarm?

Susi can try to work out these questions on her own. However there is a simpler option: She could apply standard machine-learning methods to *learn* x, y from the data. Or even better, she could apply standard tools such as logistic regression which we learn about in chapter 8 for *modelling* the break-down probability given *all* available variables and she could use proven techniques for validating her models such as cross-validation which we learn about in chapter 10. This will lead to better and more trustworthy predictions and, more importantly, it will save Susi a lot of time.

In general, the amount of data that is readily available in any given domain is growing at a rapid rate. When we as engineers consider why machine learning is important to us it is not necessarily because it will allow us to build the new self-driving car, discover the cure for cancer, or build a bridge-building robot, but because it will provide simple, off-the-shelf tools which will allow us to make efficient use of data which is already available. This book will provide an introduction to these tools.

In the following sections, we will introduce basic terminology surrounding machine learning and data mining. We will provide an overview of various forms of machine learning problems for later reference and discuss the basic machine-learning workflow.

1.2 Machine learning tasks

Some machine-learning terminology such as supervised or unsupervised learning is not fully settled in the literature and specific definitions often become overly technical. We will therefore first provide some examples of various tasks and types of learning before stating more exact definitions.

1.2.1 Supervised learning

In supervised machine learning the task is to predict a quantity based on other quantities. It is useful to distinguish between classification and regression:

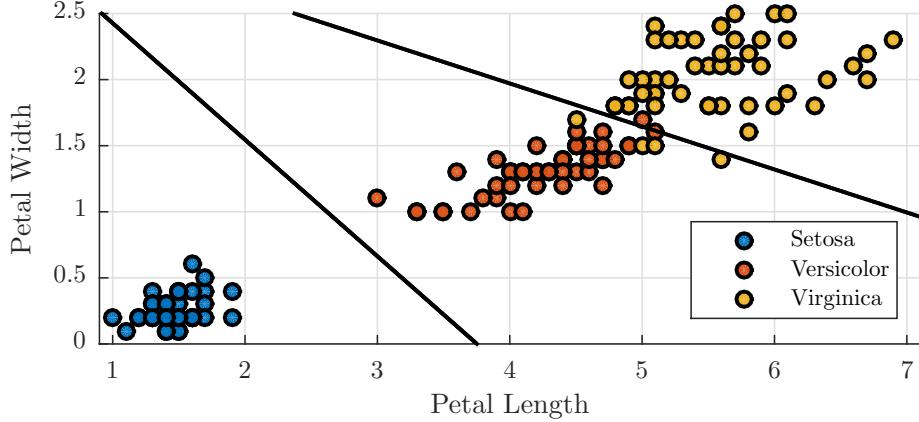


Fig. 1.1. A classification problem where we are given observations (the points) and class labels (the colors) and the goal is to come up with a rule for determining which class a point belongs to (one such rule is indicated by the lines). The rule can then be applied to new points.

Classification

In classification we are given observed values \mathbf{x} and have to predict a discrete response y . I.e., we are given discrete observations of some object and have to determine what class the object belongs to, see fig. 1.1. Examples include:

- We are given examples of hand-written digits and have to determine what number (between 0 and 9) is contained in an image. This is a multi-class classification problem since there are multiple categories to choose from.
- We are given the hospital records for a patient and have to determine if the patient will survive for one more year. This is a binary classification problem since there are only two choices (survives or dies).
- We are given a short sound-signal and have to determine which word is spoken. This is a classification problem but there are as many classes as there are words (perhaps about 20 000).
- We are given the social Facebook graph for a large group of people and have to determine how likely it is any two random people in the graph will form a friendship (or remain friends) in the next year (binary classification problem as response is discrete, i.e. link /not a link between people).
- We are shown pairs of images of faces and have to determine if the images are of the same person.

Regression

In regression problems we are given observed values \mathbf{x} and have to predict a continuous response y , see fig. 1.2. Examples include:

- We are given historical data of the stock market and have to predict the performance of a single stock the coming Monday (prediction of a single variable).
- We are given a person's height and have to determine his or her weight (prediction of a single variable from another).

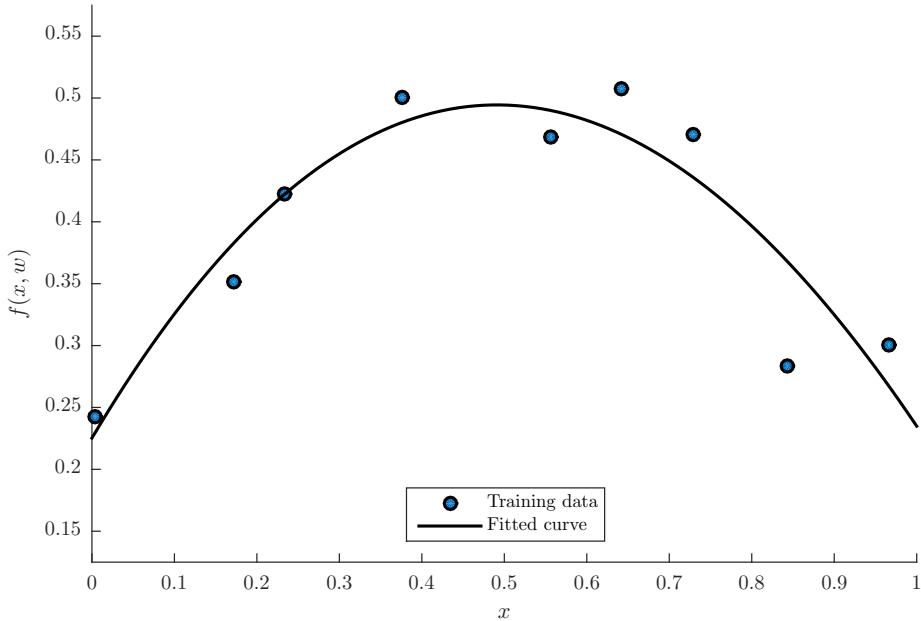


Fig. 1.2. A one-dimensional regression problem where we have to predict the y -values based on the x -values. The fitted regression model is indicated by the black line.

- We are given the performance of all stocks in the past and have to predict the performance of all stocks for the next five days (massive regression problem with many output variables).
- We are given weather information from yesterday and have to predict the temperature in five major cities tomorrow (regression of five variables).
- We are given the hospital information of a person and have to determine how many days he is going to survive (prediction of a single variable).

Notice these problems are quite different. In principle we could imagine we could solve the weather prediction problem perfectly provided our model was good enough and we had enough measurements, however we could not dream of being able to exactly predict a person's weight from his or her height, thus our prediction would in this case be guaranteed to be imperfect.

The commonality of all these tasks is that we are in all instances trying to determine a mapping where we are given observations (for instance an image of a digit) *as well as* examples of what the observation should map to (for instance the digit 4) or, in the case of regression, observations of past historical data of the patients *along with* observations of how long the patients survived. These examples are therefore attempts to directly generalize from past experience and in that sense we know what task we have to solve as well as whether we solved it correctly or not. Machine learning problems where we have both observed observations and observed target values is known as *supervised learning problems* or simply *supervised learning*.

1.2.2 Unsupervised learning

The opposite of supervised learning is *unsupervised learning*. Consider an example dataset consisting of images of animals. We may immediately consider building a machine learning method

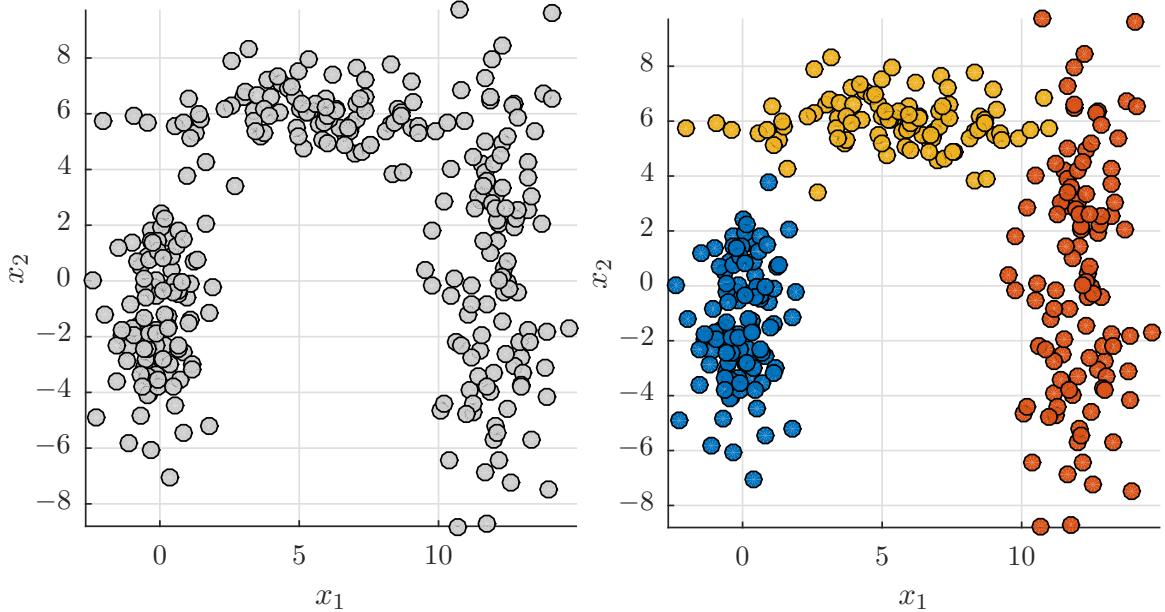


Fig. 1.3. A 2D clustering example. A clustering is given a dataset (here the 2D dataset shown in the left-hand pane) and has to estimate plausible divisions of the observations into clusters as indicated in the right-hand pane.

which tries to discover *what* animal is in the picture (a duck, a lion, an elephant, etc.). However, for most images on the internet we do not know what animal *is actually in* the image. Surely, we can sit down and label a few thousand of the images ourselves, train a supervised method (a classification method in this case) on the labelled images, and then use this method to determine the labels of all other animal pictures on the internet – but this is very boring and not reflective of how humans actually learn.

Unsupervised learning tries to solve this and similar problems where *we do not* have access to any “ground-truth” label information (such as the identity of the animal in the image) but we try to discover this labelling from the data alone. See fig. 1.3 an example of clustering where the goal is to cluster (label) the gray points in the left-hand pane and an example clustering is indicated in the right-hand pane by the coloring. Examples of clusterings include:

Clustering

- In the animal example, the goal was to group images into *clusters* such that each cluster represented a given type of animal.
- Given genetic sequence data from a number of bacteria, try to find natural groupings of the genomes (roughly corresponding to species).
- Given a large collection of documents, try to determine clusters of similar documents corresponding to topics.

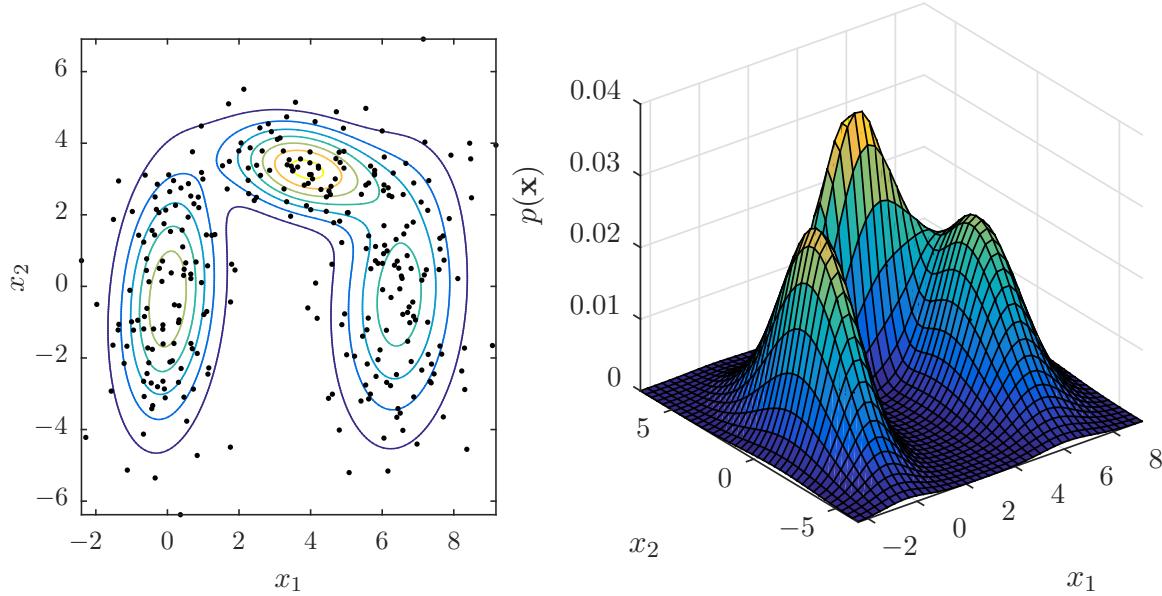


Fig. 1.4. A 2D density estimation example. Given the black points, the density is estimated and plotted in the right-hand pane.

Density estimation

In *density estimation* we try to quantify how likely (or unlikely) a given future observation is given past observations, i.e. the probability distribution of the data, see fig. 1.4. Consider the hand-written digit example. Suppose you are shown four images of hand-written digits and are told they are from the same person. Suppose then you have to determine if a fifth (until now) unobserved digit is written by the same person. This task involves estimating the relative *variability* in the person's hand-writing and how *plausibly* it is he has written a given digit – this is known as density estimation. Other examples include:

- You are at a large archeological dig and told where scientists have found archeological remains in the past. You have to decide the next place to excavate. Estimating the place with the highest probability of a new find from past finds is a density estimation problem.
- You are working for an oil company and try to estimate the drill site with the highest chance of finding oil based on past drilling.
- You are a microbiologist and you are trying to find out how typical a particular cell is given other observed cells of the same type. Being able to detect atypical cells could be relevant to determine diseases such as cancer.

Anomaly detection

Anomaly detection is figuring out which observations significantly deviate from other observations. While what constitutes a *significant deviation* is highly dependent on the context, humans nevertheless have a natural ability to carry out this task. We may for instance consider the red observation

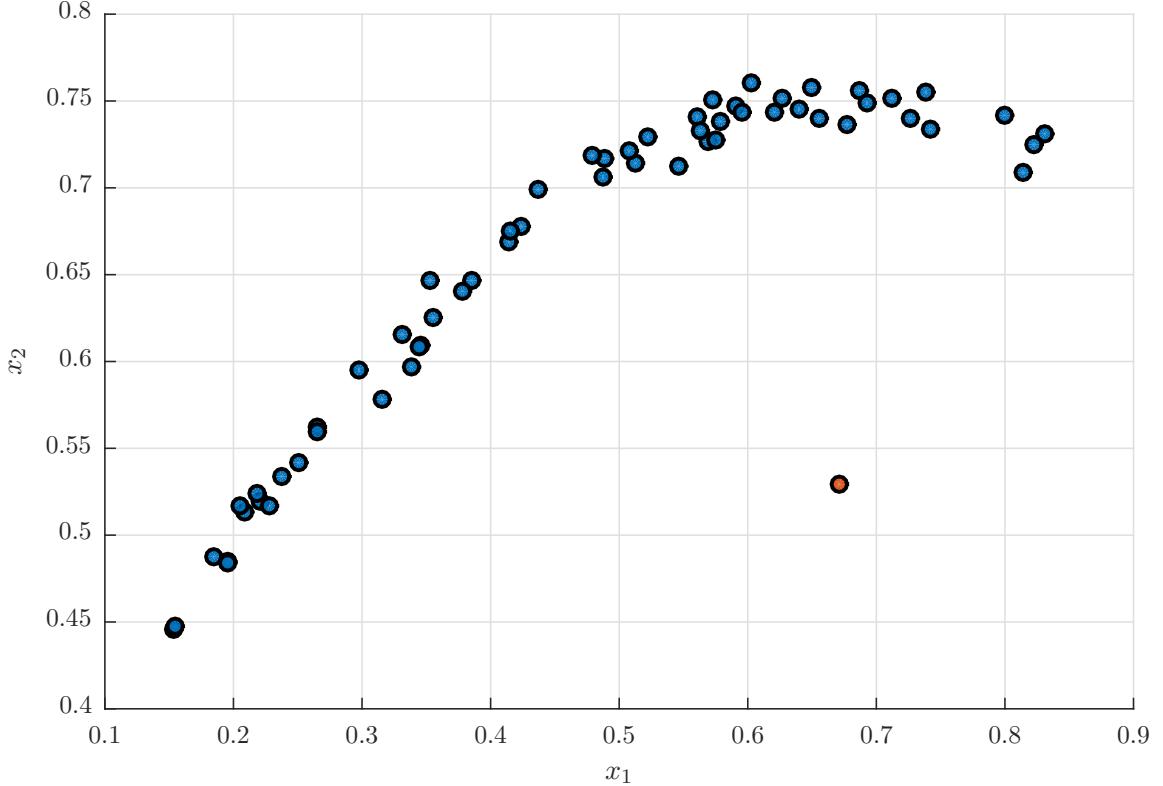


Fig. 1.5. Anomaly detection tries to discover observations that differ from the rest of the dataset.

in fig. 1.5 to be significantly different from the other observations from a number of perspectives. This includes that it is simply quite far away from its nearest neighbour relative to the other observations distance to their neighbours or that it appears not to follow the same “tendency” (the banana-shaped curve) as the other observations. Anomaly detection can be considered closely related to density estimation since an observation which is very implausible (low density) could be considered an outlier. Anomaly detection is relevant in a number of situations including:

- You work for a credit-card company and have to determine if a transaction deviates from common transactions in order to detect fraud.
- You supervise a windmill farm and based on past behaviour have to determine if a windmill is beginning to behave differently indicating it may require repair.

Association mining (rule-induction)

Association mining (or rule-induction) is figuring out rules which hold approximately in a data set, see fig. 1.6. Suppose you work for an online book seller and you are given a large dataset consisting of which books different people bought. In order to show relevant adds, you want to come up with

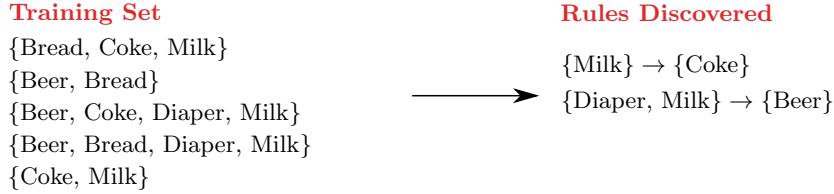


Fig. 1.6. Given examples of what people have previously bought, association mining tries to discover rules for what they will likely buy in the future. For instance a person who buys milk is likely to also buy coke.

rules such as: “*If the person bought both book X and book Y, then he is likely to buy book Z*”. This is known as association mining or rule discovery. Other applications are:

- Given which items people have historically bought in a supermarket, discover what other items each customer is likely interested in.
- Given a number of patients’ medical history, determine which past illnesses and conditions imply high risk for other, future illnesses: “*Common cold and operations imply high risk of pneumonia*”.
- Given past life experience, figure out that drinking the past night implies hangover.

Dimensionality reduction

In dimensionality reduction, we try to discover a simpler representation of a very high-dimensional dataset, see fig. 1.7. Consider for instance a dataset of faces. In one example the dataset is in a modest resolution (the images may be 500×500 pixels) and in the other example the same images are in a very high resolution (say 5000×5000). The later dataset contains 100 times more information than the former, however to a human they contain the same information: We can recognize the identity of the people from both datasets, their age, race or gender, their emotional state etc. If we think about it $M = 750\,000 = 500 \times 500 \times 3$ numbers (there are 3 color channels) is still a lot of data. If we had access to a large set of numbers such as the distance between the eyes, length of the nose, eyes and mouth, width and height of the face, the color of the skin, the curvature of the mouth, etc. we might retain most of the relevant information in the faces while using far less than M numbers. Thus dimensionality reduction is learning a representation of an M dimensional object which uses $M' < M$ numbers while retaining most of the relevant information. Other examples are:

- Lossy compression.
- Finding a summary of a sentence or book.

1.2.3 Reinforcement learning

Finally, for the sake of completeness reinforcement learning is worth mentioning. Reinforcement learning corresponds to the case where a computer has to control a robot based on sensory input and a *reward signal*. That is, at any given time the robot observes the state of the world (for instance a screenshot if the robot should learn to play a video game), selects an action (for instance pushing a particular button) and possibly receives a reward, for instance simply if the robot loses the video game or not. Reinforcement learning can be seen as a type of regression (i.e. supervised learning) where we are trying to predict the reward based on the current action and input signal.

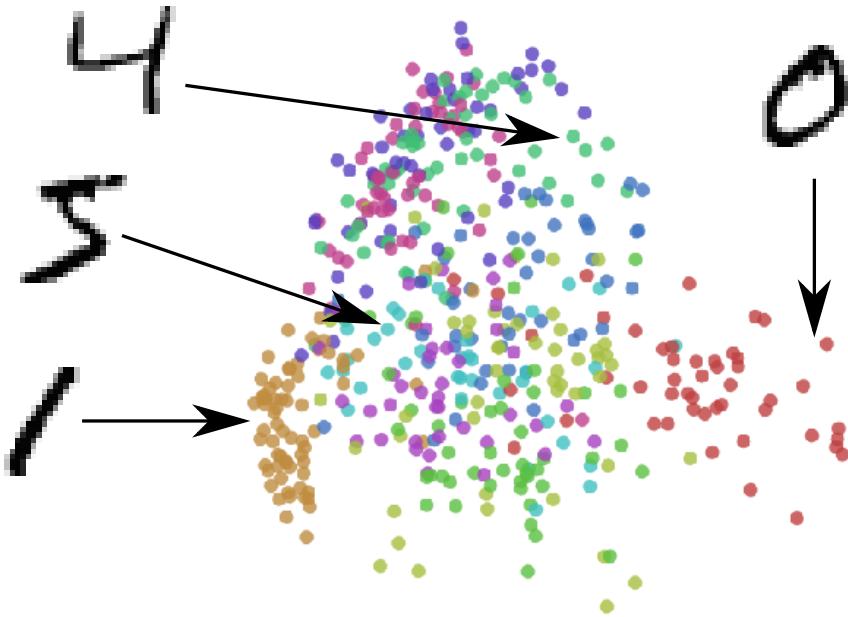


Fig. 1.7. Dimensionality reduction. High-dimensional images (each image contains 28×28 pixels corresponding to 784 dimensions) are mapped onto a 2D domain, that is compressed into a 2-dimensional vector. Colors indicate different digits ($0, 1, \dots, 9$). Notice, digits that are the most dissimilar such as 0 and 1 are mapped to points far apart.

However, as rewards are rarely observed it is usually considered to be different from supervised learning. This course will not consider reinforcement learning, however, many of the techniques used for reinforcement learning will in fact be introduced in this course.

1.2.4 The machine-learning toolbox

The above taxonomy is not exhaustive or set in stone and there are many problems which does not exactly fit into the above categories. Consider a system which has to translate from English sentences to French sentences. In some sense it is a classification problem, however there are infinitely many sentences to choose from and treating it as a generic classification problem is not helpful. Or suppose you want the computer to learn if two variables are causally related, i.e. that smoking *causes* cancer in a medical records dataset. We could naively imagine treating this as a market-basket problem, however if this is taken serious we would have to believe that, say, putting milk in someones market basket really cause them to put coke in it as well and this should hint causation is something more.¹

This might feel discouraging: Why take this course if I don't learn about the methods for the tasks I *really* think are cool. Fortunately, even the most advanced applications of machine learning rely on re-using and combining simpler tools, nearly all of which are introduced in this course

¹ This is not to say that automatic translation or inference of causation is beyond machine learning. See for instance: https://en.wikipedia.org/wiki/Neural_machine_translation and [Pearl et al., 2016].

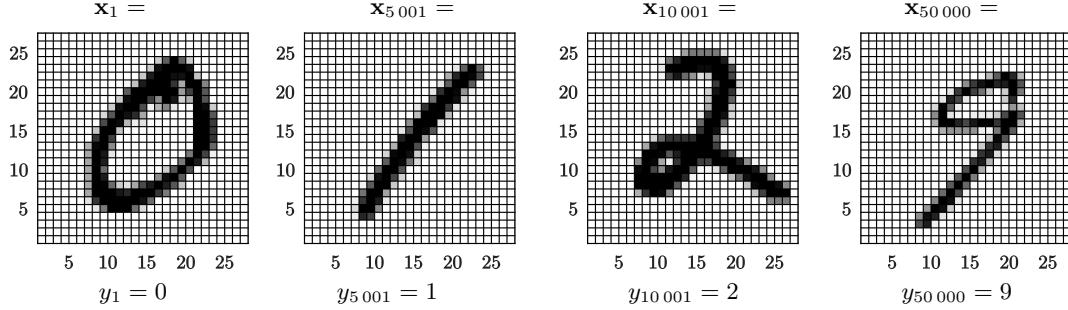


Fig. 1.8. Four observations (images of handwritten digits) from the MNIST dataset containing a total of $N = 60\,000$ handwritten digits. Each observation consist of a 784-dimensional vector \mathbf{x}_i and a label y_i which can be either $0, 1, \dots, 9$.

in some form. A useful analogy is building a machine where this course supplies the wrench, the screwdriver, the soldering iron and the other tools required to get started.

Secondly, it is worth emphasizing that the difference between the techniques supplied in this course and the absolute forefront is not as great as in other disciplines, say a basic book on mechanics and a book on general relativity. The basic architecture of one of the absolute best image-recognition network (Inception v3) is a variant of the humble simple feedforward network introduced in chapter 15.

1.3 Basic terminology

Let's make the above discussion more concrete by considering a realistic problem. Consider the handwritten digits from the MNIST dataset shown in fig. 1.8. Each digit consists of a 28×28 pixel image which, since the images are black and white, can be represented as a vector \mathbf{x} consisting of $M = 784$ real numbers² and we write this as $\mathbf{x} \in \mathbb{R}^M$. The goal is to build a machine which takes such an image \mathbf{x} and outputs the identity of the digit, i.e. if it is $0, 1, \dots, 9$. This is a non-trivial problem since there is a huge variability in how people write digits (stroke, style, open or closed digits, rotation, translation, etc.), however it is a trivial problem for humans. Our goal is therefore to construct a function f which takes a M -dimensional vector as input and returns $0, 1, \dots, 9$ depending on which digit it believes the image contains.

If we adopt a machine learning approach our goal is to construct a system which can *learn* how to solve the above problem. The system *learns* from experience. In this case the past experience (i.e. the *data*) would consist of a number of example images for each type of digit. For instance we would have 5 000 examples of the digit 0, $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{5000}$, then 5 000 examples of the digit 1: $\mathbf{x}_{5001}, \mathbf{x}_{5002}, \dots, \mathbf{x}_{10\,000}$ and so on up to image $\mathbf{x}_{50\,000}$ of the digit 9. Let $N = 50\,000$ be the number of examples, we collect all this information in an $N \times M$ matrix \mathbf{X} and a N -dimensional vector \mathbf{y} :

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_{50\,000}^T \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} 0 \\ \vdots \\ 9 \end{bmatrix},$$

² The real numbers are all numbers such as $5, -1, \pi, \frac{1}{3}, \dots$. For instance the pixel values can lie in the interval $[0, 1]$.

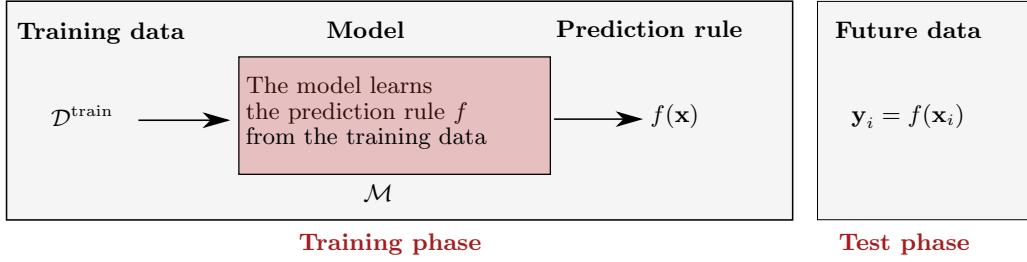


Fig. 1.9. A model in supervised learning. Given a set of training data, the model learns a prediction rule $f(\mathbf{x})$ in the training phase. Later, in the test phase, this rule can be applied to new, unobserved data.

where y_i is either $0, \dots, 9$ depending on the digit in image \mathbf{x}_i . Taken together we let $\mathcal{D}^{\text{train}} = (\mathbf{X}, \mathbf{y})$ denote all data available for training the machine learning method and $\mathcal{D}^{\text{train}}$ is known as the *training set*.

1.3.1 Models

Given the training set $\mathcal{D}^{\text{train}}$ in the above example, machine learning then consists of constructing a program which takes the training data $\mathcal{D}^{\text{train}}$ and returns a function

$$f : \mathbb{R}^M \rightarrow \{0, \dots, 9\}$$

Learning the function f from the data is known as the *training phase* or alternatively as the *learning phase* or simply *learning*, see fig. 1.9. Once this function is learned it can be used to determine the identity of unobserved images of digits. For instance if for an image \mathbf{x} we have that $f(\mathbf{x}) = 5$ this means the algorithm predicts that the image contains the digit 5. These new observations used to test the model is known as the *test set* denoted $\mathcal{D}^{\text{test}}$.

How well a model performs when evaluated on previously unseen data is known as the *generalization error*, and this key quantity in machine learning is what ultimately decides which model is better. The generalization error should be distinguished from the *training error* which is the average error on the training set; we will have much more to say about the generalization error in chapter 10. Notice that since the training set was used to train the model, we should expect the training error to be lower than the generalization error.

A computer program (along with the assumptions it relies upon) which carries out the above steps –i.e. based on a training set it constructs a function f – is known as a *model*. Different models will be denoted $\mathcal{M}_1, \dots, \mathcal{M}_S$.

1.3.2 A closer look at what a model does*

The above description present an accurate but somewhat abstract view of what machine-learning attempts to do. In this section we will try to provide an intuition of what happens “inside” the function f so as to get an understanding of what may go wrong. Note these details are obviously particular to the machine-learning method used to construct f and the discussion should therefore not be taken too literally.

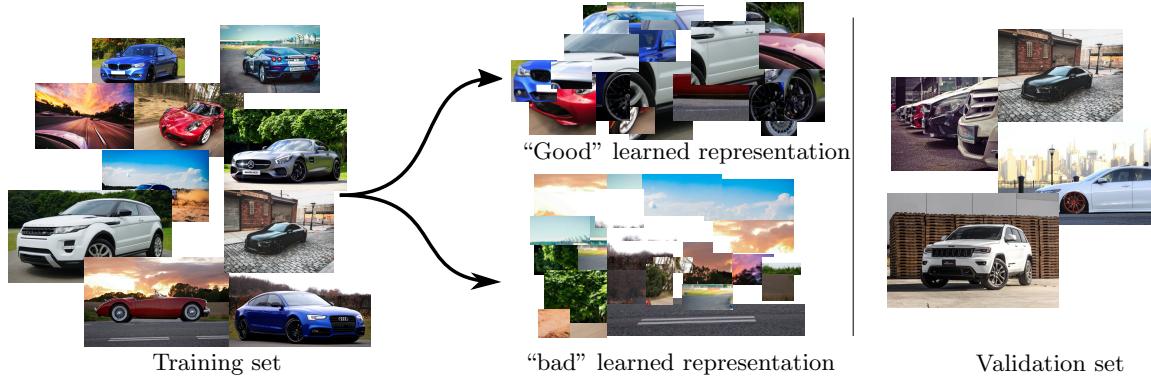


Fig. 1.10. In the broadest sense, a machine learning method takes a number of observations (here, images of cars) and build an (approximate) internal representation that captures statistical relationships in the images. One should think of each image as bringing a little piece of information about this relationship and the goodness of a method as how well it can make use of this information. Since the method does not know what it should focus on, it can either build a successfull representation (center, top), or focus on spurious properties of the training set (center, bottom). For this reason evaluating the method with data not used to train the method (the validation set) is important as these images are unlikely to conform to the (spurious) representation and will therefore offer an accurate idea about how well the method performs.

For the purpose of illustration, we will consider an example problem quite similar to the digit classification problem from the previous section, except we are now considering larger images and we wish to categorize them into classes such as *cars*, *cities*, and so on.

In fig. 1.10 we have illustrated an approximate view of what a machine learning method does in such a setting: We start out with a dataset comprised of a collection of images³ (and their labels) and the goal of the method is then to predict which category the images fall within on hitherto unseen data. This is done by somehow learn a useful internal representation based on what these images have in common. Speaking in broad terms, these learned representation of for instance the *cars*-class will correspond to a cobbled together collection of *car-like* elements and an implicit statistical relationship between them. A machine learning method trained to detect cars is likely to be *sensitive* to wheels, but will often be insensitive to the *number* of wheels and only have a very rough idea about where wheels are supposed to be located relative to each other, as illustrated in fig. 1.10 (center, top).

The role of data

Let us state a trivial but important point: these learned features/relationships are learned *from the data*. In other words, think about each observation as containing a small piece of the overall true meaning of *car*.

Machine-learning method can pick out these pieces at varying degrees of efficiency, and as a rule *less* efficient than humans. The *advantage* machine-learning methods have over humans is in making use of *more* data than a human can easily comprehend. It is therefore *the ability to make use*

³ Images obtained from <https://www.pexels.com>

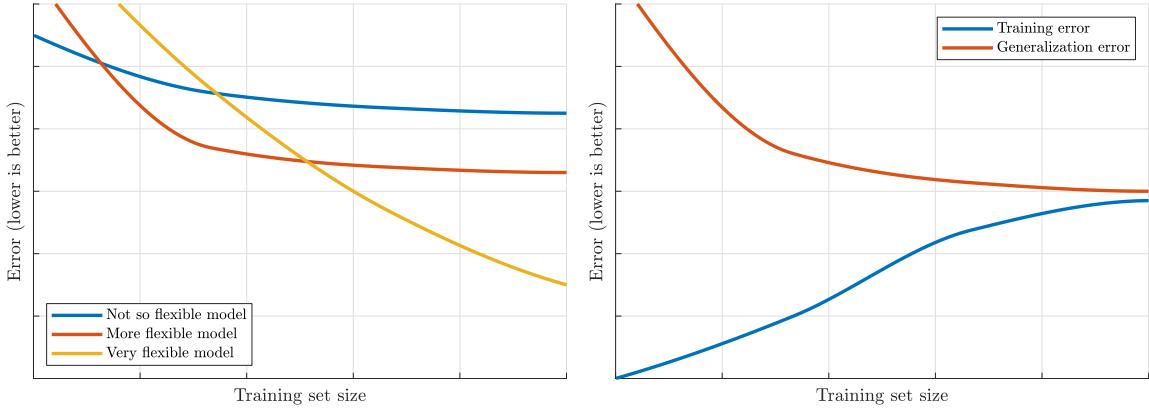


Fig. 1.11. Left: With more data we should expect the generalization error to drop, however this drop can be expected to occur sooner for less flexible models (blue and red curve) as they are less prone to learn spurious relationships in the data, however, it will be sustained longer for more flexible methods (yellow) which are able to learn a more powerful representation. Right: When a model is trained on little data, we should as a rule expect it to learn all the particulars of the dataset, including spurious relationships. It will therefore perform exceedingly well on the training set (overfit), but generalize very poorly. With more data, these two curves will approach each other as the training error increases and the generalization error drops.

of lots of data, rather than intrinsic sophistication, which makes machine-learning methods work: That is why data, specially lots of high-quality data, is so important.

When learning fails

The goal of machine learning is to find models which generalize well to future data (i.e., data the model is not trained upon), and this is measured by the generalization error which is what ultimately decide which model is better.

A model may have a high generalization error for three reasons: Firstly, it may be misapplied, but we will leave that aside. Secondly, the method can be too weak (i.e., inflexible) to learn a sufficiently rich representation to solve the problem, and thirdly, since the machine-learning methods do not have any intrinsic idea about *what* to focus on in a set of images, they might focus on the wrong things. This is illustrated in fig. 1.10 (center, below) where the method here think that the car-images has to do with things common for the images (roads, sky, tree) and not the car itself. This leads to a rule which is useful to categorize images in the training set, but which *generalize* poorly, hence high generalization error.

There are two things which determine how prone a given method is to degenerate behavior, both illustrated in fig. 1.11, namely the amount of data and how flexible the machine learning method is. In fig. 1.12 we have tried to give a rough indication of how some of the methods in this book are sorted according to complexity. One can think of any type of model in terms of how flexible a representation it can ultimately learn. With little data, a very flexible model will be able to learn any number of representations and will therefore often fail, and we say they tend to *overfit*. This means less flexible models, such as linear models or the like, will often out-compete more sophisticated choices, and methods such as regularization (which is a general technique for

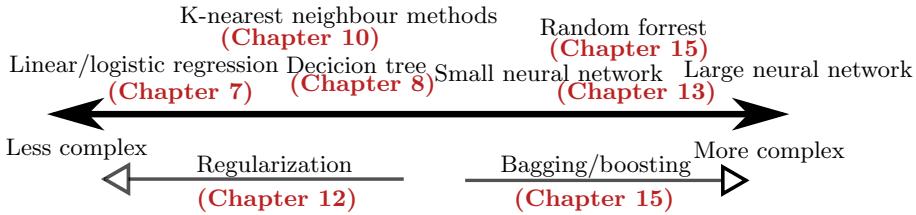


Fig. 1.12. A rough overview of the relative complexity/flexibility of models encountered in this book; note this illustration is somewhat subjective and depends on assumptions about how the methods are applied. Techniques such as regularization serve to make a model less flexible (more robust), whereas ensemble techniques such as bagging/boosting has the opposite goal.

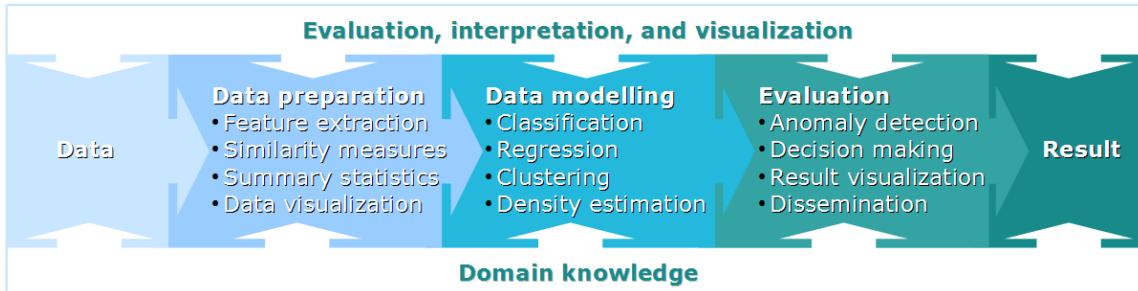


Fig. 1.13. The machine learning workflow used in this course. In the first step the problem is analysed and one obtains an overview of the data including potential issues. In the next step an appropriate machine-learning method is implemented and potentially modified and in the third step the model is evaluated. The lessons learned are used to improve on what happened in step 1 and 2 to produce a new model which is compared to the first. The process then repeats. At all times knowledge of the domain should be exploited.

reducing the flexibility of a model which we will learn about in chapter 14) will be relevant. However when more data becomes available this becomes less of an issue, and so the more flexible models will outcompete the less flexible simply because the more flexible model is able to learn a more sophisticated, and therefore more accurate, representation. The simplicity of which we can heap extra neurons into an artificial neural network (chapter 15) makes them the poster child of extremely flexible models and that, together with the availability of large datasets and powerful computers, has driven the neural network revolution of the past decade.

1.4 The machine learning workflow

To summarize the above discussion we will now present the workflow of a machine learning practitioner fig. 1.13. Suppose we are presented with a problem for the first time. The first thing we have to understand is what type of problem it is: Is it a supervised problem or unsupervised problem? Regression or classification? etc. Secondly, we should try to understand the available data: Data is the life blood of any machine learning method and without having a good handle on what our dataset is, if there are issues with our dataset and if the task appears feasible given the dataset it is

difficult to proceed. This is the data mining step and during this course we will present a number of techniques and terms for working with a dataset including visualization.

Having analysed the data and problem we must select an appropriate method and possibly apply modifications to the method. This is step 2 where we ensure the method is implemented correctly and produce an output. At this point we have our first model: \mathcal{M}_1 . To take the handwritten digit task as an example, the method may produce a prediction rule which can be applied to test data.

This brings us to step three which is absolutely critical but often neglected: Evaluation and dissemination. For a model \mathcal{M}_1 we must evaluate how well the model performs (generalizes) on previously unseen data and quantify this numerically. We should also look at examples where the model failed and try to get a idea on why this happens and how we might do better. This is also the step where we disseminate the results to be used by others either in a report or as part of a more general workflow in a scientific team or company. When this step is completed we essentially start over with whatever lessons we may have learned and see if we can do better at step 1 and 2. The result may be a new model \mathcal{M}_2 which must again be tested to see if it is better or worse than \mathcal{M}_1 .

During this course, we will learn techniques which apply to all these steps. The first section of the course relates to the first step; the two next sections treat the second and third step of the workflow, both for supervised machine-learning techniques and unsupervised machine-learning techniques.

Data and attribute types

Without data there would be no need for machine learning. In this chapter, we will define what we mean by a data set, attribute types and discuss commonly encountered data issues such as missing values. Common data issues such as missing values have been present from the day humans first began recording data but is perhaps particularly treated in depth in for instance clinical psychology where standardized treatment of corrupted data is a major concern and it is also within clinical psychology the distinction of attribute types (ration, interval, etc. which we will introduce below) was originally introduced by Stanley Smith Stevens in 1946 [Stevens, 1946].

2.1 What is a dataset?

Data, or a dataset, is a collection of electronically stored information. Quite simply, it is *what we have to work with*. Examples could be:

- Biomedical information about patients in a hospital.
- A collection of text files, for instance corresponding to news stories.
- The temporal development of 10 stocks on the New Your stock market over 50 days.
- A collection of 3D models, each model being defined as a (varying) collection of points on its surface.
- A social graph, for instance from a social network (who is friends with who).

Often, and in all cases considered in this book, a dataset can be thought of as standardized measurements of objects of the same basic type. For instance, measurements of patients, cars or documents. Each such measurement of a single object is called an *observation*. For instance, for the dataset comprised of biomedical information each observation corresponds to a patient, in the text-files example each observation is a text file, in the stock market each observation is the value of the stocks on a given day, for the 3d model case each observation is a particular model, and finally for the social graph each observation is an edge. The number of observations in a dataset is denoted N .

It is useful to distinguish between datasets which are simple and those which are complex. A simple dataset is a dataset of N observations where each observation corresponds to a fixed number of recorded values. For instance in the patients dataset, we can imagine that for each observation we record the patients name, age, weight, blood pressure and so on. Each recorded value is called

Table 2.1. Ten entries from the Cars dataset comprised of $N = 142$ observations and $M = 9$ features

ID	MPG	Cylinders	Horsepower	Weight	Year	Safety	Acceleration	Origin
1	18	8	150	3436	70	4	11	France
2	28	4	79	2625	82	4	18.6	USA
3	26	4	79	2255	76	3	17.7	USA
3	29	4	70	1937	76	1	14.2	Germany
4	NaN	8	175	3850	70	2	11	USA
5	24	4	90	2430	70	3	14.5	Germany
6	17.5	6	95	3193	76	4	17.8	USA
7	25	4	87	2672	70	-100	17.5	France
:	:	:	:	:	:	:	:	:
142	15	8	198	4341	70	2	10	USA

an *attribute* or *feature* and the number of features will be denoted by M . In table 2.1 is shown an example of a simple dataset corresponding to $N = 142$ observations of cars where for each car we record $M = 8$ features.

A non-simple dataset is a dataset where there is additional structure or complexity. This could be:

Temporal If for instance the observations are made sequentially over time (such as the stock market).

Varying number of features If each observation has a varying complexity. For instance the text files has a varying number of words, and the 3d models a varying number of surface points.

Self-referential structure For instance in the social graph, the edges refer to the same group of people.

These definitions are not set in stone and require some common sense. For instance, we *could* claim that the patients dataset was temporal since patients will arrive at different dates, however what matters is if this temporal ordering is considered important for the machine learning task. In the patients example, no doctor would consider it important if a patient arrives in May or June in terms of making his diagnoses, whereas in the stock market example there will typically be important upward/downward trends and causal structure which makes the temporal ordering a key feature.

In this course we will consider simple datasets of N observations and M features such as the Cars dataset. This may sound very restrictive; however, the techniques suitable for more complicated datasets can often be seen as specialization of the tools we here consider for the simple case. In addition, the simple case covers nearly all data that can be plugged into a spreadsheet and as we will see, it is often possible to transform data that at first glance may appear complex (for instance images or text) into the simple format.

2.1.1 Attributes

If we consider the cars dataset in table 2.1 we immediately notice the attributes are different. For instance the *Origin* feature is one of three text strings, the *ID* is an increasing sequence, presumably only used for bookkeeping purposes, *safety* is a number of stars (from 1 to 5) and so on. It is useful

to introduce some general vocabulary to describe different types of attributes. The most simple distinction is between attributes that are continuous and discrete. In particular, we define:

Continuous: A continuous attribute is one which, if it can take values a and b , then it can also take any value between a and b .

Discrete: A discrete feature is one where if it can take a value a , then there is a positive minimum distance to the nearest other value b it can take

Binary: A binary feature is a discrete feature which can only take two values. Usually these are denoted 0 (false) or 1 (true).

For instance the weight of the car is continuous since if a car can weight 1000 pounds and 2000 pounds then presumably it can also take any weight between 1000 and 2000 pounds. Notice continuous does not imply it can take *any* decimal value since the weight cannot for instance be negative and this is why the definition is somewhat technically sounding.

The definition of discrete tries to capture the intuition that the variable changes in discrete steps. Most commonly these steps are integer values such as 1, 2, 3, 4, 5 (the safety rating is such an example), however the steps need not be integer valued. For instance a safety rating which could take the possible values $0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}$ and 1 would also be discrete.

2.1.2 Attribute types

Machine learning methods operate on numbers and not text strings. Accordingly, to apply a machine-learning method to the *Cars* dataset in table 2.1 we must therefore translate the country of origin (which can be *USA*, *France* and *Germany*) into numbers (1, 2, 3). Notice that which country is assigned which number is *our* convention and should be irrelevant: If we choose to assign *USA* to 3 (and *Germany* to 1) this should not affect our subsequent computations (or so we hope!). This makes the country of origin *different* from the safety rating since a safety rating of 5 is larger than a safety rating of 3 and this feature *should* inform our machine-learning method. In other words the safety rating is ordered (smaller, larger) whereas the country of origin is not ordered. This is an example of two variables of different *types*. By convention it is common to distinguish between four different types of attributes:

Nominal: If the variable is not ordered and only uniqueness matters. An example is the country of origin or the id.

Ordinal: If the variable is ordered (smaller, larger). An example is the safety rating.

Interval: If the variable is ordered and the relative magnitude of the variable has a physical meaning. The year is interval since the difference between say 82 and 85 (three years) has the same meaning as the difference between years 77 and 80, however the safety rating is not interval since a difference in safety rating of 2 and 4 and 3 and 5 does not have a physical meaning.

Ratio: If the value 0 of the variable has a specific, physical meaning. I.e. it makes sense to say one value of the variable is “twice as large” as another. The year is not ratio since it has no particular meaning to say that 62 is twice as large as 31, whereas the volume of the engine does have a particular physical meaning since a value of 0 means the combustion chamber has volume 0 cm³.

Notice the types are a hierarchy. That is, if a variable is interval it must also be ordinal and nominal (but is not categorized as such). It is distinguished from ordinal and nominal as the relative magnitude of the variable also has a physical meaning and therefore categorized as interval. Ratio

and interval are the variables which are most easily confused. Suppose for instance we record the longitude of cities as an attribute (the longitude is the east-west location on the globe in degrees). Is this variable ratio? One could reason the answer is yes, since a longitude of 0 has the “meaning” that we are on a particular place on earth, in this case the north-south line that passes through the city of Greenwich in England. However, what the definition has in mind is that the physical meaning is not purely “by convention” but actually refers to a physical property of the globe. Since the city of Greenwich is just a convention, the correct answer is that the longitude is interval and *not* ratio. If one is in doubt if a variable is ratio or interval it is often useful to imagine that civilization had to start over and come up with the same sort of variable. If the new civilization would assign “zero” the same meaning as we do the variable is ratio, else it is interval. In the case of the longitude, if we had to come up with the longitude system anew the longitude of 0 might as well lie anywhere on the globe, and similar for the year where we could start our calendar from any other year than the year where medieval monks estimated Jesus to have been born.

If the variable is continuous or discrete is often independent of whether the variable is ordinal, nominal, interval or ratio. For instance the variable *Cylinders* is *discrete ratio* (zero cylinders has a specific meaning) and *acceleration* is *continuous ratio*. To summarize the variable types for the *Cars* example:

Origin: Discrete, nominal.

Safety: Discrete, ordinal.

Cylinders: Discrete, ratio.

Year: Discrete, interval.

Miles/Gallon, Horsepowers, Weight and Acceleration: Continuous, ratio.

2.2 Data issues

Now that we have described the features of the *Cars* dataset table 2.1 we turn our attention to the actual values and immediately notice some oddities: Car #7 has a safety rating of -100 , and for car 4 the Miles/Gallon is NaN¹. Unfortunately, such issues are surprisingly common especially when humans have to enter values and we distinguish between the following issues:

Irrelevant or spurious attributes: The ID column is irrelevant as it only depends on the ordering of the data.

Outliers: The safety rating of -100 must be due to some kind of error. We call such observations outliers.

Missing data: The Miles/Gallon attribute for car #4 is missing.

As a rule, attributes which can be known to be spurious or irrelevant such as the ID should simply be discarded, and in doing so we will reduce the number of dimensions of the dataset (M) and make the machine-learning problem easier. Missing data can be treated in four main ways:

- Some machine learning methods such as Bayesian learning methods can account for missing data if applied correctly, however, most cannot.
- If the missing data is isolated to one particular attribute and that attribute is deemed non-essential, we can choose to discard the attribute.

¹ NaN stands for “Not a number” and is one way to denote the value is ill-formed.

Table 2.2. Processed version of the Cars dataset consisting of $M = 8$ attributes and $N = 142$ observations (only 10 are shown).

MPG	Cylinders	Horsepower	Weight	Year	Safety	Acceleration	Origin
18	8	150	3436	70	4	11	3
28	4	79	2625	82	4	18.6	1
26	4	79	2255	76	3	17.7	1
29	4	70	1937	76	1	14.2	2
15.32	8	175	3850	70	2	11	1
24	4	90	2430	70	3	14.5	2
17.5	6	95	3193	76	4	17.8	1
25	4	87	2672	70	3	17.5	3
:	:	:	:	:	:	:	:
15	8	198	4341	70	2	10	1

- If we have many observations and only few have missing observations, we can discard the observations with missing values.
- If we want to keep the affected attributes and observations, we can *impute* the missing values with some kind of neutral guess.

For the last method, imputation, a neutral guess can be obtained in several ways. In the case of the Miles/Gallon attribute, we can impute the missing value with the mean of all observed instances of the Miles/Gallon attribute (in this case 15.32 Miles/Gallon). For the safety rating, it may be undesirable to impute the mean value as the safety rating is an integer. In this case we can choose to either round the safety rating, or impute the most commonly encountered safety rating in the dataset, in this case 3.

As a rule, changing the dataset by for instance throwing away “outliers” without clear reasons for doing so is a cause for concern since this may affect conclusions drawn from the data. Outliers should therefore only be removed if there are strong reasons to think the observations are erroneous such as a negative safety rating.

2.3 The standard data format

We conclude this section by relating the table to the standard data format used in the course and that we briefly saw in the introduction. Suppose we implement the changes to the *Cars* dataset discussed above, i.e. we remove the ID, replaced the country of origin with an (ordinal) numeric coding and imputed the missing values. We then obtain the new Cars dataset shown in table 2.2. The way we will represent such a dataset is as an $N \times M$ matrix \mathbf{X} where $N = 142$ observations and $M = 8$ features. The corresponding matrix is simple:

$$\mathbf{X} = \begin{bmatrix} 18 & 8 & 150 & 3436 & 70 & 4 & 11 & 3 \\ 28 & 4 & 79 & 2625 & 82 & 4 & 18.6 & 1 \\ \vdots & \vdots \\ 15 & 8 & 198 & 4341 & 70 & 2 & 10 & 1 \end{bmatrix} \quad (2.1)$$

Each observation in the dataset, i.e. a single car, is a row in \mathbf{X} . We will write \mathbf{x}_i for the i 'th observation, for instance the second observation is the $M = 8$ dimensional vector (notice the transpose):

$$\mathbf{x}_2 = [28 \ 4 \ 79 \ 2625 \ 82 \ 4 \ 18.6 \ 1]^T$$

We will refer to element i, j of the matrix \mathbf{X} as X_{ij} or $X_{i,j}$. In this way $X_{2,4} = 2625$ meaning that the second car weights 2625 pounds. To complete the discussion, recall from the introduction we might have access to additional information y_i about each feature, corresponding to either a continuous regression (target) value or a discrete class label, and we collected all these values as a N -dimensional vector \mathbf{y} . All in all this means any dataset considered in this course will be an $N \times M$ matrix \mathbf{X} and (if available) a N -dimensional vector \mathbf{y} denoting the response or output variable (we will get back to the use of \mathbf{y} in part II of the course when we cover supervised learning). To apply any of the machine-learning methods discussed in this book to a given dataset therefore consists of putting it in this \mathbf{X}, \mathbf{y} format.

2.4 Feature transformations

Feature transformations refers to the operation of changing an existing feature or adding a new feature to our dataset. There are three principal reasons why we may want to do this:

- Many machine-learning methods such as principal component analysis and K -means are sensitive to outliers or features that reside on a different scale and it may therefore be a good idea to standardize (change) features.
- Expert knowledge can be used to compute new features from existing ones, thereby (hopefully) making a simpler machine-learning method more powerful.
- Ordinal values such as *Origin* might not be appropriately encoded as an integer (see below).

To provide examples of the first type, suppose we consider the *Cars* dataset of table 2.2 (or equivalently, the processed version as the \mathbf{X} matrix in eq. (2.1)), the variable *Safety* is less than 10 whereas *Weight* is between about 2000 to 5000 and both features have a mean value greater than zero. This difference in scale can confuse some machine-learning methods and it is therefore common to *standardize* the features by either subtracting the mean, or subtracting the mean and dividing by the standard deviation. This is done by first computing the empirical mean and standard deviation for each column j :

$$\hat{\mu}_j = \frac{1}{N} \sum_{i=1}^N X_{ij}, \quad \hat{\sigma}_j = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (X_{ij} - \hat{\mu}_j)^2}$$

and then updating each column j of \mathbf{X} of the new matrices $\tilde{\mathbf{X}}$ as either:

$$\tilde{\mathbf{X}} = \begin{bmatrix} \cdots & X_{1j} - \hat{\mu}_j & \cdots \\ \cdots & X_{2j} - \hat{\mu}_j & \cdots \\ \vdots & & \vdots \\ \cdots & X_{Nj} - \hat{\mu}_j & \cdots \end{bmatrix} \text{ or } \tilde{\mathbf{X}} = \begin{bmatrix} \cdots & (X_{1j} - \hat{\mu}_j)/\hat{\sigma}_j & \cdots \\ \cdots & (X_{2j} - \hat{\mu}_j)/\hat{\sigma}_j & \cdots \\ \vdots & & \vdots \\ \cdots & (X_{Nj} - \hat{\mu}_j)/\hat{\sigma}_j & \cdots \end{bmatrix}$$

It is easy to verify that in the first case the mean of column j in the updated \mathbf{X} -matrix will be 0, and in the second case the mean will be 0 and the standard deviation 1. While subtracting mean and

standardizing columns are the most common transformations, many other could be considered. For instance, suppose \mathbf{X} represent observations about animals where one column is the weight. Some animals are very small (for instance weighing a few grams) whereas others will weight hundreds of kilos, and the same difference in weight will often be much more informative when comparing two small animals compared to comparing two large ones. In other words, if an animal weights 1020g (compared to 20g) you can tell it is not a mouse, whereas if it weight 1001kg (compared to 1000kg) this extra kilo will not tell you that it is not a rhino, however, the difference in weight in both cases is just 1kg. A possible way to make the machine-learning method respect this difference in scale is by applying the logarithm to the column:

$$\tilde{\mathbf{X}} = \begin{bmatrix} \dots \log X_{1j} \dots \\ \dots \log X_{2j} \dots \\ \vdots \\ \dots \log X_{3j} \dots \end{bmatrix}$$

because in this case the difference in weight is: $\log(1.02) - \log(0.02) \approx 3.93$ compared to $\log(1001) - \log(1000) \approx 0.0001$ and thus much more informative for the smaller animal. A feature transformation consisting of adding and multiplying a feature column x_j with constants a, b , $x_i \mapsto ax_i + b$, is called *linear* (subtracting the mean and dividing with the standard deviation being the prototypical example) whereas all other transformations, such as computing the logarithm, are called *non-linear*.

In addition to changing a feature a new feature can be *added* to the dataset computed from existing ones. Suppose we consider a hospital records dataset \mathbf{X} consisting of M attributes and that two of these features, j and k , corresponds to weight and height of the patient. Since small patients are often light and tall patients are often heavy it may be beneficial to add a new feature to the dataset which takes this into account such as the Body Mass Index (BMI). Recall the BMI is the weight divided by the square of the height, in other words we add a new column:

$$\tilde{\mathbf{X}} = \begin{bmatrix} \dots X_{1j} X_{1k}^{-2} \dots \\ \dots X_{2j} X_{2k}^{-2} \dots \\ \vdots \\ \dots X_{Nj} X_{Nk}^{-2} \dots \end{bmatrix}$$

to \mathbf{X} thereby obtaining an $N \times (M + 1)$ dataset. Adding new features in this manner can make a simple learning method more powerful and will be used extensively when we discuss linear regression in chapter 8.

2.4.1 One-out-of-K coding

The final type of feature transformation we will consider is one where the type of a feature is changed. An example of this type of transformation which will play a central role several places in this book is *one-out-of- K* coding. Suppose we have a discrete ordinal variable x which takes K discrete values, for instance $K = 4$ and $x = 1, 2, 3$ or 4 . A one-out-of- K coding of x corresponds to a vector \mathbf{z} of dimension K where entry i is 1 only if $x = i$ and otherwise 0. For instance if $x = 3$ then $\mathbf{z} = [0 \ 0 \ 1 \ 0]^T$. If we consider the *Origin*-feature in the *Cars* dataset and recall this variable could take three possibly values (USA, Germany and France) and so a one-out-of- K coding is the mapping:

$$\begin{bmatrix} 3 \\ 1 \\ 1 \\ 2 \\ 1 \\ 2 \\ 1 \\ 3 \\ \vdots \\ 1 \end{bmatrix} \leftrightarrow \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ \vdots \\ 1 & 0 & 0 \end{bmatrix} \quad (2.2)$$

What is the advantage of one-out-of- K coding? Later, we will see this representation makes certain machine-learning methods easier to describe. However, for now notice coding the country as an integer implied an ordering in the countries. For many machine-learning methods this means that it matters if for instance we let Germany correspond to 2 or 3 even though this assignment is only our convention. If we apply a one-out-of- K coding, the assignment is symmetric since no machine-learning technique will depend on the ordering of the features. For this reason a one-out-of- K coding is recommendable when one has ordinal variables which in fact represent different categories and apply a machine-learning method where the (relative magnitude) of the variable will affect what the method does. In other words, we replace the ordinal variable with three binary variables.

2.4.2 Binarizing/thresholding

Another feature transformation where the type of the feature is changed is *binarizing* or *thresholding*. Suppose we select a constant θ , then binarizing a number x at θ is simply to replace x with 1 if $x \geq \theta$ and otherwise 0. More formally for a vector \mathbf{x} this is written as

$$\mathbf{x} \leftrightarrow \begin{bmatrix} 1_{[\theta, \infty]}(x_1) \\ 1_{[\theta, \infty]}(x_2) \\ \vdots \\ 1_{[\theta, \infty]}(x_N) \end{bmatrix} \quad (2.3)$$

where

$$1_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases} \quad (2.4)$$

is known as the *indicator function*.

With all these transformations available it may appear difficult to determine what we should actually *do* with our data. The rule of thumb is to initially do as little as we can get away with. Typically this means we should *somewhat* treat missing values or (gross) outliers if they exist in our data, and then if the method we wish to apply is known to be affected by features being badly scaled or not in a 1-out-of- K coding (this would be the case with for instance PCA in the next chapter) we can apply these transformations. Once this is over, we can begin to work with our data and later, once we have set up methods for evaluating the performance of our methods, possibly consider additional transformations.

Problems

2.1. Question 1: Consider the data set described in Table 2.3. Which statement about the attributes in the data set is *correct*?

No.	Attribute description	Abbrev.
x_1	Age of Mother in Whole Years	Age
x_2	Mothers Weight in Pounds	MW
x_3	Race (1 = Other, 0 = White)	Race
x_4	History of Hypertension (1 = Yes, 0 = No)	HT
x_5	Uterine Irritability (1 = Yes, 0 = No)	UI
x_6	Number of Physician Visits First Trimester	PV
y	Birth Weight in Kilo Grams	BW

Table 2.3. Attributes in a study on risk factors associated with giving birth to a low birth weight (less than 2.5 kg) baby [Hosmer and Lemeshow, Applied Logistic Regression, 1989]. The data we consider contains 189 observations, 6 input attributes x_1-x_6 , and one output variable y .

- A Race, HT and UI are ordinal.
- B Age and PV are ratio.
- C Age is continuous and ratio.
- D MW is discrete whereas PV is continuous.
- E Don't know.

2.2. Question 2: We consider a dataset about the Galápagos islands which is the famous group of islands studied by Charles Darwin situated at the equator in the Pacific Ocean. The data is taken from <http://www.statsci.org/data/general/galapagos.html> and comprises several measurements of characteristics of twenty nine of the islands in the Galápagos. We will presently consider a subset of this data given by the seven attributes outlined in Table 2.4.

Which one of the following statements is *correct*?

No.	Attribute description	Abbrev.
x_1	Number of plant species	Plants
x_2	Number of endemic plant species	E-Plants
x_3	Area of island (in km^2)	Area
x_4	Max. elevation above sea-level (in m)	Elev
x_5	Distance to nearest island (in km)	DistNI
x_6	Distance to Santa Cruz Island (in km)	StCruz
x_7	Area of adjacent island (in km^2)	AreaNI

Table 2.4. The seven attributes of the data on a selection of 29 of the Galápagos islands.

- A All the attributes are ratio and continuous.
- B All the attributes are interval and discrete.
- C Only two of the attributes are discrete but all attributes are ratio.
- D Some of the attributes can take negative values.
- E Don't know.

2.3. Question 3:

No.	Attribute description
x_1	Liters of gasoline consumed by an engine per hour
x_2	Charge, as measured by number of ionized atoms
x_3	Order in which runners finish a marathon
x_4	Brand of car (Toyota, VW, BMW, etc.)
x_5	Longitudinal position of a city on earth

Table 2.5. Five different attributes

Consider the six attributes with their description in table 2.5. Which of the following statements about the type of the attributes is true?

- A x_1 is continuous interval, x_2 is discrete ratio and x_3 and x_4 are ordinal.
- B x_2 is discrete interval, x_3 is discrete ordinal and x_5 is continuous ratio.
- C x_1 and x_5 are continuous ratio and x_4 is nominal.
- D x_1 is continuous ratio, x_2 discrete ratio and x_5 is continuous interval.
- E Don't know.

3

Principal Component Analysis

Principal component analysis (PCA) is a widely applicable technique where the goal is to find a lower-dimensional representation of a high-dimensional dataset. A lower-dimensional representation is useful in a variety of circumstances. For instance (lossy) compression, as a pre-processing step of very high-dimensional data or as a powerful visualization technique. PCA was invented in 1901 by the Statistician Karl Pearson [Pearson, 1901], but has been re-discovered numerous times in other fields under different names such as the Kosambi-Karhunen-Loëve transform in signal processing [Kosambi, 1943], the Hotelling transform in quality control [Hotelling, 1933].

PCA builds on simpler and much earlier discovered tools from linear algebra, most importantly the Singular Value Decomposition that was discovered independently by mathematicians Eugenio Beltrami and Camille Jordan discovered in 1873 and 1874 [Stewart, 1993].

Before introducing PCA we will first recap standard definitions from linear algebra. A reader familiar with subspaces and projections can skip this section.

3.1 Projections and subspaces★

Recall an M -dimension vector space is simply the set of M -dimensional vectors \mathbf{x} where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}, \quad (3.1)$$

and we write this as $\mathbf{x} \in \mathbb{R}^M$. Further, recall that if $\mathbf{x}, \mathbf{y} \in \mathbb{R}^M$ and a, b are real numbers then the vector

$$\mathbf{z} = a\mathbf{x} + b\mathbf{y}$$

also belongs to \mathbb{R}^M and we say that \mathbb{R}^M is *closed under linear transformations*. Finally, recall that the transpose of a vector \mathbf{x} is written as \mathbf{x}^T and corresponds to flipping the vector along its diagonal:

$$\mathbf{x}^T = [x_1 \ x_2 \ \cdots \ x_M].$$

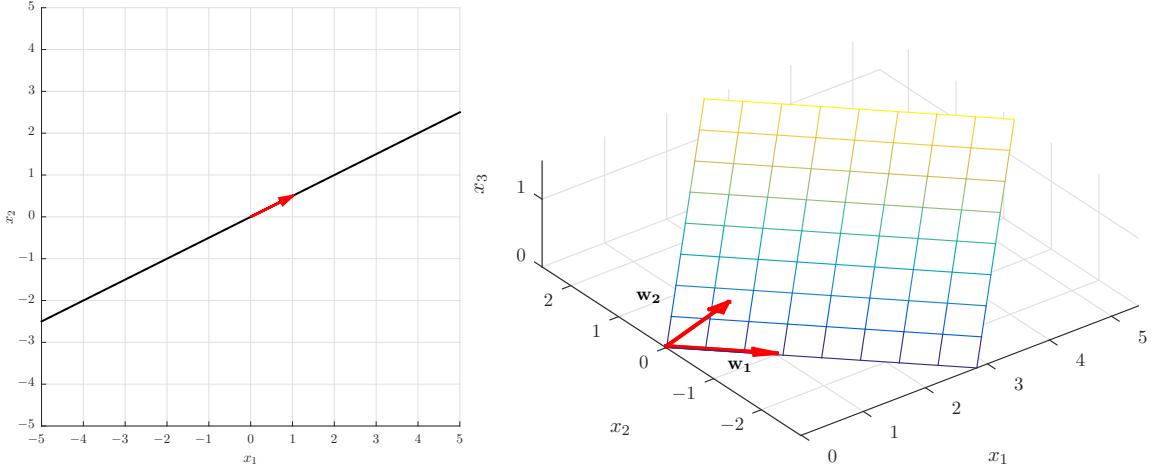


Fig. 3.1. Examples of a one- and two-dimensional subspace of \mathbb{R}^2 and \mathbb{R}^3 respectively. The subspaces can be thought of as generated by all linear combinations of the basis vectors shown as the red arrows.

3.1.1 Subspaces

A subspace of a vector space \mathbb{R}^M is a line, a plane, or their higher-dimensional generalizations. The key property of a subspace is that it is closed under linear transformations. Thus, a *subspace* V of \mathbb{R}^M is a set of M -dimensional vectors such that if $\mathbf{x}, \mathbf{y} \in V$ then

$$a\mathbf{x} + b\mathbf{y} \in V,$$

for any values of a and b . A simpler way to think of a subspace is that the subspace is *generated* by a set of vectors. Suppose $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^M$ is any set of n vectors in \mathbb{R}^M , we can then define the *span* of $\mathbf{x}_1, \dots, \mathbf{x}_n$ as all vectors \mathbf{z} that can be written as

$$\mathbf{z} = a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \dots + a_n\mathbf{x}_n \quad (3.2)$$

where a_1, \dots, a_n are arbitrary. We write this set of vectors as $V = \text{span}(\mathbf{x}_1, \dots, \mathbf{x}_n)$ and it is easy to show that V is a subspace of \mathbb{R}^M . To consider a simple example, suppose we consider the span of a single vector

$$V_1 = \text{span}\left(\begin{bmatrix} 1 \\ \frac{1}{2} \end{bmatrix}\right), \quad (3.3)$$

then V_1 corresponds to all vectors that can be written as $\begin{bmatrix} x \\ y \end{bmatrix} = a_1 \begin{bmatrix} 1 \\ \frac{1}{2} \end{bmatrix}$ (for arbitrary a_1) and they are shown as the black line in fig. 3.1 (left pane) where the red arrow is the vector $\begin{bmatrix} 1 \\ \frac{1}{2} \end{bmatrix}$. A slightly more elaborate example is shown in the right pane of fig. 3.1 where we consider the plane $V_2 = \text{span}(\mathbf{w}_1, \mathbf{w}_2)$ where

$$\mathbf{w}_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{w}_2 = \begin{bmatrix} 1 \\ 0 \\ 0.3 \end{bmatrix}. \quad (3.4)$$

Notice, nothing prevents the span of M vectors to be all of \mathbb{R}^M . To take a few more definitions, remember the length of a vector \mathbf{x} is

$$\|\mathbf{x}\| = \sqrt{\mathbf{x}^T \mathbf{x}} = \sqrt{x_1^2 + x_2^2 + \cdots + x_M^2} \quad (3.5)$$

and two vectors \mathbf{x}, \mathbf{y} are *orthogonal* if $\mathbf{x}^T \mathbf{y} = 0$. For instance

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0$$

and the reader can check these two vectors are indeed orthogonal by drawing them on a sheet of paper. A set of vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$ are said to be *linearly independent* if

$$\mathbf{0} = a_1 \mathbf{x}_1 + a_2 \mathbf{x}_2 + \cdots + a_n \mathbf{x}_n$$

implies $a_1 = a_2 = \cdots = a_n = 0$. Otherwise, they are said to be linearly dependent.

This brings us to the first central definition: A *basis* of a subspace V is a set of vectors $\mathbf{v}_1, \dots, \mathbf{v}_n$ such that

$$\text{span}(\mathbf{v}_1, \dots, \mathbf{v}_n) = V$$

and $\mathbf{v}_1, \dots, \mathbf{v}_n$ are linearly independent.

The definition of a basis simply means that $\mathbf{v}_1, \dots, \mathbf{v}_n$ are sufficient to generate V and at the same time V cannot be generated by fewer than n vectors. Notice it is always possible to find an alternative basis for a subspace (for instance, just multiply one of the basis vectors with -1). However, the *number* of vectors in the basis n will always be the same. We can therefore say that n is the *dimension* of the subspace. If we return to fig. 3.1, the red vectors form a basis for the two spaces shown in the two panes and they have dimension 1 and 2 respectively corresponding to a line and a plane.

It is often convenient that the vectors in the basis are of length 1 and pairwise orthogonal, i.e. $\mathbf{v}_i^T \mathbf{v}_j = 0$ for $i \neq j$. If this is satisfied for a basis $\mathbf{v}_1, \dots, \mathbf{v}_n$ the basis is said to be *orthonormal*. It is always possible to find an orthonormal basis for a subspace.

3.1.2 Projection onto a subspace

Suppose we consider any vector \mathbf{x} in a subspace V with orthonormal basis $\mathbf{v}_1, \dots, \mathbf{v}_n$. Then by the definition of a subspace \mathbf{x} can be written as

$$\mathbf{x} = a_1 \mathbf{v}_1 + a_2 \mathbf{v}_2 + \cdots + a_n \mathbf{v}_n,$$

for suitable choice of a_1, a_2, \dots, a_n . The reason why an orthonormal basis is so important is that it allows us to easily compute the numbers a_1, a_2, \dots, a_n . Say for instance that we wish to compute a_i , then simply multiply both sides of the above equation with \mathbf{v}_i^T to obtain:

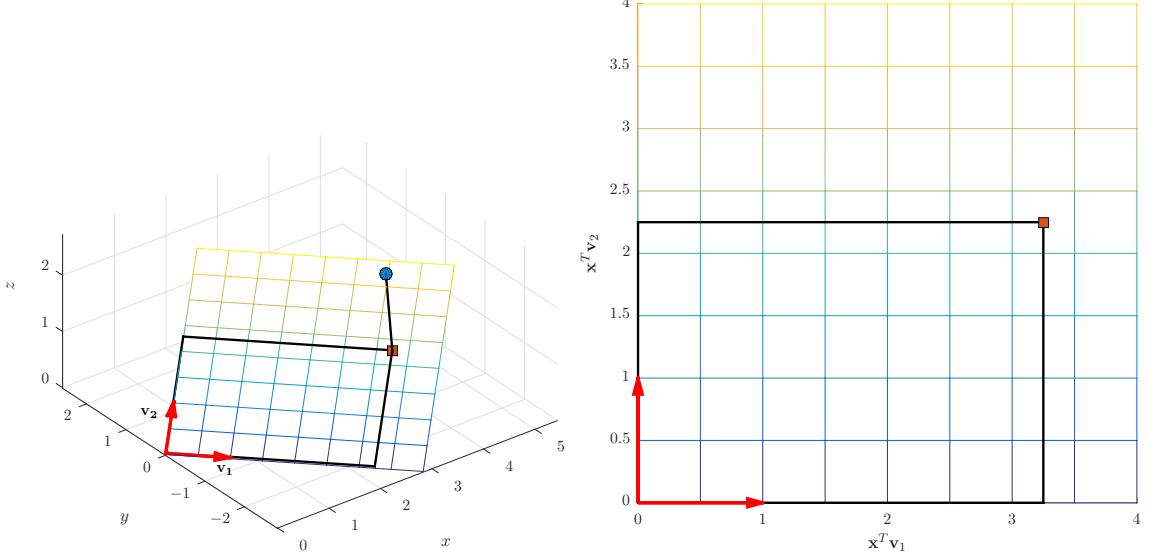


Fig. 3.2. Left pane: Projection of a 3D point \mathbf{x} (blue circle) onto the subspace spanned by the orthonormal basis $\mathbf{v}_1, \mathbf{v}_2$. The projection, shown as the red square, lies within the subspace and the right pane shows the point in the 2D coordinate system given by the basis vectors of the subspace.

$$\begin{aligned}
 \mathbf{v}_i^T \mathbf{x} &= a_1 \mathbf{v}_i^T \mathbf{v}_1 + \cdots + a_i \mathbf{v}_i^T \mathbf{v}_i + a_n \mathbf{v}_i^T \mathbf{v}_n \\
 &= a_1 \cdot 0 + \cdots + a_i \cdot 1 + \cdots + a_n \cdot 0 \\
 &= a_i
 \end{aligned} \tag{3.6}$$

and so we can find a_i by simply computing $a_i = \mathbf{x}^T \mathbf{v}_i$. However, what if \mathbf{x} does not lie in V ? We can still compute the n numbers

$$b_1 = \mathbf{x}^T \mathbf{v}_1, \quad b_2 = \mathbf{x}^T \mathbf{v}_2, \quad \dots \quad b_n = \mathbf{x}^T \mathbf{v}_n$$

and then form a new vector \mathbf{x}' which *does* lie in V :

$$\mathbf{x}' = b_1 \mathbf{v}_1 + \cdots + b_n \mathbf{v}_n. \tag{3.7}$$

One can show \mathbf{x}' is the point in V closest to \mathbf{x} and we say that \mathbf{x}' , defined above, is the *projection* of \mathbf{x} onto V . We also say the n -dimensional vector

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

is the coordinates of \mathbf{x} in the subspace V . Notice, this is an n -dimensional vector whereas the space that \mathbf{x} was in is M -dimensional.

In the left pane of fig. 3.2 is shown the projection of the blue point in \mathbb{R}^3 onto the space spanned by the two basis vectors $\mathbf{v}_1, \mathbf{v}_2$ shown as arrows. In the right pane we have plotted the projected point, \mathbf{x}' , with the coordinates in the new space V .

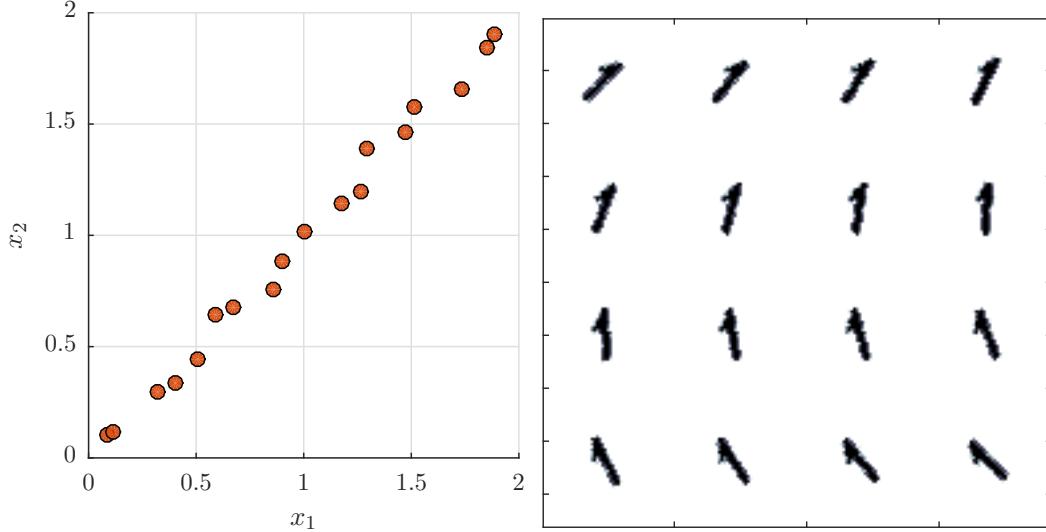


Fig. 3.3. (Left:) A simple 2D dataset \mathbf{X} comprising two features x_1 and x_2 that are noisy observations of the same quantity. Therefore, the dataset in fact only contains one “data-dimension” even though it is two-dimensional. (Right:) A more elaborate dataset example corresponding to $N = 9$ observations each corresponding to 48×48 pixel images of the same digit but rotated. From the perspective of the matrix \mathbf{X} the dataset contains $M = 2304$ attributes, (corresponding to the number of pixels), however from another perspective only *one* dimension matters, namely the rotation. PCA is able to discover the lower-dimensional representation of the dataset \mathbf{X} in the first example but not in the second.

Finally, suppose we collect the basis vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ in a matrix \mathbf{V} :

$$\mathbf{V} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \dots \ \mathbf{v}_n] \quad (3.8)$$

We can then express the coordinates of \mathbf{x} in the space V as

$$\mathbf{b}^T = \mathbf{x}^T \mathbf{V} \quad (3.9)$$

Thus, suppose the original blue points was $\mathbf{x} \in \mathbb{R}^3$, we can then express it's new two-dimensional coordinates (b_1, b_2) plotted in the right-pane of fig. 3.2 as the red square:

$$\mathbf{b}^T = [b_1 \ b_2] = \mathbf{x}^T [\mathbf{v}_1 \ \mathbf{v}_2]$$

3.2 Principal Component Analysis

Suppose we consider a two-dimensional dataset, however, unbeknownst to us the two dimensions are just noisy measurements of the same quantity. If we plot each observation in the dataset as a point we obtain a figure similar to fig. 3.3 (left pane) where the points nearly lie on a straight line. Even though the dataset is two-dimensional, there is really only a single dimension that matters,

namely the line along which the observations lie. For a more ambitious example, consider a dataset comprised of $N = 9$ observations of the *same* image of the digit 1 but rotated around the center shown in fig. 3.3 (right pane). Each image is 48×48 pixels large so if we consider each pixel as a feature we can represent an image as one long vector in a $M = 2304$ dimensional space. However, from a more human perspective the true number of dimensions is really only one, namely the angle of rotation.

That the number of “true” dimensions in the dataset is often much lower than the number of observed dimensions is a common feature of many machine learning problems and the goal of principal component analysis is to discover a lower-dimensional representation of the high-dimensional data set where it is assumed the lower-dimensional representations are linear. This is the case for 2D example in fig. 3.3 (left pane) which PCA can solve, however, not the case for the rotated digits example in the right pane which would require more advanced methods than will be considered here.

To make the discussion concrete, assume we are in the standard setting encountered previously where we are given N observations $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N \in \mathbb{R}^M$ each consisting of M features or dimensions. In principal component analysis we select a number n and then we wish to find a new n -dimensional representation

$$\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_N \in \mathbb{R}^n$$

where $n \leq M$ and such that \mathbf{b}_i “represents” the observation \mathbf{x}_i . To return to the previous example in fig. 3.3 (left pane) $M = 2$ (the number of observed features) and the representation we are interested in would have $n = 1$.

The simplest way to transform vectors from a M dimensional space to a $n \leq M$ -dimensional space is to select an orthonormal basis $\mathbf{v}_1, \dots, \mathbf{v}_n$ of a n -dimensional subspace V and define each \mathbf{b}_i as the projection of \mathbf{x}_i onto V . Since we want the projection to be invariant under addition of a constant we first subtract the mean from each \mathbf{x}_i . The general layout of the PCA algorithm is then:

- Compute the mean $\mathbf{m} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i$
- Subtract the mean from \mathbf{x}_i : $\tilde{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{m}$ (and collect all $\tilde{\mathbf{x}}_i$ into an $N \times M$ matrix $\tilde{\mathbf{X}}$)
- Project onto V : $\mathbf{b}_i^T = \tilde{\mathbf{x}}_i^T \mathbf{V}$

where

$$\mathbf{V} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \cdots \ \mathbf{v}_n]$$

is the projection matrix corresponding to the basis $\mathbf{v}_1, \dots, \mathbf{v}_n$. Notice the centering step where the mean is subtracted is the same scheme that we encountered in the previous chapter.

So how do we select the projection matrix \mathbf{V} ? We will first consider the simplest case in which $n = 1$, i.e. we are projecting onto a line, and later consider the general case $n > 2$. It is useful to consider a concrete example to get some intuition about what different choices of \mathbf{v}_1 implies. In fig. 3.4 (top panes) we have shown the projection of the same 2D dataset onto three different choices of \mathbf{v}_1 and in the bottom panes we have plotted the projected coordinates $b_i = \tilde{\mathbf{x}}_i^T \mathbf{v}_1$, i.e. the 1-dimensional “representation” of each $\tilde{\mathbf{x}}_i$. From an intuitive point of view the first projection is worse than the last since it lumps the observations together with large residuals (indicated by the blue and red lines). The same pattern is repeated in fig. 3.5, here we consider a 3d dataset and still project it onto different 1-dimensional subspaces. The first projection has large residuals and also lumps all classes together while in the third the residuals are much smaller and the data in the projection thereby more spread out thus preserving more information about the data. What these three projections have in common is that the observations, in the projected coordinates b_1, \dots, b_N ,

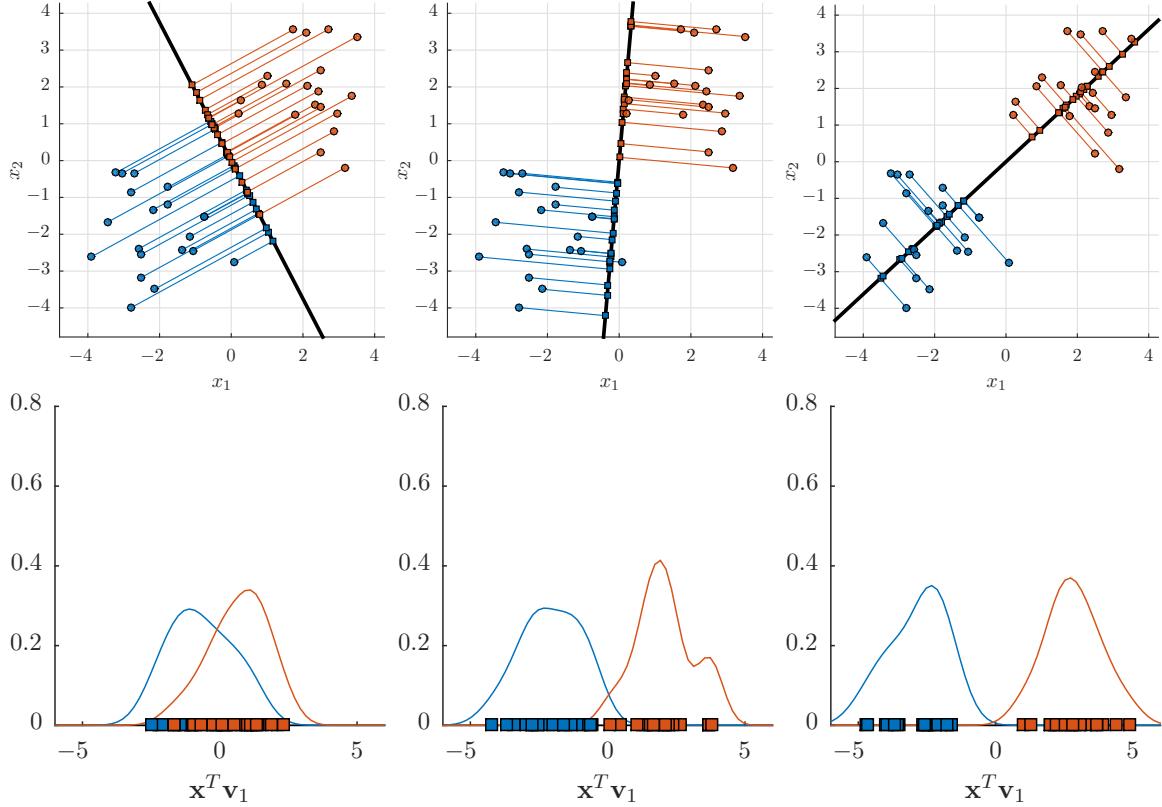


Fig. 3.4. An example 2D dataset colored for our convenience (the dots in the top panes are the same in all 3 panes) is projected onto three different 1D subspaces corresponding to different choices of basis vector \mathbf{v}_1 . The dataset in the projected coordinate system is shown in the bottom pane. As can be seen, different choices of basis vectors preserve different amounts of information, with the vector corresponding to the right-most pane preserving the most information since the dataset is the most spread out.

are more spread out (thereby providing smaller residuals indicated by the lines connecting the points to the 1-dimensional subspace). The degree to which the projected observations are spread out can be measured by the *variance* of b_1, \dots, b_N scaled by a factor of N ¹:

$$W = N \times \text{Variance}[b_1, \dots, b_N] = N \frac{1}{N} \sum_{i=1}^N (b_i - \bar{b})^2 = \sum_{i=1}^N (b_i - \bar{b})^2, \text{ where: } \bar{b} = \frac{1}{N} \sum_{i=1}^N b_i. \quad (3.10)$$

The reason for multiplying with N will be apparent later. This immediately gives us an idea for selecting \mathbf{v}_1 : We simply make it the goal of PCA to select \mathbf{v}_1 to be the vector which maximizes the spread-outness of the projected data and would therefore select the right-most figure in fig. 3.4 and fig. 3.5 over the other. To put it formally,

¹ We here consider the biased estimate of variance where we divide by N , see chapter 4.

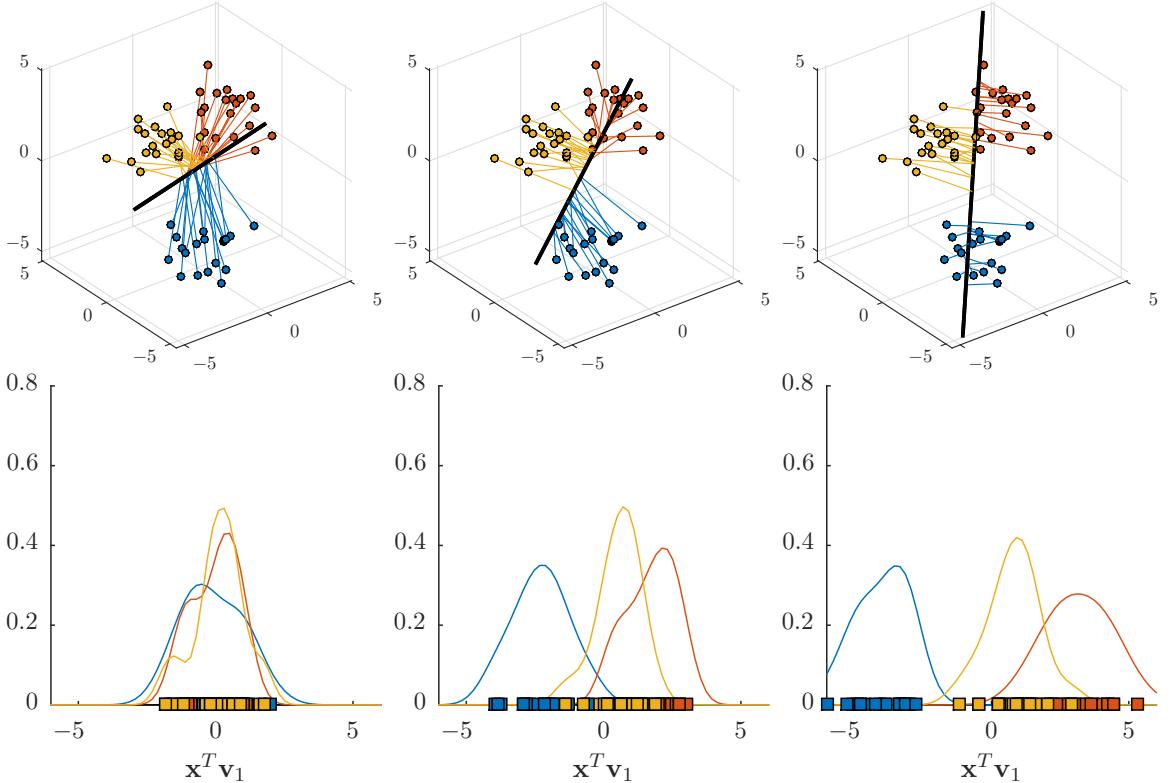


Fig. 3.5. Similar to the example fig. 3.4, a 3D dataset (colored for our convenience) is projected onto three different 1D subspaces corresponding to different choices of basis vector v_1 . The dataset in the projected coordinate system is shown in the bottom pane. As can be seen, different choices of basis vectors preserve different amounts of information, with the vector corresponding to the right-most pane preserving the most information since the dataset is the most spread out.

$$\begin{aligned} v_1 &= \text{The vector of length 1 that maximize } W \\ &= \arg \max_{v^T v=1} W. \end{aligned}$$

In the next section we will solve this maximization problem and then consider the general case where $n > 1$.

Maximizing the variance with respect to the first principal component

First notice that the mean of the projected data \bar{b} is zero since we have subtracted the mean from \mathbf{X} :

$$\bar{b} = \frac{1}{N} \sum_{i=1}^N b_i = \frac{1}{N} \sum_{i=1}^N \tilde{x}_i^T v_1 = \left(\frac{1}{N} \sum_{i=1}^N \tilde{x}_i \right)^T v_1 = \left(\left[\frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \right] - \mathbf{m} \right)^T v_1 = 0 \quad (3.11)$$

If we define the matrix $\mathbf{S} = \tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$ we can re-write the variance W to be:

$$W = \sum_{i=1}^N b_i^T b_i = \sum_{i=1}^N (\tilde{\mathbf{x}}_i^T \mathbf{v}_1)^T \tilde{\mathbf{x}}_i^T \mathbf{v}_1 = \sum_{i=1}^N \mathbf{v}_1^T \tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_i^T \mathbf{v}_1 = \mathbf{v}_1^T \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \mathbf{v}_1 = \mathbf{v}_1^T \mathbf{S} \mathbf{v}_1 \quad (3.12)$$

For \mathbf{v}_1 to be an orthonormal basis it has to have norm 1, $\|\mathbf{v}_1\|^2 = \mathbf{v}_1^T \mathbf{v}_1 = 1$. The maximization of eq. (3.12) under this constraint can be done by introducing the Lagrangian multiplier λ and maximizing the Lagrangian ²

$$\mathcal{L} = W + \lambda(1 - \|\mathbf{v}_1\|^2) = \mathbf{v}_1^T (\mathbf{S} - \lambda \mathbf{I}) \mathbf{v}_1 + \lambda \quad (3.13)$$

with respect to λ and \mathbf{v}_1 . Taking the derivatives with respect to the vector and λ we obtain:

$$\frac{\partial}{\partial \lambda} \mathcal{L} = 1 - \mathbf{v}_1^T \mathbf{v}_1 = 0 \quad (3.14)$$

$$\nabla_{\mathbf{v}_1} \mathcal{L} = (\mathbf{S} - \lambda \mathbf{I}) \mathbf{v}_1 = 0. \quad (3.15)$$

From the first equation we simply observe that \mathbf{v}_1 should be normalized. The second equation can be re-written as:

$$\mathbf{S} \mathbf{v}_1 = \lambda \mathbf{v}_1 \quad (3.16)$$

This means that \mathbf{v}_1 should be an eigenvector of \mathbf{S} with eigenvalue λ . So which eigenvector should we choose? If we look at the cost function eq. (3.12) which we wish to maximize it can be re-written:

$$W = \mathbf{v}_1^T \mathbf{S} \mathbf{v}_1 = \mathbf{v}_1^T \lambda \mathbf{v}_1 = \lambda.$$

Thus, we should select \mathbf{v}_1 as the eigenvector of \mathbf{S} corresponding to the *largest* eigenvalue.

This solves the case $n = 1$. The case $n \geq 2$ is similar but requires slightly more work. First we collect the b_i 's into an $N \times n$ matrix

$$\mathbf{B}^T = [\mathbf{b}_1 \ \mathbf{b}_2 \ \cdots \ \mathbf{b}_N]^T.$$

The projection can then be written as

$$\mathbf{B} = \tilde{\mathbf{X}} \mathbf{V}$$

Previously we defined the spread-outness of \mathbf{B} by taking the sum of squares of the entries in \mathbf{B} . We will simply re-use this idea for the case $n > 1$ and thereby define the *Frobenius norm*:

$$W = \|\mathbf{B}\|_F^2 = \sum_{i=1}^N \sum_{j=1}^n B_{ij}^2$$

We can therefore simply maximize W where we ensure $\mathbf{v}_i^T \mathbf{v}_j = 0$ for $i \neq j$ since we are looking for an orthonormal basis. To put it formally,

$$\mathbf{v}_1, \dots, \mathbf{v}_n = \text{The } n \text{ orthonormal vectors that maximize } W.$$

² If you are unfamiliar with Langrangian multipliers see Appendix A and https://en.wikipedia.org/wiki/Lagrange_multiplier.

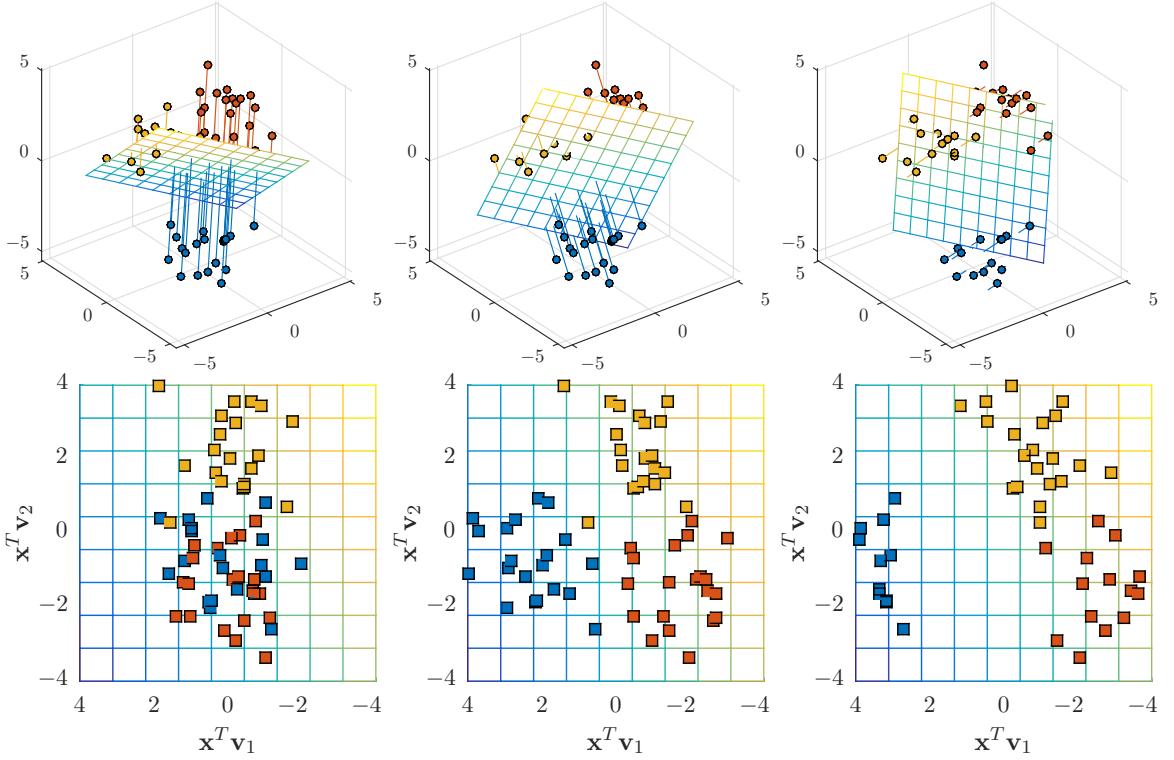


Fig. 3.6. This time, the same 3D dataset as in fig. 3.5 is projected onto three different 2D planes corresponding to the three panes in the top row. The inserts in the bottom row is the points in the coordinate system corresponding to the subspace. We again see the right most pane, where the points are the most spread out, seems to better preserve the information in the dataset.

It turns out the solution to this maximization problem is to select $\mathbf{v}_1, \dots, \mathbf{v}_n$ as the n orthonormal eigenvectors of S with the n largest eigenvalues. The case $n = 2$ is illustrated in fig. 3.6 where the right-most plane corresponds to the two largest eigenvectors and the other two panes corresponds to different choices of basis. We again see the choice of eigenvectors ensures the observations are the most spread out. Now that we have defined $\mathbf{v}_1, \dots, \mathbf{v}_n$ this formally completes the PCA algorithm. However, there is a simple (and natural) way to represent the eigenvectors using the *Singular value decomposition* (SVD) which makes PCA much easier to compute. We will therefore introduce the SVD and explain its relationship to PCA.

3.3 Singular Value Decomposition and PCA

The *singular value decomposition* (SVD) provides an easy way to compute the n eigenvectors corresponding to the n largest eigenvalues. For *any* $N \times M$ matrix \mathbf{X} , the SVD computes three matrices

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_M \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}, \quad \mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_N] \quad \mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_M]$$

such that

$$\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T = \mathbf{X}$$

and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_M$ are known as the *singular values* of \mathbf{X} . Notice these conditions implies that $\mathbf{v}_i^T \mathbf{v}_j = 0$ if $i \neq j$ and otherwise 1; in other words the columns of \mathbf{V} are orthonormal. This has some interesting consequences. For instance if we compute:

$$\mathbf{V}^T \mathbf{v}_i = \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_i^T \\ \vdots \\ \mathbf{v}_M^T \end{bmatrix} \mathbf{v}_i = \begin{bmatrix} \mathbf{v}_1^T \mathbf{v}_i \\ \vdots \\ \mathbf{v}_i^T \mathbf{v}_i \\ \vdots \\ \mathbf{v}_M^T \mathbf{v}_i \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \mathbf{e}_i$$

This can be used to show:

$$(\mathbf{X}^T \mathbf{X}) \mathbf{v}_i = (\mathbf{V} \boldsymbol{\Sigma}^T \mathbf{U}^T \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T) \mathbf{v}_i = \mathbf{V} \boldsymbol{\Sigma}^T \boldsymbol{\Sigma} \mathbf{e}_i = \mathbf{V} \sigma_i^2 \mathbf{e}_i = \sigma_i^2 \mathbf{v}_i$$

So each \mathbf{v}_i is an eigenvector of $\mathbf{X}^T \mathbf{X}$ with associated eigenvalue σ_i^2 and the eigenvectors are sorted according to their eigenvalues. This should sound very familiar. Indeed, by our previous definition the first n columns of \mathbf{V}

$$\mathbf{V}_n = [\mathbf{v}_1 \ \mathbf{v}_2 \ \cdots \ \mathbf{v}_n],$$

are by definition the first n eigenvectors and, by simply using the definition of projection onto a subspace eq. (3.9), the projection of a single observation \mathbf{x}_i^T onto the subspace spanned by the first n principal components can therefore be written as $\mathbf{b}_i^T = \mathbf{x}_i^T \mathbf{V}_n$.

3.3.1 The PCA algorithm

Gathering the previous discussion, we can define the PCA algorithm on a matrix \mathbf{X} where the dataset is projected onto the first n components as follows:

- Subtract the mean: $\tilde{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{m}$, $\mathbf{m} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i$
- Divide by standard deviation (Optional): $\tilde{x}_{ij} = \frac{\tilde{x}_{ij}}{s_k}$, where $s_k = \sqrt{\frac{1}{N-1} \sum_{i=1}^N \tilde{x}_{ik}^2}$
- Compute the SVD: $\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T = \tilde{\mathbf{X}}$
- The n first principal components are $\mathbf{v}_1, \dots, \mathbf{v}_n$ and coordinates of observation i when projected onto the subspace spanned by the first n principal components are

$$\mathbf{b}_i^T = \tilde{\mathbf{x}}_i^T \mathbf{V}_n \text{ or alternatively } \mathbf{B} = \tilde{\mathbf{X}} \mathbf{V}_n$$

where $\mathbf{V}_n = [\mathbf{v}_1, \dots, \mathbf{v}_n]$.

In the above steps are included the optional step indicated in gray to not only center the data but also normalize each attribute by further dividing each attribute by its standard deviation. This optional step may be relevant when the attributes have different scales, i.e. if one attribute have very large variance compared to the other attributes the first principal component will be highly driven by this attribute in order to account for as much of the variance as possible in the data.

3.3.2 Variance explained by the PCA

The vectors \mathbf{b}_i in the matrix \mathbf{B} represents the coordinates of the vector \mathbf{x}_i when it is projected onto the n -dimensional subspace, i.e. the bottom row in fig. 3.5. What if we want to know what vector in the original space $\mathbf{b}_i = [b_{i1} \ b_{i2} \ \cdots \ b_{in}]^T$ corresponds to? I.e. the intersection with the line in the top-row of fig. 3.5? Similar to eq. (3.7) we can find the projected coordinates in the original space as:

$$\mathbf{x}'_i = b_{i1}\mathbf{v}_1 + \cdots + b_{in}\mathbf{v}_n = \mathbf{V}_n\mathbf{b}_i,$$

or if we choose this can be written more condensed for all observations as

$$\mathbf{X}' = (\mathbf{V}_n \mathbf{B}^T)^T = \tilde{\mathbf{X}} \mathbf{V}_n \mathbf{V}_n^T. \quad (3.17)$$

The case $n = M$ is worth mentioning. In this case by definition $\mathbf{V}_n = \mathbf{V}$ and so, by orthonormality of \mathbf{V} , we obtain:

$$\mathbf{X}' = \tilde{\mathbf{X}} \mathbf{V} \mathbf{V}^T = \tilde{\mathbf{X}} \mathbf{I} = \tilde{\mathbf{X}}$$

that is, if all M principal directions are used the projection does not “lose” any information. In this case the projected matrix \mathbf{B} is still different from $\tilde{\mathbf{X}}$ — it is exactly corresponding to “rotating” all observations (in M dimensions!) and then, when translating back to \mathbf{X}' , we just “rotate back” without losing information.

In general, the reconstructed matrix \mathbf{X}' will have lost information if $n < M$ (or alternatively, variability in the data) compared to $\tilde{\mathbf{X}}$ — in linear algebra terms we lose all variability of $\tilde{\mathbf{X}}$ which is orthogonal to the space \mathbf{V}_n because we project those directions away. A natural way to measure how much variance — or information about $\tilde{\mathbf{X}}$ — is retained in a reconstruction based on n principal components is the *variance explained* computed using the Fröbenius norm:

$$\text{Variance Explained} = \frac{\|\mathbf{X}'\|_F^2}{\|\tilde{\mathbf{X}}\|_F^2}$$

Using the fact that $\|\mathbf{X}\|_F^2 = \text{trace}(\mathbf{X}^T \mathbf{X})$ and plugging in the definition of the SVD (exploiting $(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$ and $\text{trace}(\mathbf{AB}) = \text{trace}(\mathbf{BA})$) one can show that $\|\mathbf{X}'\|_F^2 = \sum_{i=1}^n \sigma_i^2$ and $\|\tilde{\mathbf{X}}\|_F^2 = \sum_{i=1}^M \sigma_i^2$. Plugging this in we get the formula for the variance explained by the n first principal components:

$$\text{Variance Explained} = \frac{\sum_{i=1}^n \sigma_i^2}{\sum_{i=1}^M \sigma_i^2}. \quad (3.18)$$

As we expect the case where $n = M$ guarantees that all variance will be conserved, however, if some σ_i are zero we can still perfectly represent all variance with fewer than M directions. Why? This case corresponds to the dataset residing in a subspace of V having dimension less than M . In two dimensions, this could be if the observations fall exactly on a line.

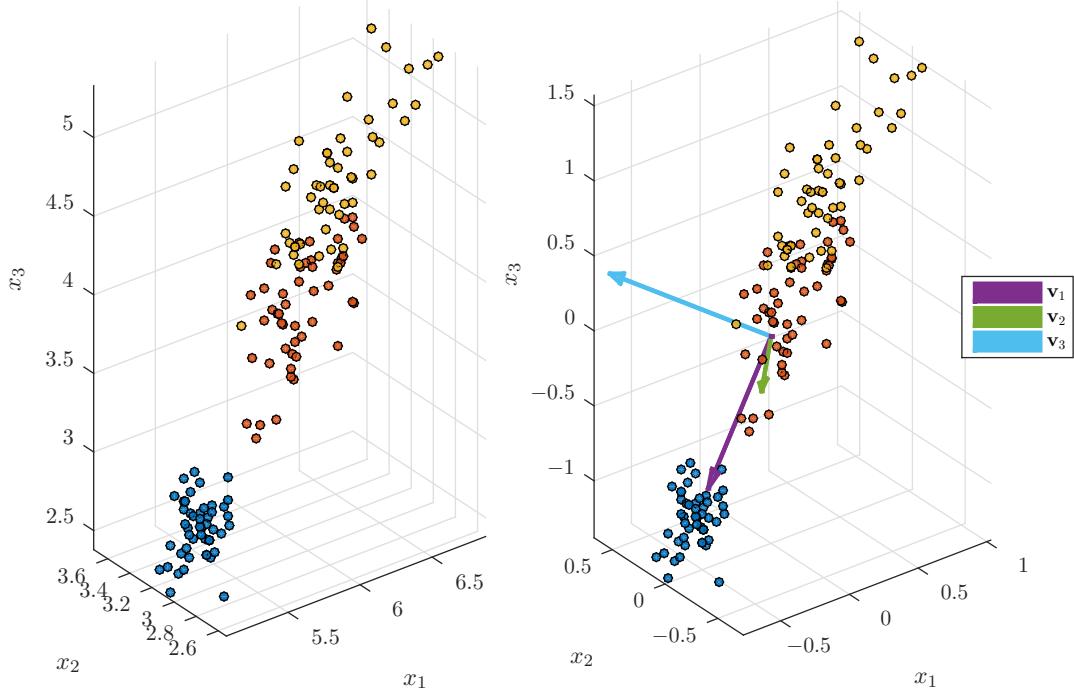


Fig. 3.7. The Fisher Iris dataset consisting of $N = 150$ observations of three types of flowers. In the left pane the dataset is plotted as a scatter plot whereas in the right pane the mean of the dataset has been subtracted to obtain the centered matrix $\tilde{\mathbf{X}}$ and the three principal directions are plotted as unit vectors.

3.4 Applications of principal component analysis

For our first application of PCA we will consider the Fisher Iris dataset consisting of $N = 150$ observations of three different types of flowers. The original data contains four features but we will presently only consider three of the features in order to visualize the data prior to the PCA, thus each observation consists of $M = 3$ features. The dataset, labelled according to flower type, is shown in fig. 3.7 (left pane). The first step of the PCA analysis is to subtract the mean of the dataset to obtain the centered matrix $\tilde{\mathbf{X}}$ ($\tilde{X}_{ij} = X_{ij} - \frac{1}{N} \sum_{i=1}^N X_{ij}$). Then, a SVD is performed to obtain the decomposition $\tilde{\mathbf{X}} = \mathbf{U}\Sigma\mathbf{V}^T$. The mean vector \mathbf{m} and the principal component directions, i.e. columns of $\mathbf{V} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \mathbf{v}_3]$, are approximately:

$$\mathbf{m} = \begin{bmatrix} 5.8 \\ 3.1 \\ 3.8 \end{bmatrix}, \quad \mathbf{v}_1 = \begin{bmatrix} 0.4 \\ -0.1 \\ 0.9 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} -0.6 \\ -0.7 \\ 0.2 \end{bmatrix}, \quad \mathbf{v}_3 = \begin{bmatrix} -0.7 \\ 0.7 \\ 0.3 \end{bmatrix},$$

and are shown in fig. 3.7 as colored vectors. Notice the first principal direction follows the main direction of variability in the data.

Let's consider the projections onto the principal directions. These can be found in fig. 3.8 where for instance the middle figure shows the projection onto the first and second principal direction found by computing the vectors $\tilde{\mathbf{X}}\mathbf{v}_1$ and $\tilde{\mathbf{X}}\mathbf{v}_2$ and plotting these as a scatter plot. As can be

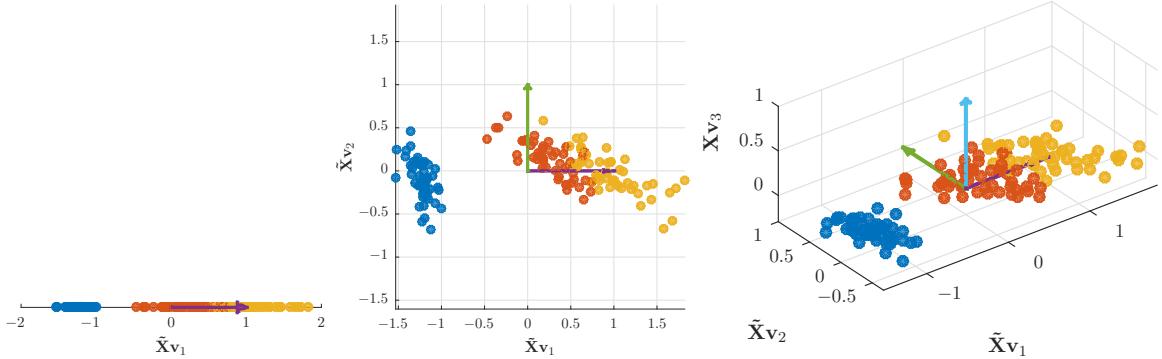


Fig. 3.8. The Fisher Iris dataset of fig. 3.7 projected onto the principal directions. In the first pane only the first principal direction is used corresponding to a large loss of information. In the second pane two principal directions are used better preserving the structure of the data and finally the third pane use all principal directions and therefore only corresponds to a rotation.

seen the right-most figure, corresponding to using all principal directions, simply corresponds to rotating the dataset in fig. 3.7 until the PCA directions are oriented along the axis. The other plots are obtained by projecting away principal direction v_3 and then v_3 and v_2 and therefore lose information. Finally let's turn to the variance explained by the principal directions. This can be

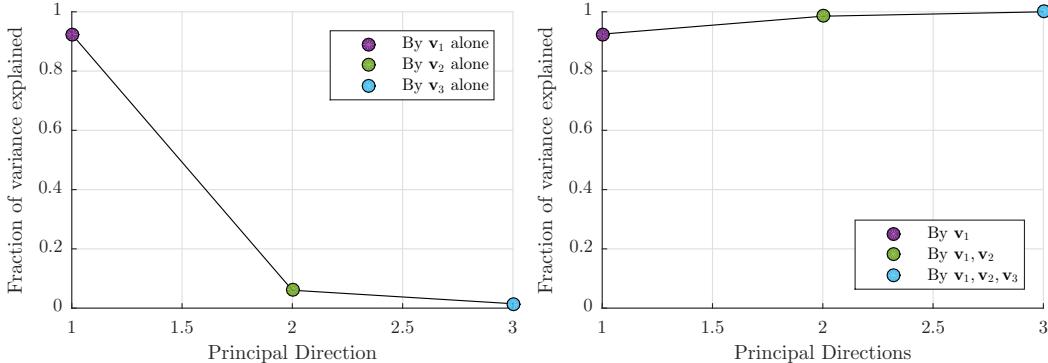


Fig. 3.9. (Left:) Variance explained by each of the three principal directions from the Fisher Iris example of fig. 3.7. Nearly all variation is explained by the first and second principal directions which can be interpreted as the original dataset residing on a 2D plane spanned by the first and second principal directions. (Right:) Cumulative variance explained by the subspaces spanned by only v_1 , then v_1 and v_2 and finally v_1 , v_2 and v_3 . Compare to fig. 3.8.

computed from Σ and in our example

$$\Sigma = \begin{bmatrix} 11.7 & 0 & 0 \\ 0 & 3.0 & 0 \\ 0 & 0 & 1.5 \end{bmatrix},$$

and the variance explained by each component can be seen in fig. 3.9. For instance the first component alone explains $\frac{11.7^2}{11.7^2+3^2+1.5^2} \approx 92\%$ of the variance.

3.4.1 Example 1: Interpreting PCA components

The PCA components, along with label information, can often be given a physical interpretation. To see this, we will consider a subset of the Cities dataset³. We will consider the $M = 5$ attributes *health*, *crime*, *transportation*, *education*, and *arts* and the dataset consist of ratings of $N = 329$ US cities in these categories. Higher rating means better.

We apply PCA as described in section 3.3.1 where we first standardize by subtracting the mean, and include the step where we divide by the standard deviation since the ratings have substantially different scale. After carrying out an SVD, we obtain the matrices \mathbf{U} , Σ , and \mathbf{V} , where as usual the two first columns of \mathbf{V} are the first two principal components:

$$\mathbf{v}_1 = \begin{bmatrix} 0.55 \\ 0.31 \\ 0.42 \\ 0.37 \\ 0.54 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} -0.11 \\ 0.78 \\ 0.02 \\ -0.61 \\ 0.06 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 29.8 & 0 & 0 & 0 & 0 \\ 0 & 17.6 & 0 & 0 & 0 \\ 0 & 0 & 14.5 & 0 & 0 \\ 0 & 0 & 0 & 13.8 & 0 \\ 0 & 0 & 0 & 0 & 6.3 \end{bmatrix}.$$

Using eq. (3.18) we see these two components accounts for 54 and 19 percent of the variance respectively.

To interpret any of these components, for instance the first principal component \mathbf{v}_1 , it is useful to imagine what properties an observation which have a very large (or very small) projection onto \mathbf{v}_1 , i.e. an observation $\mathbf{x} = \mathbf{v}_1$ or $\mathbf{x} = -\mathbf{v}_1$, would have.

To obtain a large projection onto \mathbf{v}_1 , we see all the coordinates should have the same sign and be positive and in a similar vein, to get a very small projection all coordinates should be negative. In other words, the feature of a city \mathbf{v}_1 expresses is that cities that have a high value of one coordinate generally also have a high value of the other coordinates and visa versa. To put this in simpler terms, PC one seems to measure if a city is good (versus bad), and cities that are good typically have many hospitals, lower crime, a lot of transportation options, art, and so on.

If we then focus on \mathbf{v}_2 , we see only two attributes, crime and education, are relevant as their magnitude is quite a lot bigger than the rest. In this case they have opposite signs, which means an observation with a large projection onto \mathbf{v}_2 would have high value of crime and low value of education (i.e., an orderly city with poor education options) and the other way around for an observation with a negative projection (a relatively criminal city with good education options). This might seem somewhat puzzling, so we project the dataset onto the first two principal components and label the cities with the highest/lowest value of the second principal component, see fig. 3.10. We see the second principal component might express a big-city effect, such that the two large cities (Las Vegas and New York) are singled out as very orderly cities with a neglected school system, whereas those with the lowest projection (Cumberland and Scranton) are much smaller cities.

Obviously these specific interpretation are quite subjective and should only be treated as a hypothesis at this point. Furthermore, as we consider $\mathbf{v}_3, \mathbf{v}_4$ and so on they will generally be more difficult to interpret and may simply capture noise.

³ See <https://mathworks.com/help/stats/sample-data-sets.html>

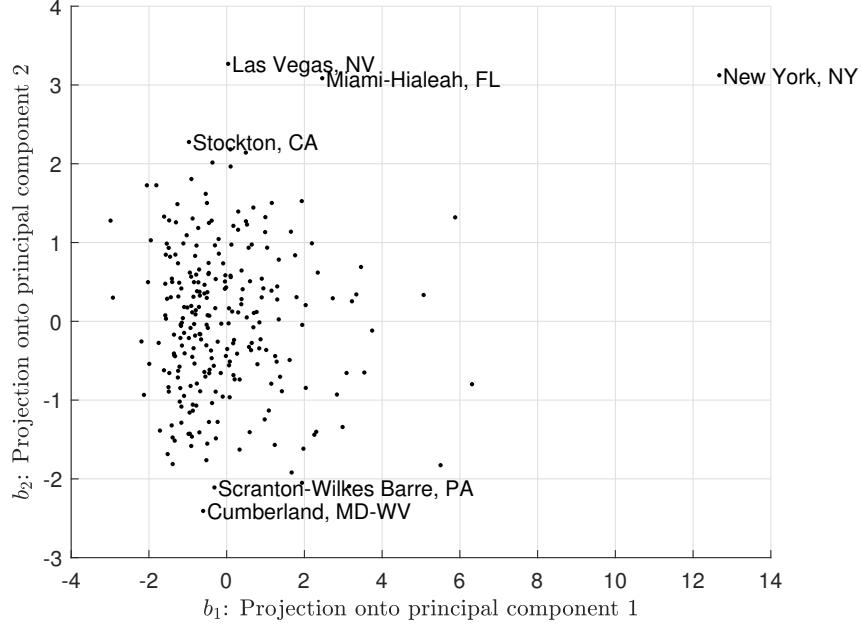


Fig. 3.10. Projection of the cities dataset onto the first two principal components. Names of cities with the largest/smallest projection onto the second principal component have been inserted.

3.4.2 Example 2: A high-dimensional example

In this example, we will consider PCA applied to a very high-dimensional problem. We will consider the MNIST dataset and first limit ourselves to $N = 1000$ observations corresponding to images of the digits 0 and 1. Recall the MNIST dataset contains 28×28 pixel images and thus the dataset considered corresponds to a matrix \mathbf{X} of size $N \times M$ where $M = 784$. Suppose we compute the first two principal components of $\tilde{\mathbf{X}}$ using the PCA algorithm and plot $\tilde{\mathbf{X}}$ projected onto these first two components, $\tilde{\mathbf{X}}\mathbf{V}_2$. A plot of the projected data can be seen in the top of fig. 3.11 where the red dots correspond to 1's and the blue dots to 0's. From the plot we learn that the first two principal components, and in fact only the first which is plotted along the x -axis, is enough to distinguish these two classes fairly well. To get an idea about what the two principal components consist of we place a grid on top of the projected dots (top right pane) and select those observations nearest to the grid points. The corresponding observations are plotted in the bottom left pane of fig. 3.11. From this plot we learn that the first principal component (as we suspected) captures the variability between 1 and 0, meanwhile the second principal component appears to capture how *slanted* the digit is (this is particularly apparent for the digit 1) and how *bold* the digit is. In the bottom right-most pane we have plotted the same images as in the left pane but projected back into the original space using eq. (3.17) (we have added back the mean value for easier visualization). This plot provides a visualization of what we “see” when we consider only the first two principal components. The plot confirms our interpretation from before, however, we also see that the two first principal components arguably correspond to a significant loss of information, exactly as we can expect when we project a high dimensional dataset onto only the first two principal components.

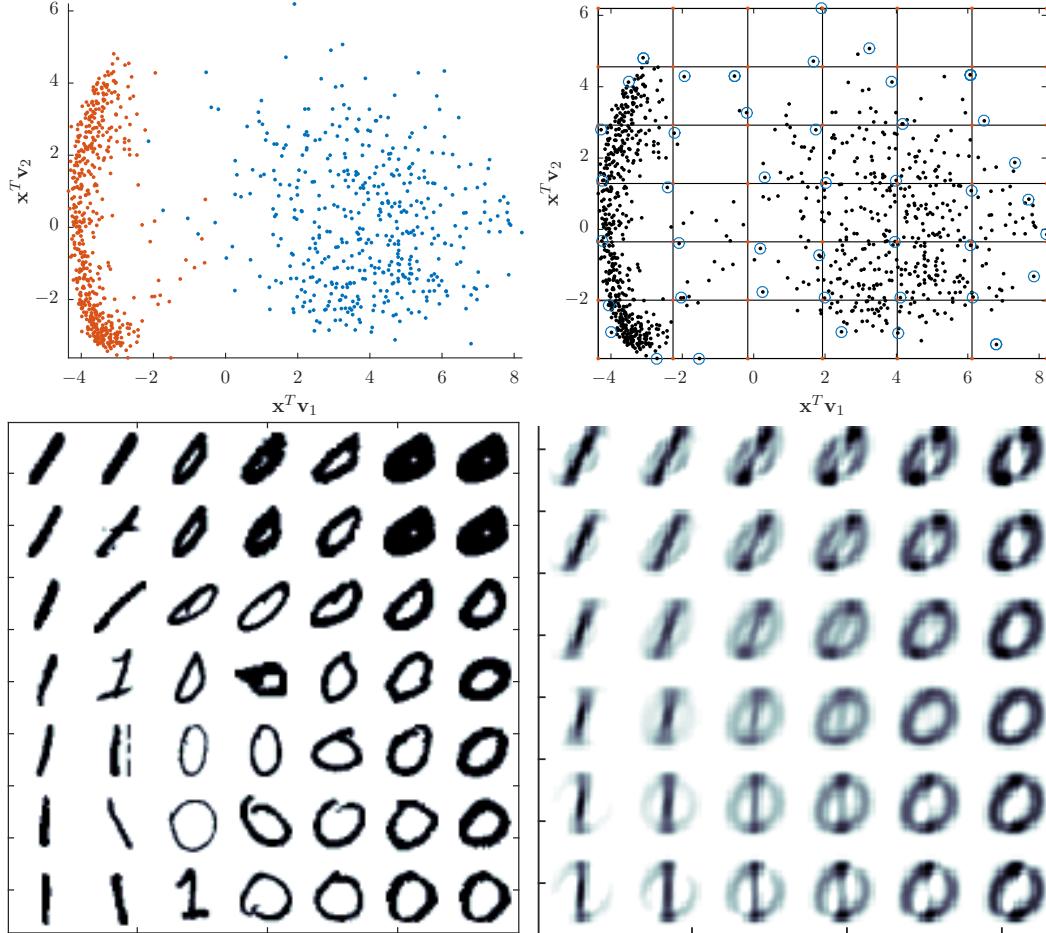


Fig. 3.11. $N = 1000$ observations from the MNIST dataset of the digits 0 and 1 are projected onto the first 2 principal directions as shown in the top-left pane. The red dots correspond to the digit 1 and we see the first principal direction easily allows us to distinguish the digits. If we select the observations the closest to the grid points in the top right pane and plot the observations in the bottom left pane we see the first principal direction separates 1s from 0s while the second corresponds to *slanting* and *bolding* (vertical direction). In the bottom-right pane, we have plotted the PCA projections; we see much of the information is lost when projecting onto the first 2 principal directions.

To get an idea about how much information we lose, we consider the *full* MNIST dataset containing 10 classes of digits and a total of $N = 60000$ observations. In the top row of fig. 3.12 we plot 10 randomly selected digits from each of the 10 classes, and in the following rows we plot the PCA reconstruction based on $n = 2, 5, 20, 50, 100$ principal directions. As expected, the reconstructions are awful when only a few directions are used. Around $n = 20$ directions most of the digits can be distinguished and for $n = 50$ (corresponding to a compression level of almost 16) the digits are clearly distinguishable. For $n = 100$ only a small amount of smudge separate the reconstruction from the true images; this corresponds to a compression level of almost 8. In

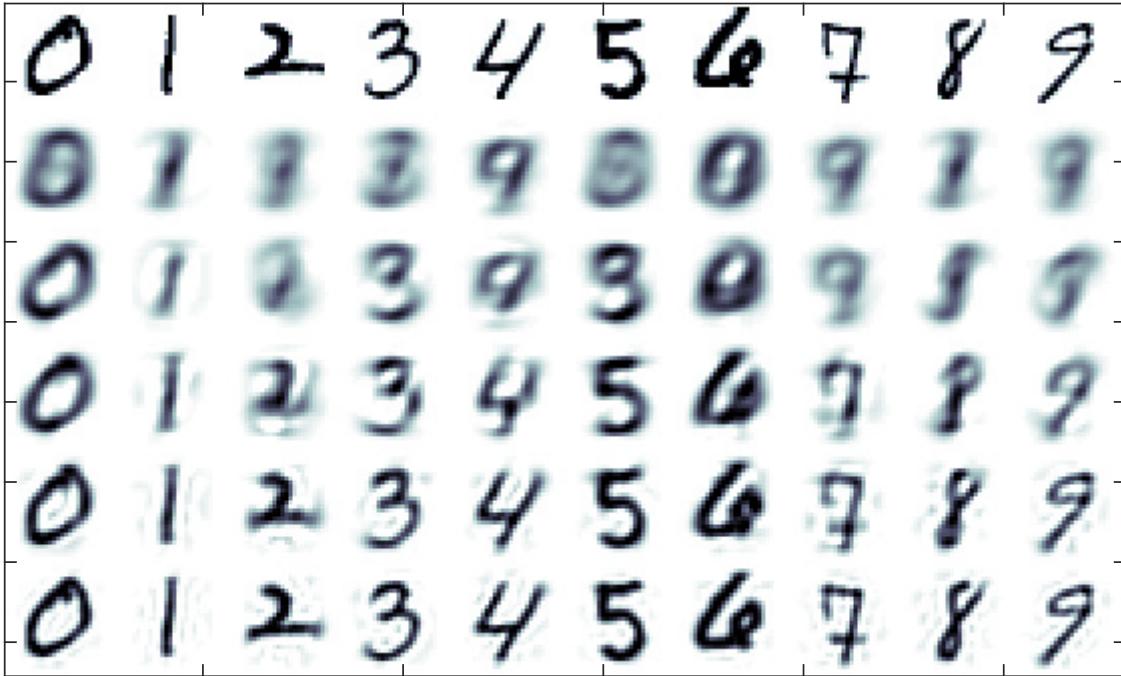


Fig. 3.12. Top row: 10 randomly chosen digits from the MNIST dataset. They are then reconstructed from $n = 2, 5, 20, 50, 100$ PCA components in the next 5 rows. We see that about 50 principal directions is enough to make a fairly good reconstruction of the images corresponding to a compression factor of about 16.

fig. 3.13 and fig. 3.14 we have plotted a subset of 5000 digits projected onto the first two principal components as well as when projected onto the space spanned by component 1 and 3 and 2 and 3. We again see the first plot easily allows 0 and 1 to be separated, however, the other digit classes have significant overlap.

3.4.3 Uses of PCA

PCA can be used in a variety of circumstances and ways. For instance

Visualization PCA is a easily applicable tool for data visualization. For instance in the MNIST dataset it allows us to distinguish the easy classification tasks (0 vs. 1) and the harder tasks.

Feature extraction Applying PCA provides new features that can be used for other machine-learning techniques.

Compression As we saw in the MNIST case, PCA provides a very simple yet efficient *lossy* compression method.

Despite these valid points, it is important to remember that PCA, when $n < M$, implies a *loss* of information. In the MNIST case this loss of information is very significant when $n < 50$, and one should therefore not automatically apply PCA. We will later consider more in depth how to

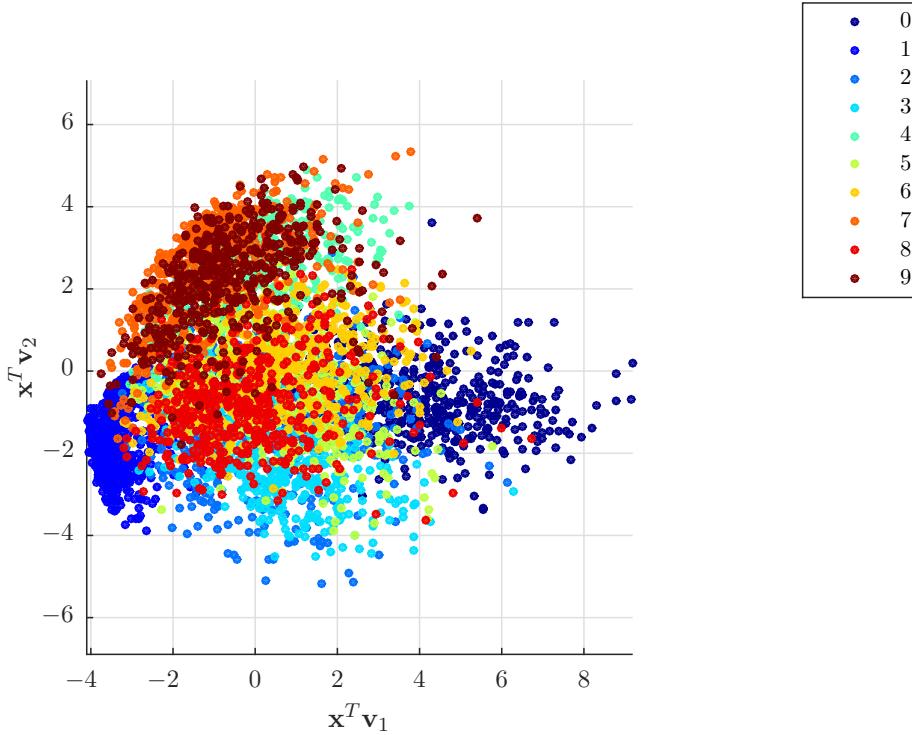


Fig. 3.13. $N = 2000$ observations of the MNIST dataset projected onto principal direction \mathbf{v}_1 and \mathbf{v}_2

validate machine-learning techniques and this will provide insight into how to determine if PCA is applicable and also how the number of principal components n should be selected. Note, as PCA is optimal for data compression optimized to describe as much of the data variance as possible the principal components do not necessarily provide features that are good for classification.

As a final comment, the normalization step in PCA can be of great importance. If the coordinates represent the same type of quantity and are on the same scale, for instance pixel intensities as in this example, the normalization step should likely be excluded. However, if the coordinates represent different things and are on vastly different scales the normalization step is important. For instance, suppose one dataset record the height of a person in meters as well as the person's annual income in USD. PCA will then accurately detect that the variance in the dataset is primarily in the annual income, after all this quantity will vary with many thousands between subjects while the height will only vary with about 1 meter. In this manner, the first PC (trivially) becomes the income and PCA will not tell us anything about the interaction between the variables. Meanwhile, normalizing this dataset with the standard deviation will ensure the height and income have the same scale, and PCA will easier pick out potential correlations.

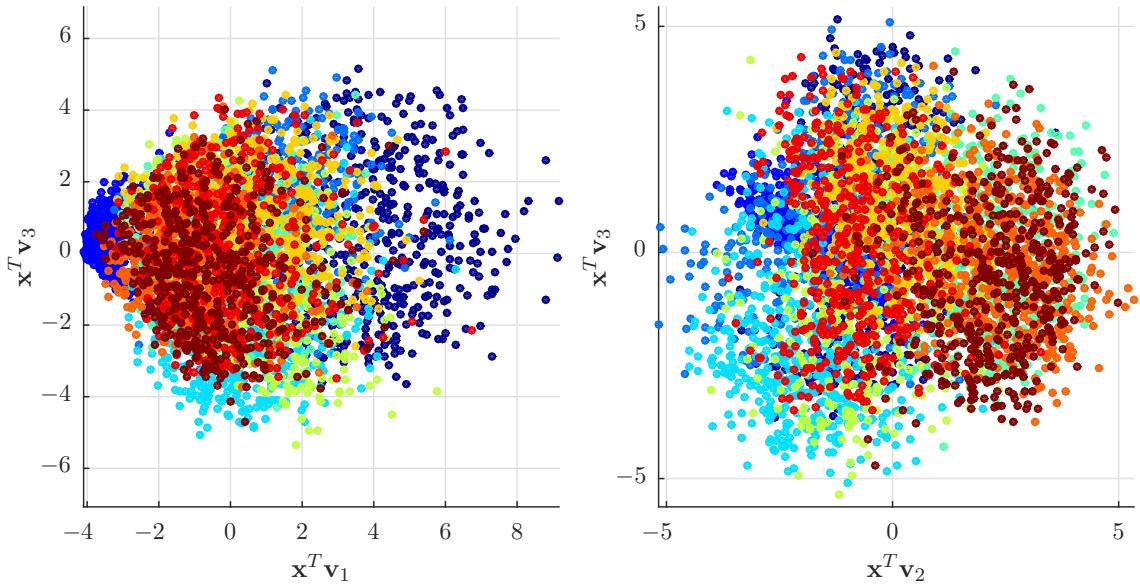


Fig. 3.14. $N = 2000$ observations of the MNIST dataset projected onto principal direction $\mathbf{v}_1, \mathbf{v}_3$ (left pane) and $\mathbf{v}_2, \mathbf{v}_3$ (right pane)

Problems

3.1. Question 1: The first and second principal components directions of the data in the RAT dataset (considering only the attributes $x_1 - x_{13}$) in Table 3.1 are:

$$\mathbf{v}_1 = \begin{bmatrix} 0.0247 \\ -0.0388 \\ -0.3288 \\ -0.2131 \\ 0.0477 \\ -0.4584 \\ 0.2683 \\ -0.0838 \\ -0.5020 \\ -0.0200 \\ -0.3091 \\ -0.2588 \\ 0.3714 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} -0.0764 \\ 0.5675 \\ -0.0550 \\ 0.2449 \\ 0.3115 \\ -0.1999 \\ 0.1738 \\ 0.3668 \\ -0.0737 \\ 0.2988 \\ 0.0628 \\ 0.4446 \\ 0.1051 \end{bmatrix}.$$

and in Figure 3.15 the data projected onto the first two principal components is plotted against the average consumer ratings (RAT). Which of the following statements is *correct*?

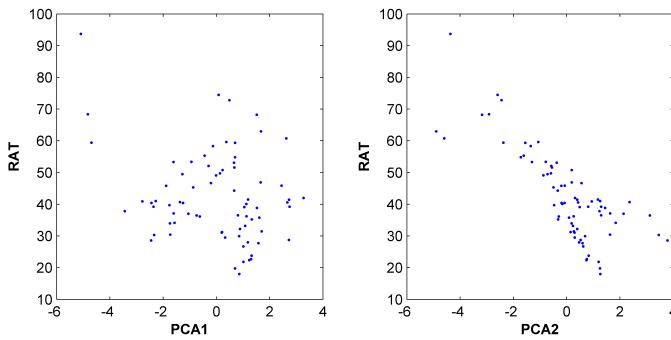


Fig. 3.15. The output RAT plotted against the first and second principal component respectively.

No.	Attribute description	Abbrev.
x_1	Type (0 = served cold, 1 = served hot)	TYPE
x_2	Calories per serving	CAL
x_3	Grams of protein	PROT
x_4	Grams of fat	FAT
x_5	Milligrams of sodium	SOD
x_6	Grams of dietary fiber	FIB
x_7	Grams of complex carbohydrates	CARB
x_8	Grams of sugars	SUG
x_9	Milligrams of potassium	POT
x_{10}	Vitamins and minerals in 0%, 25%, VIT or 100% of FDA recommendations	VIT
x_{11}	Shelf position (1, 2, or 3, counting from the floor)	SHELF
x_{12}	Weight in ounces of one serving	WEIGHT
x_{13}	Number of cups in one serving	CUPS
x_{14}	Name of cereal brand	NAME
y	Average rating of the cereal (from 0 to 100)	RAT

Table 3.1. Attributes in a study of cereals (i.e. breakfast products, taken from <http://lib.stat.cmu.edu/DASL/Datafiles/Cereals.html>). The data we consider has 74 observations (i.e., the original data has 77 observations but three observations have been removed due to missing values). The data has 14 input attributes $x_1 - x_{14}$ and one output variable y which defines the average rating of the cereal product given by the consumers.

A Relatively high values of CAL, PROT, FAT, FIB, SUG, POT, VIT, SHELF, and WEIGHT and low values of TYPE, SOD, CARB, and CUPS will result in a negative projection onto the first principal component.

B PCA2 primarily discriminates between relatively low values of PROT and high values of SHELF.

C The data projected onto the second principal component (i.e., PCA2) is positively correlated with RAT.

D The principal component directions are not guaranteed to be orthogonal to each other since the data has been standardized.

E Don't know.

3.2. Question 2: Consider the data set described in Table 3.2. Each attribute in the data set is standardized, and we carry out a principal component analysis (PCA) on the standardized input data, $x_1 - x_6$. The singular values obtained are: $\sigma_1 = 17.0$, $\sigma_2 = 15.2$, $\sigma_3 = 13.1$, $\sigma_4 = 13.0$, $\sigma_5 = 11.8$, $\sigma_6 = 11.3$. The first and second principal component directions are:

$$\mathbf{v}_1 = \begin{bmatrix} 0.5238 \\ 0.5237 \\ -0.3491 \\ 0.1981 \\ -0.3369 \\ 0.4204 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} -0.2948 \\ 0.3452 \\ 0.3584 \\ 0.6808 \\ -0.3049 \\ -0.3302 \end{bmatrix}.$$

Which one of the following statements is *incorrect*?

No.	Attribute description	Abbrev.
x_1	Age of Mother in Whole Years	Age
x_2	Mothers Weight in Pounds	MW
x_3	Race (1 = Other, 0 = White)	Race
x_4	History of Hypertension (1 = Yes, 0 = No)	HT
x_5	Uterine Irritability (1 = Yes, 0 = No)	UI
x_6	Number of Physician Visits First Trimester	PV
y	Birth Weight in Kilo Grams	BW

Table 3.2. Attributes in a study on risk factors associated with giving birth to a low birth weight (less than 2.5 kg) baby [Hosmer and Lemeshow, Applied Logistic Regression, 1989]. The data we consider contains 189 observations, 6 input attributes x_1-x_6 , and one output variable y .

- A The first three principal component account for more than 90% of the variation in the data.
- B Relatively heavy, old and white mothers that frequently goes to the physician and have a history of hypertension but do not have uterine irritability will have a positive projection onto the first principal component.
- C Relatively young, heavy mothers that are not white and have a history of hypertension but infrequently goes to the physician and do not have a uterine irritability will have a positive projection onto the second principal component.
- D Since the data is standardized we do not need to subtract the mean when performing the PCA but can directly carry out the singular value decomposition on the standardized data.
- E Don't know.

3.3. Question 3: All the attributes of the Galápagos data are standardized and a principal component analysis carried out on the standardized attributes x_1-x_7 . Using the singular value decomposition on the standardized data matrix of size 29×7 we obtain for the matrices S and V :

$$S = \begin{bmatrix} 9.7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6.7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5.7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.7 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.50 & -0.02 & 0.31 & -0.26 & 0.22 & -0.47 & 0.57 \\ 0.51 & -0.04 & 0.26 & -0.30 & 0.18 & 0.05 & -0.74 \\ 0.45 & -0.01 & -0.02 & 0.78 & -0.32 & -0.26 & -0.11 \\ 0.51 & -0.15 & -0.25 & 0 & -0.02 & 0.75 & 0.32 \\ -0.06 & -0.70 & 0.20 & -0.25 & -0.64 & -0.05 & 0.01 \\ -0.11 & -0.70 & -0.10 & 0.28 & 0.64 & -0.07 & -0.04 \\ 0.17 & -0.06 & -0.85 & -0.29 & -0.08 & -0.37 & -0.10 \end{bmatrix}$$

Which one of the following statements regarding the principal component analysis is *correct*?

- A The first principal component accounts for more than 55% of the variance in the data.
- B The first three principal components accounts for more than 90% of the variance in the data.
- C The last principal component accounts for more than 1% of the variance in the data.
- D Five principal components are required in order to account for more than 95% of the variance in the data.
- E Don't know.

3.4. Question 4: A principal component analysis is applied to a dataset composed of 1000 data points. After applying the principal component analysis, the data-points (gray points) along with the right singular vectors (i.e. the principal directions) (black arrows) are plotted. Which of the subplots A, B, C, D of figure fig. 3.16 corresponds to this description?

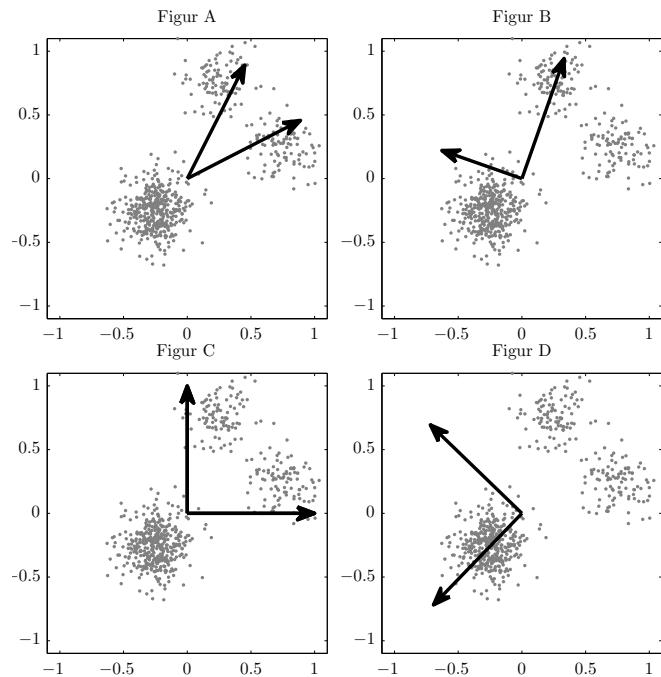


Fig. 3.16. Dataset of 1000 data points along with four possible choices of right singular vectors

- A Figure A
- B Figure B
- C Figure C
- D Figure D
- E Don't know.

3.5. Question 5: A principal component analysis is carried out on a dataset comprised of three data points $\mathbf{x}_1, \mathbf{x}_2$ and \mathbf{x}_3 collected in a $N \times M$ matrix \mathbf{X} such that

each row of the matrix is a data point. Suppose the matrix $\tilde{\mathbf{X}}$ corresponds to \mathbf{X} with the mean of each columns substracted i.e.

$$\mathbf{X} = \begin{bmatrix} 3.00 & 2.00 & 1.00 \\ 4.00 & 1.00 & 2.00 \\ 0.00 & 1.00 & 2.00 \end{bmatrix}, \quad \tilde{X}_{nm} = X_{nm} - \frac{1}{N} \sum_{k=1}^N X_{km}.$$

and suppose $\tilde{\mathbf{X}}$ has the singular value decomposition:

$$\tilde{\mathbf{X}} = \mathbf{U} \Sigma \mathbf{V}^\top, \quad \mathbf{U} = \begin{bmatrix} -0.26 & 0.77 & 0.58 \\ -0.54 & -0.61 & 0.58 \\ 0.80 & -0.16 & 0.58 \end{bmatrix},$$

$$\Sigma = \begin{bmatrix} 2.96 & 0.00 & 0.00 \\ 0.00 & 1.10 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} -0.99 & -0.13 & -0.00 \\ -0.09 & 0.70 & -0.71 \\ 0.09 & -0.70 & -0.71 \end{bmatrix}$$

What is the (rounded to two significant digits) coordinates of the first observation \mathbf{x}_1 projected onto the 2-Dimensional subspace containing the maximal variation?

- A $[-3.06 \ 0.31]^\top$
- B $[-0.78 \ 0.85]^\top$
- C $[-1.07 \ 0.21]^\top$
- D $[-3.16 \ 0.23]^\top$
- E Don't know.

3.6. Question 6: A principal component analysis is carried out on the wholesale data based on $x_1 \dots x_6$. The mean is substracted from each attribute and the singular value decomposition (SVD) is applied to the data matrix of size 440×6 . From the SVD we obtain for the matrices

\mathbf{S} and \mathbf{V} :

$$\mathbf{S} = 10^5 \cdot \begin{bmatrix} 2.69 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2.53 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.05 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.83 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.49 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.31 \end{bmatrix},$$

$$\mathbf{V} = \begin{bmatrix} -0.98 & -0.11 & -0.18 & -0.04 & 0.02 & -0.02 \\ -0.12 & 0.52 & 0.51 & -0.65 & 0.20 & 0.03 \\ -0.06 & 0.76 & -0.28 & 0.38 & -0.16 & 0.41 \\ -0.15 & -0.02 & 0.71 & 0.65 & 0.22 & -0.01 \\ 0.01 & 0.37 & -0.20 & 0.15 & 0.21 & -0.87 \\ -0.07 & 0.06 & 0.28 & -0.02 & -0.92 & -0.27 \end{bmatrix}.$$

We note that both \mathbf{S} and \mathbf{V} above have been rounded to the first couple of significant digits. Which one of the following statements regarding the principal component analysis is *correct*?

- A The first principal component accounts for less than 40 % of the variance in the data.
- B The first three principal components account for more than 95 % of the variance in the data.
- C The last two principal component account for more than 2 % of the variance in the data.
- D The fourth principal component accounts for more than 5 % of the variance in the data.
- E Don't know.

4

Summary statistics and measures of similarity

In this chapter, we will consider more elementary properties and definitions of a dataset. We will consider ways to summarize (or describe) attributes of the dataset but more importantly ways to compare observations.

4.1 Attribute statistics

When working with variables it is convenient to be able to summarize them using elementary statistical measures such as the mean and variance. Suppose we record observations x_1, \dots, x_N of a particular attribute x , for instance corresponding to the weight of $N = 20$ schoolchildren. Since we only have access to a small sample of schoolchildren the average weight of the N observations will only be an *estimate* of the true average weight of all schoolchildren and we therefore call the average computed from the $N = 20$ schoolchildren, denoted by $\hat{\mu}$, the *empirical mean* and use the hat-symbol to signify it is only a “*best guess based on the available information*”. In this manner we define the empirical mean, variance and standard deviation as follows:

$$\text{Emperical mean of } x: \quad \hat{\mu} \approx \frac{1}{N} \sum_{i=1}^N x_i \quad (4.1)$$

$$\text{Emperical variance of } x: \quad \hat{s} \approx \frac{1}{N-1} \sum_{i=1}^N (x_i - \hat{\mu})^2 \quad (4.2)$$

$$\text{Emperical standard deviation of } x: \quad \hat{\sigma} \approx \sqrt{\hat{s}} \quad (4.3)$$

Notice that for the estimate of the variance (and therefore also for the standard deviation) we divided by $N - 1$ and not N . This is because if we divide by N the estimate of the variance will be unrealistically small as we have used the data to also estimate the mean value ¹. The estimates eq. (4.2) and eq. (4.3) are therefore called *unbiased* and for a small sample they are considered superior to the *biased* estimators ²:

¹ For completeness it should be mentioned that if $N = 1$ it is common to set $\hat{s} = \hat{\sigma} = 0$.

² Note, if the true mean value μ is provided and not empirically estimated from the data this estimator is no longer biased and we should therefore use this estimate based on dividing by N .

$$\hat{s}_B \approx \frac{1}{N} \sum_{i=1}^N (x_i - \hat{\mu})^2, \quad \hat{\sigma}_B \approx \sqrt{\hat{s}_B}.$$

The mean value provides important information about a sample, however, it is also affected by outliers. A way to get around this is the median which corresponds to the value of x such that “half the observations” are greater than x and “half” are lower; i.e. the value of x that’s “in the middle” of the dataset. To put this formally, we first sort the values of x, x_1, x_2, \dots, x_N in ascending order, i.e. as $x'_1 \leq x'_2 \leq \dots \leq x'_N$. The median is then defined as the value of x such that “half” of the observations is less than x and “half” of the observations are greater than x . If N is odd this is just $x'_{(N+1)/2}$, and if N is even we compute the average of the two middle values:

$$\text{Median of } x: \quad \text{median}[x] = \begin{cases} x'_{(N+1)/2} & \text{if } N \text{ is odd} \\ \frac{1}{2} (x'_{N/2} + x'_{N/2+1}) & \text{if } N \text{ is even.} \end{cases} \quad (4.4)$$

Percentile

The concept of the median can be generalized to *percentiles*. The exact definition of percentiles is somewhat technical, but the concept is easy to understand: Given a sample \mathbf{x} , the p 'th percentile is a number x_p such that p percent of the observations are less than x_p . Consider for instance $N = 200$ university students where x_i denotes the grade average of student i , $i = 1, \dots, N$. The $p = 90\%$ 'th percentile is then a value of the grade average $x_{p=90\%}$, for instance $x_{p=90\%} = 11.7$, such that 180 students have a grade average *less* than $x_{p=90\%}$ and 20 students have a grade average *greater* than $x_{p=90\%}$. If we use the notation introduced for the median³ we might reasonably expect $x_{p=90\%} \approx x'_{\lceil Np \rceil}$, compare this to the definition of median which is obtained when $p = 50\%$. However, we have to use the approximately equal sign because the definition is slightly ambiguous. If 180 students has a grade average less than 11.7, presumably the same 180 students has a grade average less than 11.7001 and just as for the median we therefore has to select a reasonable value of $x_{p=90\%}$ somewhere between the grade of student $x_{\lfloor Np \rfloor}$ and $x_{\lceil Np \rceil}$. There are different ways to accomplish this interpolation in practice, all somewhat notationally involved, and the details need not concern us here⁴.

³ The notation $\lceil a \rceil$ rounds a upwards to the nearest integer whereas $\lfloor a \rfloor$ rounds a downwards to the nearest integer. For instance $\lceil 2.8 \rceil = 3$ and $\lfloor 2.8 \rfloor = 2$.

⁴ Nevertheless, as a punishment to the curious, a popular interpolation method is linear interpolation defined as follows: Suppose the dataset is sorted as $x'_1 \leq x'_2 \leq \dots \leq x'_N$. The percentile function for a percentile p can then be defined by first introducing the “granulated percentiles” $p_i = \frac{1}{N}(i - \frac{1}{2})$ for $i = 1, \dots, N$ and the function $X(z)$:

$$X(z) = x'_{\lfloor z \rfloor} + (z - \lfloor z \rfloor)(x'_{\lfloor z \rfloor + 1} - x'_{\lfloor z \rfloor}).$$

Notice if z is an integer $X(z) = x'_z$. The p 'th percentile can then be defined as $x_p = X(z)$ where z is selected as

$$z = \begin{cases} Np + \frac{1}{2} & \text{if } p_1 \leq p \leq p_N \\ 1 & \text{if } 0 \leq p < p_1 \\ N & \text{if } p_N < p \leq 1. \end{cases} \quad (4.5)$$

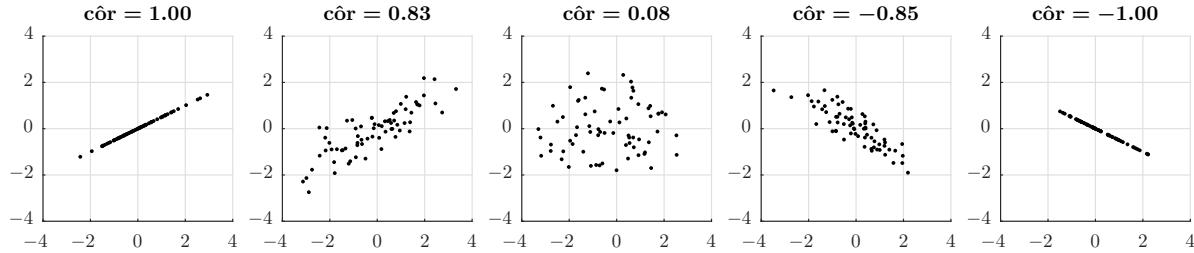


Fig. 4.1. Five two-dimensional datasets and their correlation as estimated from eq. (4.7).

Mode

We also define the *mode* as the most frequently occurring value of x_1, \dots, x_N . The mode may not be unique, for instance for the dataset

$$1, 2, 2, 4, 4$$

both 2 and 4 occur two times. In this case we say both 2 and 4 are the mode of the dataset and that the dataset is *multimodal*. For a value of x , we say that the number of times x occur in the dataset is the *frequency* of x . In the previous example the frequency of 1 is 1, the frequency of 2 and 4 is 2 and the frequency of 7 is 0.

4.1.1 Covariance and Correlation

Covariance measures how much one variable y can be expected to change when another variable x changes and visa-versa. Suppose we have a dataset containing two attributes x and y with recorded values x_1, x_2, \dots, x_N and y_1, y_2, \dots, y_N . If we let $\hat{\mu}_x$ and $\hat{\mu}_y$ denote the empirical mean of the two attributes the covariance of attribute x and y can be estimated as

$$\text{Empirical covariance of } x, y: \quad \text{c̄ov}[x, y] = \frac{1}{N-1} \sum_{i=1}^N (x_i - \hat{\mu}_x)(y_i - \hat{\mu}_y). \quad (4.6)$$

Notice that $\text{cov}[x, x] = \text{Var}[x]$. Given a dataset of M attributes, x_1, \dots, x_M , we can compute the pairwise covariance between any two attributes $\text{cov}[x_i, x_j]$ and collect all these in an $M \times M$ matrix $\tilde{\Sigma}$ where $\tilde{\Sigma}_{ij} = \text{cov}[x_i, x_j]$. This matrix is known as the *covariance matrix*. A drawback of the covariance as a summary statistic is that it is affected by the scale of each attribute. This can be overcome by standardizing with the empirical standard deviation of the two attributes, $\hat{\sigma}_x$ and $\hat{\sigma}_y$, leading to the *correlation* of x and y :

$$\text{Empirical correlation of } x, y: \quad \text{c̄or}[x, y] = \frac{\text{c̄ov}[x, y]}{\hat{\sigma}_x \hat{\sigma}_y} = \frac{1}{N-1} \sum_{i=1}^N \frac{(x_i - \hat{\mu}_x)(y_i - \hat{\mu}_y)}{\hat{\sigma}_x \hat{\sigma}_y}. \quad (4.7)$$

Correlation tells us how (linearly) related attributes are. A correlation of 0 means that x tells us nothing about y , a positive correlation tells us that when x is large y is also likely to be large and a negative correlation tells us that if x is large y will typically be small.

Table 4.1. A term document matrix corresponding to three lines from the tale

	assembl	beauti	court	courtyard	father	gave	hors	king	mount	princess	servant	watch	win	woo
0	1	0	0	1	1	1	0	0	1	0	0	1	0	
0	0	1	0	0	0	0	1	0	1	0	0	0	1	
1	0	0	1	0	0	1	0	1	0	1	1	0	0	

An instructive example is if we assume $y = ax + b$. In this case $\hat{\mu}_y = a\hat{\mu}_x + b$ and $\hat{\sigma}_y = |a|\hat{\sigma}_x$. We then obtain:

$$\begin{aligned}\text{côr}[x, y] &= \frac{\frac{1}{N-1} \sum_{i=1}^N (x_i - \hat{\mu}_x)(ax_i + b - (a\hat{\mu}_x + b))}{\hat{\sigma}_x \hat{\sigma}_y} \\ &= \frac{\frac{1}{N-1} \sum_{i=1}^N a(x_i - \hat{\mu}_x)(x_i - \hat{\mu}_x)}{|a| \hat{\sigma}_x \hat{\sigma}_x} \\ &= \text{sign}(a)\end{aligned}$$

So the correlation is -1 if $a < 0$ and 1 if $a > 0$. See fig. 4.1 for further examples.

4.2 Term-document matrix

Suppose we have a collection of documents (for instance news stories) and we wish to analyse them using machine learning. A problem is that the news stories will contain a different number of words and so they cannot naturally be represented as an $N \times M$ matrix. A way to overcome this is to represent the documents as what is known as the term-document matrix. Suppose as an example we consider three “documents” corresponding to three lines from the tale *Clumsy Hans* by H.C. Andersen

$$\begin{aligned}d_1 &= \left\{ \begin{array}{l} "I \text{ shall win the Princess!}" \text{ they both said, as their} \\ \text{father gave each one of them a beautiful horse.} \end{array} \right\} \\ d_2 &= \{ "To \text{ the King's court, to woo the Princess.} \} \\ d_3 &= \left\{ \begin{array}{l} All \text{ the servants assembled in the courtyard to watch} \\ them mount their horses, \end{array} \right\}\end{aligned}$$

In the term-document representation, we count the number of times each word in our vocabulary occurs in the documents. Each document can then be represented as a vector (of the same length as our vocabulary) where most of the entries are zero. In addition, it is common to remove words that are very common such as *an* or *the* (these are called stop words) as well as remove the tense of the words by removing the last letters (called stemming). For instance *horse* and *horses* are considered to count as the same stem *hors*. Doing this we obtain the table seen in table 4.1 where each row correspond to d_1 , d_2 and d_3 . This table can then be considered as the dataset matrix \mathbf{X} (i.e. N corresponds to the number of documents and M , the number of features, to the number of

word-stems) and is known as the term-document matrix⁵. For instance the first column of \mathbf{X} is $[0 \ 0 \ 1]^T$ because only the last document contains the word “assembled”.

4.3 Measures of distance

The concept of distance and similarity play a crucial role in machine learning. Suppose we have to determine if an image contains a picture of a cat or a dog. One way to phrase this problem is that we have to compare the image to what a cat ought to look like and what a dog ought to look like and determine which of the two the image is the most similar to. A computer learning method will often do something similar, and for that reason studying measure of distance and similarity more explicitly is useful. No simple definition exist for what a distance measure is except it is some function of two observations \mathbf{x}, \mathbf{y} such that the value is large when they are very dissimilar and small when they are very similar, however we are usually interested in measures of distance d which obey the following rules:

$$\text{non-negativity} \quad d(\mathbf{x}, \mathbf{y}) \geq 0, \quad (4.8)$$

$$\text{identity of indiscernibles} \quad d(\mathbf{x}, \mathbf{y}) = 0 \text{ if and only if } \mathbf{x} = \mathbf{y}, \quad (4.9)$$

$$\text{symmetry} \quad d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}), \quad (4.10)$$

$$\text{triangle inequality} \quad d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y}). \quad (4.11)$$

For instance, the triangle inequality is saying the distance from home to the workplace is not greater than the distance from the workplace to the baker and from the baker to home. A measure of distance which obey the above rules is called a *metric*. A common way to define distances is as the magnitude of the difference⁶ of observations, $\mathbf{x} - \mathbf{y}$ which, in a vector space, is called a *norm* and is denoted by $\|\mathbf{x}\|$. It must in turn obey:

$$\text{non-negativity} \quad \|\mathbf{x}\| > 0 \text{ if } \mathbf{x} \neq \mathbf{0}, \quad (4.12)$$

$$\text{scaling} \quad \|a\mathbf{x}\| = |a|\|\mathbf{x}\| \quad (4.13)$$

$$\text{triangle inequality} \quad \|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|. \quad (4.14)$$

Then we can define the distance from the norm as

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|. \quad (4.15)$$

No doubt, the most familiar norm is the Euclidian norm. Given a vector \mathbf{x} it is defined as

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_M^2}. \quad (4.16)$$

⁵ The reader may wonder why it is not called the document-term matrix when it has dimensions documents \times terms. This is because it is common in text analysis represents documents as the transpose of \mathbf{X} , and we have decided to re-use the terminology.

⁶ Naturally, this assume we can meaningfully subtract the two observations \mathbf{x}, \mathbf{y} from each other without getting into trouble. As an example of a way to get into troubles is if an attribute is nominal, such as the Origin-attribute from the Cars-dataset we encountered in chapter 2, in which case the particular (arbitrary) numeric encoding of the countries as integers should not affect their difference. The remedy is ofcourse to apply a 1-of- K encoding to this attribute.

More generally, we have the L_p norm (or simply p -norm) which, for any number $p \geq 1$ is defined as:

$$\|\mathbf{x}\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_M|^p)^{\frac{1}{p}}. \quad (4.17)$$

It is common to extend this definition to $p = \infty$ by the definition:

$$\|\mathbf{x}\|_\infty = \max\{|x_1|, |x_2|, \dots, |x_M|\}. \quad (4.18)$$

Based on the norm, we can define the p -distance as

$$d_p(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p = \begin{cases} \left(\sum_{i=1}^M |x_i - y_i|^p \right)^{\frac{1}{p}} & \text{if } 1 \leq p < \infty \\ \max\{|x_1 - y_1|, |x_2 - y_2|, \dots, |x_M - y_M|\} & \text{if } p = \infty. \end{cases}$$

Note in the particular case of the $p = \infty$ norm, the distance measures the largest difference in coordinates.

In fig. 4.2 we have plotted a large number of observations and colored those red which have a p -distance less than 1 to $(0, 0)$, i.e. $d_p(\mathbf{x}, \mathbf{0}) \leq 1$, for 6 different values of p . Note we have included $p = \frac{1}{2}$ for completeness (see also Technical Note 4.3.1).

Finally, we should also mention the Fröbenius norm which we encountered earlier in chapter 3:

$$\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^N \sum_{j=1}^M X_{ij}^2} = \sqrt{\text{trace}(\mathbf{X}^T \mathbf{X})}. \quad (4.19)$$

A useful way to think of the Fröbenius norm is as measuring the magnitude of the entire dataset: $\|\mathbf{X}\|_F = \sum_{i=1}^N \|\mathbf{x}_i\|^2$.

Technical note 4.3.1: The case $p < 1$ of the p -norm

For technical reasons the $p < 1$ case is more difficult. In general, we define the p -distance when $0 < p < 1$ as:

$$d_p(\mathbf{x}, \mathbf{y}) = |x_1 - y_1|^p + |x_2 - y_2|^p + \cdots + |x_M - y_M|^p, \quad (4.20)$$

and for the particular case $p = 0$ it is common to define $0^0 = 0$ and then call the function

$$\|\mathbf{x}\|_0 = |x_1|^0 + |x_2|^0 + \cdots + |x_M|^0, \quad (4.21)$$

which counts the number of non-zero coordinates of \mathbf{x} the $p = 0$ norm. Note from a mathematical standpoint this is a misnomer since it does not obey the mathematical properties of a norm.

4.3.1 The Mahalanobis Distance

Suppose we are given a covariance matrix Σ , for instance estimated from a dataset as in eq. (4.6). We can then define the *Mahalanobis distance* as

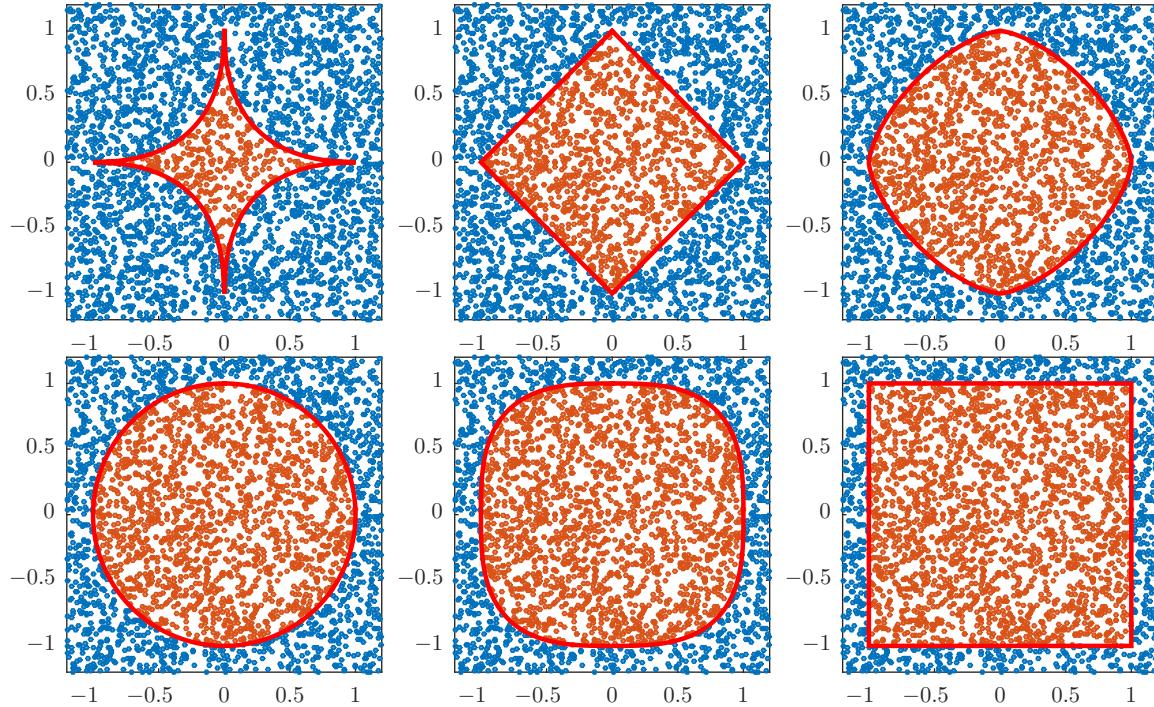


Fig. 4.2. Illustration of the p -distance for various values of p . A point \mathbf{x} is colored red if its p distance to the center $\mathbf{0}$ is less than 1, $d_p(\mathbf{x}, \mathbf{0}) \leq 1$. Top row: $p = \frac{1}{2}, 1, \frac{3}{2}$ and bottom row: $p = 2, 3, \infty$. The red line is the decision boundary $d_p(\mathbf{x}, \mathbf{0}) = 1$. Increasing p corresponds to “inflating” the red region.

$$d_M(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{y})}$$

Notice, if $\boldsymbol{\Sigma} = \mathbf{I}$ the Mahalanobis distance reduces to the Euclidian distance: $d_M(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \mathbf{I} (\mathbf{x} - \mathbf{y})} = \|\mathbf{x} - \mathbf{y}\|_2$. If $\boldsymbol{\Sigma}$ is estimated from a dataset as in eq. (4.6), what the corresponding Mahalanobis distance takes into account is (very roughly said) that the distance between two points should be lower when the points lie within the point cloud of the dataset. For instance in fig. 4.3 the distance between the two red points according to the Mahalanobis distance is 13 but only 4.15 between the blue points, however in both cases the Euclidian distance is roughly 5.65.

4.4 Measures of similarity

A measure of similarity is a function which takes two observations \mathbf{x}, \mathbf{y} as input and is large when \mathbf{x} is very similar to \mathbf{y} . Obviously, a measure of similarity can be constructed from a distance measure by a simple mapping. The most simple way is to define $s(\mathbf{x}, \mathbf{y}) = -d(\mathbf{x}, \mathbf{y})$, however, often it is desirable to have a measure of similarity on a scale that goes from 0 to 1 and so one could choose:

$$s(\mathbf{x}, \mathbf{y}) = \frac{a}{d(\mathbf{x}, \mathbf{y}) + a},$$

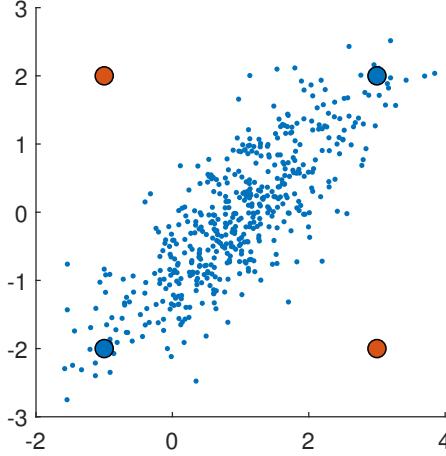


Fig. 4.3. A simple 2D dataset to illustrate the Mahalanobis distance. If we estimate the covariance matrix from the dataset, the Mahalanobis distance between the red points is 13 but only 4.15 between the blue points.

for a constant $a > 0$. Defining measures of similarity from distance is arguably a bit silly, but for some types of observations, measures of similarity may be easier to define than measure of dissimilarity. Consider the important situation where \mathbf{x} is binary, i.e. $x_i = 0, 1$ for all i . Suppose we define

$$\begin{aligned} f_{11} &= \{\text{Number of entries } i \text{ where } x_i = 1 \text{ and } y_i = 1\}, \\ f_{10} &= \{\text{Number of entries } i \text{ where } x_i = 1 \text{ and } y_i = 0\}, \\ f_{01} &= \{\text{Number of entries } i \text{ where } x_i = 0 \text{ and } y_i = 1\}, \\ f_{00} &= \{\text{Number of entries } i \text{ where } x_i = 0 \text{ and } y_i = 0\}. \end{aligned}$$

Then notice $M = f_{11} + f_{10} + f_{01} + f_{00}$ and we can then define the following measures:

$$\text{Simple Matching Coefficient} \quad \text{SMC}(\mathbf{x}, \mathbf{y}) = \frac{f_{11} + f_{00}}{M} \quad (4.22)$$

$$\text{Jaccard Similarity} \quad J(\mathbf{x}, \mathbf{y}) = \frac{f_{11}}{f_{11} + f_{10} + f_{01}} \quad (4.23)$$

$$\text{Cosine similarity} \quad \cos(\mathbf{x}, \mathbf{y}) = \frac{f_{11}}{\|\mathbf{x}\| \|\mathbf{y}\|} \quad (4.24)$$

For general vectors \mathbf{x}, \mathbf{y} the cosine similarity and the *extended Jaccard similarity* can also be defined as:

$$\text{Extended Jaccard Similarity} \quad \text{EJ}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^T \mathbf{y}}{\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - \mathbf{x}^T \mathbf{y}} \quad (4.25)$$

$$\text{Cosine similarity} \quad \cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^T \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \quad (4.26)$$

Why do we need three different measures of similarity? It is useful to consider the qualitative difference between these measures of similarity in context of the term-document example of table 4.1. We first notice the SMC makes no difference between 0 and 1; i.e. if we flip the zeros and ones it makes no difference to the similarity: $\text{SMC}(1 - \mathbf{x}, 1 - \mathbf{y}) = \text{SMC}(\mathbf{x}, \mathbf{y})$. This is useful in some cases, for instance if the zeros and ones arise only by convention, say, we record a “zero” if a patient is a male and “one” if the patient is a female (or visa-versa), and we do not want our machine-learning method to be influenced by this rather arbitrary choice.

However, for some datasets what is a zero and what is a one has an assymetric meaning. Consider the term-document example. For documents there will typically be many more 0s than 1s since a document only use a fraction of the vocabulary and so, since the 0s are counted as “matches”, the SMC will be large even for documents that have nothing in common, simply because they *don't* contain many of the same words: According to the SMC a recipe for ice-cream and the US constitution is quite similar since they don't use words like Armadillo, lumberjack or vacuum cleaner.

The Jaccard and cosine similarity gets around these problems by focusing on positive matches (i.e. words the documents *do* have in common), however with an important twist: Suppose we compare two documents \mathbf{x} and \mathbf{y} which are on the same topic, but \mathbf{x} is much longer than \mathbf{y} . In this case \mathbf{x} will (likely) contain many words not found in \mathbf{y} simply because the author wrote more text for document \mathbf{x} , and the Jaccard similarity will pick up on this and believe the documents are quite dissimilar, which may not be desirable. If we normalize by the document length, as is done in the Cosine similarity, this will somewhat correct for this problem and is therefore more suitable if some vectors has much larger magnitude (in the binary case more 1s) than others and this difference is not in itself considered very informative.

Problems

4.1. Question 1:

In table 4.2 is given the pairwise cityblock distances between 8 observations along with a description of the dataset. What can be concluded about the similarity of observation o_1 and o_3 ?

	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8
o_1	0	4	7	9	5	5	5	6
o_2	4	0	7	7	7	3	7	8
o_3	7	7	0	10	6	6	4	9
o_4	9	7	10	0	8	6	10	9
o_5	5	7	6	8	0	8	6	7
o_6	5	3	6	6	8	0	8	11
o_7	5	7	4	10	6	8	0	7
o_8	6	8	9	9	7	11	7	0

Table 4.2. Pairwise Cityblock distance, i.e $d(o_i, o_i) = \|\mathbf{x}_i - \mathbf{x}_j\|_1 = \sum_{k=1}^M |x_{ik} - x_{jk}|$, between 8 observations. Each observation o_i corresponds to a $M = 15$ dimensional binary vector, $x_{ik} \in \{0, 1\}$. The blue observations $\{o_1, o_2, o_3, o_4\}$ belong to class C_1 and the black observations $\{o_5, o_6, o_7, o_8\}$ belong to class C_2 .

A $\text{COS}(o_1, o_3) = 0.533$

B $J(o_1, o_3) = 0.533$

C $SMC(o_1, o_3) = 0.533$

D There is insufficient information to draw specific conclusions.

E Don't know.

4.2. Question 2: We will let $J(A, B)$, $SMC(A, B)$, and $\cos(A, B)$ denote the Jaccard Coefficient, Simple Matching Coefficient and Cosine Similarity respectively between observation A and B . We will consider the data in Table 4.3 containing 10 observations denoted NS1, NS2, NS3, NS4, NS5, AS1, AS2, AS3, AS4, and AS5 such that the first observation is given by $NS1 = \{1, 0, 0, 1, 0, 1, 1, 0\}$. Which one of the following statements is *correct*?

	CD_Y	CD_N	AST_Y	AST_N	SI_Y	SI_N	HF_Y	HF_N
NS1	1	0	0	1	0	1	1	0
NS2	0	1	1	0	1	0	1	0
NS3	1	0	0	1	0	1	1	0
NS4	0	1	1	0	0	1	1	0
NS5	1	0	1	0	1	0	1	0
AS1	0	1	1	0	0	1	1	0
AS2	0	1	1	0	0	1	1	0
AS3	0	1	1	0	0	1	1	0
AS4	0	1	0	1	1	0	0	1
AS5	1	0	1	0	0	1	1	0

Table 4.3. Given are the first five subjects with normal semen (denoted NS1, NS2, ..., NS5) as well as the first five subjects with abnormal semen (denoted AS1, AS2, ..., AS5) including whether these subjects have had a childhood disease or not (CD_Y , CD_N), accident or serious trauma or not (AST_Y , AST_N), serious injury or not (SI_Y , SI_N), and high fever or not (HF_Y , HF_N).

A $J(NS1, NS2) = SMC(NS1, NS2)$

B $\cos(NS4, NS5) = \frac{1}{8}$

C $J(NS5, AS5) = SMC(NS5, AS5)$

D $\cos(NS5, AS5) = \frac{3}{4}$

E Don't know.

4.3. Question 3: We will let $J(A, B)$, $SMC(A, B)$, and $\cos(A, B)$ denote the Jaccard Coefficient, Simple Matching Coefficient and Cosine Similarity respectively between observation A and B . We will consider the data in Table 4.4 containing 10 observations denoted S1, S2, S3, S4, S5, NS1, NS2, NS3, NS4, and NS5 such that the first observation is given by $S1 = \{1, 0, 1, 0, 1, 0\}$. Which one of the following statements is *correct*?

	YAY	YAN	OAY	OAN	PAY	PAN
S1	1	0	1	0	1	0
S2	1	0	1	0	0	1
S3	0	1	0	1	1	0
S4	0	1	1	0	1	0
S5	0	1	1	0	1	0
NS1	0	1	1	0	1	0
NS2	0	1	0	1	1	0
NS3	1	0	0	1	0	1
NS4	0	1	1	0	1	0
NS5	0	1	1	0	1	0

Table 4.4. Given are five subjects that survived in Haberman's study (denoted S1, S2, ..., S5) as well as the five subjects that did not survive in Haberman's study (denoted NS1, NS2, ..., NS5) including whether these subjects are young or old (YAY , YAN), were operated after 1960 or not (OAY , OAN), and had positive axillary nodes or not (PAY , PAN).

A Using the Jaccard coefficient S1 is more similar to S2 than to NS1, i.e. $J(S1, S2) > J(S1, NS1)$.

B Using the Simple Matching coefficient S1 is more similar to S2 than to NS1, i.e. $SMC(S1, S2) > SMC(S1, NS1)$.

C The Jaccard coefficient between S1 and S2 is identical to the Cosine Similarity between S1 and S2, i.e. $J(S1, S2) = \cos(S1, S2)$.

D The Simple Matching coefficient between S1 and S2 is identical to the Cosine Similarity between S1 and S2, i.e. $SMC(S1, S2) = \cos(S1, S2)$.

E Don't know.

Discrete probabilities and information

Correct reasoning is central to many areas of intellectual activity, including philosophy (how ought we reason?), cognitive science (how do we reason?), science (what does reason tell us about theories given experimental evidence?), and artificial intelligence (how do we build reasoning machines?).

Probability theory has something to say about all these subjects, however, it is particularly relevant for machine learning for two reasons:

- All modern approaches to machine learning makes use of probabilities as will all chapters of this book.
- Probabilities have something fundamental to say about *reasoning under uncertainty*. In certain important situations, if we want to build an optimal reasoning machine, it ought to reason according to probability theory.

The reader no doubt has some familiarity with probability theory, however it is our experience it is the single subject which causes the most difficulties. We will therefore provide a fairly detailed introduction to probabilities and probabilistic concepts, with a focus on probabilities as plausible reasoning. The reader should note we will introduce probabilities as the probability of true/false statements (i.e., *the probability it will rain tomorrow*), rather than statements about sets and stochastic variables, which is customary in statistics. This perspective might be a bit different from how probabilities are usually introduced in statistics (usually, based on set theory), but we believe this approach is both closer to how we think about probabilities informally, notationally simpler, and not less formally correct.

bibliographical remarks

Bayes' theorem was first described by Thomas Bayes who considered a problem very akin to the binomial distribution example we will consider in 6.4 but his work was only published after his death in 1763 [Bayes and Price, 1763], and the subject was significantly expanded by other early pioneers such as Pierre-Simon Laplace and many others. The approach to probability theory, including the interpretation as quantifying rational thought, was revitalized in the first half of the 20th century by Bruno de Finetti [De Finetti, 1937, Barlow, 1992], Harold Jeffreys [Jeffreys, 1939] and Richard T. Cox [Cox, 1946]. Information theory, which will be covered in the last section, is due to the seminal work by Claude Shannon in 1948 [Shannon, 1948]. The normalized version of mutual information, which builds onto Shannon's work, is due to Strehl and Ghosh [2002].

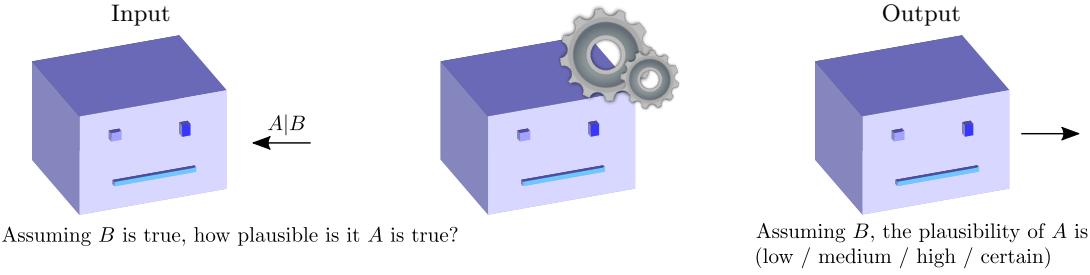


Fig. 5.1. The robot is given a query in the form $A|B$ (assume B is true, how plausible is A), the robot then reason about the answer, and outputs the plausibility of A given B .

5.1 Probability basics

An important part of what an intelligent robot should do is to reason correctly in light of evidence. Now, reasoning can mean many things, but the specific sense we are interested in is something akin to the legal or scientific sense, in which multiple pieces of evidence are weighted so as to determine the plausibility of a conclusion. In a legal context, the proposition we might be interested in could be:

$$G : \text{The accused is guilty.} \quad (5.1a)$$

$$E_1 : \text{A car similar to his was seen at the crime scene.} \quad (5.1b)$$

$$E_2 : \text{His mom says he was home on the night.} \quad (5.1c)$$

$$E_3 : \text{A large sum of money was found in his posession.} \quad (5.1d)$$

$$E_4 : \text{His fingerprints was found at the door of the bank.} \quad (5.1e)$$

If the robot was on the jury, we would ask it to determine the truth of G in light of the evidence E_1, \dots, E_4 . That is, the robot should assume E_1, \dots, E_4 are true, and based on this form a belief about whether G is true or false¹, see fig. 5.1. We can even simplify this query by defining a new proposition E as:

$$E \equiv E_1 \text{ and } E_2 \text{ and } E_3 \text{ and } E_4 \quad (5.2)$$

Since E is true only when E_1, \dots, E_4 are true we can simply ask the robot about G in light of E . In fact, we will assume all inputs to the robot is of the form:

$$A|B : \text{Determine the plausibility of } A \text{ assuming } B \text{ is true}$$

where it is assumed A and B can be any proposition which is either true or false. For instance, in fig. 5.1, our query to the robot would be of the form $G|E$.

¹ It is assumed there exists a more complete description of the symbols G , E_1 , etc.. For instance, G would refer to a particular bank heist, etc. etc.

5.1.1 A primer on binary propositions*

Since all queries to the robot are binary true/false propositions (sometimes also called Boolean propositions), it is useful to introduce some basic notation governing these. In the following sections, upper-case Latin letters A, B, C , etc. will denote binary propositions, i.e. statements that are either true or false. We will introduce the following shorthand to represent the operations **not**/**and/or**:

- Negation: \bar{A} (*logical not; true if A is false*)
- Conjunction: AB (*logical and; True if A and B are both true*)
- Disjunction: $A + B$ (*logical or; True if A or B is true*)

This notation allow us to write E from eq. (5.2) symbolically

$$E = E_1 E_2 E_3 E_4.$$

The notation allows us to conveniently query the robot about hypothetical situations. For instance, if the search on the suspects apartment had *not* turned up money, the evidence would be written as $E' = E_1 E_2 \bar{E}_3 E_4$ and we would make the query $G|E'$.

In addition to these operations, we will also define the following two special propositions:²

$$1 : A \text{ proposition which is always true} \quad (5.3a)$$

$$0 : A \text{ proposition which is always false} \quad (5.3b)$$

The following identities should be intuitively obvious:

$$A1 = A, \quad A + \bar{A} = 1, \quad \bar{\bar{A}} = A$$

In addition, we have the distributive rule:

$$A(B_1 + B_2 + \dots + B_n) = AB_1 + AB_2 + \dots + AB_n$$

It is easy to verify this is the case. If for instance the left-hand side is true, then both A and at least *one* of the B_i 's must be true, but then the right-hand side is also true. The following identity will also be useful:

$$A + B = \bar{A} \bar{B}.$$

It is easy to verify this by considering the different possibilities for A and B and observe both sides of the equality sign agree in all four case. If, for instance, both A and B are false then: $A + B = 0 + 0 = 0$ and $\bar{A} \bar{B} = \bar{0} \bar{0} = \bar{1} = 0$.

5.1.2 Probabilities and plausibility

Continuing the trial example, we cannot logically *deduce* G is true based on the evidence E (after all, there might be an innocent explanation of all the evidence), however certainly the evidence *increases*

² Note we could have used other symbols than 0 and 1, for instance K_0 and K_1 . If these two propositions cause the reader any issues, he or she can mentally substitute them for propositions which are indeed always true or false, for instance $2 + 2 = 4$ and $1 + 1 = 4$.

our confidence G is true. It is exactly this *degree-of-belief* in one proposition given another we model based on probabilities. Specifically, we denote by the *number*

$$P(G|E_1 E_2 E_3 E_4)$$

the *degree-of-belief* G is true given E_1 , E_2 , E_3 , and E_4 are true. If we leave aside the issue of how we compute the number, we can at least represent statements such as *the fingerprints are incriminating*:

$$P(G|E_1 E_2 E_3 E_4) > P(G|E_1 E_2 E_3).$$

The degree of belief is a number between 0 and 1, with 0 and 1 representing certainty

$$\begin{aligned} P(A|B) = 0 &\quad (\text{interpretation: given } B \text{ is true, } A \text{ is certainly false}) \\ P(A|B) = 1 &\quad (\text{interpretation: given } B \text{ is true, } A \text{ is certainly true}) \end{aligned}$$

along with the convention that the plausibility of something which is absolutely certain is 1: $P(1|A) = 1$.

It is worth stressing the symbol $P(A|B)$ represents a *state of knowledge* of the agent, and to be very careful symbols are not dropped. To take the guilty-example, the fact the suspects mom provides an alibi counts against him being guilty. However, notice that:

$$P(G|E_1 E_2) < P(G|E_1) < P(G|E_1 \bar{E}_2) \tag{5.4}$$

Why is this? It should be obvious the first number is smaller than the last, exactly because in the first case we *know* his mom provide an alibi (E_2), whereas in the last we know the mom *did not* provide an alibi (\bar{E}_2), and all being equal an alibi is exonerating; in the case of the middle-most number, the mom *might* or *might not* have provided an alibi, but the information is not specified in the query. We stress the point is not that eq. (5.4) is a rule which is always true (it is not), but rather that the reader take great care in following the rules when later manipulating probabilistic statements.

Technical note 5.1.1: Are probabilities and plausibility really the same?

A reader might at this point have two concerns: firstly, that by saying *probabilities* represent *plausibility*, we are abusing the meaning of probability and giving it a non-scientific meaning. Secondly, Why should it be exactly *probabilities* that quantify plausibility? Couldn't it be some other mathematical theory? We offer the following points:

- Isn't it just true? When we speak about probability, it seems like we are making statements about how plausible certain statements are given what we know. For instance: "It is very **probable/plausible** Manchester United will not win Champions League 2024".
- What is the alternative? The most common alternative interpretation of what a probability *is*, is the number of occurrences divided by the number of repeats of an experiment. But in the case of Manchester United, there is only a single, future event. Which two numbers are we supposed to divide? [Hájek, 1997, 2009]
- More importantly, it can be shown that if we make certain assumptions about what plausible reasoning should obey, only probability theory can implement those assumptions [Cox, 1946, Jaynes, 2003].

5.1.3 Basic rules of probability

To summarize the rules of probability: A probability is always written in the form $P(A|B)$ where A and B can be any binary propositions, as long as $B \neq 0$. A probability is always a number between 0 and 1, which measures our degree-of-belief in A if we assume B is true. Finally, probabilities *always* obey the following two rules:

$$\text{The sum rule: } P(A|C) + P(\bar{A}|C) = 1 \quad (5.5a)$$

$$\text{The product rule: } P(AB|C) = P(B|AC)P(A|C) \quad (5.5b)$$

Where A , B and C can be *any* three propositions.

Note in particular we could select C as the logical constant true, $C = 1$. In that case, C has no bearing on the truth of A and B ; after all, we always know the true constant is true, and we will use the shorthand $P(A|1) = P(A)$. We therefore have as a special case:

$$P(A) + P(\bar{A}) = 1, \quad P(AB) = P(B|A)P(A).$$

This is quite remarkable: Reasoning under uncertainty, and most of what we will learn about machine learning in this course, will come down to these two simple rules applied in different ways, and we invite the reader to vigilantly observe if we hold good on this promise.

5.1.4 Marginalization and Bayes' theorem

We will begin by deriving Bayes' theorem as an example of how non-trivial results can be obtained from just the sum and product rule. First, with some creative application of the sum and product rule we obtain:

$$\begin{aligned} P(B|C) &= P(B|C) [P(A|BC) + P(\bar{A}|BC)] = P(AB|C) + P(\bar{A}B|C) \\ &= P(B|AC)P(A|C) + P(B|\bar{A}C)P(\bar{A}|C). \end{aligned}$$

Next, if we then apply the product rule twice to $P(AB|C)$ we obtain

$$\begin{aligned} \text{The product rule: } &P(AB|C) = P(B|AC)P(A|C) \\ \text{The product rule again: } &P(AB|C) = P(A|BC)P(B|C). \end{aligned}$$

Observing the two right-hand expressions are equal and dividing both sides with $P(B|C)$ we obtain our first non-trivial result:

$$\begin{aligned} \text{Bayes' theorem: } P(A|BC) &= \frac{P(B|AC)P(A|C)}{P(B|C)} \\ &= \frac{P(B|AC)P(A|C)}{P(B|AC)P(A|C) + P(B|\bar{A}C)P(\bar{A}|C)}. \end{aligned}$$

Example 1: The taxicab accident

So why is Bayes' theorem so useful? Consider the following example due to Kahneman et al. [1982]:

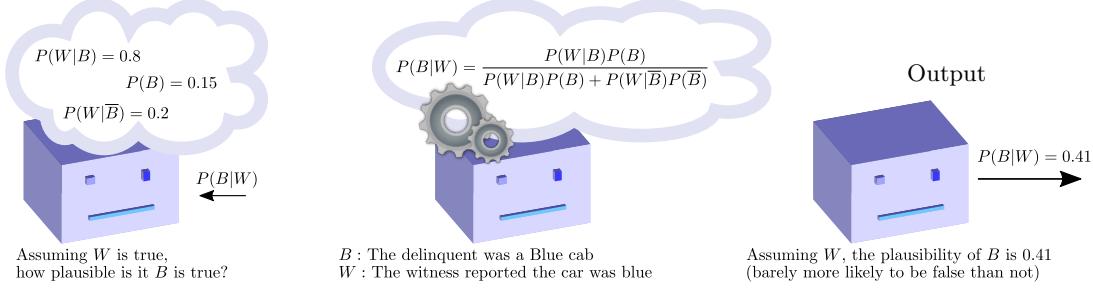


Fig. 5.2. Example of how our robot in fig. 5.1 might reason about the hit-and-run incident. First, the available information is transformed into probabilities and then, the query is expressed using the known probabilities and an answer is computed

Example 5.1.1: The taxicab accident

A cab was involved in a hit and run accident at night. Two cab companies, the Green and the Blue, operate in the city. You are given the following data:

- 85% of the cabs in the city are Green and 15% are Blue.
- A witness identified the cab as Blue. The court tested the reliability of the witness under the same circumstances that existed on the night of the accident and concluded that the witness correctly identified each one of the two colors 80% of the time and failed 20% of the time.

What is the probability that the cab involved in the accident was Blue rather than Green?

To solve the problem, we define the two binary propositions:

B : *The delinquent was a Blue cab.*

W : *The Witness reported the car was blue.*

Since cabs can only be green and blue, \bar{B} is the event the cab is green. We are interested in computing the probability the cab was Blue given the witness said it was blue $P(B|W)$. We first assign these numerical values using the information in the text (see also left-most pane of fig. 5.2), then we use the rules of probability to write an expression for the probability we are interested in (here, Bayes theorem, see middle pane of fig. 5.2), and finally compute an answer:

$$P(B|W) = \frac{P(W|B)P(B)}{P(W|B)P(B) + P(W|\bar{B})P(\bar{B})} \quad (5.6)$$

$$= \frac{0.8 \times 0.15}{0.8 \times 0.15 + 0.2 \times 0.85} \approx 41\% \quad (5.7)$$

So despite the witness testimony, the hit-and-run cab is more likely to be Green than Blue, i.e. $P(\bar{B}|W) = 1 - P(B|W) = 0.59$.

5.1.5 Mutually exclusive events

Probability as considered so far is only defined for binary events, however we can easily expand our notation to cover events with more than one outcome just using the basic rules of probability theory. Suppose we roll an ordinary die. There are then six possible outcomes corresponding to the six events

$$\begin{aligned} A_1 &: \text{The side } \square \text{ face up.} & A_2 &: \text{The side } \square \text{ face up.} \\ A_3 &: \text{The side } \square \text{ face up.} & A_4 &: \text{The side } \square \text{ face up.} \\ A_5 &: \text{The side } \square \text{ face up.} & A_6 &: \text{The side } \square \text{ face up.} \end{aligned} \quad (5.8)$$

Since a die can only show one face up at a time, no two of these propositions can be true at the same time, and they are said to be *mutually exclusive*. This means that for any values of i and j :

$$A_i A_j = \begin{cases} A_i & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

In general, assuming n events A_1, \dots, A_n are mutually exclusive, one can show the following generalization of the sum rule (see Technical Note 5.1.2):

$$P(A_1 + A_2 + \dots + A_n | C) = P(A_1 | C) + P(A_2 | C) + \dots + P(A_n | C). \quad (5.9)$$

As a special case, consider the case the events are also *exhaustive*, meaning that i.e. $A_1 + \dots + A_n = 1$. This is the case for the die where we know one of the six propositions A_1, \dots, A_6 has to be true because one side must be facing up. In this case the left-hand side of eq. (5.9) is equal to $P(1|C) = 1$ and therefore:

$$1 = P(A_1 | C) + P(A_2 | C) + \dots + P(A_n | C). \quad (5.10)$$

Furthermore, if A_1, \dots, A_n are both mutually exclusive and exhaustive then, for any B and C :

$$\begin{aligned} P(B|C) &= P(B|C) \cdot 1 = P(B|C) \left[\sum_{i=1}^n P(A_i|BC) \right] = \sum_{i=1}^n P(A_i|BC)P(B|C) \\ &= \sum_{i=1}^n P(BA_i|C) = \sum_{i=1}^n P(B|A_iC)P(A_i|C). \end{aligned} \quad (5.11)$$

This general procedure will be used many times in the following and is known as *marginalization*. Our first use of marginalization will be to generalize Bayes theorem to many events: Suppose A_1, \dots, A_n is a set of mutually exclusive and exhaustive hypothesis and B is some piece of evidence, we then have

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B)} = \frac{P(B|A_i)P(A_i)}{\sum_{j=1}^n P(B|A_j)P(A_j)}.$$

Technical note 5.1.2: Derivation of the sum rule for multiple events

Suppose A and B are two mutually exclusive events, i.e. both cannot happen at the same time. Consider the event A or B occurs written as $A + B$. Recall this can be written as:

$$A + B = \overline{A} \overline{B}$$

thus, using the sum rule, $P(A + B) = P(\overline{A} \overline{B}) = 1 - P(\overline{A} \overline{B})$. We can then compute

$$\begin{aligned} P(A + B) &= 1 - P(\overline{A} \overline{B}) &= 1 - P(\overline{A}|\overline{B})P(\overline{B}) \\ &= 1 - [1 - P(A|\overline{B})] P(\overline{B}) = P(B) + P(A\overline{B}) \\ &= P(B) + P(\overline{B}|A)P(A) &= P(B) + [1 - P(B|A)] P(A) \\ &= P(A) + P(B) - P(AB). \end{aligned} \tag{5.12}$$

In the case of the die this gives $P(A_1 + A_2) = P(A_1) + P(A_2) - P(A_1A_2)$, however, since the same die cannot show two faces up at once we know $P(A_1A_2) = P(0) = 0$ and this simplifies to

$$P(A_1 + A_2) = P(A_1) + P(A_2).$$

Repeated applications of this result show that for n mutually exclusive events we get eq. (5.9)

$$P(A_1 + A_2 + \dots + A_n|C) = P(A_1|C) + P(A_2|C) + \dots + P(A_n|C).$$

5.1.6 Equally likely events

We have purposefully introduced probabilities without reference to specific numbers, exactly to make it clear our two rules of probability are true regardless of what the propositions refer to or what their specific values are. That being said, we obviously need a way to assign numerical values to probabilities, which is what we will address here.

There is a tendency, which we warn against, to think familiar counting rules such as

$$P(\text{Positive}) = \frac{\{\text{Number of positive cases}\}}{\{\text{Number of trials}\}} \tag{5.13}$$

has a *fundamental* status in probability theory. Obviously, we don't want to say these rules are false, but a rule such as the above is true only because we can somehow *demonstrate* it is true. Doing that has the benefit of giving us a clear idea of when it is applicable, and when it is not.

Let's begin with a simple case, namely the die from the previous section. Now, it no doubt seems intuitively obvious that the chance side ☰ or ☱ comes up is:

$$P(A_4) = P(A_3) = \frac{1}{6}$$

but *why* is this true? Let us carefully go through the steps involved:

- We don't have any information about the die, or how it was rolled, which makes it more plausible A_4 will occur than A_3 or visa versa

- As no *information* makes us prefer A_4 over A_3 (or visa-versa), they are equally plausible. As probability measures plausibility, we conclude $P(A_4) = P(A_3)$
- Re-doing this argument for all pairs we conclude $P(A_i) = P(A_j)$ for all i, j .
- Since the events are mutually exclusive, we know from eq. (5.10) that

$$1 = P(A_1) + \cdots + P(A_6) = 6P(A_1)$$

- Therefore, $P(A_1) = \frac{1}{6}$, and $P(A_i) = \frac{1}{6}$ for all i since they are equally plausible

Based on this example, we can conclude that in an experiment with N mutually exclusive outcomes, where we have no information that makes us prefer one outcome over another, the outcomes have probability $\frac{1}{N}$.

Let us consider a slightly more elaborate example. Suppose we ask the probability the next roll of the die will be a prime. If we denote this event by R , we see it can be written as:

$$R = \{\text{Next roll is prime}\} = \{\text{Next roll is 2, 3 or 5}\} = A_2 + A_3 + A_5$$

Since these events are mutually exclusive, we can use eq. (5.9) to get:

$$P(R) = P(A_2 + A_3 + A_5) = P(A_2) + P(A_3) + P(A_5) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{3}{6} = \frac{1}{2}. \quad (5.14)$$

Importantly, in this example we ended up doing exactly the same as eq. (5.13), however, the *way* we arrived at the result was by showing each event was equally probable, and then applying the rule for adding mutually exclusive probabilities.

As another example, consider once more the cars example from table 2.1. Recall the dataset consisted of data from $N = 142$ cars; if we focus on the Cylinders-attribute, and suppose there are $c_4 = 95$ cars with 4 cylinders, it seem intuitive to compute the probability a car has 4 cylinders as:

$$P(\{4 \text{ cylinders}\}) = \frac{c_4}{N} = \frac{95}{142}$$

Probabilities of this form is commonly called the *empirical frequency* or *empirical estimates*. But what, exactly, does this probability refer to? In light of the previous example, this probability corresponds exactly to the case where we select a cars-instance from the dataset with equal probability, $P(\{\text{Car } i\}) = \frac{1}{N}$, ask the probability such an instance has four cylinders, written as

$$F_1 + F_2 + \cdots + F_{142}$$

(where F_i is the proposition car i has four cylinders) and then apply the same computation that lead to eq. (5.14) to $P(F_1 + \cdots + F_{142})$.

This may seem like a rather long discussion to arrive at something intuitively obvious, however, see Example 5.1.2 for a non-trivial combination of simple counting arguments and the basic rules of probability, or section 5.2.1 for a continuation of the cars-example where we derive counting rules for estimating conditional probabilities.

Finally, the probability of four cylinders in the cars-dataset refers to a fact about the particular dataset, and not in and by itself about the general prevalence of cylinders in cars, which is what we really wish to know. This type of generalization can be though of as our first instance of *learning*, and is one we will return to several times in the coming two chapters, see section 5.4 and section 6.4.

Example 5.1.2: Two dice★

Consider the following problem: *Suppose we roll two dice. If at least one of the die show five, what is the chance both show five?*

We can easily compute this using the previous ideas. Let A_i be the event die 1 shows face i and B_j the event die 2 shows j . Using the product rule, and the obvious fact if both dice show five at least one must show five,

$$\begin{aligned} P(\{\text{Both } \boxtimes\} | \{\text{At least one } \boxtimes\}) &= \frac{P(\{\text{Both } \boxtimes\} \{\text{At least one } \boxtimes\})}{P(\{\text{At least one } \boxtimes\})} \\ &= \frac{P(\{\text{Both } \boxtimes\})}{P(\{\text{At least one } \boxtimes\})} \end{aligned}$$

If we apply the marginalization rule eq. (5.11) twice we get:

$$\begin{aligned} P(\{\text{At least one } \boxtimes\}) &= \sum_{i=1}^6 \sum_{j=1}^6 P(\{\text{At least one } \boxtimes\} | A_i B_j) P(A_i B_j) \\ &= P(A_1 B_5) + P(A_2 B_5) + P(A_3 B_5) + P(A_4 B_5) + P(A_5 B_5) + P(A_6 B_5) \\ &\quad + P(A_5 B_1) + P(A_5 B_2) + P(A_5 B_3) + P(A_5 B_4) + P(A_5 B_6) \\ &= \frac{11}{36} \end{aligned}$$

where we used $P(A_i B_j) = P(A_i)P(B_j) = \frac{1}{36}$, and that when we know the outcome of both rolls, the conditional probability is either 0 or 1. A similar argument gives $P(\{\text{Both } \boxtimes\}) = \frac{1}{36}$, and therefore

$$P(\{\text{Both } \boxtimes\} | \{\text{At least one } \boxtimes\}) = \frac{\frac{1}{36}}{\frac{11}{36}} = \frac{1}{11}.$$

Example 2: The Monty Hall game show★

As an illustration of the sum rule consider the following more elaborate problem originally posed by Steve Selvin in 1975 [Selvin et al., 1975]:

Example 5.1.3: The Monty Hall game show

Suppose you're on a game show, and you're given the choice of three doors 1, 2, 3. Behind one door is a car; behind the others, goats. You pick a door, say 1, and the host, who knows what's behind the doors, opens another door, say 3, which has a goat. He then says to you, "Do you want to pick door 2?". If you know the host never opens the door with a car is it then to your advantage to switch your choice?

It is tempting to solve the problem with reasoning along the following lines:

Independent of what door we choose, there is a $\frac{1}{3}$ chance door 1 contains the car and $\frac{1}{3}$ that door 2 contains the car. That the host later tells us something about door 3 does not shuffle the goat and car around and so they remain equally likely to be behind the first two doors. Thus the chance the car is behind door 1 is still $\frac{1}{2}$ and there is no advantage in switching.

This line of reasoning refers to several facts about the problem which are not in dispute (such as the initial probabilities being $\frac{1}{3}$). Do you think it is true? If the argument is true, it should not hurt to examine it with more rigor. To do so, let us define the four variables:

A_1, A_2, A_3 : The car is behind door 1, 2 and 3 respectively

R_{g3} : The host reveals a goat behind door 3.

Solving the riddle boils down to computing $P(A_1|R_{g3})$, namely the probability the car is behind door 1 given we initially selected door 1 and the host subsequently revealed the goat was behind door 3. Using our newly derived version of Bayes theorem with mutually exclusive hypothesis we get:

$$P(A_1|R_{g3}) = \frac{P(R_{g3}|A_1)P(A_1)}{P(R_{g3}|A_1)P(A_1) + P(R_{g3}|A_2)P(A_2) + P(R_{g3}|A_3)P(A_3)} \quad (5.15)$$

Since the car is initially placed randomly we have $P(A_1) = P(A_2) = P(A_3)$. Then notice

- If the car is behind door 1 and we selected door 1, then $P(R_{g3}|A_1) = \frac{1}{2}$ as the host choose randomly between door 2 and 3 which both contains goats.
- If the car is behind door 2 and we selected door 1, then $P(R_{g3}|A_2) = 1$ as the host cannot open our door (containing a goat) or the door with a car.
- If the car is behind door 3 and we selected door 1, then $P(R_{g3}|A_3) = 0$ as the host will never open the door with a car

If we plug this information into eq. (5.15) we have:

$$\begin{aligned} P(A_1|R_{g3}) &= \frac{P(R_{g3}|A_1)P(A_1)}{P(R_{g3}|A_1)P(A_1) + P(R_{g3}|A_2)P(A_2) + P(R_{g3}|A_3)P(A_3)} \\ &= \frac{P(R_{g3}|A_1)}{P(R_{g3}|A_1) + P(R_{g3}|A_2) + P(R_{g3}|A_3)} \\ &= \frac{\frac{1}{2}}{\frac{1}{2} + 1 + 0} = \frac{1}{3} \end{aligned}$$

Since the car is either behind the first or second door, then $P(A_2|R_{g3}) = 1 - P(A_1|R_{g3}) = \frac{2}{3}$ and so it is clearly in your best interest to switch doors.

So what went wrong with the initial argument? The argument (subtly) relied on the idea that probabilities referred to the actual state of the world and so should only change when the state of the world changes (the goat and car cannot change places because of what the game host does). In actuality, probabilities refer to our state of knowledge, and when we are given relevant knowledge about the problem, our assignment of probability may change even if the world remains the same.

5.2 Discrete data and stochastic variables

Data will often come in numerical form, in which case it is common to express probabilities using *stochastic variables*. This notation is often the cause of unnecessary pain, likely because the name *stochastic* makes one think about chance. This is *not* the case, rather, *stochastic variables are simply a notational shortcuts that makes it easier to define binary propositions of the kind we have already encountered.*

More specifically, suppose in some situation we are measuring a quantity X . A way to think about that is (for instance) a robot that has a sensor which keep track of the acceleration (on a discrete scale), or in another example, a variable which corresponds to the number of children a family has.

Specifically, when we write $X = x$, where x is a number, we define that to be the binary proposition X_x :

$$X_x : \{\text{The binary event that the quantity } X \text{ is equal to the number } x\} \quad (5.16)$$

In other words, whenever the reader encounters the statement $X = x$, simply make the above mental substitution and we are back in the usual language of binary propositions.

Let us make this concrete with a few examples. First, referring back to the Monty-Hall problem, we can re-express the problem using the stochastic variables A and R :

$$A = i \equiv \{\text{the car is behind door number } i; \text{ equivalent to } A_i\}$$

$$R = j \equiv \{\text{the goat was revealed to be behind door number } j\}$$

Using this notation, we can express the solution to the Monty Hall example eq. (5.15) as:

$$P(A_1|R_{g3}) \equiv P(A = 1|R = 3) = \frac{P(R = 3|A = 1)P(A = 1)}{\sum_{i=1}^3 P(R = 3|A = i)P(A = i)}.$$

As another example, we will consider the *silly die*. Suppose we take an ordinary die and paint each side with the numbers $-2, 1, 1, 4, 4$ and 10 . When we then roll the silly die and read the number on the side facing up, we thereby generate a random number. This outcome can be described as a random variable X that takes one of the four different values

$$x_1 = -2, \quad x_2 = 1, \quad x_3 = 4, \quad \text{and} \quad x_4 = 10$$

the probability of each outcome being (see also fig. 5.3)

$$p(X = x_1) = p(X = x_4) = \frac{1}{6}, \quad p(X = x_2) = p(X = x_3) = \frac{1}{3} \quad (5.17)$$

Note in this example, we chose to *enumerate* the events as x_1, \dots, x_4 rather than having to write their numerical value again and again.

This type of notation is so convenient we will use it from now one. Therefore, suppose X and Y are stochastic variables and they each take values x_1, x_2, \dots and y_1, y_2, \dots respectively. We can easily re-express for instance the product-rule to say that for any i and j :

$$\text{Product rule, normal version:} \quad P(X_{x_i} Y_{y_j}) = P(X_{x_i} | Y_{y_j}) P(Y_{y_j}) \quad (5.18)$$

$$\text{Stochastic variable version:} \quad P(X = x_i, Y = y_j) = P(X = x_i | Y = y_j) P(Y = y_j) \quad (5.19)$$

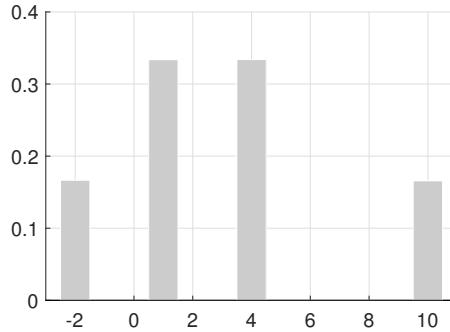


Fig. 5.3. Illustration of the density $p(X)$ of the silly die, note the numbers sum to 1.

Table 5.1. Summary counts of the cars dataset

Origin	Four cylinders	Six cylinders	Eight cylinders
France	11	10	9
Germany	17	12	2
USA	28	21	32

Where, once more, by X_{x_i} we simply mean that the variable X , for instance the roll of the silly die, took value x_i .

To make matters slightly more complicated, it is common to drop the stochastic variable if it is clear from the context, and for instance write eq. (5.19) as:

$$p(x_i, y_j) = p(x_i|y_j)p(y_j).$$

summary box 5.2.1 re-states the rules of discrete probabilities we have previously derived in this notation.

5.2.1 Example: Bayes theorem and the cars dataset

To familiarize ourselves with the new notation, we will now re-visisit the *Cars* example from section 5.1.6. Suppose an inspection of the data in table 2.1 reveal the counts in table 5.1. These should be read as saying there are 2 cars which are both made in Germany and have eight cylinders. Now, suppose we are interested in the question

What is the probability a car is from Germany given it has eight cylinders?

To solve this, the first thing we should do is define relevant stochastic variables O and C :

$$\text{country of origin: } O = 1, 2, 3 \quad (5.20)$$

$$\text{number of cylinders: } C = 4, 6, 8. \quad (5.21)$$

Where $O = 1$ is USA, 2 is Germany, and 3 is France. The query of interest is then $P(O = 2|C = 8)$. Notice that according to the product rule, this can be written as:

$$P(O = 2|C = 8) = \frac{P(O = 2, C = 8)}{P(C = 8)}.$$

These two probabilities can be computed from the data using the methods in section 5.1.6, eq. (5.13); i.e. we assume each car observation has a probability of $\frac{1}{142}$ and sum those together that matches the query we are interested in. Specifically:

$$P(O = 2, C = 8) = \frac{2}{142}, \quad P(C = 8) = \frac{9 + 2 + 32}{142}, \quad P(O = 2|C = 8) = \frac{\frac{2}{142}}{\frac{43}{142}} = \frac{2}{43}.$$

Let us check this is consistent with Bayes' theorem. To do so, we need to compute the probabilities:

$$P(O = 1) = \frac{11 + 10 + 9}{142} = \frac{30}{142}, \quad P(O = 2) = \frac{17 + 12 + 2}{142} = \frac{31}{142}, \quad P(O = 3) = \frac{28 + 21 + 32}{142} = \frac{81}{142}.$$

In addition, we also need the conditional probabilities

$$P(C = 8|O = 1) = \frac{9}{30}, \quad P(C = 8|O = 2) = \frac{2}{31} \quad P(C = 8|O = 3) = \frac{32}{81}.$$

Therefore, we obtain:

$$\begin{aligned} P(O = 2|C = 8) &= \frac{P(C = 8|O = 2)P(O = 2)}{\sum_{o=1}^3 P(C = 8|O = o)P(O = o)} \\ &= \frac{\frac{2}{31} \times \frac{31}{142}}{\frac{9}{30} \times \frac{30}{142} + \frac{2}{31} \times \frac{31}{142} + \frac{32}{81} \times \frac{81}{142}} = \frac{2}{43}. \end{aligned}$$

That these two ways of computing the probability agree match should not come as a surprise: Fundamentally, it derives from our assumption each car-observation has a probability of $\frac{1}{142}$, and then applying the basic rules of probability. Since the rules of probability are always true, *obviously* the result must be consistent!

Technical note 5.2.1: More comments on notation

The keen reader will observe we have changed to a lower-case p . This change reflects that from a mathematical perspective, we can simply think of $p(y_j)$ and $p(x_i, y_j)$ as the *evaluation* of the function $p(\cdot)$ and $p(\cdot, \cdot)$ which compute the probability. Note the notation is heavily overloaded, and unless it is evident from the context we will sometimes write

$$p_X(x_i), \quad p_{X,Y}(x_i, y_j) \quad \text{and} \quad p_{X|Y}(x_i|x_j)$$

to represent the probabilities

$$P(X = x_i), \quad P(X = x_i, Y = y_j) \quad \text{and} \quad P(X = x_i|Y = y_j)$$

Algorithm 1: Generate a random sample from a discrete probability distribution

Require: Probability of each outcome $p(x_i) = p_i$ and T , the number of samples to generate
Ensure: Generate a random sample $\tilde{x}_1, \dots, \tilde{x}_T \sim p(\cdot)$

```

for  $t = 1, \dots, T$  do
    Generate  $u$  as a random number in the unit interval
    Select  $k$  as the highest value such that  $\sum_{i=1}^{k-1} p_i \leq u$ 
    Set  $\tilde{x}_t = x_k$ 
end for

```

Summary 5.2.1: Rules of probability, discrete version

Consider three stochastic variables X, Y , and Z and suppose x_i, y_j and z_j are three *numbers* representing values taken by each variable. Then

$$p(x_i|y_j) \equiv P(X = x_i|Y = y_j) \equiv P(X_{x_i}|Y_{y_j})$$

represents the probability that X takes value x_i given that Y takes value y_j . In this notation, the sum/product rule is

$$\text{The sum rule:} \quad \sum_{i=1}^{\infty} p(x_i|z_j) = 1 \quad (5.22a)$$

$$\text{The product rule:} \quad p(x_i, y_j|z_k) = p(x_i|y_j, z_k)p(y_j|z_k) \quad (5.22b)$$

As important special cases, we mention Bayes' theorem and marginalization:

$$p(y_j|x_i, z_k) = \frac{p(x_i|y_j, z_k)p(y_j|z_k)}{\sum_{j'=1}^{\infty} p(x_i|y_{j'}, z_k)p(y_{j'}|z_k)}, \quad p(x_i|z_k) = \sum_{j=1}^{\infty} p(x_i|y_j, z_k)p(y_j|z_k).$$

Note in all these rules, z_k may be omitted provided it is done on both sides of the equality sign.

5.2.2 Generating random numbers★

Often, we will talk about a sample of random numbers generated from a probability distribution. Suppose a random variable X have N possible outcomes x_1, \dots, x_N , and the probability of each outcome is $p(X = x_i) = p_i$. We can imagine each p_i is a small stick of length p_i meters, and if we place the N sticks next to each other they have a combined length of 1 meter. If we then pick a random point uniformly within this large stick, it will select stick i with probability p_i , and if we do this T times we get our random sample, commonly written using the tilde-symbol:

$$\tilde{x}_1, \dots, \tilde{x}_T \sim p_X(\cdot).$$

Where obviously \tilde{x}_t is equal to one of the possible outcomes, x_1, \dots, x_N . A concrete implementation of the procedure can be found in algorithm 1 and an example using the silly die in fig. 5.4.

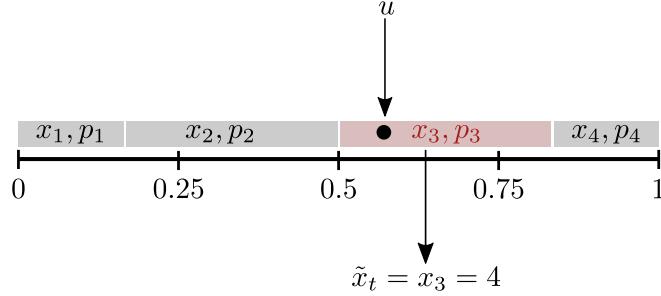


Fig. 5.4. Considering each bar in the histogram as a stick, we can generate a random sample \tilde{x}_t from the silly die (a roll) using a random uniform number $u \in [0, 1]$.

5.2.3 Expectations, mean and variance

If we roll a die a large number of times and compute the average, the average will eventually get arbitrarily close to the *mean* of a die roll which is

$$\frac{1 + 2 + 3 + 4 + 5 + 6}{6} = \frac{7}{2}$$

This procedure can be generalized as follows: Suppose X is a discrete random variable taking the possible values x_1, \dots, x_n and f is an arbitrary function. The expectation of f is then defined as:

$$\text{Expectation: } \mathbb{E}[f] = \sum_{i=1}^N f(x_i)p(x_i). \quad (5.23)$$

A useful intuition about the expectation is that if we let

$$\tilde{x}_1, \dots, \tilde{x}_T \sim p_X$$

be a random sample generated from p length T (see section 5.2.2), the simple average will approach the expectation when T becomes large enough just as the case of the die:

$$\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T \tilde{f}(x_t) = \mathbb{E}[f].$$

Two expectations are of particular importance namely the mean and variance. These can be obtained by setting $f(x) = x$ and $f(x) = (x - \mu)^2$ where μ is the mean of x . In particular we write:

$$\text{mean: } \mathbb{E}[x] = \sum_{i=1}^N x_i p(x_i), \quad \text{Variance: } \text{Var}[x] = \sum_{i=1}^N (x_i - \mathbb{E}[x])^2 p(x_i). \quad (5.24)$$

We have briefly illustrated the mean/variance definitions above in Example 5.2.1. Note that since the expectation can be generalized to continuous variables, and the rules for expectations are the same in both cases, useful identities will be given later in section 6.2.

Example 5.2.1: Mean and variance

These definitions may appear somewhat abstract and so they are worth illustrating with two examples. First, suppose all outcomes are equally probable such that $p(x_i) = \frac{1}{N}$. In this case:

$$\mathbb{E}[x] = \frac{1}{N} \sum_{i=1}^N x_i, \quad \text{Variance: } \text{Var}[x] = \frac{1}{N} \sum_{i=1}^N (x_i - \mathbb{E}[x])^2.$$

As another example, consider the *silly die* from eq. (5.17) and recall the probability of each outcome was:

$$p(X = -2) = p(X = 10) = \frac{1}{6}, \quad p(X = 1) = p(X = 4) = \frac{1}{3}$$

Using the definitions we can compute the mean and variance of the silly die as:

$$\begin{aligned} \mathbb{E}[x] &= \frac{1}{6}(-2) + \frac{1}{3}1 + \frac{1}{3}4 + \frac{1}{6}10 = 3 \\ \text{Var}[x] &= \frac{1}{6}(-2 - 3)^2 + \frac{1}{3}(1 - 3)^2 + \frac{1}{3}(4 - 3)^2 + \frac{1}{6}(10 - 3)^2 = 14. \end{aligned}$$

5.3 Independence and conditional independence

Consider three random variables X, Y and Z and denote their values by x_i, y_j and z_k respectively. Independence and conditional independence is then the mathematical relationships:

$$\text{Independent: } p(x_i, y_j) = p(x_i)p(y_j) \quad (5.25a)$$

$$\text{Conditionally independent given } z_k : \quad p(x_i, y_j | z_k) = p(x_i | z_k)p(y_j | z_k) \quad (5.25b)$$

Which must be true for all i and j . In case of conditional independence, if the relationship hold for a particular z_k we say they are independent given $Z = z_k$, and if it holds for all z_k we say they are independent given Z . If two variables are not (conditionally) independent, they are said to be (conditionally) *dependent*.

Conditional independence will later play an important role in structuring our machine-learning models and will usually be based on assumptions about the data-generating mechanism. Note that a reader should take great care not to assume that independence implies conditional independence or visa-versa as Example 5.3.1 illustrates.

Example 5.3.1: Independence

Since probabilities has to do with knowledge, conditioning on something can render seemingly independent events dependent and visa-versa.

Independence therefore does not imply conditional independence: Consider two events corresponding to the number of children of Bob and Maria; these two events can be assumed to be independent:

$$p(\text{Bob has } i \text{ children, Maria has } j \text{ children}) = p(\text{Bob has } i \text{ children})p(\text{Maria has } j \text{ children})$$

But if we condition on the third variable Z , Bob and Maria are married, the number of children each of them have will be highly correlated and therefore:

$$p(i, j | \text{Married} = \text{True}) \neq p(i | \text{Married} = \text{True})p(j | \text{Married} = \text{True}).$$

Conditional independence does not imply independence: Alternatively, ones mathematical skill and height are two highly dependent variables, because young children are usually small and poor at math. But if we condition on age, it is reasonable to think they are independent.

5.4 The Bernoulli, categorical and binomial distributions

As we saw in section 5.1.6, one idea for *obtaining* probabilities is simply to estimate them from the data matrix \mathbf{X} . For instance, suppose $M = 2$ and the two columns correspond to random variables X_1, X_2 and denote their possible values by $x_k^{(1)}, x_k^{(2)}$ respectively, the *empirical estimate* is then

$$\tilde{P}(X_1 = x_1^{(1)}, X_2 = x_1^{(2)}) = \frac{\left\{ \text{Number of rows } i \text{ where } X_{1i} = x_1^{(1)} \text{ and } X_{2i} = x_2^{(2)} \right\}}{N}. \quad (5.26)$$

While this approach is useful in some cases, it has at least three problems:

- For all but the most trivial datasets, it is not feasible to compute/store this many numbers
- Even if we try, the estimates are likely to be very poor
- More importantly, the whole *idea* of learning is the dataset can be summarized by a few, well-chosen parameters.

Probabilistic models, that is, models that depends on parameters, are a solution to these problems. Obviously, how to build these models will be a subject we return to several times in later chapters, however in this chapter we will be concerned with introducing a few building blocks which we will use many times over when constructing more elaborate models.

5.4.1 The Bernoulli distribution

Consider a setting where we consider a single, binary variable b which can be either false, $b = 0$, or true, $b = 1$.

The prototypical example of a binary event is a coin flip where $b = 0$ denote the event the coin landed tails and $b = 1$ the event the coin landed heads, but the setup applies to all simple

classification problems with two outcomes, for instance we could denote the event a treatment cures a patient such that $b = 0$ if the patient is not cured and $b = 1$ if the patient is cured. The Bernoulli distribution is the assumption the probability that $b = 0$ or $b = 1$ depends on a number $0 \leq \theta \leq 1$ as:

$$\text{Bernoulli distribution: } p(b|\theta) = \theta^b(1-\theta)^{1-b}.$$

It is worth going over this in some detail. The left hand side says that *given* we know θ , then the probability of b (which has two outcomes) is the expression on the right. Notice that:

$$\begin{aligned} p(b=0|\theta) &= \theta^0(1-\theta)^{1-0} = 1-\theta \\ p(b=1|\theta) &= \theta^1(1-\theta)^{1-1} = \theta \end{aligned}$$

and therefore, we can interpret θ as simply being the probability $b = 1$. As an example, we can compute the mean and variance of the Bernoulli distribution:

$$\mathbb{E}[b] = \sum_{b=0}^1 bp(b|\theta) = 0 \times (1-\theta) + 1 \times \theta = \theta, \quad (5.27a)$$

$$\text{Var}[b] = (0-\theta)^2 \times (1-\theta) + (1-\theta)^2 \times \theta = \theta(1-\theta). \quad (5.27b)$$

A notational problem we can anticipate is that when we use p to denote all sorts of probability densities, it can become difficult to figure out exactly which we refer to. It is therefore common to introduce special notation for familiar densities. We will therefore sometimes write:

$$\text{Bernouilli}(b|\theta) = p(b|\theta)$$

to signify the Bernoulli distribution.

5.4.2 The categorical distribution

We will often consider situations with more than two outcomes, for instance a roll of a die (six outcomes), multiple possible diagnosis (as many outcomes as there are diagnoses) or which category an image belongs to (as many outcomes as there are image categories). Suppose there are M possible outcomes, and let $y = 1, \dots, M$ denote the outcome of the experiment, the categorical distribution is then defined as:

$$\text{Categorical distribution: } \text{Catagorical}(y|\boldsymbol{\theta}) = \theta_1^{\delta_{y,1}} \theta_2^{\delta_{y,2}} \times \dots \times \theta_K^{\delta_{y,K}} \quad (5.28)$$

where $\sum_{k=1}^K \theta_k = 1$ and $\theta_k \geq 0$. We have here used that $\delta_{y,k} = 1$ if $y = k$ and zero otherwise. This notation is slightly cumbersome, but note once again it simply means that the chance $y = k$ is

$$p(y|\boldsymbol{\theta}) = \theta_k$$

To simplify the notation, it is common to 1-out-of- K encode the variable k . Specifically, we introduce K new variables, $z_i = \delta_{y,i}$, such that

$$y = k \Leftrightarrow \begin{bmatrix} z_1 \\ \vdots \\ z_k \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

in which case we can write the categorical distribution as:

$$\text{Categorical distribution: } \text{Catagorical}(y|\boldsymbol{\theta}) = \prod_{k=1}^K \theta_k^{z_k} \quad (5.29)$$

5.4.3 Parameter transformations

The parameter θ in the Bernoulli distribution is easy to interpret as the probability of $b = 1$.

A *disadvantage* is θ belongs to the interval $[0, 1]$ and therefore, when we apply numerical methods this constraint has to be taken into account. A way around this problem is to replace θ with a function of another parameter x , $\theta = h(x)$. As long as the domain of h is the unit interval, this substitution lead to a well-defined probability density. The most common choice is the logistic function $\theta = \sigma(x) = \frac{1}{1+e^{-x}}$ in which case we can write the density of the Bernoulli distribution as:

$$p(b|x) = (1 - \sigma(x))^{1-b} \sigma(x)^b \quad (5.30)$$

and now x can be any real number. A similar trick can be applied to the categorical function. In this case we introduce K new parameters, x_1, \dots, x_K and re-define:

$$\theta_k = \frac{e^{x_k}}{\sum_{c=1}^K e^{x_c}}$$

this operation, when applied to the entire parameter vector, will be written as

$$\boldsymbol{\theta} = \text{softmax}(\boldsymbol{x}) = \left[\frac{e^{x_1}}{\sum_{c=1}^K e^{x_c}} \cdots \frac{e^{x_K}}{\sum_{c=1}^K e^{x_c}} \right]^T$$

and therefore

$$p(y|\boldsymbol{x}) = p(y|\boldsymbol{\theta} = \text{softmax}(\boldsymbol{x}))$$

where the right-hand side is just the ordinary categorical distribution. While this is today the most common way to re-parameterize the categorical distribution, the reader should be aware we can alternatively choose to just use $K - 1$ parameters, x'_1, \dots, x'_{K-1} and define:

$$\theta_k = \begin{cases} \frac{e^{x'_k}}{1 + \sum_{c=1}^{K-1} e^{x'_c}} & \text{if } k \leq K - 1 \\ \frac{1}{1 + \sum_{c=1}^{K-1} e^{x'_c}} & \text{if } k = K. \end{cases} \quad (5.31)$$

5.4.4 Repeated events

Suppose we flip the coin from before N times or, alternatively, we administer the treatment to N patients and observe the outcome. In this case we have N binary (Bernoulli) events b_1, \dots, b_N , corresponding to the outcome of each event.

When the events are based on the same process, it is reasonable to assume each event occurs with a probability θ , but when we know θ the outcome of different coin flips or patients are independent. In other words, the outcomes are *conditionally independent* given θ . If we then simply apply the definition of conditional independence (eq. (5.25b)) the probability of an entire sequence becomes:

$$\begin{aligned} p(b_1, \dots, b_N | \theta) &= \prod_{i=1}^N p(b_i | \theta) = \prod_{i=1}^N \theta^{b_i} (1-\theta)^{1-b_i} = \theta^{\sum_{i=1}^N b_i} (1-\theta)^{N-\sum_{i=1}^N b_i} \\ &= \theta^m (1-\theta)^{N-m}, \quad m = b_1 + b_2 + \dots + b_N. \end{aligned} \quad (5.32)$$

From this, we learn the important factor the probability of a particular sequence of outcomes only depend on the length N and positive outcomes m . For convenience, we will refer to the sequence of flips using the symbol $\mathbf{b} = [b_1 \ b_2 \ \dots \ b_N]$ and write eq. (5.32) as $p(\mathbf{b}|\theta) = \theta^m (1-\theta)^{N-m}$.

5.4.5 A learning principle: Maximum likelihood

Continuing the above example, the probability of the N flips \mathbf{b} was:

$$p(\mathbf{b}|\theta) = \theta^m (1-\theta)^{N-m}. \quad (5.33)$$

When we vary θ , this probability changes. The exact way to think about this relationship is that various values of θ makes the occurrence of the data more or less plausible.

An idea is therefore to select θ as the value that maximizes the probability (plausibility) of the data, and this principle is so often invoked the probability of the data given the parameters is called the *likelihood* and denoted by the function \mathcal{L} :

$$\mathcal{L}(\theta) = p(\mathbf{b}|\theta).$$

When we try to implement this idea, we often run into a practical issue, namely that probabilities will often be very small. For instance, suppose $\theta = 0.9$ and $N = 1000$. Then if $m = 900$, we have

$$p(\mathbf{b}|\theta) \approx 6.6 \cdot 10^{-142}.$$

For this reason it is very common to work with *logarithms* of probabilities, called the log likelihood. In our case the log likelihood is:

$$\log \mathcal{L}(\theta) = \log p(\mathbf{b}|\theta) = m \log \theta + (N-m) \log(1-\theta)$$

Maximizing the probability is the same as maximizing the likelihood, and learning θ by maximizing the likelihood is known as the *maximum likelihood* principle.

In fig. 5.5 we have illustrated the likelihood function $\mathcal{L}(\theta) = p(\mathbf{b}|\theta)$ for different values of θ , N and m . We see that as N increases, a smaller range of values of θ makes the data remotely possible. The keen eyed reader will also observe the value of θ that maximizes the probability, called the *maximum likelihood estimate* and referred to as θ^* , seems to be $\frac{m}{N}$, i.e. the empirical frequency. We can readily verify this is indeed the case by taking the derivative of the log likelihood (i.e., as the logarithm is a monotonic function the optimum of the likelihood does not change when taking the logarithm) and setting this derivative equal to zero:

$$0 = \frac{d \log \mathcal{L}(\theta)}{d\theta} = \frac{m}{\theta} - \frac{N-m}{1-\theta}, \quad \text{implies: } \theta^* = \frac{m}{N}.$$

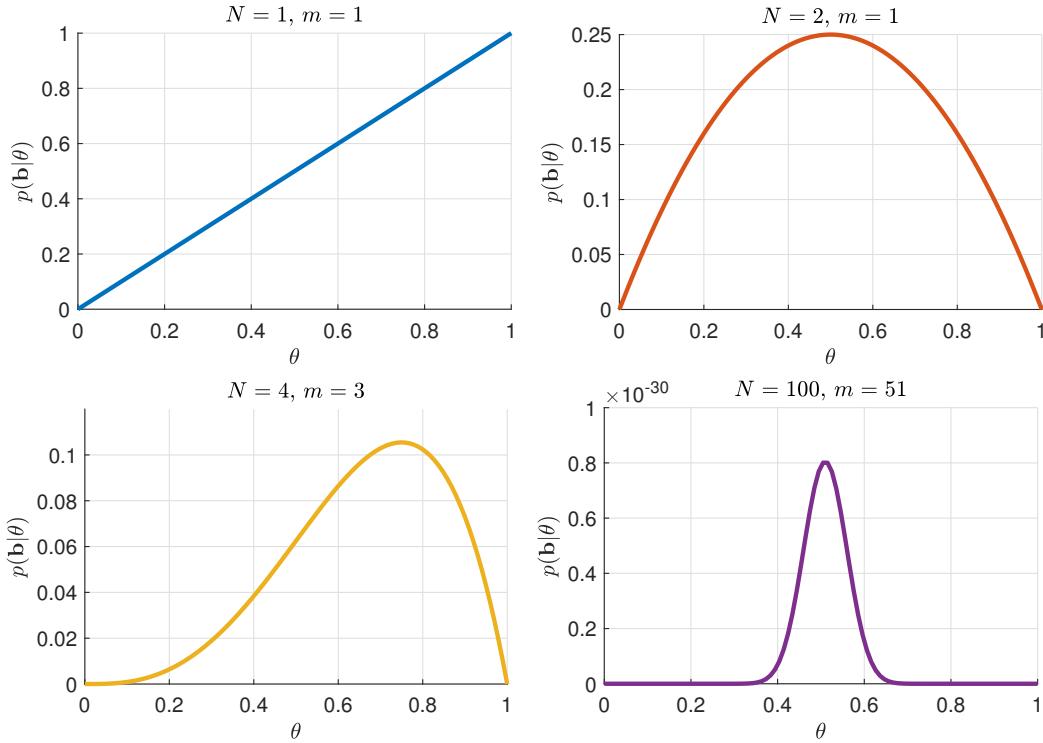


Fig. 5.5. Examples of the likelihood $\mathcal{L}(\theta) = p(\mathbf{b}|\theta)$ from eq. (5.33) for different numbers of flips N and different number of heads m . The top left figure corresponds to *heads*, the top-right to *heads, tails*, bottom left to *heads, tails, heads* and bottom right to $N = 100$ flips where $m = 51$ came up heads. Notice the dramatic change of scale on the y -axis.

Summary 5.4.1: Common notation

When Bayes' theorem is used as a learning principle, for instance as in the Monty-Hall example, it is common to give the different expressions names. Specifically, suppose x play the role of data, y the role of a hypothesis, and we apply Bayes' theorem to find the probability of our hypothesis y given data x :

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

It is then common to use the following names for the terms:

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}.$$

Even though these names may make it appear something more complicated is going on, each of the terms are just familiar probability densities.

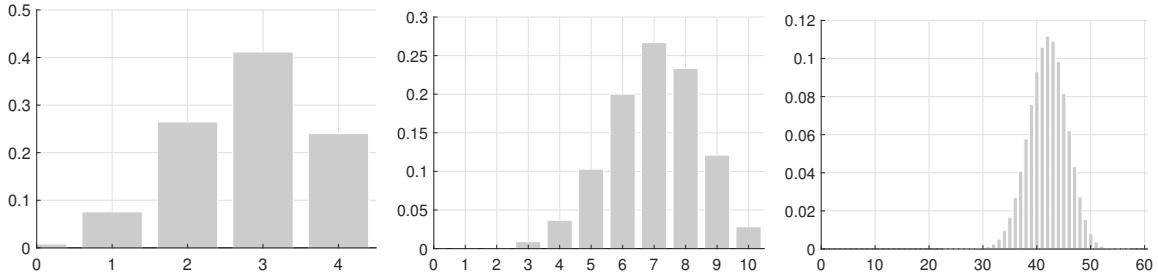


Fig. 5.6. The binomial distribution $p(m|N, \theta)$ for $N = 4, 10, 60$ and $\theta = 0.7$.

5.4.6 The binomial distribution★

The binomial distribution will play a minor role in this course, and a reader may choose to skip this section. Briefly, suppose once more we have a sequence b_1, \dots, b_N of Bernoulli events. As we have seen many times, their probability is

$$p(b|\theta) = \theta^m(1 - \theta)^{N-m}.$$

However, suppose we wish to compute $p(m|\theta)$, namely the probability of observing m positive outcomes in a sequence of N Bernoulli events that each occur with probability θ . This probability can be computed using the sum rule:

$$\begin{aligned} p(m|N, \theta) &= \{\text{Sum of the probability of all sequences of length } N \text{ with } m \text{ positive outcomes}\} \\ &= \left\{ \begin{array}{l} \text{Number of sequences of length } N \\ \text{with } m \text{ positive outcomes} \end{array} \right\} \times \left\{ \begin{array}{l} \text{Probability of a sequence} \\ \text{with } m \text{ positive outcomes} \end{array} \right\} \\ &= \left\{ \begin{array}{l} \text{Number of sequences of length } N \\ \text{with } m \text{ positive outcomes} \end{array} \right\} \times \theta^m(1 - \theta)^{N-m} \end{aligned} \quad (5.34)$$

Computing the quantity in the bracket requires a combinatorial argument, but it can be shown to be equal to $\binom{N}{m} = \frac{N!}{m!(N-m)!}$ where $n! = n(n-1) \times \dots \times 1$. We therefore have:

$$\text{Binomial distribution: } p(m|N, \theta) = \binom{N}{m} \theta^m(1 - \theta)^{N-m}, \quad (5.35)$$

In fig. 5.6 we have shown the probability density function of m computed from eq. (5.35) when $\theta = 0.7$ and $N = 4, 10$ and 60 .

5.5 Information Theory★

Shannon's theory of information attempts to describe the information content in a random variable [Shannon, 1948]. While information theory is an important topic in machine learning, it will play a minor role for the subjects we will discuss, and a reader may choose to simply use Box 5.5.1 as a reference and skip the following justifications. With these warnings out of the way, the measures which we will introduce are:

Information How much information is contained in observing a single outcome x_i of a random variable?

Entropy The complexity of the distribution of a random variable, measured in *bits*

Mutual information How much information is *shared* between two random variables, measured in bits. Put differently, if we learn the state of one random variable, how many bits of information does this tell us about the other

Normalized mutual information Same as mutual information, but re-scaled by the information content in the two random variables.

5.5.1 Measuring information

How do we measure information? First, we must recognize that *information* is a word that can take many meanings, and information theory is a theory which explores one, particular, meaning of the word.

According to information theory, we consider the information contained in repeated, random events. For instance, we can imagine a weather-station which each day sends home a weather-report consisting of one of the following three events:

E_1 : The weather is stormy.

E_2 : The weather is windy but not exceedingly so.

E_3 : The weather is fair.

What we specifically wish to quantify is the *amount of information* the receivers of the weather-report obtains when they learn for instance E_3 occurred.

Since the events are discrete, they happen with some probability, which we will write in the usual way:

$$P(E_1) = p_1, \quad P(E_2) = p_2, \quad P(E_3) = p_3.$$

Let's suppose there exists such a measure of information which we will write as

$I(E)$: The information obtained by hearing E occurred.

Obviously, this measure of information can't take into account the specifics of what the event was because that is a convention, however, it *should* take into account what *probability* the event occurred with: There is little information in knowing about things that are very likely to happen, but a lot of information in very infrequent events. For instance, if it is stormy nearly every day, learning it is stormy again today does not tell us much, but on the other hand, if we learn something surprising, like the weather was good, that contains a lot of information.

Accordingly, our measure of information is a function of the probability of E :

$$I(E) = I(P(E)) = I(p), \quad \text{where } p = P(E).$$

Let's consider two important examples: Suppose we know an event will occur ($P(E) = 1$). The information we obtain by hearing that E occurred is then zero, $I(E) = I(1) = 0$.

Next, consider two events E_1 and E_2 which are *independent*, meaning:

$$P(E_1 E_2) = P(E_1)P(E_2) = p_1 p_2.$$

However, since the events are independent, the information in knowing both events happened should be the sum of the information of both events taken independently. That is, we conclude our measure I should obey:

$$I(p_1 p_2) = I(p_1) + I(p_2),$$

and in the particular case where $p_1 = p_2 = p$ (for instance, E_1 and E_2 might be independent flips of the same coin) then we have

$$I(p^2) = I(p) + I(p) = 2I(p).$$

In general, if we consider n events we obtain:

$$I(p^n) = I(p \cdot p^{n-1}) = I(p) + I(p^{n-1}) = nI(p). \quad (5.36)$$

To proceed we will use a small trick. Notice for any integer $m \geq 0$ and probability p we can write $p = p^{\frac{m}{m}} = \left(p^{\frac{1}{m}}\right)^m$. Using eq. (5.36) we get:

$$I(p) = I\left(\left(p^{\frac{1}{m}}\right)^m\right) = mI\left(p^{\frac{1}{m}}\right), \quad (5.37)$$

or more conveniently this can be written as $I\left(p^{\frac{1}{m}}\right) = \frac{1}{m}I(p)$. Let's then consider any rational number $r = \frac{n}{m}$ where n, m are integers. Combining eq. (5.37) and eq. (5.36) we obtain:

$$I\left(p^{\frac{n}{m}}\right) = I\left(\left(p^{\frac{1}{m}}\right)^n\right) = nI\left(p^{\frac{1}{m}}\right) = \frac{n}{m}I(p). \quad (5.38)$$

If we assume the measure of information is continuous, it therefore holds for general positive x that $I(p^x) = xI(p)$. If we differentiate according to x we get:

$$I'(p^x)p^x \log p = I(p).$$

Since this is true for any x it follows $I'(p^x) = A\frac{1}{p^x}$ for an unknown constant A . From this we conclude: $I'(z) = A\frac{1}{z}$ which implies

$$I(z) = A \log(z) + B$$

where B is a constant. However since $I(1) = 0$ we can conclude $B = 0$. It is convenient to have $I(z) \geq 0$, and we therefore select $A < 0$, however, any number will in principle do. A particular convenient choice is $A = -\frac{1}{\log 2}$ which ensures information is measured in *bits*³ We have now derived that the information content of an event that occurs with probability p is:

$$I(p) = -\frac{1}{\log 2} \log p + 0 = -\log_2(p) \quad (5.39)$$

where $\log_2(p)$ is the base-2 logarithm. For instance, suppose you flip a single, unbiased coin. The amount of information obtained is then $I(\frac{1}{2}) = -\log_2 \frac{1}{2} = 1$ bit and the amount of information in N such coins is $\log_2 \frac{1}{2^N} = n$ bits.

³ Alternatively, if we choose $A = -1$ the information is said to be measured in *nats*.

5.5.2 Entropy

Suppose we consider a source of random events, for instance a die (the random events are which of the six sides face upwards in a roll) or the next character in a newspaper article. In case of the die there are 6 events, each occurring with probability p_1, \dots, p_6 , and for the next letter in a sentence there are 26 outcomes (letters in the alphabet) each occurring with probability p_1, \dots, p_{26} . Such a source is quantified by the *average* information content it produces which is simply the average of the information

$$H[p_1, \dots, p_n] = \sum_{i=1}^n p_i I(p_i) = - \sum_{i=1}^n p_i \log p_i$$

This quantity is known as the *entropy* and is a measure of the amount of uncertainty associated with events produced from a given distribution⁴. Since we don't want to write the probabilities every time, let's suppose the probabilities relates to a random quantity $k = 1, \dots, K$ and that we write

$$p_k(1), p_k(2), \dots, p_k(K)$$

for the probability of each event $k = 1, \dots, K$. We will then write $H[p_k]$ for the entropy of the random variable k defined as

$$H[p_k] = - \sum_{k=1}^K p_k(k) = - \sum_{k=1}^K p_k(k) \log p_k(k)$$

Example 5.5.1: Example 1: Entropy of a coin flip

The entropy of a single (biased) coin $c = 0, 1$ is given by writing the probability of heads and tails as $p_c(0), p_c(1) = 1 - p_c(0)$. The entropy is then:

$$H[p_c] = -p \log p - (1-p) \log(1-p), \quad \text{where } p = p_c(0).$$

This entropy is zero when $p = 0$ or $p = 1$ (i.e. we know the outcome of flipping the coin beforehand) and maximal when $p = \frac{1}{2}$. In this case the entropy is $H[p_c] = \log 2$ or $H[p_c] = \log_2 2 = 1$ if we use the base-two logarithm. Intuitively, this is saying that a random, binary event where we are completely uncertain about the outcome beforehand contains 1 bit of information.

This definition easily allows us to consider the entropy of multiple variables. Suppose we have a density of two quantities $k = 1, \dots, K$ and $m = 1, \dots, M$. We define their joint density as:

$$p_{km}(k, m), \quad \text{for } k = 1, \dots, K \text{ and } m = 1, \dots, M$$

We stress that p_{km} is just the regular old probability and the subscript km are only there to make referencing easier later. For instance, the marginal density of k, m is as usual given by the sum rule:

⁴ Two comments: Firstly, we are no longer using the base 2 logarithm but the regular logarithm, however, as we saw in the previous section, this is only a matter of scale, and the natural logarithm is somewhat easier to write. Secondly, if an event occurs with $p = 0$, we use the convention $0 \times \log 0 = 0$

$$p_k(k) = \sum_{m=1}^M p_{km}(k, m) \quad \text{and} \quad p_m(m) = \sum_{k=1}^K p_{km}(k, m).$$

In the case of two variables the entropy is simply:

$$H[p_{km}] = - \sum_{k=1}^K \sum_{m=1}^M p_{km}(k, m) \log p_{km}(k, m).$$

The following two examples illustrates how the entropy can be calculated for distributions of two variables

Example 5.5.2: Example 2: Entropy of two variables

Since it will be relevant later, let's consider an example with two variables. Suppose $M = K = 2$ and

$$p_{km}(1, 1) = 0.4, \quad p_{km}(1, 2) = 0.2, \quad p_{km}(2, 1) = 0.1 \quad \text{and} \quad p_{km}(2, 2) = 0.3$$

the entropy is then:

$$\begin{aligned} H[p_{km}] &= - \sum_{k=1}^K \sum_{m=1}^M p_{km}(k, m) \log p_{km}(k, m) \\ &= -0.4 \log 0.4 - 0.2 \log 0.2 - 0.1 \log 0.1 - 0.3 \log 0.3 \\ &\approx 1.28. \end{aligned}$$

For K outcomes, the entropy is largest when all events has the same probability, and in general the entropy becomes lower when one particular outcome has high probability. This is simply saying that the uncertainty in a random phenomenon is smaller when we know one outcome is very likely to occur.

5.5.3 Mutual information

Recall two variables X and Y are independent if

$$p(X = x_i, Y = y_i) = p(X = x_i)p(Y = y_i)$$

and otherwise dependent. A way to view mutual information is as a way to quantify *how* dependent two variables are; if the two variables are independent, the mutual information is 0, and otherwise greater than zero. More specifically, Mutual information tell us how many *bits* of information we learn about X if we know Y .

Specifically, given a probability assignment $p_{km}(k, m)$ the *Mutual information* is defined using the entropy as:

$$\text{MI}[p_{km}] = H[p_k] + H[p_m] - H[p_{km}]$$

A loose justification of this definition is that it computed the sum of information in the two quantities, and then subtract a term which is low if the two variables are highly redundant. That is, the mutual information becomes *low* if the two variables are highly redundant, and otherwise it will be high. Note the mutual information can be re-written as:

$$\text{MI}[p_{km}] = \sum_{k=1}^K \sum_{m=1}^M p_{km}(k, m) \log \frac{p_{km}(k, m)}{p_k(k)p_m(m)}.$$

To get some more intuition, we will consider two extreme cases. First, suppose k and m are completely unrelated such that $p_{km}(k, m) = p_k(k)p_m(m)$. In this case:

$$\text{MI}[p_{km}] = \sum_{k=1}^K \sum_{m=1}^M p_k(k)p_m(m) \log \frac{p_k(k)p_m(m)}{p_k(k)p_m(m)} = \sum_{k=1}^K \sum_{m=1}^M p_k(k)p_m(m) \log 1 = 0$$

This makes sense intuitively: If k and m are not informative about each other, the mutual information (information shared between them) should be zero. On the other hand assume k actually determines m , for instance that the two variables are the same quantity measured twice. In this case $p_{km}(k, m) = p_k(k) = p_m(m)$ and so:

$$\text{MI}[p_{km}] = \sum_{k=1}^K \sum_{m=1}^M p_k(k) \log \frac{p_k(k)}{p_k(k)p_m(m)} = \sum_{m=1}^M p_k(k) \log \frac{1}{p_m(m)} = H[p_k], \quad (5.40)$$

which also makes sense from an intuitive standpoint: If k tells us what m is (for instance if they are the same), then the mutual information is all the information in k or $H[p_k]$.

5.5.4 Normalized mutual information

Sometimes, the mutual information is normalized to lie on a scale from 0 to 1 to make it easier to interpret. While there are several ways of doing so we will here consider the method of Strehl and Ghosh [2002] which is reminiscent of the definition of the correlation we saw earlier in eq. (4.7) in chapter 4

$$\text{NMI}[p_{km}] = \frac{\text{MI}[p_{km}]}{\sqrt{H[p_k]}\sqrt{H[p_m]}}. \quad (5.41)$$

We see now that if k determines m and visa-versa we obtain:

$$\text{NMI}[p_{km}] = \frac{\text{MI}[p_{km}]}{\sqrt{H[p_k]}\sqrt{H[p_m]}} = \frac{H[p_m]}{H[p_m]} = 1.$$

Therefore, an NMI of 0 means the variables are independent, and 1 means they are maximally dependent.

Method 5.5.1: Information theory

The *information* contained in an event which occur with probability p is

$$I(p) = -\log(p)$$

Next, consider two random quantities m and n , such that k can take values $k = 1, \dots, K$ and m can take values $m = 1, \dots, M$. We assume we know the joint distribution of n, m , which is the $K \times M$ matrix:

$$p_{km}(k, m), \quad \text{for } k = 1, \dots, K \text{ and } m = 1, \dots, M$$

Based on this matrix, we can define the marginal distributions as the K and M -dimensional vectors:

$$p_k(k) = \sum_{m=1}^M p_{km}(k, m), \quad p_m(m) = \sum_{k=1}^K p_{km}(k, m)$$

The *Entropy* in the 1 and 2d-case is then defines as:

$$H[p_k] = - \sum_{k=1}^K p_k(k) \log p_k(k). \quad H[p_{km}] = - \sum_{k=1}^K \sum_{m=1}^M p_{km}(k, m) \log p_{km}(k, m).$$

In both cases, it measures the *complexity* of p_k and p_{km} in *bits*. In addition, the *mutual information* and *normalized mutual information* is defined as:

$$\begin{aligned} \text{MI}[p_{km}] &= H[p_k] + H[p_m] - H[p_{km}] \\ \text{NMI}[p_{km}] &= \frac{\text{MI}[p_{km}]}{\sqrt{H[p_k]} \sqrt{H[p_m]}}. \end{aligned}$$

When we use the natural logarithm ($\log(x)$), The information, entropy, and mutual information are all measured in *nats*. If we instead use the base-2 logarithm $\log_2(x)$, or alternatively simply divide each quantity by $\log(2)$, the quantities are measured in *bits*.

6

Densities and models

So far we have considered the probability of binary events such as A , B , A_i and so on. When working with machine-learning models, input is usually defined as continuous numbers and so we have to work with the probability of continuous quantities. We do so by using *probability densities* defined on continuous numbers. These may appear so different from probabilities they signify a departure from the basic sum-and-product rules applied to binary events, however, we will here stress how densities, and the rules relating to densities, follow from the basic rules and therefore do not signify any new formalisms.

We will begin by providing an intuitive introduction to this connection, and subsequently discuss the most commonly used continuous density, the multivariate normal distribution, and the connection between continuous probabilities and learning. The end-result will be a general recipe for building machine-learning models we will return to later in the course.

6.1 Probability densities

Let us consider a simple example. Suppose we denote by r the amount of daily rainfall in Denmark. Clearly, r is random since the amount of rain varies for different days. However, the problem is that it does not strictly speaking make sense to talk about the probability r takes a specific value. After all, suppose we ask:

What is the probability there will be $r = 2.3\text{mm}$ of rain a given day?

We could just as well have asked for $r = 2.31$ or $r = 2.299\text{mm}$, and so a little thought reveals the probability there will be *exactly* $r = 2.3\text{mm}$ of rain must be zero. The way we overcome this is to rather ask

What is the probability there will be between 2 and 3mm of rain?.

This is a proper yes/no question that can be formulated by introducing a binary variable

$$A_{[a,b]} : \text{There will be between } a \text{ and } b \text{ mm of rain, i.e. } r \in [a, b],$$

and then simply write $P(A_{[2,3]})$ for the probability $r \in [a, b]$. Importantly, this probability is non-zero, and of the usual kind encountered in the previous chapter. In fig. 6.2 (left) we have tabulated

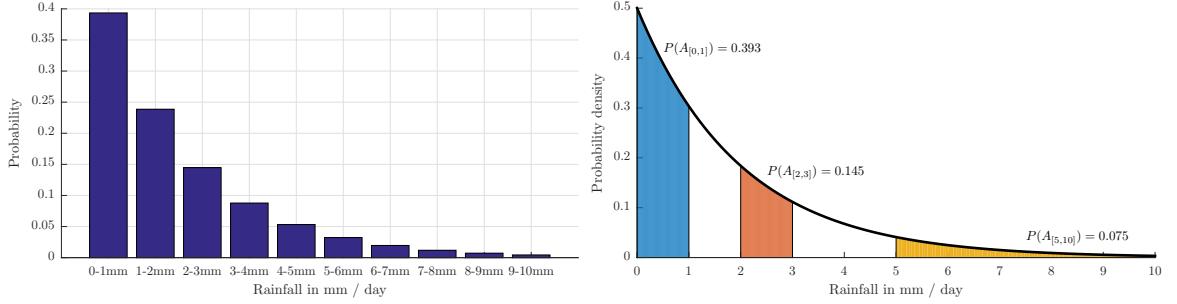


Fig. 6.1. Left: The probability of rainfall per day illustrated as a histogram. Each bar denotes the event that on a given day there are between $0 - 1\text{mm}$ of rain, $1 - 2\text{mm}$ of rain, $2 - 3\text{mm}$ of rain etc. These well-defined binary events can be estimated from historical rainfall records. Right: If we introduce a function $p(r)$, we can define the probability of the event $A_{[a,b]}$ (that there was between a and b mm of rainfall a given day) as $p(A_{[a,b]}) = \int_a^b p(r)dr$. The three colored regions thus correspond to three events, and each area corresponds to their probability.

different intervals of amounts of rainfall in Denmark and their respective probabilities. However, keeping track of rainfall using such a histogram is not very practical. After all, suppose someone ask for $P(A_{[2.5,3.5]})$? We can perhaps make a qualified guess at this variable by eye-balling neighboring bins in the histogram (a reasonable guess would be around 12%), however, we would like a convenient and exact way to represent *all* such binary variables. This is exactly the task a *probability density function* accomplishes. A probability density function is simply a function p that is non-negative and integrates to one. Using this function, we can then *represent* the probability of a particular event such as $A_{[a,b]}$ as the integral

$$P(A_{[a,b]}) = \int_a^b p(x)dx. \quad (6.1)$$

In fig. 6.1 (right) is shown the probability of the events $A_{[2,3]}$, $A_{[0,1]}$ and $A_{[5,\infty]}$ with their respective probabilities corresponding to the area under the density function.

6.1.1 Multiple continuous parameters

As we have seen, probability densities are simply functions that are useful as bookkeeping devices to make statements about continuous random variables, such as the amount of rainfall r . In this section, we will use this connection to, by means of the sum-and-product rule for *binary* variables, *derive* similar looking rules densities must therefore obey. We stress these new rules are not some new postulate of probability theory, but merely a consequence of the binary sum-and-product rules.

To do so, consider a very small interval of width dx , $A_{[x,x+dx]}$ (for instance $dx = 0.1$). Since p will be nearly flat assuming dx is small enough then (see fig. 6.2)

$$P(A_{[x,x+dx]}) \approx p(x)dx.$$

In general, suppose we have two variables x, y . In direct generalization of the 1d case, the probability that x, y both fall within some 2d subset D of \mathbb{R}^2 is then

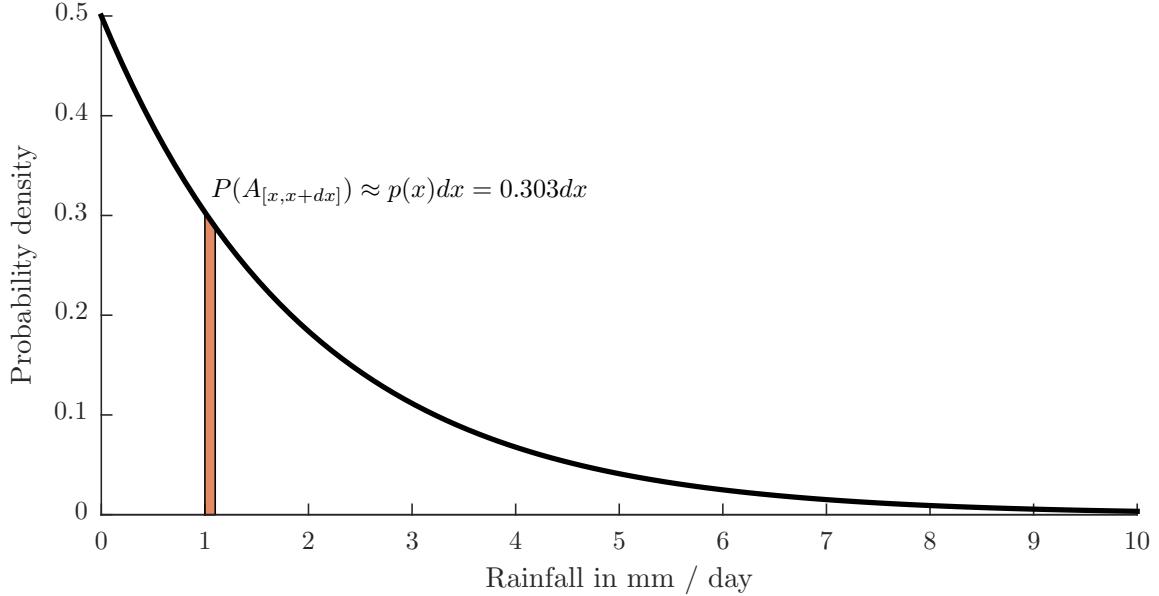


Fig. 6.2. Continuing further the rainfall example, rather than talking about the probability there will be exactly x mm of rain, we can talk about the probability there will be between x and $x+dx$ mm of rain. This can be approximated as $P(A_{[x,x+dx]}) \approx p(x)dx$ which becomes more and more exact when dx approaches 0.

$$P((x, y) \in D) = \int_{(x,y) \in D} p(x, y) dx dy \quad (6.2)$$

However similar to the 1d case, we can consider the event x lies in the interval $[x, x + dx]$ and y in the interval $[y, y + dy]$, $A_{[x,x+dx]}$ and $B_{[y,y+dy]}$, see fig. 6.3 where this corresponds to the red area. If we shut off our brain and apply the product rule:

$$P(A_{[x,x+dx]} B_{[y,y+dy]}) = P(B_{[y,y+dy]} | A_{[x,x+dx]}) P(A_{[x,x+dx]}) \quad (6.3)$$

However using that dx, dy are both small we can again approximate this as

$$P(A_{[x,x+dx]} B_{[y,y+dy]}) \approx dx dy p(x, y), \quad (6.4)$$

$$P(B_{[y,y+dy]} | A_{[x,x+dx]}) \approx dy p(y|x), \quad (6.5)$$

$$P(A_{[x,x+dx]}) \approx dx p(x), \quad (6.6)$$

where $p(y|x)$ is a new function of two parameters. If we plug these definitions into eq. (6.3) and divide both sides by $dxdy$ we obtain the more familiar form:

$$p(x, y) = p(y|x)p(x).$$

We say that $p(x, y)$ is the *joint* density of x, y and $p(y|x)$ is the *density* distribution of y given x . In general, we can define the sum and product rules for continuous densities:

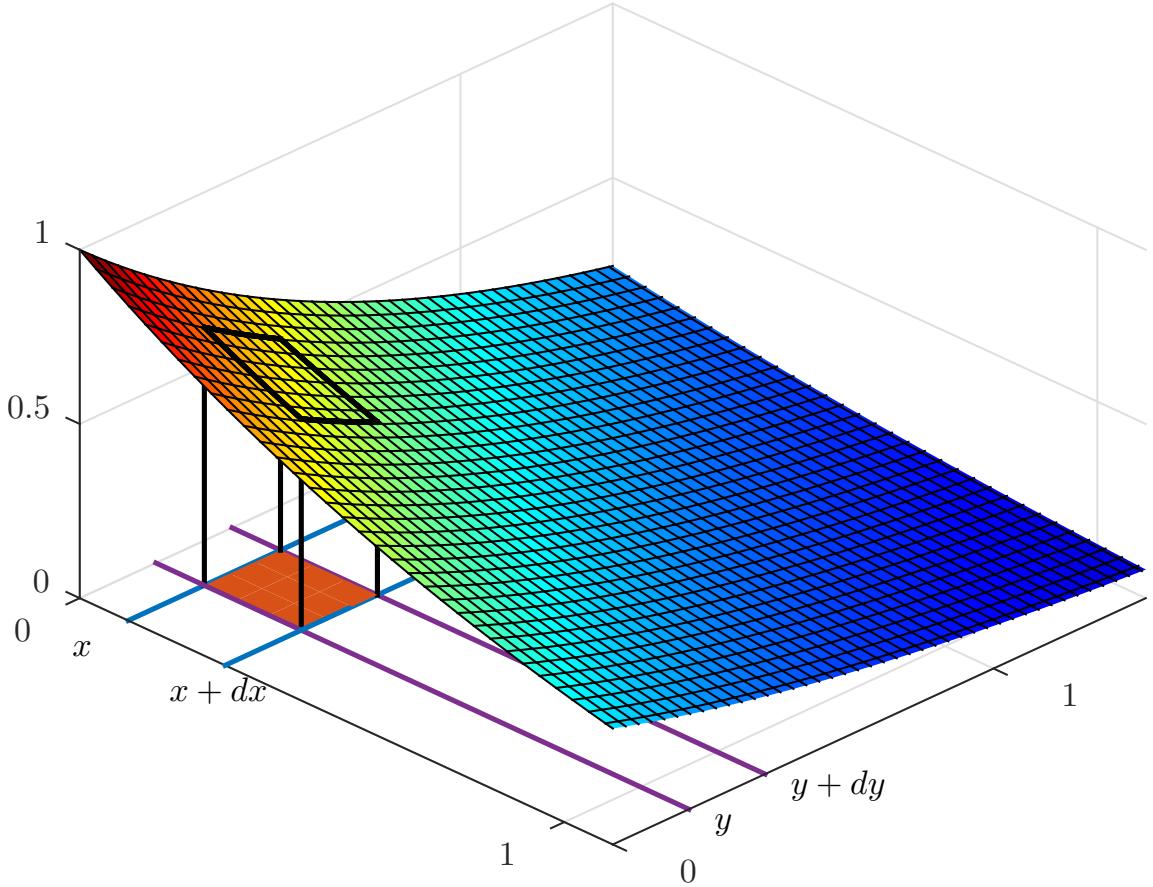


Fig. 6.3. Suppose we have a 2d density $p(x, y)$. For a subset $D \subset \mathbb{R}^2$ we can define the probability (x, y) lies in D as $p(A_D) = \int_{(x,y) \in D} p(x, y) dx dy$. For small but non-zero values of dx and dy , and the case where D is the rectangle $[x, x + dx] \times [y, y + dy]$ (the red area), the probability of D (the volume indicated by the black lines) can be approximated as $p(A_D) \approx p(x, y) dx dy$

$$\text{The sum rule:} \quad \int p(x|z) dx = 1, \quad (6.7)$$

$$\text{The product rule:} \quad p(x, y|z) = p(y|x, z)p(x|z), \quad (6.8)$$

where again we will often omit z for simplicity. Since the machine-learning models we are interested in use continuous variables we will in the coming sections and chapters mostly use these rules as applied to densities. However, it is worth stressing this is not because these rules are more fundamental or a departure from a simpler theory about binary probabilities: rather, they are consequences of the simple sum-and-product rules introduced at the very beginning of this chapter provided we choose to represent probabilities using densities.

Notice, x, y, z in the above can also be vectors or discrete variables in which case we only have to modify the integral in the sum rule to be either an integral over vectors or a sum over discrete

variables as we have already encountered. For completeness we also provide Bayes' theorem for continuous variables in summary box 6.1.1.

Summary 6.1.1: Rules of probability, continuous version

Consider three stochastic variables X , Y , and Z which are now considered to take *continuous* values x , y , and z respectively. Note these are just numbers. We assume the joint density is written as $p(x, y, z)$ (in this case, a function of three variables). The sum and product rule is:

$$\text{The sum rule:} \quad \int p(x|z)dx = 1 \quad (6.9a)$$

$$\text{The product rule:} \quad p(x, y|z) = p(x|y, z)p(y|z) \quad (6.9b)$$

As important special cases, we mention Bayes' theorem and marginalization:

$$p(y|x, z) = \frac{p(x|y, z)p(y|z)}{\int p(x|y', z)p(y'|z)dy'}, \quad p(x|z) = \int p(x|y, z)p(y|z)dy.$$

Finally, note that:

- These rules also hold for blocks of variables, for instance $\iiint p(x, y, z)dxdydz = 1$ and $p(x, y, z, v) = p(x, y|z, v)p(z, v)$
- The variable z may be omitted, for instance $p(x, y) = p(x|y)p(y)$
- We are allowed to mix discrete and continuous variables, for instance: $p(x, y, Z = z_k) = p(Z = z_k, x|y)p(y)$ and $\sum_{k=1}^{\infty} p(Z = z_k, x) = p(x)$
- Independence/conditional independence is defined exactly as in the continuous case

6.2 Expectations, mean and variance

Expectations, means and variances work just like discrete probabilities, except we replace the sum signs with integrals. Specifically, for any function f and random quantity x we have:

$$\mathbb{E}[f] = \int f(x)p(x)dx \quad (6.10)$$

provided p is the density of x . Once more, mean and variance can be obtained by setting $f(x) = x$ and $f(x) = (x - \mu)^2$ where μ is the mean of x . In particular we write:

$$\text{mean: } \mathbb{E}[x] = \int xp(x)dx, \quad \text{variance: } \text{Var}[x] = \int (x - \mathbb{E}[x])^2 p(x)dx \quad (6.11)$$

The rules governing expectations are the same in both cases. The reader is therefore invited to use either eq. (6.11) or eq. (5.24) to verify the following useful identities:

$$\mathbb{E}[ax + b] = a\mathbb{E}[x] + b, \quad \text{Var}[ax + b] = a^2 \text{Var}[x].$$

where a, b are constants.

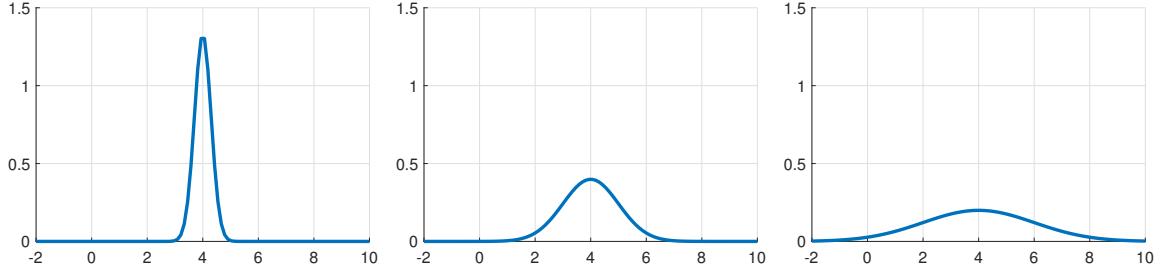


Fig. 6.4. The density of the normal distribution $\mathcal{N}(x|\mu, \sigma^2)$ for $\mu = 4$ and $\sigma = 0.3, 1$ and 2 . Note the density can be greater than 1.

Multidimensional expectations

Because it will be particularly relevant later, we will consider the more general case of a distribution of several variables, say, x, y . In this case the concept of expectation generalizes:

$$\mathbb{E}_{x,y}[f(x, y)] = \iint f(x, y)p(x, y)dxdy$$

notice we have introduced the subscript to indicate we take the expectation over x and y . An important special case is when we consider the sum of several variables. In this case:

$$\begin{aligned} \mathbb{E}_{x,y}[x + y] &= \iint (x + y)p(x, y)dxdy = \iint xp(x, y)dxdy + \iint yp(x, y)dxdy = \int xp(x)dx + \int yp(y)dy \\ &= \mathbb{E}_x[x] + \mathbb{E}_y[y] \end{aligned} \quad (6.12)$$

More generally, if x_1, \dots, x_n are n random variables with joint density p , then for constants a_1, \dots, a_n we have:

$$\mathbb{E}\left[\sum_{i=1}^n a_i x_i\right] = \sum_{i=1}^n a_i \mathbb{E}[x_i] \quad (6.13)$$

Furthermore, if x_1, \dots, x_n are independent, that is, $p(x_1, \dots, x_n) = \prod_{i=1}^n p(x_i)$:

$$\text{Var}\left[\sum_{i=1}^n a_i x_i\right] = \sum_{i=1}^n a_i^2 \text{Var}[x_i]. \quad (6.14)$$

Both of these identities also hold in the discrete case.

6.3 Examples of densities

In this section, we will consider two examples of densities: The normal distribution (and its generalization, the multivariate normal distribution) which provides a convenient density for K -dimensional vectors. Secondly, we will consider a simpler density, namely the Beta density which is defined on the unit interval.

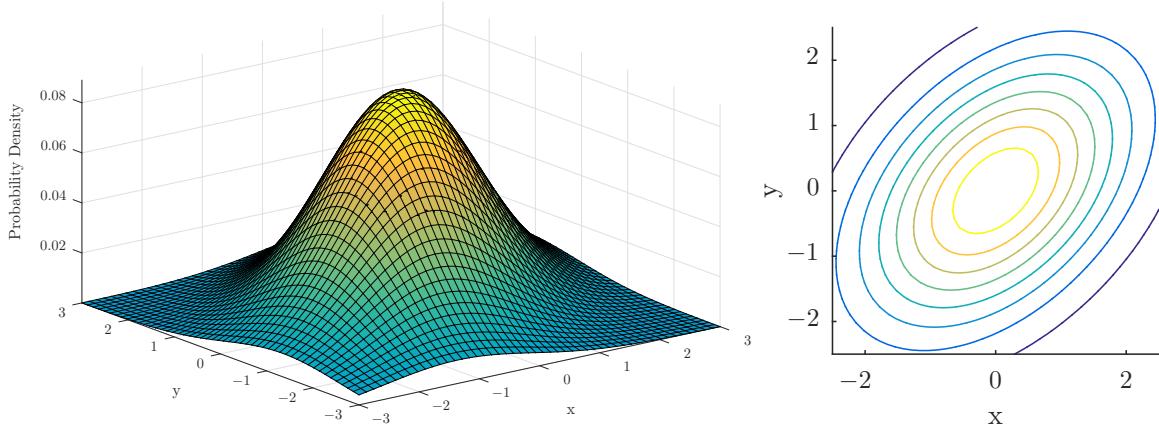


Fig. 6.5. Example of the probability density function of a 2d multivariate normal distribution. In left it is plotted as a function of $\mathbf{x} = [x \ y]^T$, i.e. $p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, whereas on the right the same distribution is shown as a contour plot.

6.3.1 The normal and multivariate normal distribution

In one dimension, the normal distribution with mean μ and variance σ^2 has the density:

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

The symbol \mathcal{N} is simply used for convenience; the normal density is nothing but a function which depends on the three numbers x , μ and σ , and in more familiar notation we would write it as $p(x|\mu, \sigma) = \mathcal{N}(x|\mu, \sigma^2)$. Obviously, it follows from the sum rule eq. (6.9a):

$$\int \mathcal{N}(x|\mu, \sigma^2) = 1.$$

The normal distribution has the familiar bell shaped curve seen in fig. 6.4. The parameters get their name because if we computed the mean and variance using eq. (6.11):

$$\mathbb{E}[x] = \mu, \quad \text{and} \quad \text{Var}[x] = \sigma^2.$$

The normal distribution can be generalized to the multivariate normal distribution which is a distribution over d -dimensional vectors

$$\mathbf{x} = [x_1 \ x_2 \ \cdots \ x_d]^T,$$

and is defined by the density:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})},$$

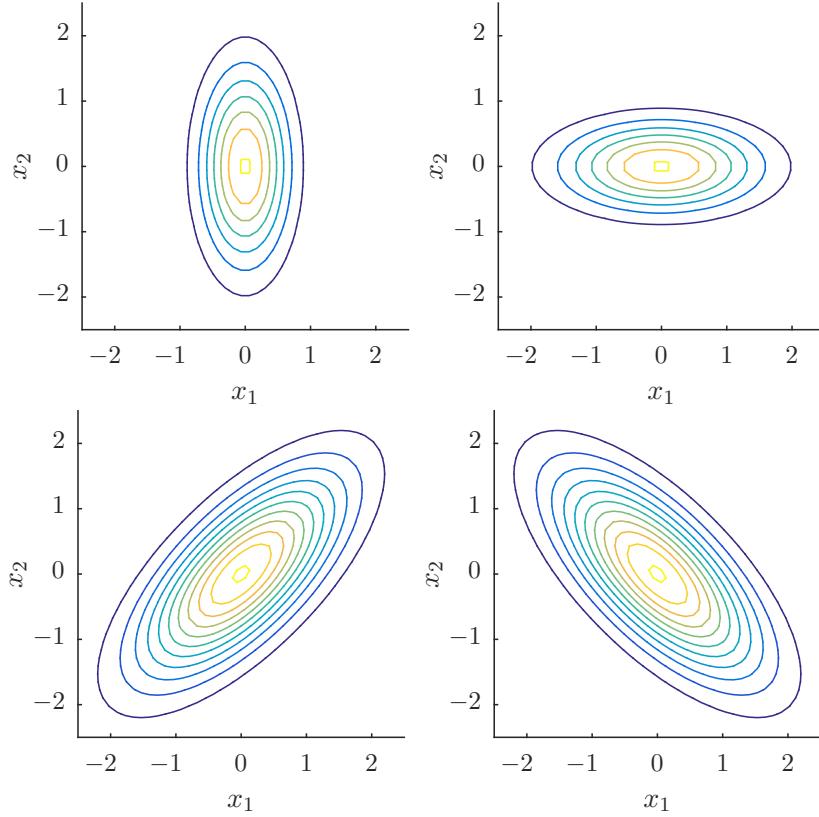


Fig. 6.6. Example of multivariate Gaussians when the covariance matrix is given as $\Sigma_1, \Sigma_2, \Sigma_3$ and Σ_4 in eq. (6.15). When the covariance matrix is diagonal, the multivariate Gaussians are “cigars” oriented along either of the two axis indicating that x_1, x_2 are independent (top row), however with off-diagonal elements, the multivariate Gaussian indicate a dependence between x_1, x_2 . Notice the sign determines how they are slanted.

where Σ is known as the covariance matrix which must be symmetric and positive definite and $|\Sigma|$ is the determinant of Σ and μ is known as the mean of the multivariate normal distribution¹. In fig. 6.5 is shown the multivariate normal distribution corresponding to

$$\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix} \quad \text{and} \quad \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Notice, the distribution for this choice of μ is centered on $(0,0)$; this is no accident. For a general probability density p we can define the covariance matrix C as:

¹ The determinant quantifies the volume of the column-vectors of Σ . In 2d it corresponds to the cross-product, i.e. $\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$. In higher dimensions, the reader should consult a linear-algebra textbook or use a numerical library to compute the determinant, however, see also section 6.3.2.

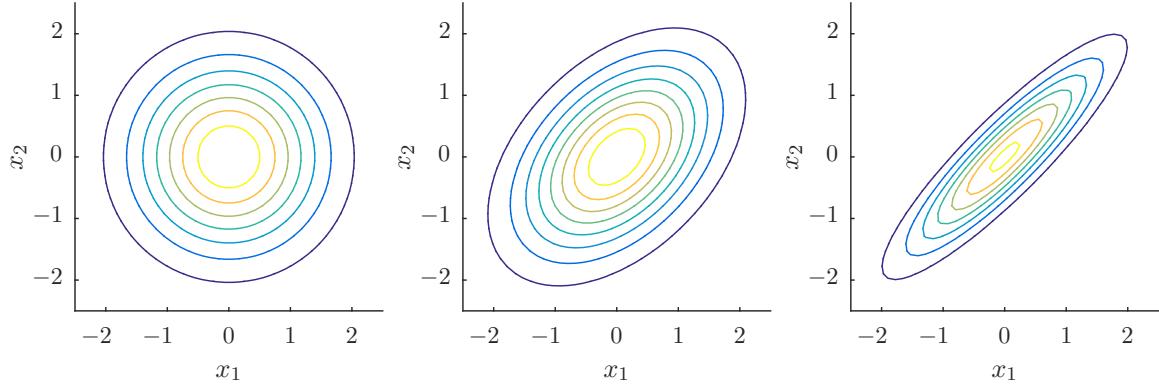


Fig. 6.7. Example of multivariate Gaussians when the covariance matrix is given as Σ_1 , Σ_2 and Σ_3 in eq. (6.16). Increasing the off-diagonal elements increases the covariance.

$$C_{ij} = \text{cov}(x_i, x_j) = \mathbb{E}_{\mathbf{x}} [(x_i - \mathbb{E}[x_i])(x_j - \mathbb{E}[x_j])] = \int (x_i - \mathbb{E}[x_i])(x_j - \mathbb{E}[x_j]) p(\mathbf{x}) d\mathbf{x}.$$

For the special case of the multivariate normal distribution, it holds that

$$\boldsymbol{\mu} = \mathbb{E}[\mathbf{x}] \quad \text{and} \quad \boldsymbol{\Sigma} = \mathbf{C}.$$

This can allow us to get insight into how the multivariate normal distribution behaves for different choice of covariance matrix. Different examples are illustrated in fig. 6.6 where in each instance the mean is chosen as $\boldsymbol{\mu} = [0 \ 0]^T$ and

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.2 & 0 \\ 0 & 1 \end{bmatrix}, \quad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 1 & 0 \\ 0 & 0.2 \end{bmatrix}, \quad (6.15a)$$

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 1 \end{bmatrix}, \quad \boldsymbol{\Sigma}_4 = \begin{bmatrix} 1 & -0.7 \\ -0.7 & 1 \end{bmatrix}. \quad (6.15b)$$

Let us consider one final example where we vary the off-diagonal elements (see fig. 6.7) corresponding to

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & 0.45 \\ 0.45 & 1 \end{bmatrix} \quad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}. \quad (6.16)$$

Notice, the distribution becomes more slanted (*skewed*) when the off-diagonal elements increase.

6.3.2 Diagonal covariance

Consider the case where covariance matrix $\boldsymbol{\Sigma}$ is diagonal

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_d^2 \end{bmatrix},$$

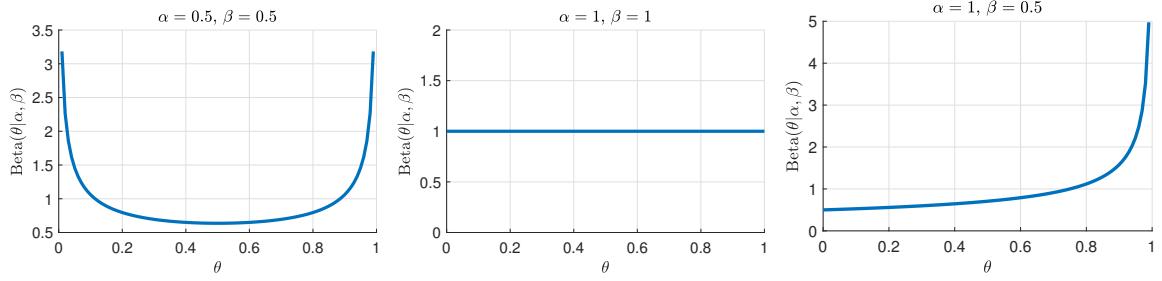


Fig. 6.8. Examples of the beta prior density of eq. (6.17) for different choices of α, β . These two numbers control the mean and variance of the beta distribution, in particular notice the choice $\alpha = \beta = 1$ corresponding to the flat prior.

In this case, the determinant is $|\Sigma| = \prod_{i=1}^d \sigma_i^2$ and therefore:

$$\begin{aligned} p(\mathbf{x}|\boldsymbol{\mu}, \Sigma) &= \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})} \\ &= \frac{1}{\sqrt{(2\pi)^d \prod_{i=1}^d \sigma_i^2}} e^{-\sum_{i=1}^d \frac{1}{2}(x_i - \mu_i)^2 / \sigma_i^2} \\ &= \prod_{i=1}^d \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}} \\ &= \prod_{i=1}^d p(x_i|\mu_i, \sigma_i^2) \end{aligned}$$

Thus, in this case the multivariate normal distribution is just a product of univariate normal distributions, i.e. the x_i 's are independent.

6.3.3 The Beta distribution

The normal distribution gave us a distribution defined for all real numbers. For reasons that will be apparent in a moment, we will be particularly interested in quantities θ that are known to lie between 0 and 1. A particularly interesting family of distributions for such a quantity is the so-called *Beta distribution*. This distribution depends on two parameters $\alpha, \beta > 0$ and has density²

$$\text{Beta density: } \text{Beta}(\theta|\alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1}. \quad (6.17)$$

The two parameters α and β are related to the mean and variance as:

² In the definition, $\Gamma(x)$ is the Gamma function. If x is an integer then $\Gamma(x) = (x-1)!$ and ! denotes the factorial function, i.e., $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$, and $0! = 1$. For further details see https://en.wikipedia.org/wiki/Gamma_function

$$\mathbb{E}_{p(\theta|\alpha,\beta)}[\theta] = \frac{\alpha}{\alpha + \beta}, \quad \text{Var}_{p(\theta|\alpha,\beta)}[\theta] = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}. \quad (6.18)$$

A reader should be warned these integrals are not easy to compute analytically. More insight in α and β can be obtained by plotting realizations of the beta density for different values such as in fig. 6.8. Notice in particular the choice $\alpha = \beta = 1$, which exactly corresponds to a prior where no value of θ is preferred. This is often called the *uniform* prior on the unit interval and is sometimes written as $\mathcal{U}([0, 1])$.

6.3.4 The cumulative density function

Consider once again our rain-fall example from the beginning of this chapter, and recall that if X denotes the amount of rain-fall and $p(x)$ is the density at $X = x$, then the chance there will be between a and b millimeters of rainfall is given by the integral indicated in eq. (6.1)

$$p(a \leq x \leq b) = \int_a^b p(x)dx = \int_{-\infty}^b p(x)dx - \int_{-\infty}^a p(x)dx \quad (6.19)$$

where in the last equality sign we have assumed that $p(x) = 0$ for $x < 0$ and used basic properties of the integral. This inspires the definition of the *cumulative density function* as:

$$\text{Cumulative Density Function} \quad \text{cdf}(z) = \int_{-\infty}^z p(x)dx \quad (6.20)$$

this allows us to write $p(a \leq x \leq b) = \text{cdf}(b) - \text{cdf}(a)$. As an example, consider the Beta distribution:

$$p(\theta) = \text{Beta}(\theta|\alpha = 7, \beta = 3)$$

we have illustrated the CDF defined as $\text{cdf}(\theta) = \int_0^\theta \text{Beta}(z|\alpha = 7, \beta = 3)dz$ in fig. 6.9 (left).

Note the cumulative density function, being defined as an integral of a density, will be increasing. We can therefore invert the cumulative density function to obtain the inverse cumulative density function cdf^{-1} . It is defined as:

$$\text{cdf}^{-1}(y) = \{\text{the value } x \text{ such that } \text{cdf}(x) = y\}. \quad (6.21)$$

We have illustrated the inverse of the CDF in fig. 6.9 (left), but we will also provide a more concrete example. Suppose x is a random quantity with range $[0, 1]$ and density

$$p(x) = \text{Beta}(x|\alpha = 3, \beta = 1) = 3x^2, \quad 0 \leq x \leq 1. \quad (6.22)$$

We can now compute the cumulative density function explicitly as:

$$\text{cdf}(x) = \int_0^x p(z)dz = 3 \int_0^x z^2 dz = x^3 \quad (6.23)$$

If we want to invert the cumulative density function, then the value $\text{cdf}^{-1}(y)$ is found by solving $\text{cdf}(x) = y$ for x . Note that $y = \text{cdf}(x) = x^3$ implies $x = y^{1/3}$ and therefore

$$\text{cdf}^{-1}(y) = y^{1/3}. \quad (6.24)$$

So why is this useful? Consider once more a general density $p(x)$, and suppose we want to find an interval $[x_L, x_U]$ where x is likely to reside. We can make this concrete by saying:

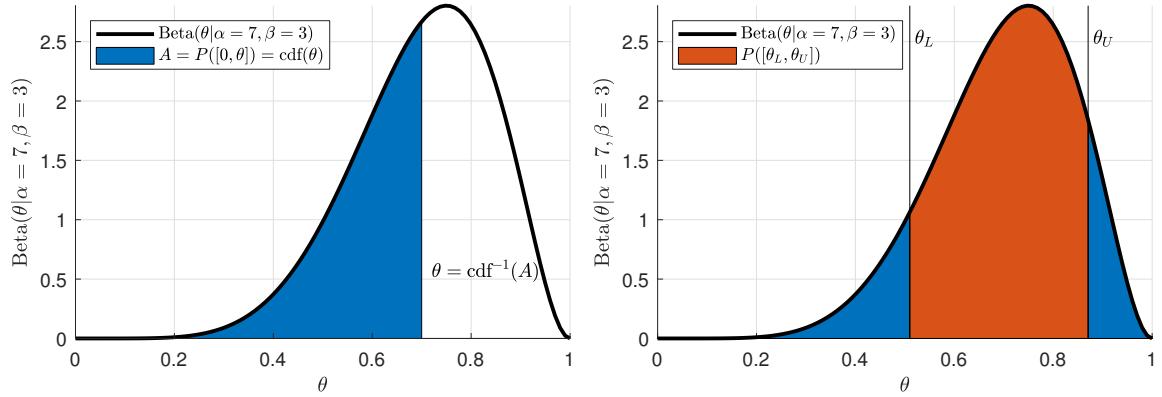


Fig. 6.9. Left: Illustration of the cumulative density function. For a given value of θ , then $\text{cdf}(\theta)$ is the area under the curve up to θ . For a given value of the area A , then the *inverse* of the cumulative density function $\text{cdf}^{-1}(A)$ computes θ such that $\text{cdf}(\theta) = A$. Right: An interval where θ is likely to reside can be defined using the cumulative density function as $[\theta_L, \theta_U]$ where θ_L, θ_U are defined as in eq. (6.26) with $\alpha = 0.2$.

- The chance x belongs to the interval $[x_L, x_U]$ should be high, i.e. equal to $1 - \alpha$ for a small value of α
- We want the interval to be symmetric, in other words the chance $x < x_L$ should be equal to the chance $x > x_U$

We can translate these statements into statements about the cumulative density function as

$$p(x_L \leq x \leq x_U) = 1 - \alpha \quad \text{implies} \quad 1 - \alpha = \text{cdf}(x_U) - \text{cdf}(x_L) \quad (6.25a)$$

$$p(x < x_L) = p(x > x_U) = \frac{\alpha}{2} \quad \text{implies} \quad \frac{\alpha}{2} = \text{cdf}(x_L), \quad 1 - \frac{\alpha}{2} = \text{cdf}(x_U) \quad (6.25b)$$

We have here used that the three above probabilities must sum to 1.

In other words, for a given density p , we can compute cdf and therefore cdf^{-1} . Then we can find x_L and x_U from the last line as:

$$\theta_L = \text{cdf}^{-1}\left(\frac{\alpha}{2}\right), \quad \text{and} \quad \theta_U = \text{cdf}^{-1}\left(1 - \frac{\alpha}{2}\right). \quad (6.26)$$

In other words, once we know cdf^{-1} , we can find a symmetric interval where x resides with a probability $1 - \alpha$ according to $p(x)$. Many confidence intervals are based on intervals of this form, a topic we will return to in chapter 11. An illustration of this type of interval can be found in fig. 6.9 (right) where we have used $\alpha = 0.2$. In other words, according to the density, there is a $1 - \alpha = 0.8$ chance θ is found in the middle (red) area.

6.3.5 The central limit theorem★

The central limit theorem is a key theorem in statistics and provides both an explanation for why the normal distribution is so common, but also a justification for why variance is reduced when

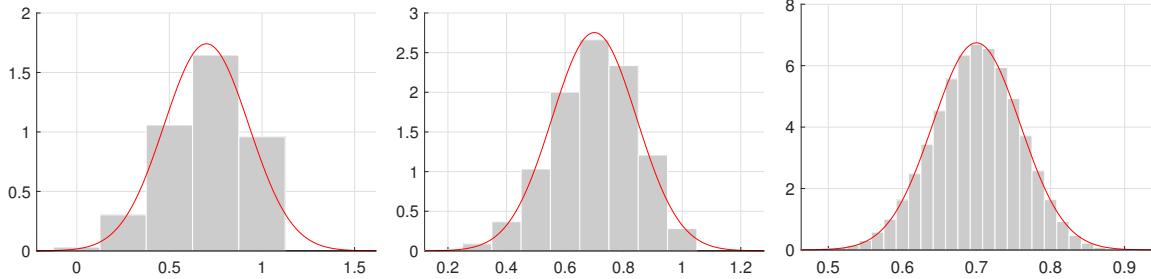


Fig. 6.10. The distribution of $\nu = \frac{m}{N}$ where m follows a binomial distribution $p(m|N, \theta)$ for $N = 2, 10, 60$ and $\theta = 0.7$. The inserted red curve is the normal approximation.

computing averages; as so many quantities in machine learning and statistics are averages, this makes the central limit theorem a key theoretical concept.

It is easier to introduce the central limit theorem by way of example. Once more, suppose we flip N weighted coins and count the number of heads m . As we saw in the previous chapter, the distribution of m is the Binomial distribution eq. (5.35)

$$p(m|\theta, N) = \binom{N}{m} \theta^m (1-\theta)^{N-m}$$

and in fig. 5.6 we have shown the probability density function of m when $\theta = 0.7$ and $N = 4, 10$ and 60 . If we look at these plots from arms length, they sort of look like the normal distributions approximately centered at $N\theta$. In fact, this may give us an idea. Suppose we define the (random) quantity

$$\nu \equiv \frac{m}{N} = \sum_{i=1}^N \frac{b_i}{N}$$

Since we are re-scaling by $\frac{1}{N}$, we should expect it to have mean θ . We can in fact compute both the mean and variance analytically using first eq. (6.13) and eq. (6.14), and next eq. (5.27). Specifically we get:

$$\mathbb{E}[\nu] = \mathbb{E}\left[\sum_{i=1}^n \frac{1}{N} b_i\right] = \sum_{i=1}^n \frac{1}{N} \mathbb{E}[b_i] = \sum_{i=1}^n \frac{1}{N} \theta = \theta \quad (6.27a)$$

$$\text{Var}[\nu] = \text{Var}\left[\sum_{i=1}^n \frac{1}{N} b_i\right] = \sum_{i=1}^n \frac{1}{N^2} \text{Var}[b_i] = \sum_{i=1}^n \frac{1}{N^2} \theta(1-\theta) = \frac{\theta(1-\theta)}{N} \quad (6.27b)$$

This tell us two things. First, that we are right ν has mean θ , but also that the variance is inversely proportional to $\frac{1}{N}$. We should therefore expect the normal density

$$\mathcal{N}(\nu|\mu=\theta, \sigma^2 = \theta(1-\theta)N^{-1}) = \frac{1}{\sqrt{2\pi\frac{\theta(1-\theta)}{N}}} e^{-\frac{(\nu-\theta)^2}{2\theta(1-\theta)}} = \sqrt{\frac{N}{2\pi\theta(1-\theta)}} e^{-\frac{N(\nu-\theta)^2}{2\theta(1-\theta)}}$$

to at least provide an approximate match to the true density of ν . We have plotted both these densities in fig. 6.10 for $N = 4, 10, 60$. While it should *not* be surprising the mean and variance

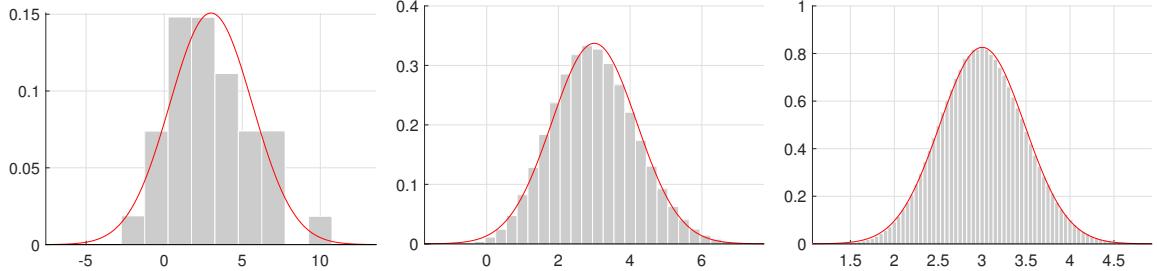


Fig. 6.11. The average of N rolls of the silly die for $N = 2, 10, 60$ and inserted normal approximations. Notice average of the rolls converge rapidly to the normal distribution.

match (we just showed this to be the case in eq. (6.27)), it *should* be surprising that the exact *shape* of the curves seem to match perfectly as N increases.

We might suspect this has something to do with the Bernoulli distribution, but this is not the case. To see this, let us turn to the silly die from section 5.2: Suppose we roll the silly die N times and compute the mean value of the roll ν :

$$\nu = \sum_{i=1}^N x_i$$

For instance, let $N = 3$ and consider the following two example roll sequences:

$$\text{Roll sequence 1: } [10 \ 1 \ 4], \quad \text{mean of sequence 1: } \nu = \frac{13}{3} \quad (6.28)$$

$$\text{Roll sequence 2: } [4 \ 2 \ 1], \quad \text{mean of sequence 2: } \nu = \frac{7}{3} \quad (6.29)$$

The distribution of ν for general N , $p(\nu|N)$, is difficult to derive. For instance if $N = 2$ the chance of getting $\nu = 10$ (two 10's) is $p(\nu = 10|N = 2) = p(\nu = 10|N = 1)p(\nu = 10|N = 1) = \frac{1}{6^2}$, however it is easy to simulate many realizations of the distribution, compute ν , and plot the estimated $p(\nu|N)$ for different N . The result is shown in fig. 6.11 for $N = 2, 10, 60$ as the gray histogram.

While the distribution is difficult to derive, a computation similar to eq. (6.27), along with the mean/variance computed in Example 5.2.1, show that

$$\mathbb{E}[\nu] = \{\text{Mean of a single die}\} = 3, \quad \text{Var}[\nu] = \frac{\{\text{Variance of a single die}\}}{N} = \frac{14}{N}$$

Both the true (simulated) density ν and the normal approximation $\mathcal{N}(\nu|\mu = \mathbb{E}[\nu], \sigma^2 = \text{Var}[\nu])$ is plotted in fig. 6.11. As we can clearly see in the figure, the normal approximation is *very nearly identical* to the true distribution. This is the content of the central limit theorem:

Consider N random variables X_1, \dots, X_N , each taking values x_1, \dots, x_N . If we then define z as the mean:

$$z = \frac{1}{N} \sum_{i=1}^N x_i,$$

Then, as N increases, the distribution of z will be closer and closer to a normal distribution $\mathcal{N}(z|\mu, \sigma^2)$ with mean/variance:

$$\mu = \mathbb{E}[z] = \frac{1}{N} \sum_{i=1}^N \mathbb{E}[x_i], \quad \sigma = \sqrt{\text{Var}[z]} = \sqrt{\frac{1}{N} \sum_{i=1}^N \text{Var}[x_i]}. \quad (6.30)$$

It is this normal distribution, and in particular that the variance shrinks towards zero as N increases, that guarantees averages converge in statistics to well-defined values. If the central limit theorem did not hold there would be no reason to increase the number of patients in a clinical trial to get a better idea about the average effect or increase the number of test examples in machine learning to better judge the performance of a method, and it implies the normal distribution can be expected to pop up in all kinds of circumstances because most quantities are (in one way or another) representative of average effects.

6.4 Bayesian probabilities and machine learning

In this section, we will consider a very basic learning problem which nevertheless encapsulates how probabilities are applied in general in machine learning. There are basically two steps when applying probabilities in machine learning:

- Write up a probability distribution for all relevant quantities of interest (data and parameters).
- Formulate the machine-learning task (prediction, classification, etc.) in terms of a probability which can be derived using the sum and product rule.

We will illustrate this with a very simple inference problem. Suppose your friend shows you a coin he bought at a flea market. The salesman informed him that it might be a magic coin (a magic coin is a coin where one side comes up more often than 50%), but he wasn't sure and in either case he doesn't know which side is supposed to come up more often.

A simple learning problem could be to flip the coin a number of times and use the information to figure out the chance it will come up heads in a new flip. As usual, let $b_i = 0$ be the event the coin comes up tails in flip i and $b_i = 1$ heads. We once more represent the sequence *heads, tails, heads* as

$$b_1 = 1, \quad b_2 = 0, \quad b_3 = 1,$$

which we will write as a vector \mathbf{b} . Recall the by now well-known probability:

$$p(\mathbf{b}|\theta) = \theta^m(1-\theta)^{N-m} \quad (6.31)$$

Phrased in this way, what we are interested in is learning the chance the coin comes up heads θ given the particular sequence of flips. In section 5.4.5 we saw one idea, namely to maximize the likelihood function $\mathcal{L}(\theta) = p(\mathbf{b}|\theta)$ and thereby obtain $\theta^* = \frac{m}{N}$. A moments thought will reveal this answer is quite plainly *wrong*. For instance, if we observe just $N = 3$ flips of a coin, nobody in their right mind would rule out the possibility the coin was fair (i.e. $\theta = \frac{1}{2}$), but the only possible values of θ^* we could compute would be θ^* are 0, $\frac{1}{3}$, $\frac{2}{3}$ and 1. Obviously something has gone wrong!

The problem is maximum likelihood is a *principle*, in the sense of a sometimes-good-idea, and not something we have derived from first principles. The *correct* way to proceed is the *only* way to proceed, namely to ask our rational robot from fig. 5.1 which only operates according to the rules

of probability. What we wish to learn is θ based on a sequence of coin flips \mathbf{b} . Our belief in theta given \mathbf{b} is exactly $p(\theta|\mathbf{b})$, and simply plugging this into Bayes' theorem we get:

$$p(\theta|\mathbf{b}) = \frac{p(\mathbf{b}|\theta)p(\theta)}{\int p(\mathbf{b}|\theta')p(\theta')d\theta'} \quad (6.32)$$

Note we did *not* have to check or verify anything to use Bayes' theorem: Rather, this is something we can *always* do because the rules of probability are *always* true. Inspecting the above, we see that in order to proceed, we must specify $p(\theta)$. Since we know just one distribution for a quantity defined on the unit interval, the beta distribution, we will assume θ is beta distributed with (so far) unknown parameters $\alpha, \beta > 0$

Technical note 6.4.1: Deriving the posterior density of the coin

Returning to our coin, to compute the posterior $p(\theta|\mathbf{b})$, we need to compute the numerator and denominator of eq. (6.32). Beginning with the numerator, using the likelihood eq. (6.31) and the Beta prior eq. (6.37) we obtain

$$p(\mathbf{b}|\theta)p(\theta) = p(\mathbf{b}|\theta)p(\theta|\alpha, \beta) \quad (6.33)$$

$$\begin{aligned} &= \theta^m(1-\theta)^{N-m} \times \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1} \\ &= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{m+\alpha-1}(1-\theta)^{N-m+\beta-1} \end{aligned} \quad (6.34)$$

The denominator is obtained by integrating this expression with respect to θ ; the integral is somewhat complicated analytically, but nevertheless based on well-known rules. We get:

$$\begin{aligned} \int p(\mathbf{b}|\theta)p(\theta)d\theta &= \int_0^1 p(\mathbf{b}|\theta)p(\theta|\alpha, \beta)d\theta = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \int_0^1 \theta^{\alpha+m-1}(1-\theta)^{\beta+N-m-1}d\theta \\ &= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(\alpha+m)\Gamma(\beta+N-m)}{\Gamma(\alpha+\beta+N)} \end{aligned} \quad (6.35)$$

Inserting eq. (6.35) and eq. (6.34) into eq. (6.32) we obtain:

$$\begin{aligned} p(\theta|\mathbf{b}, \alpha, \beta) &= \frac{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha+m-1}(1-\theta)^{\beta+N-m-1}}{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(\alpha+m)\Gamma(\beta+N-m)}{\Gamma(\alpha+\beta+N)}} \\ &= \frac{\Gamma(\alpha+\beta+N)}{\Gamma(\alpha+m)\Gamma(\beta+N-m)}\theta^{\alpha+m-1}(1-\theta)^{\beta+N-m-1}. \end{aligned} \quad (6.36)$$

All that remains is to note eq. (6.36) has the same form as the Beta density eq. (6.37), but with new parameters $a = \alpha + m$, $b = \beta + N - m$.

$$p(\theta) \equiv p(\theta|\alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1} \quad (6.37)$$

We now have expressions for all terms on the right-hand side of eq. (6.32); simply plugging these in and computing the integral we see:

$$\begin{aligned} p(\theta|\mathbf{b}) &= \text{Beta}(\theta|a, b), \quad a = \alpha + m, \quad b = \beta + N - m \\ &= p(\theta|\mathbf{b}, \alpha, \beta) \end{aligned} \quad (6.38)$$

In the last line, we include α and β to signify the posterior depends on these two numbers (see Box section 6.4 for details on the derivation).

In other words, when we use a $\text{Beta}(\cdot|\alpha, \beta)$ prior for θ , the posterior density remains a Beta distribution. This important property is known as *conjugacy* and is one of the motivations for using a Beta prior. Furthermore, notice α and β plays roughly the same role as the number of flips that come up heads m or tails $N-m$. In other words, we can interpret α and β in the prior as a number of *pseudo-counts* where the specific case of the flat prior $\alpha = \beta = 1$ corresponds to observing two coin flips, one positive and one negative.

6.4.1 Choosing the prior

Continuing the example of the flat prior ($\alpha = \beta = 1$), if we plug these values into either eq. (6.36) or eq. (6.38) we obtain

$$p(\theta|\mathbf{b}, \alpha = \beta = 1) = \frac{(N+1)!}{m!(N-m)!} \theta^m (1-\theta)^{N-m}. \quad (6.39)$$

In fig. 6.12 this figure is plotted for different sequences of flips, starting with just one flip that came up heads $N = 1, m = 1$ and ending with $N = 100$ flips where $m = 51$ came up heads. Several important aspects can be read off from this example. We observe that the distribution of θ is peaked at around the expected value, i.e. $\frac{m}{N}$. However when the number of observations increase we become more and more certain that θ is *near* to this value. For instance in the $N = 4, m = 3$ we can compute the probability θ is in the interval $[0.4, 0.6]$ as:

$$P(\theta \text{ is between } 0.4 \text{ and } 0.6 | N = 4, m = 3) = \int_{0.4}^{0.6} p(\theta|\mathbf{b}) \approx 0.25,$$

whereas for $N = 100, m = 51$ this chance is more than 0.95!. This answer may not feel as satisfying as a single number (“ θ is really 0.5”), or the familiar confidence interval from classical statistics (“with a confidence level of 95% θ is in the interval $0.5 \pm \theta_0$ ”), however, the Bayesian answer is more general and later in chapter 10 we will see there exist a simple recipe for constructing the Bayesian equivalent of the confidence interval, the *credibility interval*.

6.5 Bayesian learning in general

Let us turn to the general problem of *learning* parameters \mathbf{w} from a dataset $\mathcal{D} = (\mathbf{X}, \mathbf{y})$ consisting of N observations in the usual format. First, the density we are interested in is

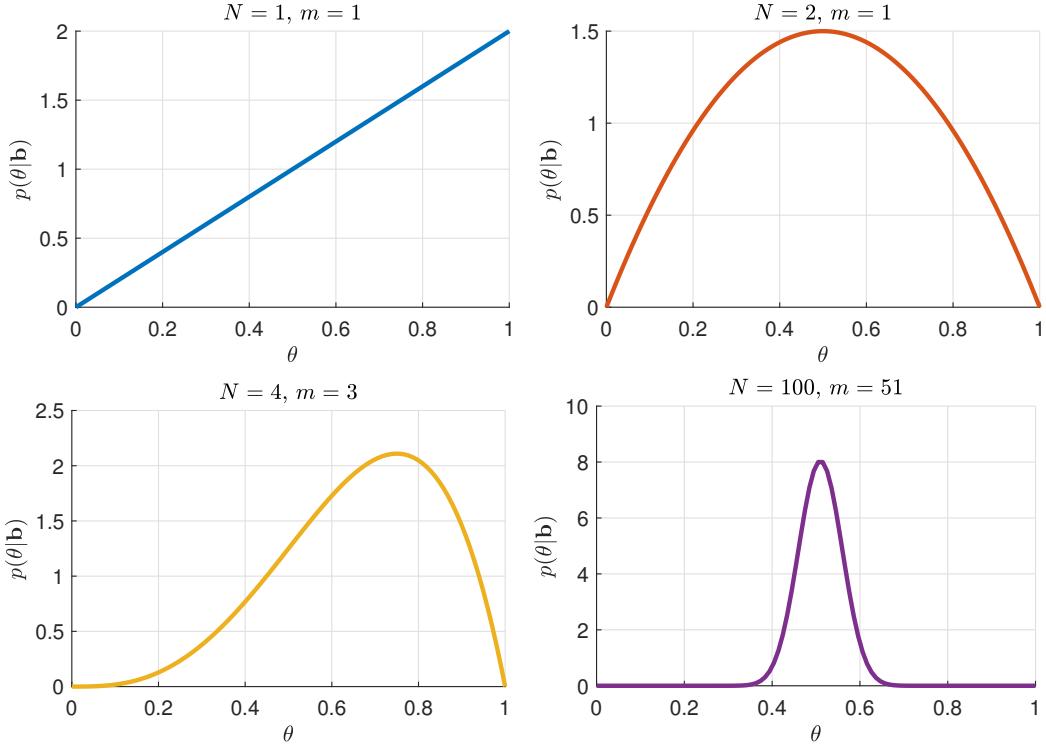


Fig. 6.12. Examples of the posterior probability $p(\theta|\mathbf{b})$ from eq. (6.39) for different numbers of flips N and different number of heads m . The top left figure corresponds to *heads*, the top-right to *heads, tails*, bottom left to *heads, tails, heads* and bottom right to $N = 100$ flips where $m = 51$ came up heads.

$$p(\mathbf{w}|\mathcal{D}) = p(\mathbf{w}|\mathbf{y}, \mathbf{X}) \quad (6.40)$$

We will make the following two *assumptions*:

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{i=1}^N p(y_i|\mathbf{x}_i, \mathbf{w}) \quad (6.41a)$$

$$p(\mathbf{w}|\mathbf{X}) = p(\mathbf{w}) \quad (6.41b)$$

What the first assumption tells us is that when we know the parameters \mathbf{w} and \mathbf{x}_i , the other observations are irrelevant in terms of predicting y_i . The second assumption encode the idea \mathbf{X} alone does not tell us anything about \mathbf{w} .

At this point, the reader no doubt expect what will happen: Applying Bayes' theorem to eq. (6.40), and the then the two assumptions eq. (6.41a) and eq. (6.41b), we get:

$$\begin{aligned} p(\mathbf{w}|\mathbf{X}, \mathbf{y}) &= \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w}|\mathbf{X})}{p(\mathbf{y}|\mathbf{X})} \\ &= \frac{\prod_{i=1}^N p(y_i|\mathbf{x}_i, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})} \end{aligned} \quad (6.42)$$

There are two general ways to proceed from here. In the coin-example, we proceeded by simply computing the numerator and simplifying the expression. In that case, \mathbf{w} was equal to θ , there was no \mathbf{X} , $p(y_i|\mathbf{x}_i, \mathbf{w}) = p(b_i|\theta)$ and $p(\mathbf{w}) = \text{Beta}(\theta|\alpha, \beta)$ was a Beta distribution.

The second approach, which is simpler and the one we will consider in this book, is to *maximize* the above expression with respect to \mathbf{w} to find the most likely value of \mathbf{w} . Maximizing eq. (6.42) is equivalent to maximizing the logarithm, which can be written as:

$$\text{Maximize: } \log \mathcal{L}(\mathbf{w}) + \log p(\mathbf{w}), \quad \text{where } \mathcal{L}(\mathbf{w}) = \sum_{i=1}^N \log p(y_i|\mathbf{x}_i, \mathbf{w}) \quad (6.43)$$

Sometimes, the prior term $\log p(\mathbf{w})$ (which does not depend on the data) will be ignored, and sometimes not.

It will often be more convenient to formulate the maximization problem as instead the corresponding minimization problem of the cost-function obtained by multiplying eq. (6.43) with -1 . Furthermore, to easier compare the cost-function for models trained on different-size dataset, the cost function is re-scaled by $\frac{1}{N}$ so that it measures a cost-per-observation. In that case the minimization problem becomes:

$$\text{Minimize: } E(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N \log p(y_i|\mathbf{x}_i, \mathbf{w}). \quad (6.44)$$

Sometimes, we will add a *regularization term* to the right-hand side of this expression (see chapter 14). This discussion has been somewhat abstract, but inspecting eq. (6.44), we see that one way to do machine learning is to first come up with an expression for the probabilities $p(y_i|\mathbf{w}, \mathbf{x}_i)$, and then a numerical recipe for carrying out the minimization in eq. (6.44) to learn \mathbf{w} . We will see examples for how this can be implemented in chapter 8, chapter 14, and chapter 15. We summarize this discussion in summary box 6.5.1.

Summary 6.5.1: Maximum likelihood framework

Many machine-learning methods can be motivated within the following, general, framework. Given data \mathbf{X}, \mathbf{y} , we *select*, based on expert knowledge, the likelihood density function:

$$p(y_i | \mathbf{w}, \mathbf{x}_i). \quad (6.45)$$

Here, \mathbf{w} are the parameters in the model. We then learn the weights \mathbf{w} by letting them be equal to the value \mathbf{w}^* found by either:

$$\text{Maximize: } \mathbf{w}^* = \arg \max_{\mathbf{w}} \left[\log p(\mathbf{w}) + \sum_{i=1}^N \log p(y_i | \mathbf{x}_i, \mathbf{w}) \right] \quad (6.46)$$

$$\text{Minimize: } \mathbf{w}^* = \arg \min_{\mathbf{w}} E(\mathbf{w}) \quad (6.47)$$

$$\text{where: } E(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N \log p(y_i | \mathbf{x}_i, \mathbf{w}) + \{\text{Optional regularization term}\}.$$

If we ignore either the prior or regularizations term (or they have the same analytically form), these two formulations are equivalent, and can be derived using Bayes' rule and the maximum likelihood principle, see eq. (6.42) for further details.

Problems

6.1. Question 1: A factory produces cars. We consider three properties of the cars produced by the factory and each property can only take two values:

- The *color* which can be either *red* or *blue*.
- The *weight* which can be either *heavy* or *light*.
- The *model* which can be either *2-doors* or *4-doors*.

There are thus $2^3 = 8$ possible car types such as (*red, heavy, 2-doors*) or (*blue, light, 4-doors*).

Suppose you are given the following information about cars produced from the factory:

- The probability a car has four doors is 0.5
- The probability a car is heavy given it has four doors is 0.8
- The probability a car is heavy given it has two doors is 0.2
- The probability a car is heavy and red is 0.1

Given the above information, what is the probability a car is blue given it is heavy?

- A 0.2
- B 0.5
- C 0.8
- D 0.9
- E Don't know.

6.2. Question 2: In the study it was found that

- 88 pct. of the persons have normal semen.
- 12.5 pct. of the persons that have normal semen have had a childhood disease.
- 16.7 pct. of the persons that have abnormal semen have had a childhood disease.

What is the probability that a person that has had a childhood disease will have normal semen according to the study?

- A 12.50 %
- B 74.85 %

- C 84.59 %
- D 88.00 %
- E Don't know.

6.3. Question 3: Based on Haberman's Survival Data found in Table 6.1 it is found:

- 56 pct. of the subjects had positive axillary nodes detected.
- 36 pct. of the subjects that had positive axillary nodes detected survived.
- 14 pct. of the subjects that did not have positive axillary nodes survived.

What is the probability that a subject that has survived would have positive axillary nodes according to the study by Haberman?

No.	Attribute description	Abbrev.
x_1	Young (< 60 years), $x_1 = 0$ or Old (≥ 60 years), $x_1 = 1$	Age
x_2	Operated before, $x_2 = 0$ or after 1960, $x_2 = 1$	OpT
x_3	Positive axillary nodes detected No, $x_3 = 0$ or Yes, $x_3 = 1$	PAN
y	Lived after 5 years No, $y = 0$ or Yes, $y = 1$	Surv

Table 6.1. A modified version of Haberman's Survival Data taken from <http://archive.ics.uci.edu/ml/machine-learning-databases/haberman/haberman.names>. The attributes x_1-x_3 denoting the age, operation time and cancer size as well as the output denoting survival after five years are binary. The data contains a total of $N = 306$ observations.

- A 20.2 %
- B 36.0 %
- C 56.0%
- D 76.6%
- E Don't know.

Data Visualization

“The drawing shows me at one glance what might be spread over ten pages in a book.” wrote Ivan S. Turgenev in 1862¹, thereby repeating the literary trope a picture is worth a thousand words. It is worth reflecting on why this is so widely thought to be the case. One idea is that, simply put, more of the brain is adapted to the direct processing of visual information than any other type of sensory information. This in turn means we can distinguish between two senses a picture is worth a thousand words: the first is it allows us to quickly comprehend and therefore learn new information visually than from any other source and, the second, that it allows us to find and learn new patterns in data; it is notable we often refer to this as *seeing* a new pattern.

We will therefore decided to dedicate an entire chapter to visualizations, but split into two parts reflecting these two senses of seeing: The first sections will treat to the classical use of visualizations, namely communicating information to a reader. The remaining sections will re-visit the machine-learning workflow of fig. 1.13, with a focus on the second use of using visualizations to see — namely as a window into what machine-learning does. We will also use this as an opportunity to discuss the machine-learning workflow in slightly more details to provide a backdrop for the remaining sections of this book.

7.1 Basic plotting

Visualization can be thought of as compressing a large quantity of information into a few visual elements. This section will review some ways this can be accomplished roughly ordered according to how much compression is desired.

Visualization of a single attribute

Consider each of the four attributes of the Fisher Iris dataset and suppose we wish to visualize a single attribute. The histogram allows us to represent multiple observations in a limited space while preserving nearly all the information. A histogram is constructed in two steps. The first step is to divide the entire range of value of the variable into a series of intervals (referred to as *bins*),

¹ Quote taken from *Fathers and Sons* in <http://www.phrases.org.uk/meanings/a-picture-is-worth-a-thousand-words.html>

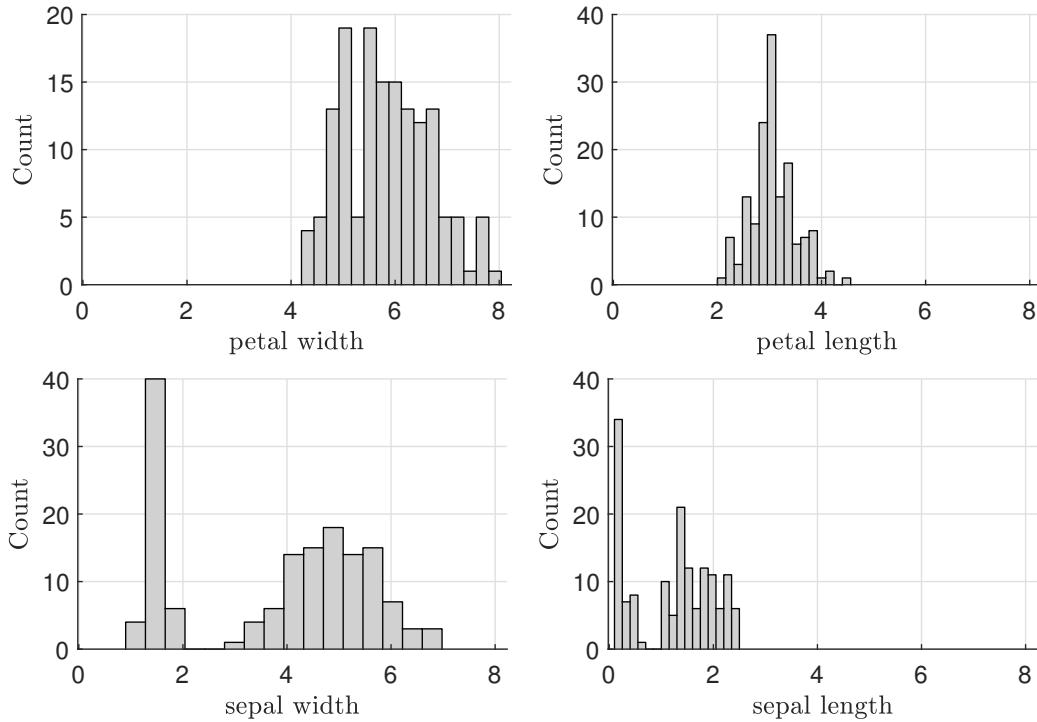


Fig. 7.1. Histograms based on $N = 16$ bins of the four features in the Iris dataset.

most often of equal length. We then count how many observations in the dataset fall within each such bin and draw a rectangle where the base of the rectangle is the interval and the height is the number of observations that fall into the interval. That is, the sum of the height of all rectangles will be the number of observations. This procedure is also known as *binning*.

In fig. 7.1 is shown the histograms of all four attributes of the Iris dataset. We see some of the histograms look roughly symmetric and bell-shaped (this indicates the attribute is likely normally distributed) whereas for instance the sepal width has two humps (it is multimodal). The advantage of the histogram is that it tells us nearly all there is to know about a variable, the disadvantage is that they take up quite a lot of space and that we have to select the number of bins manually and too many or too few will create uninformative histograms. A more parsimonious representation of the distribution of an attribute can be obtained with a boxplot. Boxplots of the four attributes of the Fisher Iris data is shown in fig. 7.2 (left). Here, the middle red line corresponds to the median (the $p = 0.5$ percentile), the upper and lower bounds, l_{75} and l_{25} , of the blue box is the $p = 0.75$ and 0.25 percentile, the black lines are known as the whiskers and attempt to outline how wide the distribution is. The upper/lower whiskers are defined as:

$$\text{upper whisker: } \min(l_{75} + \frac{3}{2}(l_{75} - l_{25}), v_N), \quad (7.1)$$

$$\text{lower whisker: } \max(l_{25} - \frac{3}{2}(l_{75} - l_{25}), v_1), \quad (7.2)$$

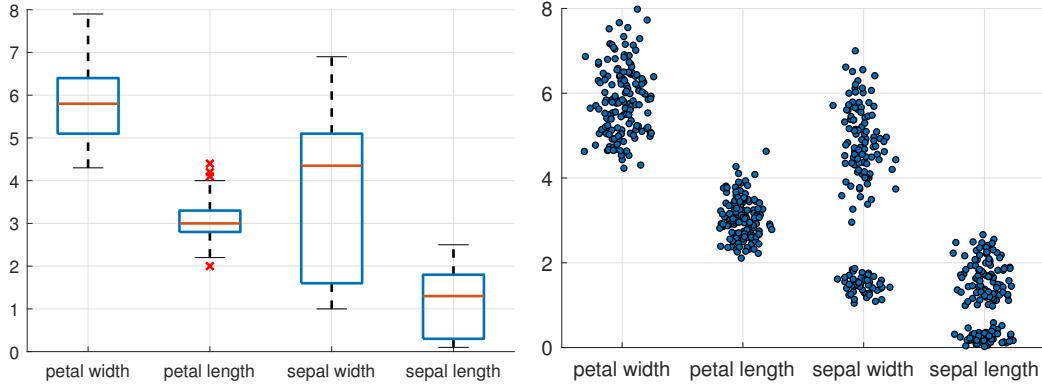


Fig. 7.2. A boxplot (left pane) is a way to condense the information in a histogram into a stereotypical representation. An advantage of the boxplot is it allows us to read off relevant quantities of the dataset, such as the medium value, however, compared to the histograms in fig. 7.1 we also loose information such as the bimodality of the sepal width. Notice how these features affect the symmetry of the boxplot. The right-most pane provides an example of how the same information can be communicated with a simpler visual element, namely by simply plotting each observation as a point and adding a bit of x, y jitter to make the points distinct.

where v_N denotes the value of the largest observation, and v_1 the value of the smallest observation. Observations that fall outside these bounds are marked as red crosses and they are said to be outliers insofar as the boxplot is concerned. A similar effect to the boxplot and histogram can be obtained with a bit of simple data-processing. In fig. 7.2 (right) we have simply plotted each value of the attributes plus a bit of random noise applied to the x and y coordinate. This allows us to distinguish each individual point and convey similar information as that found in the boxplot and the histogram.

Visualization of one-dimensional data

Suppose we have to visualize a single 1d dataset, for instance, the sale of widgets produces by a company over 12 months. The three most common ways to visualize this is shown in fig. 7.3 where we have illustrated the line plot, a “dot” plot and a bar chart. Notice the bar chart start at 0 and so should primarily be considered for variables which are ratio, i.e. 0 has some specific meaning. The use of lines often help to “ground” the eye and provide guidance when there is correspondence between observations whereas the dot and bar chart are easy to read and compare different values. Also notice the bar chart and the other plots tend to guide the reader to different aspects of the data.

The first two plots would be useful for pointing out the variability of the data, whereas the bar chart would be useful for pointing out the variability in absolute terms. Having in mind *what* we want to communicate should always inform us about what graphical elements we choose and how we decide to select or scale for instance the y -axis. In the bottom-right pane we have indicated the chart we would likely prefer for this situation: We focus on the variability in the data and use a line to indicate the months are connected while the large dots indicate the individual measurements and points out to the reader that we only have a month-by-month dataset with few observations. We

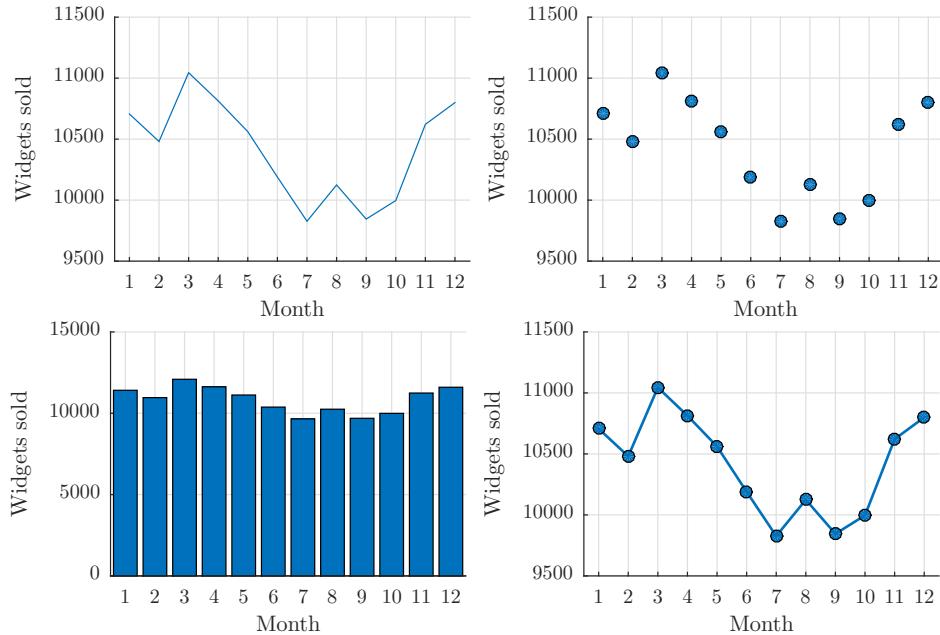


Fig. 7.3. Different attempts to visualize a simple 1D dataset corresponding to widget sale over months. Top row is the line plot and the dot-plot. Bottom left is the bar plot (notice the y -axis start at 0) and an attempt to combine the line and dot plot to better guide the reader's eye.

have also increased the line width slightly. In all four charts, we use grid lines to guide the reader's eye making it easier to compare the first months to the last.

Several one-dimensional series

Suppose we tests four different models on eight datasets and for each model we obtain a performance rating. We believe our method (Method 1) is the better. How do we best communicate this? Of course, we should include a table in the report. However, suppose we want to include a visualization of this data in for instance a presentation or in the main pages of the report and wish to include the table as an appendix. In fig. 7.4 are four attempts to visualize this dataset. Notice the three methods we have seen so far are all fairly difficult to read. The lines cross many times for the line plots, the dot plots too seem difficult to compare and the bar chart has an almost psychedelic effect. One strategy to fix this is to sort the datasets in descending order. In this way, connecting the datasets with lines makes the (relative) performance easier to read. It is now fairly apparent the yellow method seems to have some benefits, especially for the medium-difficult problems whereas the blue method seems to perform worse. We have also sorted the methods such that the first (best!) method is first in the legend and in addition (try to zoom in) we make sure to plot the graphs such that the first method are on top of the others. This is a rather subtle effect but it does make a noticeable difference in terms of what the reader is focused on.

This is an example where arranging the data is important for easier communication. In addition to what we have already done, one could try to select a color or line scheme for the other method

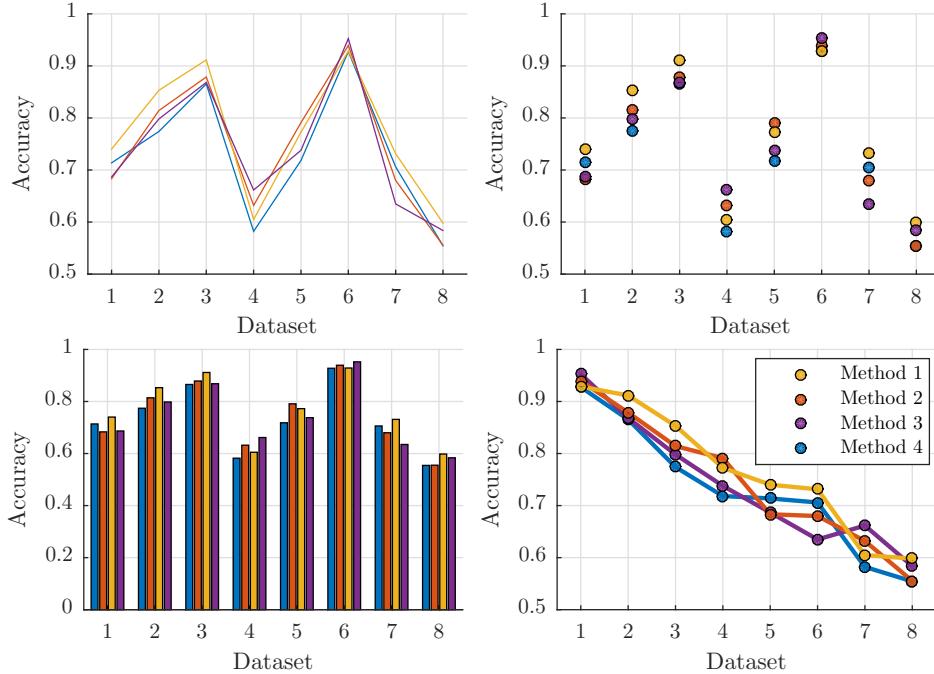


Fig. 7.4. Illustration of four 1D datasets corresponding to the performance of four machine-learning methods on eight datasets. In the bottom right pane we have tried to sort the datasets and use lines to connect related datasets. Which are easier to read?

that made them stand out less. For instance by marking them all in graytone, which would further emphasize that we were comparing our method against three others.

Visualizing higher-dimensional data

Likely, the best and certainly the simplest way to visualize 2D data is the scatter plot. In fig. 7.5 we have plotted two coordinates of the Fisher Iris data against each other and used colors to denote the classes. A 2D plot quickly provides an overview of how spread out the data is and in this case, it immediately tells us that determining if a flower is setosa (as opposed to the two other types) is a trivial problem whereas the other two classes, insofar as these two features are concerned, are more difficult to discern. When making 2D scatter plots, be aware of the scaling of the axes; if the units of the axes are the same (length) then it may be sensible to ensure they have the same scale.

A difficulty in the 2D scatter plot is that we only see two dimensions at the same time. This can (to some extend) be overcome by plotting all dimensions against each other in pairs constituting what is known as a matrix plot, see fig. 7.6. An advantage of this type of plot is that we no longer have to select two particular dimensions; a disadvantage is that this is only possible to display for a limited number of attributes. What we perhaps learn from this plot is that sepal width and sepal length may be two features useful for distinguishing versicolor and virginica, which may leave us even more optimistic in terms of the classification problem. However, one cannot conclude that because no two features in and by themselves can be used to separate two classes then the problem

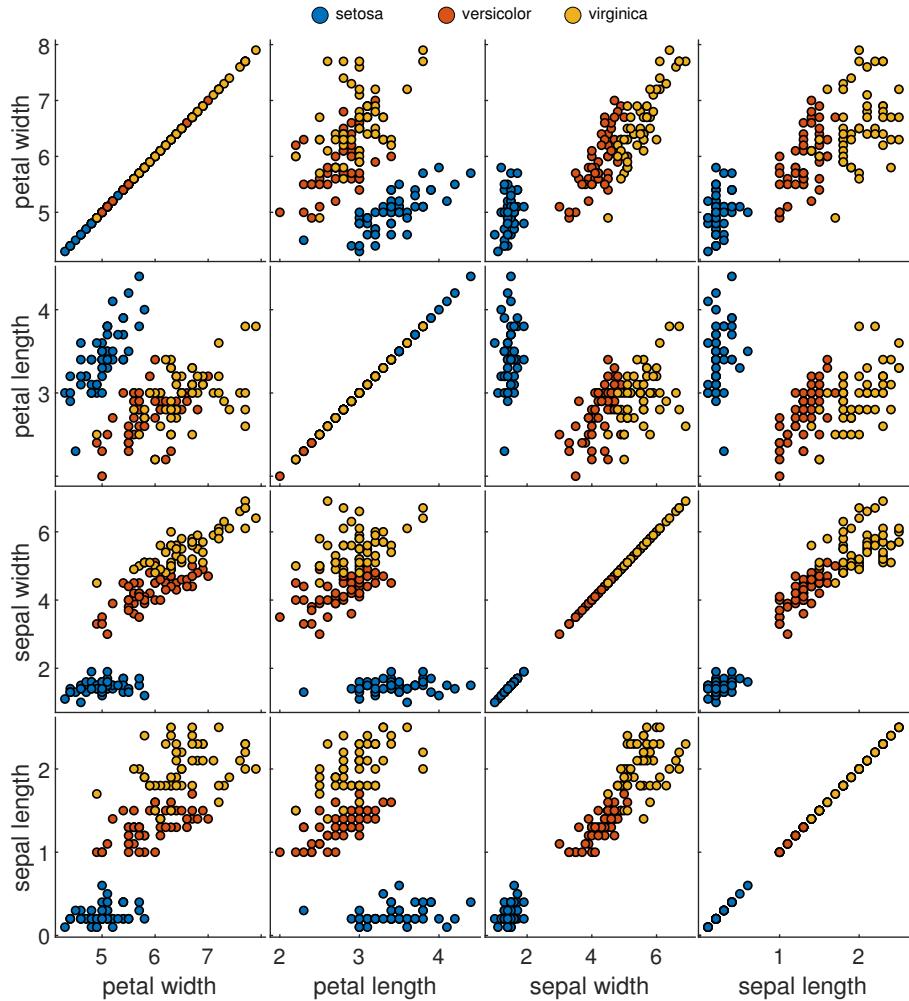


Fig. 7.6. A matrix plot in which the four attributes are plotted pairwise against each other and colors are used to indicate class labels

is impossible to solve: Firstly, this tells us nothing about how three features can perform, and secondly it only tells us about certain (axis-oriented) projections onto a 2D plane.

Higher-dimensional observations

To go beyond 2d requires either changing the visual element or accept some distortion. for instance one can attempt a 3D scatter plot of the data where we consider three features together as shown in fig. 7.7. The problem with 3D plots is that they have to be projected onto a 2D screen or paper which ruins most of the benefits of the 3D plot. Another type of technique for high-dimensional data is to represent each multi-dimensional observation by a more complex visual element than a point. One such example is the *coordinate plot* illustrated in fig. 7.7 (right) where an observation

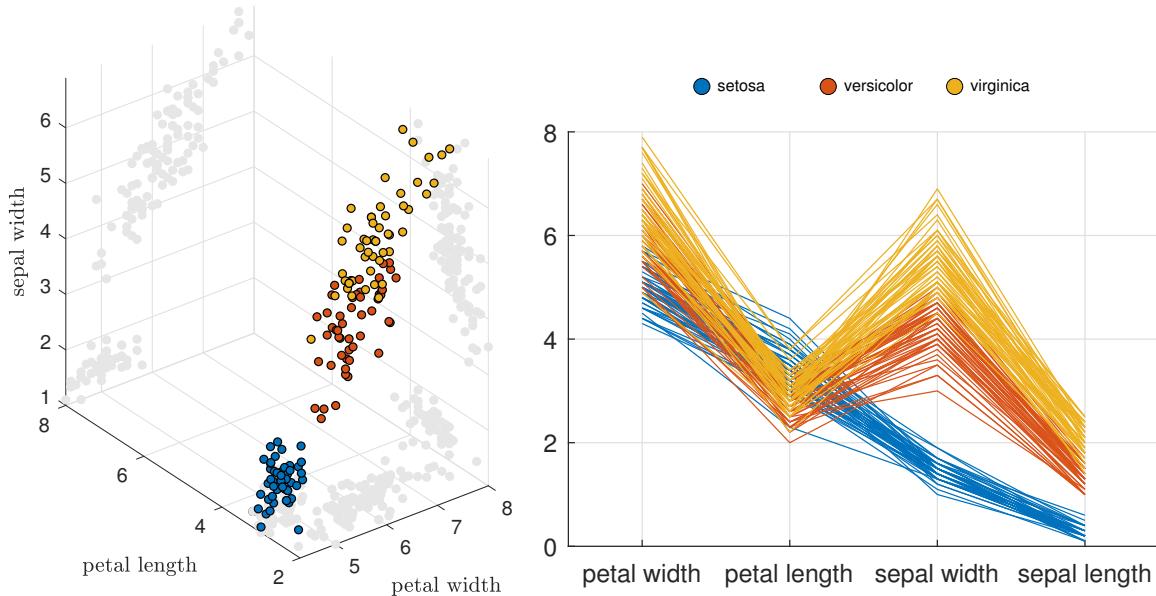


Fig. 7.7. Higher-dimensional objects are difficult to visualize meaningfully. To the left is shown a 3D scatter plot where three attributes are plotted against each other. 3D plots are best when one can interactively move the camera around since their 2D projection onto paper necessarily ruins much of the information that can be extracted. For more dimensions some creativity is required, for instance changing a 4D point to a line intersecting points given by coordinate number and the corresponding value as shown to the right.

is represented by a line passing through the 4 points with coordinates (coordinate \times value). As a rule, one should consider using selection or projection of the data set before including plots of high-dimensional observations.

7.2 What sets apart a good plot?

Having introduced the basic plots, a natural question to ask is when to use which plot type. Beyond the basic requirement our visualizations should provide a truthful summary of the data there is no single optimal answer to this question, however, there are useful guidelines².

A useful analogy is to consider technical writing. Suppose we are writing a section in a report. What are the relevant questions to keep in mind? Arguably, the first, and most important, question is what the *point* of the section actually is: What particular *question* are we hoping to answer? If we are unsure about what point we are trying to convey, the text will only confuse the reader, and we should be better off discarding the section entirely.

When we know which point we wish to convey the next element is *how* the section should address the question: Should we use examples? An abstract definition first and then illustrations? Draw on other parts of the text? Perhaps begin by explaining the reader why he or she should care?

² The guidelines illustrated here are adopted from http://junkcharts.typepad.com/junk_charts/

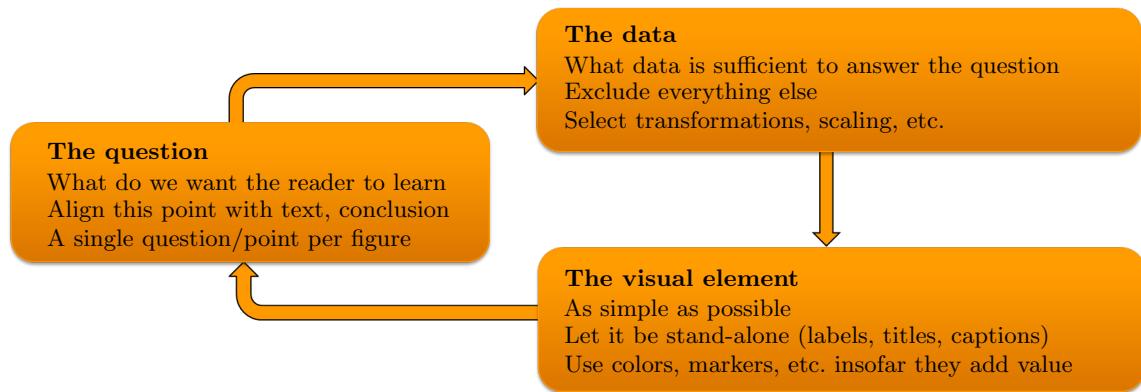


Fig. 7.8. A visualizations is first and foremost about providing a truthful summary of the data, but the clarity of a visualization can be greatly increased by carefully considering the following three points: (i) what question are we trying to answer? i.e. what conclusion do we wish to convey to the reader using the figure (ii) what is the minimal amount of data which will make that conclusion clear and how should it be pre-processed (if necessary) (iii) what visual element, colors, arrangement, etc. is more suitable.

After we have narrowed in on which question we want to answer, and how (in the broad picture) this will be accomplished, we get to the low-level issue of putting our thoughts into well-structured and readable English sentences. While this is arguably the least important aspect of writing³, it is certainly important to ensure the text is enjoyable or, as a bare minimum, readable.

For visualizations we can imagine a similar thought-process illustrated in fig. 7.8

The question: The most important aspect of a visualization is what question the graphical element will pose and answer. A figure should attempt to convey one (or a very few) interesting facts to the reader and nothing more. It should be aligned with the main conclusions of the text and be interesting enough to warrant the space.

The data: Next we should determine what data is useful to answer the question and possibly what transformations should be applied to the data to (say) reduce noise, change scale, etc. The rule is to go with the bare minimum of data transformations.

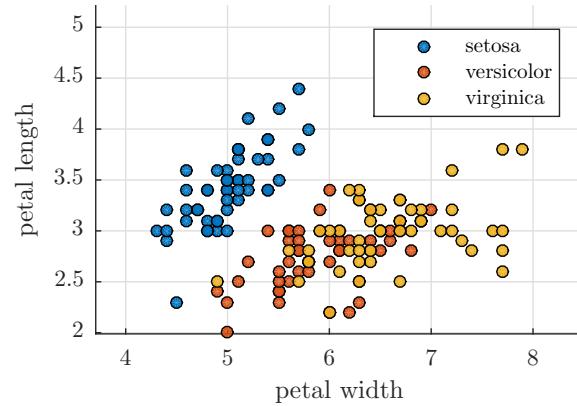


Fig. 7.5. Scatter plot of two attributes of the Fisher Iris data. Colors are used to visualize the three classes.

³ Or so we hope

The visuals: Lastly, what visual element (i.e. the type of plot) should be used to represent the data and answer the question. Preference should be given to simplicity. Consider how to make important visual feature stand out, use correct labels to guide the reader, etc.

There are many answers as to how these steps⁴. However, the main point is that *some thought is given to the process of making visualizations*. For instance, no person would hand in a report written in a font that was unreadable. However, it is common to see plots where axis or labels are quite literally impossible to read due to pixelization and poor font size choices. Errors of this kind, as well as the related mistake of omitting labels to axis where their labeling is non-obvious, are easily avoided with a minimum of afterthought.

As a final recommendation, consider in the future to occasionally ask yourself if a particular graphical element in a book or slide show works well (or not!) and ask yourself why and if any of the ideas are worth copying.

7.3 Visualizing the machine-learning workflow★

In this section, we will focus our attention on how visualizations can help us at the various stages of the machine-learning workflow we first encountered in fig. 1.13. Our hope is to convince the reader that *visualizations are useful when working with a practical machine-learning problem* as encountered later during the course and, therefore, building visualizations for our *own* benefit should become a natural part of our machine-learning toolkit and workflow.

As this is more of a general suggestion than a single, practical method we have chosen to make the point early and at once rather than scattering it throughout the text. However, this means the suggestions presented in this section will not in and by themselves be very helpful at this exact moment, and that throughout the section we will, for illustrative purpose, refer to machine-learning methods only introduced later. We emphasize a first-time reader is not supposed to understand the methods fully at this point, and a reader should focus their reading on the general point we try to convey rather than the specifics. Notice that this section (including subsections) is marked by a ★ indicating it is not necessary to understand the main text.

7.3.1 Visualizations to understand loss

To meaningfully distinguish between any two methods you need a well-defined goal. Therefore, begin by figuring out a quantifiable way to express one model is preferable to another. Sometimes this is trivial, but sometimes it is much harder: Suppose you are designing a method to automatically re-stock a supermarket. Is the goal to ensure *on average* there are always n packs of ground beef on the shelves, that on average no more than m packages of ground beef are discarded as waste, or that no more than p percent of the customers are unable to buy the product they came for?

⁴ An interested reader can consult the work of Edward Tufte (Tufte et al. [1990], see also <https://www.edwardtufte.com/tufte/>) or the ACCENT principles [Burn, 1993]

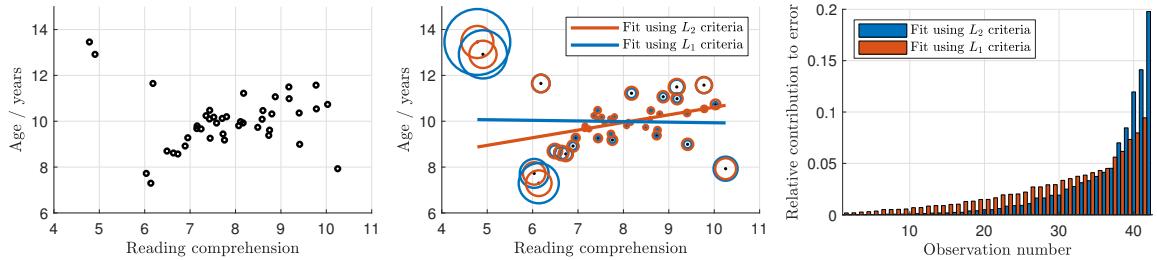


Fig. 7.9. Example of how the performance metric has a qualitative impact on the choice of model. For the 2D dataset shown in the left-most pane, we consider a simple model (a line) but using two performance metrics: One is the standard L_2 error typically used in regression, the other is the L_1 error defined as the sum-of-distances between the points and the line. The middle figure illustrates the best linear models using these two methods and the circles the *relative* contribution of each point to the total error (the histogram in the right-hand pane are the same values sorted in ascending order). The *optimal* model change qualitatively between a slightly positive/negative trend because the L_2 error is much more affected by outliers.

A more concrete example is given in fig. 7.9 which illustrates how different choices of loss shifts emphasis between outlier errors vs. typical error. The figure shows a simple prediction problem where the interpolating line is selected so as to minimize the sum-of-distances between the line and the observations, however, we try different measures of distance as defined by the L_p norm (see eq. (4.17) in chapter 4).

The figure illustrates the standard L_2 error (square loss) and the L_1 error (the sum-of-distances to prediction) and how this results in qualitatively different predictions.

A temptation is to take a wait-and-see attitude where several losses are computed, however this encourage working in an ad-hoc manner where one is unable to reliably track progress. Alternatively, it may seem impossible to find such a number, in which case one should wonder if the problem is too poorly specified.

Having a single, well-defined loss does not mean you cannot change your loss if it turns out to be poorly selected, or you should not compute alternative metrics as you go along. Doing so provides important information about what your method does and perhaps *does wrong*.

7.3.2 Use visualizations to understand mistakes

A simple but versatile idea is to visualize the outcome of the machine learning method. Suppose we are trying to distinguish between cars and cities. A thing that can always be done is to plot a meaningful number of members of these two classes to get an idea about what the images look like. Even more useful, once a method is set up plot the images that are *wrongly* classified. This is illustrated⁵ in fig. 7.10 where we have plotted 6 members of each class here assumed to be wrongly classified: we can immediately notice some interesting facts, such as the “cars” class contains images of the interior of cars, things that does not look like cars at all (lower, right), wrongly labeled images (lower, left or upper, right) and a motorbike. Similarly the city images contains things that do not look very city like; from this we can conclude (i) we should not expect our method to be 100%

⁵ Images obtained from <https://www.pexels.com>



Images from “car” class mislabelled by method



Images from “city” class mislabelled by method

Fig. 7.10. A simple way to visualize the result of a method is to picture wrongly classified observations. For instance suppose the above 12 images show (wrongly) classified instances of the cars and city class; this would tell us there are issues with the labelling (the motorcycle and truck), as well as hint as a possible methodological problem of assigning a single label to an image. One should perhaps also wonder if the method has a problem with very dark images; notice this could be further illuminated with other visualizations.

accurate (ii) perhaps it is more useful to measure performance by looking at the top-5 predicted labels as the labels are somewhat ambiguous.

7.3.3 Visualization to debug methods

Suppose your method does not work as well as you expect. Two recommended approaches is to attempt to use a simpler version of the method, for instance by reducing the number of neurons or layers in a neural network, or applying the method on a simplified version of the data set. Assuming this seems to improve the performance, one should suspect something is going wrong in the training of the method. Once more, we stress visualizations are the single most useful way to understand what might go wrong. Simply put, extracting information about what the method does internally should be the go-to technique, and doing so in a visual format is often the easiest and most re-useable way. An example is shown in fig. 7.11; we assume an artificial neural network (see chapter 15) is applied to a dataset and performs poorly. Neural networks often consist of thousands (or millions) of weights which are hard to visualize, but in the figure we try to plot the output of each layer as a histogram. We see that for the first layer, the histogram show a variety of activation of the neurons indicating they maintain some information about the input. In the third and 10th layer things look a bit askew: Neurons are now either fully on or off (-3 and 3) and we should wonder why. Is this natural? could it be this is arbitrary? where does the number 3 come from? Finally in layer 10 we see neurons all have nearly no activation, and once more we should wonder if this is what we expect.

More experience with neural networks will tell us a histogram such as layer 3 can be expected if the layer weights are initialized to have a too high value, and layer 10 if they have a too low initial value, how the true benefit of such a visualization is it allows us to *learn* such experience: it can readily be re-used when the network is applied to a simpler dataset where it is perhaps known to work well, and then we can see, even if highly approximately, what a good network is supposed to look like. if all layers look more or less as the layer 1 histogram, this should inform us something is going wrong, and we can begin to prod and poke the network to see what that is. Simply put,

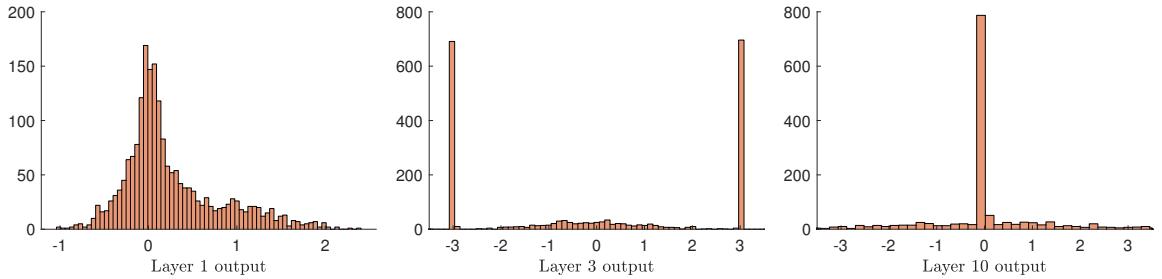


Fig. 7.11. Even models with millions of parameters can be meaningfully visualized with some creativity and with the right visual element. This example illustrate the activity of the neurons in three layers of a neural network. In the first layer, we see a broad distribution of activity indicating the neurons are probably exhibiting varying degrees of activity for the input as we would hope. The center and right-most pane show behaviour that should make us slightly suspicious: in the center pane, neurons are either fully on or off (obviously, we should ask ourselves if the number 3 is what we expect), and in the right-most pane nearly all neurons have no activity. Notice these histograms are not supposed to be a gold-standard for visualizing networks, but an illustration of what can be done with simple means.

when things go wrong, attempt to figure out the problem by writing code to *see* what goes wrong rather than sit in the armchair and figure out what *might* be wrong.

7.3.4 Use visualization for an overview

Visualizations can often provide an immediate overview of what to expect in terms of performance. When first encountering a machine-learning problem, we suggest a reader try to estimate two quantities:

The baseline performance

The *baseline performance* (or simply *baseline*) refers to the performance of a naive method. A well-chosen baseline should be quick to compute, simple and foolproof; think of it as what you would do if you had 10 minutes to solve the problem. Examples could be:

Classification Classify everything as belonging to the class with the most elements

Regression Output the mean of the sample (i.e. make a constant prediction)

Density estimation Return a uniform density, i.e. all outcomes are equally likely

Outlier detection Mark everything (or nothing) as outliers

These examples are very naive and refers to the situation where we simply ignore \mathbf{X} in our learning problem and only focus on \mathbf{y} . When working with more elaborate methods, for instance an artificial neural network (see chapter 15), it is common to use a simple linear model (see chapter 8) as baseline. Regardless of ones choice, the point of the baseline is twofold: First, to be able to recognize when a method is not learning anything or very little. This may seem silly, but it is not at all uncommon to encounter situations where a neural network is trained for hundreds of hours on giant datasets and still only outperform a linear classifier with a few percent.



Fig. 7.12. Scatterplot of $M = 11$ attributes considered pairwise and colored according to class labels. The inserted yellow line indicate the decision boundary of a simple classification method, logistic regression, and the corresponding errors are shown as inserts.

The target, or ceiling, performance

The baseline provides a lower bound of performance and the target (or ceiling) performance refers to an upper bound. Such an upper bound may arise for different reasons depending on the situation: For instance, it may refer to how well a human performs at the task, or how well we *have* to perform the task for our machine-learning method to be useful, or the performance of a comparable method from the literature. Alternatively, it may reflect some inherent noise in the problem, for instance we *cannot* expect to predict a persons test score with higher accuracy than the inherent noise in the test (such noise arise due to the limited number of questions or possibly arbitrariness of the

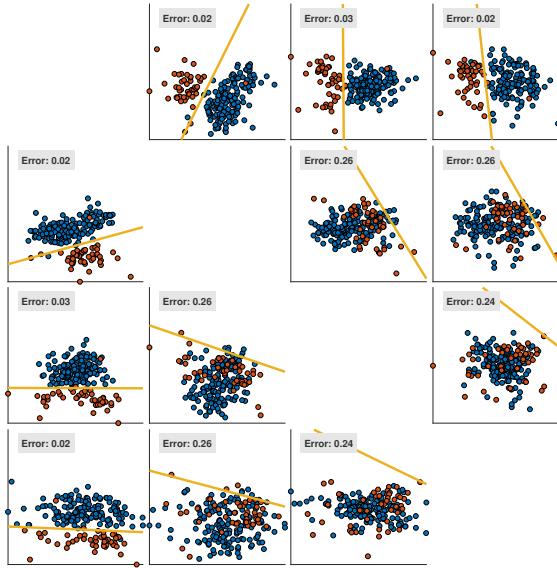


Fig. 7.13. Continuing the example of fig. 7.12, the figure indicates the pairwise plots of a dataset projected onto $K = 4$ PCA components. Note the first two components easily separate the classes and give rise to a very low error, whereas the other attributes appear to be fairly uninformative. A plot such as this may reveal important information about how relatively easy/difficult the classification problem is.

scoring), or predict the *exact* number of goals in a highly random game like soccer, or if 10% of the observations in a dataset are mislabeled we should not expect to be more than 90% accurate⁶.

The point of the baseline and ceiling performance is to get an intuitive feeling for what our, specific, performance means as well as how easy the problem is. Suppose we accurately predict if an image contains either a dog or a cat 91% of the time, but 90% of the dataset is comprised of dogs; then the performance improvement by using machine learning is about 1%, probably within chance levels and we should not be too happy. On the other hand, suppose the dataset contains 100 classes, each with the same number of elements, then the naive baseline level is about 1% and a performance of 15% mean the method is doing *something* (we will return to the point of class imbalance in chapter 16).

Example:

To make this concrete we will consider a 12-feature classification problem⁷ where the goal is to predict the wine type based on 12 different features.

In fig. 7.12 we have made a scatter plot of each pair of feature and colored the two classes. If we inspect the plots we can see several things: (i) some of the features (for instance feature 8 and 3) pairwise allow good separation (ii) others (such as feature 11 and 4) does not (the two point clouds

⁶ A technical point: If the observations are mislabeled in a predictable manner, for instance all wolves are mislabeled as dogs, we can learn to accurately predict the *wrong* label if that is all we have access to. What we refer to here is the *true* class.

⁷ The Wine dataset was collected by Cortez et al. [2009] and obtained from <http://archive.ics.uci.edu/ml/datasets/Wine+Quality> (note dataset has been processed to remove outliers).

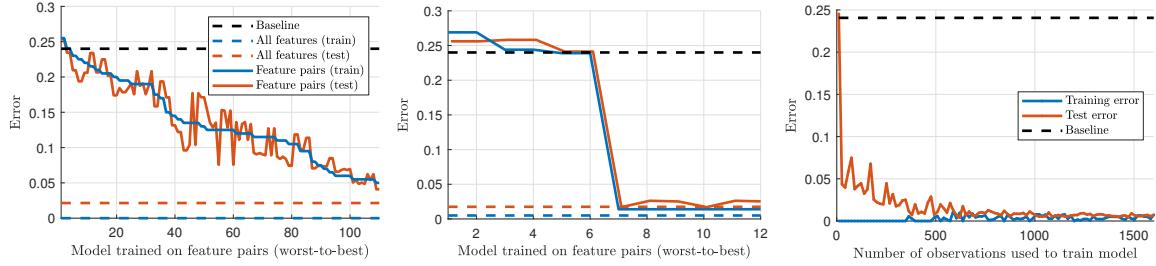


Fig. 7.14. Illustration of the relative difficulty of the classification task in fig. 7.12. Left: Dotted lines indicate baseline model as well as a model trained on all features. The blue and red curves indicates the classification error of each of the 55 plots in fig. 7.12; as shown these ranges from *worse* than random guessing to nearly as good as using all the data. A similar plot for the PCA-projected version of the same dataset is shown in fig. 7.13. The last figure illustrates how the method improves (in terms of test error) as more data is used to train a linear model using all the features.

are on top of each other). The lesson we can immediately draw from this illustration is the problem is feasible and relatively easy; it is more or less a matter of finding the right pair of features.

On the other hand, suppose we only had access to the pair of features shown in the bottom-right corner of fig. 7.12. In this case, we can see the two classes are so intermixed, and nothing suggests they can be separated with any sort of rule, that we should *at the very least* conclude the problem is very difficult and throwing one advanced method at the problem after another is unlikely to do us any good.

Obviously, for high dimensional data, eye-balling becomes infeasible and dimensionality reduction methods may be of use. In fig. 7.13 we have illustrated the data projected onto the first 4 principal components and again we clearly see the first two principal components would easily solve the problem to a high accuracy.

7.3.5 Illustration of baseline and ceiling performance

The simplest baseline is obtained by classifying everything as belonging to the *largest* class. In the wine example, such a baseline obtains an error of just 25%⁸; however the degree to which the classes are separated in for instance the first subplot of fig. 7.13 suggests we can do a lot better. For illustrative purpose we have visualized a simple, linear classifier (specifically, logistic regression, which we will introduce in chapter 8) where everything on one side of the line is classified as belonging to one class and everything on the other. The error of the classifier is shown for each pair of coordinates and we see the best pair of PCA components obtain an error of just 2% on the training set.

The error of all linear classifiers using pairs of features are shown in fig. 7.14 (left and middle figures). These plot show the baseline error 25% as well as the error obtained by training a model on all features (about 2.5%). Visually inspecting the randomness in the data in fig. 7.12 an error of 2.5% is probably quite close to the best we can hope to do on this dataset.

Between these two quantities we have illustrated the performance of a linear classifier trained on each pair of features on both a training and test set; we note the performance of these range

⁸ Because the largest class contains 75% of the observations

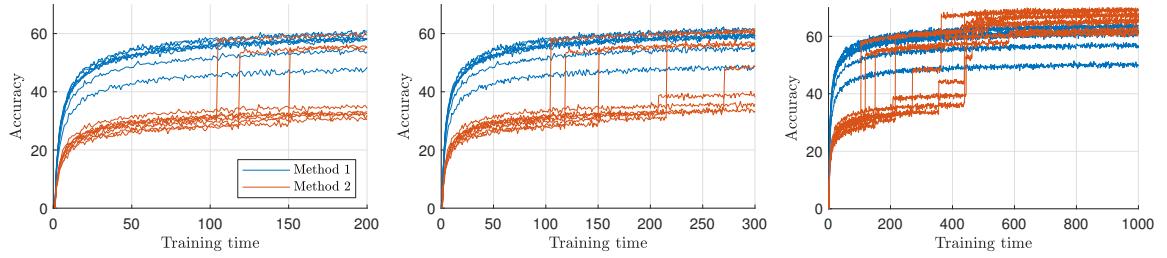


Fig. 7.15. While it is important to quantify model performance using a single number, visualizing learning curves can often reveal important facts about our method including how much we should trust these numbers as providing an absolute truth. The examples indicate separate runs of two methods, and the performance of the methods would normally be computed after learning (at $T = 200, 300, 1000$ respectively) and compared using a standard test. Such a test would reveal method 1 perform better than 2 in the first two panes ($p = 0.012$ and $p = 0.048$) ($T = 200, 300$), however inspecting the learning curve reveals such a conclusion would be highly spurious due to the non-normal behaviour of method 2 (the jumps) as well as the error is obviously not stationary. In $T = 1000$ we in fact see that method 2 seems to perform better than method 1 ($p = 0.0071$).

from abysmal (worse than the trivial baseline) to nearly as good as when using all features. For the PCA projections (middle) this is even more pronounced. What we learn from this example is the wine dataset represents a fairly simple problem and a linear classifier should (and will) do well.

7.3.6 Visualizing learning curves

We previously argued it is important to quantify the performance of a given method as a single number, however it is worth emphasizing that a single number can often hide important information about the method which visualizations can make apparent.

Example 1:

Continuing our wine-example from the previous section, in fig. 7.14 (right pane) we have shown how the error (on a training and test set) of a linear classifier depend on the number of training observations. What we see is an important, general, feature of a well-functioning machine-learning method: When using very few training observations, the method can fit the training data perfectly, but does not learn to generalize to the test set (because there is insufficient data to learn the true underlying statistical features of the problem; we say the method *overfits*). When more data becomes available, the error on the training set actually increases slightly (because the training set becomes more diverse and eventually exceeds the flexibility of a linear model), however the test error (which, recall, is an estimate of the generalization error which is the quantity we are interested in) drops. We will return to how such learning curves are interpreted in section 10.4.

Example 2:

A plot may provide insights which are not apparent from a statistical test, both because it is often intuitively easier to judge a plot than a p -value, but also more importantly, that the distribution of the error of our method may (and often, will) violate assumptions of the statistical test in more

or less pronounced ways, thereby significantly weakening (or completely invalidating) the utility of the statistical test.

An example of this is shown in fig. 7.15. This curve illustrates the performance (here: accuracy, higher is better) of two methods as a function of training time (it is assumed we are using a model that benefits from more training time; a large neural network is an example of such a method). A statistical test will show that at both time $T = 200$ and $T = 300$ method 1 outperforms method 2, however if we inspect the learning curves we see method 2 behaves differently than method 2; it makes discrete “jumps”. This kind of behaviour is in fact quite common for methods which operate on a discrete internal representation or in a discrete environment. At any rate, if we look at these curves we should quickly draw the conclusions that

- The error of method 2 is not normally distributed meaning our statistical test is only suggestive
- The need to evaluate method 2 longer and be aware performance of the two methods may shift depending on initial conditions
- there are good reasons to think tweaking of method 2, such as initial conditions or other learning parameters, can dramatically improve its relative performance.

Problems

7.1. Question 1: We will only use the attributes $x_1 - x_{10}$ as well as the output y in our modeling of the data. The attributes $x_1 - x_{10}$ are standardized (i.e., the mean has been subtracted each attribute and the attributes divided by their standard deviations). In Figure 7.16 a boxplot of the standardized data is given. Which of the following statements is *correct*?

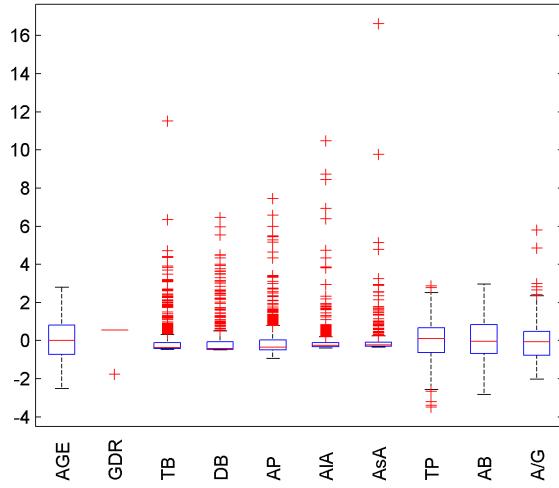


Fig. 7.16. Boxplots of the 10 attributes $x_1 - x_{10}$ where the data has been standardized.

- A The value of the 50th and 75th percentiles of the attribute DB coincides.
- B Even though the distribution of A1A and AsA may have a similar shape this does not imply that the two attributes are correlated.
- C The attribute TB is likely to be normal distributed.
- D The attribute GDR has a clear outlier that should be removed.
- E Don't know.

7.2. Question 2: A 1-dimensional dataset is composed of $N = 60$ observations; exactly 40 of these observations take the value 1, 10 take the value 2 and the remaining 10 observations take the value 3. Which of the four boxplots in fig. 7.17 is a boxplot of the dataset?

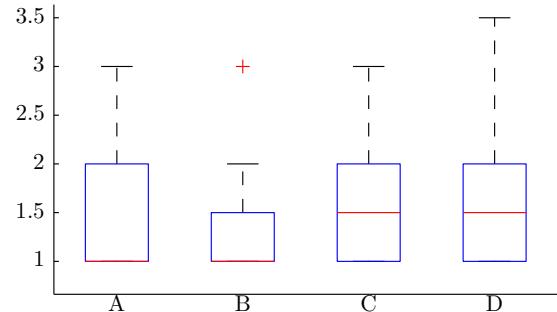


Fig. 7.17. Boxplots

- A Boxplot A
- B Boxplot B
- C Boxplot C
- D Boxplot D
- E Don't know.

7.3. Question 3: We will consider a dataset on wholesale taken from <http://archive.ics.uci.edu/ml/datasets/Wholesale+customers>. The data set includes 440 customers. The customers are from Lisbon and Oporto in Portugal as well as one additional region here denoted Other. The data provides the costumers' annual expenditures in monetary units of fresh products (FRESH), milk products (MILK), grocery products (GROCERY), frozen products (FROZEN), detergents and paper products (PAPER), and delicatessen products (DELI). The attributes of the data and their abbreviations are also given in Table 7.1.

In Figure 7.18 is given a boxplot of the six input attributes of the data. Which one of the following statements is *correct*?

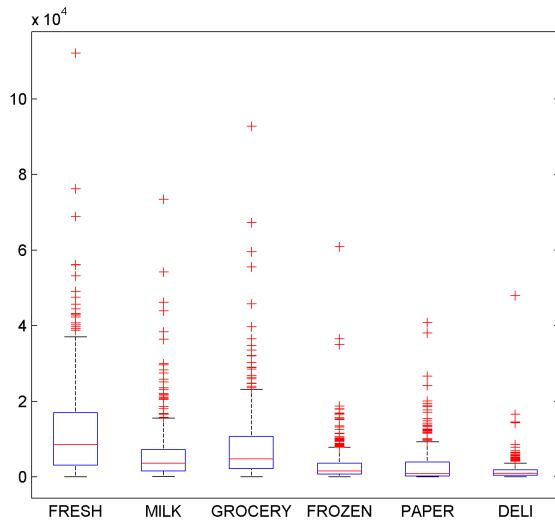


Fig. 7.18. Boxplot of the six input attributes x_1-x_6 of the wholesale data.

Table 7.1. The six input attributes x_1-x_6 denoting the annual consumption in monetary units of customers as well as the output y denoting which of the three regions; Lisbon, Oporto, and one additional region denoted Other, the customers came from in the wholesale customer data.

- A The boxplot contains prominent outliers that must be removed.
- B All the attributes appear to be normal distributed.
- C If we do not standardize the data (i.e., for each attribute subtract the mean and divide by the standard deviation) a PCA would give equal importance to all the attributes.
- D The mean and median values are not likely to be very close to each other for any of the attributes.
- E Don't know.

Part II

Supervised learning

Introduction to classification and regression

We will now turn our attention to *supervised learning*. In supervised learning we are given a training set comprised of N observations, $\mathbf{x}_1, \dots, \mathbf{x}_N$ and N targets y_1, \dots, y_N and we wish to come up with a way to predict y from \mathbf{x} :

$$y = f(\mathbf{x}, \mathbf{w}) + \varepsilon, \quad (8.1)$$

where \mathbf{w} is a vector of tunable parameters and ε represents a noise term. Learning then consists of selecting the parameters \mathbf{w} based on the training data \mathbf{X}, \mathbf{y} . If y is a continuous parameter, for instance the price of a stock, we will say that the model (denoted \mathcal{M}) is a regression model. On the other hand if y is discrete, i.e. $y = 1, 2, \dots, C$ as in the MNIST example we encountered in chapter 1, we will say \mathcal{M} is a classification model. In this chapter, we will discuss the linear and logistic regression models for regression and classification, starting by first explaining what $f(\mathbf{x}, \mathbf{w})$ looks like and then show how probabilities can be used to treat the noise term ε .

The history of linear regression can be traced back to mathematicians Adrien-Marie Legendre and Carl Friedrich Gauss who independently in 1805 and 1809 applied linear regression models to astronomical observations of the orbit of planets [Legendre, 1805, Gauß, 1809]. Meanwhile logistic regression, which we will consider in a later section, has its origin with the discovery of the logistic function by Pierre François Verhulst and Adolphe Quetelet in 1838 [Garnier and Quetelet, 1838] where it was originally applied to growth curves of populations.

8.1 Linear models

Despite having different goals, linear and logistic regression are closely related by virtue of using a linear transformation of the input features which will be our natural starting point. Recall in a linear model, the output y in eq. (8.1) is modelled as a *linear combination* of the input features:

$$f(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_M x_M, \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}. \quad (8.2)$$

The reason this is known as a linear model is that it is a *linear function* of the input features \mathbf{x} . To

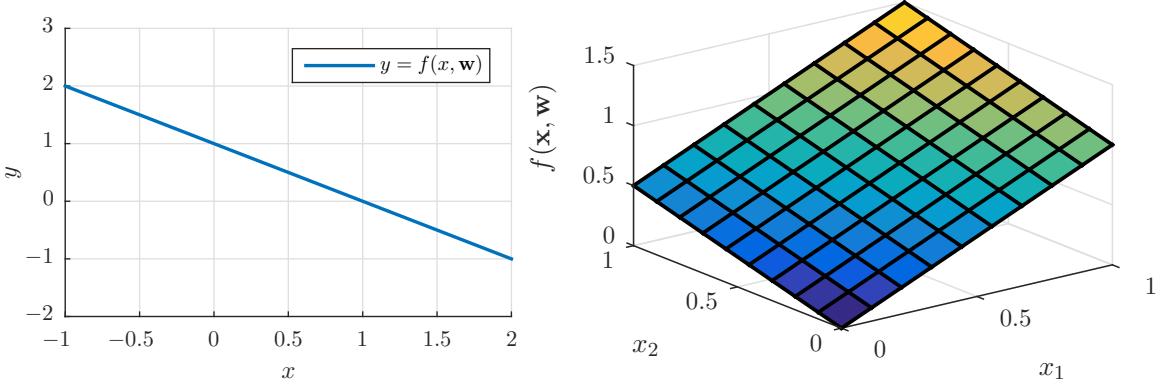


Fig. 8.1. The linear regression models prediction is a linear combination of the features $f(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \dots + w_M x_M$. This allows for lines (left pane), $y = w_0 + w_1 x$, planes (right pane) $y = w_0 + w_1 x_1 + w_2 x_2$ and in general hyperplanes.

consider a very simple example, consider the linear model with $w_0 = 1, w_1 = -1$ shown in fig. 8.1 as the blue line

$$y = f(x, \mathbf{w}) = 1 - x.$$

This naturally also extends to multiple input features. In the right-hand pane of fig. 8.1 is shown the two-dimensional regression example with $w_0 = 0, w_1 = 1, w_2 = \frac{1}{2}$.

$$y = f(\mathbf{x}, \mathbf{w}) = 0 + x_1 + \frac{1}{2}x_2 = x_1 + \frac{1}{2}x_2.$$

More generally, we can consider a feature transformation of \mathbf{x} such that

$$y = f(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

where $\phi_1(\mathbf{x}), \dots, \phi_{M-1}(\mathbf{x})$ are $M - 1$ basis functions. If we define

$$\phi(\mathbf{x}) = \begin{bmatrix} 1 \\ \phi_1(\mathbf{x}) \\ \vdots \\ \phi_{M-1}(\mathbf{x}) \end{bmatrix}$$

(that is, the first basis function is just a constant) we can write the linear regression model more compactly as simply

$$y = f(\mathbf{x}, \mathbf{w}) = \phi(\mathbf{x})^T \mathbf{w} \quad \text{and} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{M-1} \end{bmatrix}. \quad (8.3)$$

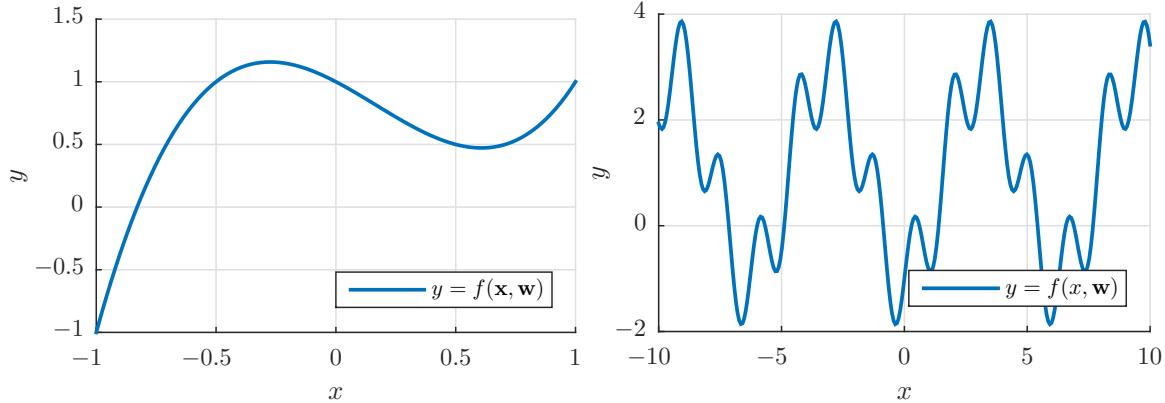


Fig. 8.2. Applying a non-linear transformation to the input x allows much more complicated curves to be fitted by the linear regression model. In the left-hand pane is shown a polynomial $y = w_0 + w_1x + w_2x^2 + w_3x^3$ and in the right-hand pane a sinusoidal model $y = w_0 + w_1 \cos(x) + w_2 \sin(4x)$.

The use of non-linear basis functions allow the linear regression model to model non-linear features. In fig. 8.2 is shown two such examples, the first is a 3rd degree polynomial corresponding to

$$\phi(x) = [1 \ x \ x^2 \ x^3]^T \quad \text{and} \quad \mathbf{w} = [1 \ -1 \ -1 \ 2]^T. \quad (8.4)$$

The second example corresponds to two trigonometric functions suitable for a periodic signal

$$\phi(x) = [1 \ \cos(x) \ \sin(4x)]^T \quad \text{and} \quad \mathbf{w} = [1 \ -2 \ 1]^T. \quad (8.5)$$

Since the transformation by a basis function does not change the linearity in \mathbf{w} the discussion in this chapter will be independent on the choice of basis functions. In practical terms, applying a basis functions to a dataset \mathbf{X} to obtain the transformed dataset $\tilde{\mathbf{X}}$ is equivalent to applying feature transformations such as the ones we encountered in chapter 2

$$\tilde{\mathbf{x}}_i = \phi(\mathbf{x}_i), \quad \text{and} \quad \tilde{\mathbf{X}} = \begin{bmatrix} \phi(\mathbf{x}_1)^T \\ \phi(\mathbf{x}_2)^T \\ \vdots \\ \phi(\mathbf{x}_N)^T \end{bmatrix}$$

and we can write the prediction of observation i as $y_i = \tilde{\mathbf{x}}_i^T \mathbf{w}$.

8.1.1 Training the linear regression model

To learn the parameters of the linear regression model, we will follow the general procedure outlined in section 6.5, in particular eq. (6.47). The first step of which is to come up with an expression for $p(y|\mathbf{x}, \mathbf{w})$ (see eq. (6.45)).

To this end, note the linear regression model eq. (8.3) is an ideal relationship between the input \mathbf{x} and the target y . An actual observation will be noisy which we will capture with a noise parameter ε :

$$y = f(\mathbf{x}, \mathbf{w}) + \varepsilon = \tilde{\mathbf{x}}^\top \mathbf{w} + \varepsilon$$

Since we don't know what ε is, we have to model it with our only tool for handling unknown quantities: probabilities. Therefore, assume that for each observation ε follows a normal distribution with mean 0 and variance σ^2 . Using this assumption, the probability density of y is then a normal distribution centered around $\tilde{\mathbf{x}}^\top \mathbf{w}$:

$$\begin{aligned} p(y|\mathbf{x}, \mathbf{w}, \sigma) &= \mathcal{N}(y - \tilde{\mathbf{x}}^\top \mathbf{w} | \mu = 0, \sigma^2) \\ &= \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y - \tilde{\mathbf{x}}^\top \mathbf{w})^2}{2\sigma^2}} \end{aligned} \quad (8.6)$$

Our objective (see eq. (6.40)) is to find \mathbf{w} as the value which is most plausible given the data, i.e. as maximizing $p(\mathbf{w}|\mathbf{X}, \mathbf{y})$. Using Bayes' theorem this can be written as

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})}. \quad (8.7)$$

As we saw earlier (see eq. (6.46)), this is equivalent to selecting \mathbf{w}^* as the value which *maximizes*

$$\mathbf{w}^* = \arg \max_{\mathbf{w}} \left[\log p(\mathbf{w}) + \sum_{i=1}^N \log p(y_i|\mathbf{x}_i, \mathbf{w}) \right] \quad (8.8)$$

$$= \arg \max_{\mathbf{w}} \left[\log p(\mathbf{w}) - \frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^N (y_i - \tilde{\mathbf{x}}_i^\top \mathbf{w})^2 \right]. \quad (8.9)$$

We will now assume the prior of \mathbf{w} is flat, $p(\mathbf{w}) = 1$, which is also known as the *uniform* or *improper* prior¹, and furthermore we will choose to formulate the problem as a minimization problem by dropping constant terms and re-scaling the above expression. The problem of finding \mathbf{w}^* is therefore equivalent to *minimizing* the *sum-of-squares* error function (compare to eq. (6.47)):

$$\begin{aligned} \mathbf{w}^* &= \arg \max_{\mathbf{w}} p(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \arg \min_{\mathbf{w}} E(\mathbf{w}) \\ \text{where } E(\mathbf{w}) &= \frac{1}{N} \sum_{i=1}^N (y_i - \tilde{\mathbf{x}}_i^\top \mathbf{w})^2. \end{aligned} \quad (8.10)$$

As we know from analysis, this can be accomplished by setting the derivative of E equal to zero and solving for \mathbf{w} . If we differentiate eq. (8.10) with respect to a weight w_j we obtain:

$$\frac{\partial}{\partial w_j} E(\mathbf{w}) = \frac{2}{N} \sum_{i=1}^N (y_i - \tilde{\mathbf{x}}_i^\top \mathbf{w}) \tilde{\mathbf{x}}_{ij} = \frac{2}{N} \sum_{i=1}^N y_i \tilde{X}_{ij} - \frac{2}{N} \left[\sum_{i=1}^N X_{ij} \tilde{\mathbf{x}}_i^\top \right] \mathbf{w} \quad (8.11)$$

The gradient is therefore

¹ Notice the choice $p(\mathbf{w}) = 1$ strictly speaking does not make sense since the density is no longer normalized: $\int p(\mathbf{w}) d\mathbf{w} = \infty$. The prior can however be understood as saying that no particular value of \mathbf{w} is preferred over another or more formally it can be understood as using the prior $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, I\delta)$ throughout the derivation and then taking the limit $\delta \rightarrow \infty$.

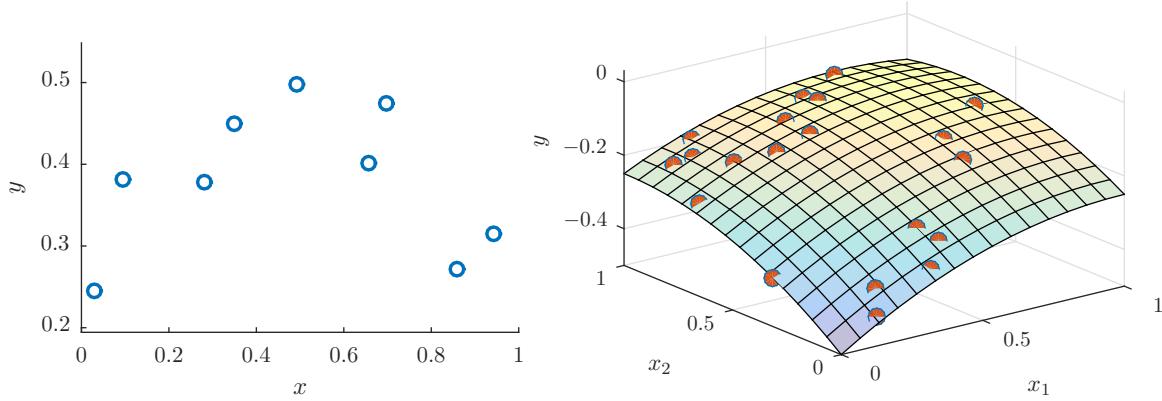


Fig. 8.3. Examples of two datasets for which we will apply linear regression. In the left-hand pane is a 1-d dataset comprised of x , in the right-hand side a 2d dataset comprised of red dots lying on a curved plane.

$$\nabla E(\mathbf{w}) = \begin{bmatrix} \frac{\partial E(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial E(\mathbf{w})}{\partial w_M} \end{bmatrix} = \frac{2}{N} \tilde{\mathbf{X}}^T \mathbf{y} - \frac{2}{N} (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}) \mathbf{w}. \quad (8.12)$$

Setting the gradient equal to zero and solving we obtain

$$\mathbf{w}^* = (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \mathbf{y} = (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}) \setminus \tilde{\mathbf{X}}^T \mathbf{y}. \quad (8.13)$$

Thus, training the linear regression model can be accomplished using one line of code in many computing environments. Since the linear regression model is so basic and important we will illustrate it in two scenarios in the following.

Example 1: Linear regression applied to a 1d dataset

Consider the 1d dataset shown in the left-pane of fig. 8.3. Suppose we wish to fit two models to the dataset, one corresponding to plain linear regression and the other to feature transforming the data to correspond to a second-degree polynomial.

For the first order polynomial linear regression case, this is accomplished by applying the (identity) feature transformation:

$$\tilde{\mathbf{X}}_{(1)} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} \quad (8.14)$$

then we compute $\mathbf{w}_{(1)} = (\tilde{\mathbf{X}}_{(1)}^T \tilde{\mathbf{X}}_{(1)}) \setminus \tilde{\mathbf{X}}_{(1)}^T \mathbf{y}$ and the red curve for an arbitrary point x can be predicted as $y = f(x, \mathbf{w}_{(1)}) = [1 \ x] \mathbf{w}_{(1)}$. In the right-hand pane of fig. 8.4 we illustrate the second-degree polynomial linear regression corresponding to the feature transformation:

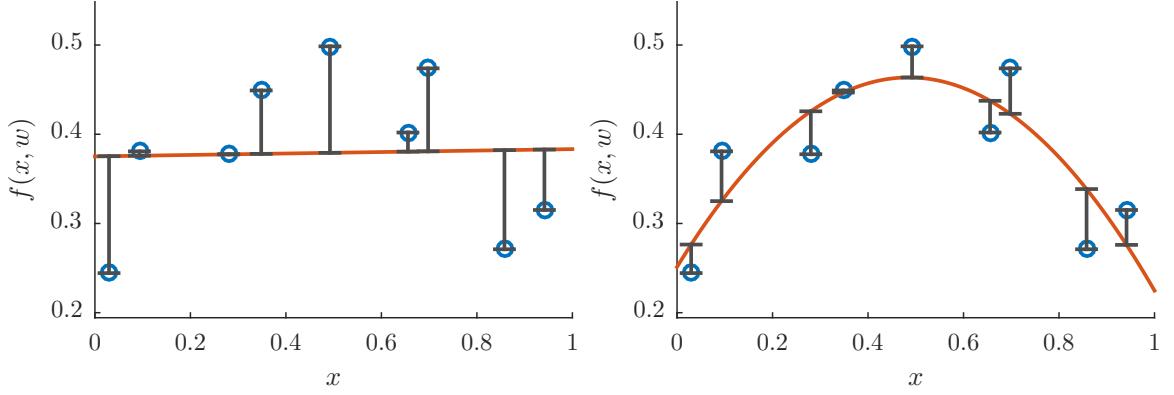


Fig. 8.4. Examples of applying the linear regression model to the dataset shown in the left-hand pane of fig. 8.3. The two panes respectively show a basic linear regression model $\mathbf{y} = \tilde{\mathbf{X}}_{(1)}\mathbf{w}_{(1)}$ without feature transformations, and linear regression model with feature transformation by adding the feature x^2 to produce a second-polynomial curve, $\mathbf{y} = \tilde{\mathbf{X}}_{(2)}\mathbf{w}_{(2)}$. See text for details.

$$\tilde{\mathbf{X}}_{(2)} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{bmatrix} \quad (8.15)$$

then we compute $\mathbf{w}_{(2)} = (\tilde{\mathbf{X}}_{(2)}^T \tilde{\mathbf{X}}_{(2)}) \setminus \tilde{\mathbf{X}}_{(2)}^T \mathbf{y}$ and the red curve for an arbitrary point x can be predicted as $y = f(x, \mathbf{w}_{(2)}) = [1 \ x \ x^2] \mathbf{w}_{(2)}$.

Example 2: Linear regression applied to a 2d dataset

In the second example, we will consider the 2d dataset shown in the right-hand pane of fig. 8.3. We will again consider two models, one corresponding to plain linear regression and the other to feature transforming the data to correspond to a second order Taylor expansion.

In the left-hand pane of fig. 8.5 we illustrate the second-degree polynomial linear regression corresponding to the (trivial) feature transformation:

$$\tilde{\mathbf{X}}_{(1)} = \begin{bmatrix} 1 & X_{11} & X_{12} \\ 1 & X_{21} & X_{22} \\ \vdots & \vdots & \vdots \\ 1 & X_{N1} & X_{N2} \end{bmatrix}, \quad (8.16)$$

then we compute $\mathbf{w}_{(1)} = (\tilde{\mathbf{X}}_{(1)}^T \tilde{\mathbf{X}}_{(1)}) \setminus \tilde{\mathbf{X}}_{(1)}^T \mathbf{y}$ (notice \mathbf{w} is three-dimensional) and for an arbitrary point \mathbf{x} we can predict $y = f(\mathbf{x}, \mathbf{w}_{(1)}) = [1 \ x_1 \ x_2] \mathbf{w}_{(1)}$. This is used to generate the plane shown in the left-hand pane of fig. 8.5.

In the second example, we will attempt to fit a second-order expansion to the same dataset. This is accomplished by the feature transformation:

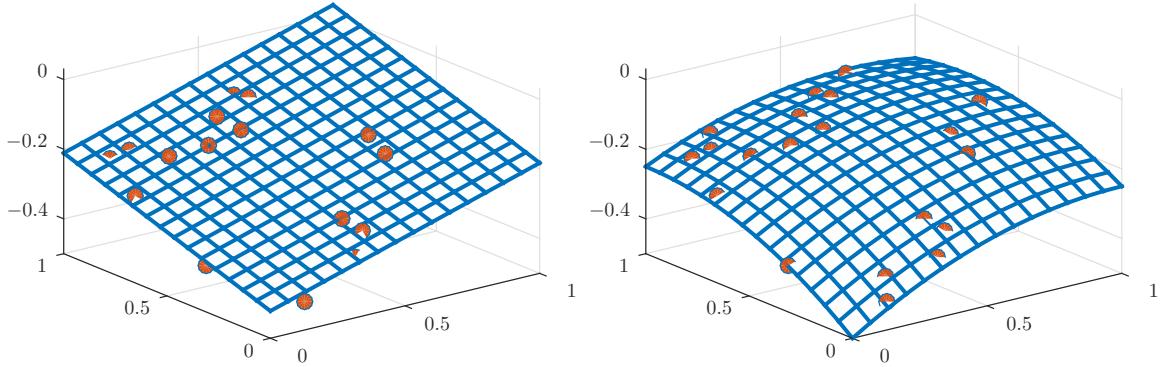


Fig. 8.5. Examples of applying the linear regression model to the dataset shown in the right-hand pane of fig. 8.3. The left-hand pane shows the basic linear regression model $\mathbf{y} = \tilde{\mathbf{X}}_{(1)}\mathbf{w}$, and in the right-hand pane we make a feature transformation to include second order terms corresponding to $\mathbf{y} = \tilde{\mathbf{X}}_{(2)}\mathbf{w}$. See text for details.

$$\tilde{\mathbf{X}}_{(2)} = \begin{bmatrix} 1 & X_{11} & X_{12} & X_{11}^2 & X_{12}^2 & X_{11}X_{12} \\ 1 & X_{21} & X_{22} & X_{21}^2 & X_{22}^2 & X_{21}X_{22} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_{N1} & X_{N2} & X_{N1}^2 & X_{N2}^2 & X_{N1}X_{N2} \end{bmatrix} \quad (8.17)$$

Again, learning \mathbf{w} (which is now six-dimensional!) can be accomplished as $\mathbf{w}_{(2)} = (\tilde{\mathbf{X}}_{(2)}^T \tilde{\mathbf{X}}_{(2)}) \backslash \tilde{\mathbf{X}}_{(2)}^T \mathbf{y}$ and predictions, as shown in the right-hand pane of fig. 8.5, for a new point $\mathbf{x} = [x_1 \ x_2]^T$ can be made as

$$y = f(\mathbf{x}, \mathbf{w}_{(2)}) = [1 \ x_1 \ x_2 \ x_1^2 \ x_2^2 \ x_1x_2] \mathbf{w}_{(2)}.$$

In the later case the found value of $\mathbf{w}_{(2)}$ is

$$\mathbf{w}_{(2)} = [-0.5 \ 0.5 \ 0.5 \ -0.25 \ -0.25 \ -0.125]^T,$$

which is exactly (at this precision at least) equal to the value of \mathbf{w} used to generate the data.

8.2 Logistic Regression

The goal of classification and regression may seem very different, however, it turns out linear regression can easily be extended to classification by the use of probabilities. Consider a binary classification task where $y = 0$ corresponds to the negative class and $y = 1$ to the positive class. We will now proceed exactly as we did in the case of linear regression by first finding an expression for

$$p(y|\mathbf{x}, \mathbf{w}) \quad (8.18)$$

and apply the maximum likelihood framework from section 6.5.

Since y is binary, our immediate idea should be to model its density eq. (8.18) as a Bernoulli variable. However, as the output of a linear model is a general continuous number, and the parameter

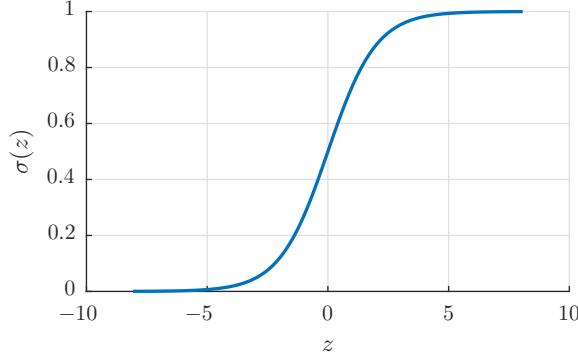


Fig. 8.6. The logistic sigmoid $\sigma(z) = (1 + e^{-z})^{-1}$. Notice as $z \rightarrow -\infty$ then $\sigma(z) \rightarrow 0$ and when $z \rightarrow \infty$ then $\sigma(z) \rightarrow 1$.

θ of the Bernoulli distribution belongs to the unit interval $[0, 1]$, we re-parameterize the Bernoulli distribution using the sigmoid function. Recall that according to eq. (5.30) from section 5.4.3 the Bernoulli density could be written as:

$$p(b|z) = \text{Bernoulli}(b|\theta = \sigma(z)) = \sigma(z)^b(1 - \sigma(z))^{1-b}, \quad \sigma(z) = \frac{1}{1 + e^{-z}}. \quad (8.19)$$

Where the function $\sigma(z)$ is the *logistic sigmoid* and is shown in fig. 8.6. The way we will proceed is to define z as being equal to the output of a standard linear model $\tilde{\mathbf{x}}^\top \mathbf{w}$. Specifically, define:

$$\hat{y}_i = \sigma(\tilde{\mathbf{x}}_i^\top \mathbf{w})$$

The probability density of a given observation y_i is then:

$$p(y_i|\mathbf{x}_i, \mathbf{w}) = \text{Bernoulli}(y_i|\hat{y}_i) = \hat{y}_i^{y_i}(1 - \hat{y}_i)^{1-y_i}. \quad (8.20)$$

If we then proceed exactly as in the case of the linear regression model, by using eq. (8.20) and eq. (6.47), we see we should select the parameters \mathbf{w}^* as the solution of the optimization problem:

$$\begin{aligned} \mathbf{w}^* &= \arg \min_{\mathbf{w}} E(\mathbf{w}) \\ \text{where: } E(\mathbf{w}) &= -\frac{1}{N} \log \left[\prod_{i=1}^N p(y_i|\mathbf{x}_i, \mathbf{w}) \right] \\ &= -\frac{1}{N} \sum_{i=1}^N [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)], \quad \hat{y}_i = \sigma[\tilde{\mathbf{x}}_i^\top \mathbf{w}], \end{aligned} \quad (8.21)$$

However, at this point our discussion has to depart from linear regression: there is no closed-form analytical solution for the minimum of the error function. How we find \mathbf{w}^* in practice will be discussed later in the chapter for neural networks where we will consider a general method for minimizing error functions, however, until then simply rely on the commands build into your computing environment for solving logistic regression problems.

As mentioned, all the tricks of feature-transforming \mathbf{X} also “work” for logistic regression and we will therefore only consider two simple examples where \mathbf{X} has not had feature-transformations

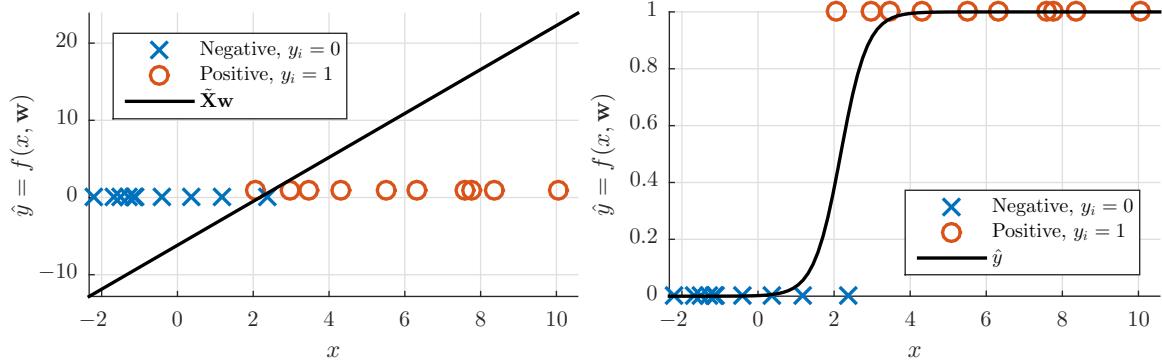


Fig. 8.7. A one-dimensional logistic regression example. The left-hand pane shows the intermediate (linear) output $\mathbf{X}\mathbf{w}$ and the right-hand pane the true logistic-regression decision boundary $\hat{y} = \sigma(\mathbf{X}\mathbf{w})$.

applied to it aside from being pre-fixed with 1s as is required for any regression problem. In fig. 8.7 is shown a basic logistic regression example for a simple 1-dimensional dataset. The procedure is exactly similar to linear regression. We first define:

$$\tilde{\mathbf{X}}_{(1)} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} \quad (8.22)$$

Then, the left-hand pane shows the intermediate linear regression value $z = \tilde{\mathbf{X}}_{(1)}\mathbf{w}$ as the black line, and the right-hand pane the predicted probabilities $\hat{y} = \sigma(\tilde{\mathbf{X}}_{(1)}\mathbf{w})$ are plotted as a black line. As expected for such a simple problem the logistic regression learns how to separate the two classes. For completeness, fig. 8.8 illustrates a 2d example, where the two classes are fitted with a logistic regression model and the decision surface \hat{y} is shown. In the example, the class-membership probabilities are computed as:

$$\hat{y}_i = \sigma(\tilde{\mathbf{x}}_i^\top \mathbf{w}) = \sigma([1 \ X_{i1} \ X_{i2}] \mathbf{w}).$$

Notice the decision boundary is linear and quite steep. Logistic regression will have linear decision boundaries unless we apply (non-linear) feature transformations to our dataset.

8.2.1 The confusion matrix

While the error function $E(\mathbf{w})$ could be used to evaluate a logistic regression model it is important to keep in mind that the error function measures the *probability* of the data, however, at the end of the day, we are probably more interested in how often the logistic regression model classifies correctly and how often it classifies wrongly. To this end, we must turn the output of the logistic regression (which is a probability) into a class label. This can be accomplished by simply thresholding at 0.5 (we will return to how the threshold should be selected in chapter 10) and predict that observation i belongs to the positive class if $\hat{y}_i > \frac{1}{2}$ and the negative class if $\hat{y}_i \leq \frac{1}{2}$.

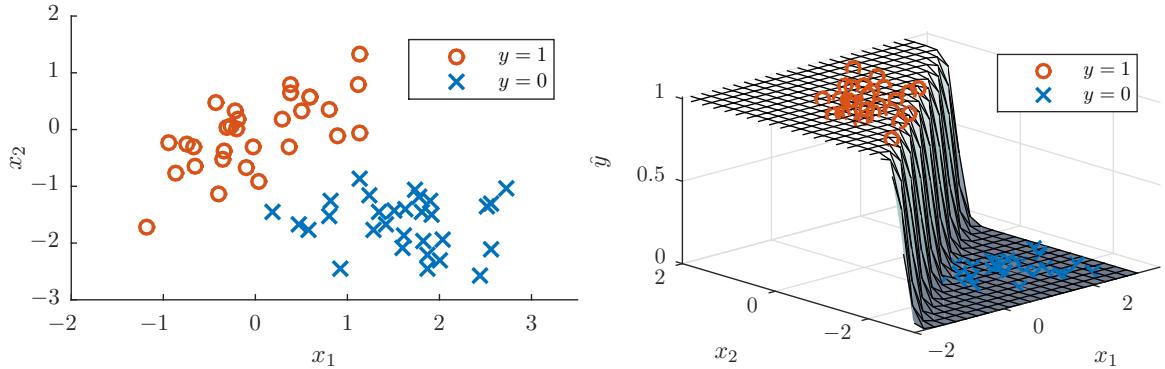


Fig. 8.8. 2d logistic regression example. The dataset shown in the left-hand pane is fitted with a logistic regression model and the class-membership prediction \hat{y} is shown in the right-hand pane.

There are now four different combinations of what class an observation actually belongs to and what it is predicted to belong to by the model. They are called:

True Positives, TP: Number of observations which are in fact positive $y_i = 1$ which the classifier correctly labels as positive $\hat{y}_i > \frac{1}{2}$

False Positives, FP: Number of observations which are in fact negative $y_i = 0$ which the classifier incorrectly labels as positive $\hat{y}_i > \frac{1}{2}$

False Negatives, FN: Number of observations which are in fact positive $y_i = 1$ which the classifier incorrectly labels as negative $\hat{y}_i < \frac{1}{2}$

True Negatives, TN: Number of observations which are in fact negative $y_i = 0$ which the classifier correctly labels as negative $\hat{y}_i < \frac{1}{2}$

These are illustrated in fig. 8.9. In the left-hand pane is shown a binary classification problem of $N = 10$ observations where the colors indicate the predictions made by a logistic regression model (blue corresponds to the positive class and red to the negative). In the right-hand pane the true positives, false negatives, etc. are collected in what is known as the *confusion matrix*, where the inserted ticks on colored background indicate which observations counts towards which numbers. As mentioned we will return to the meaning of these numbers in more detail in chapter 10, however, for now we will simply focus on *how often the classifier is right* which is known as the *accuracy* (or equivalently *how often the classifier is wrong* which is known as the *error rate*):

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{N}, \quad \text{Error rate} = \frac{\text{FP} + \text{FN}}{N} = 1 - \text{Accuracy}.$$

For instance the accuracy of the logistic regression model in fig. 8.9 is $\frac{5+2}{10} = \frac{7}{10}$.

8.3 The general linear model★

The overall form of the cost-function in the linear and logistic regression models can be generalized into what is known as the *general linear model*. Since this is the form of these models which is mostly commonly encountered in a computing environment, we will briefly discuss it here. Basically,

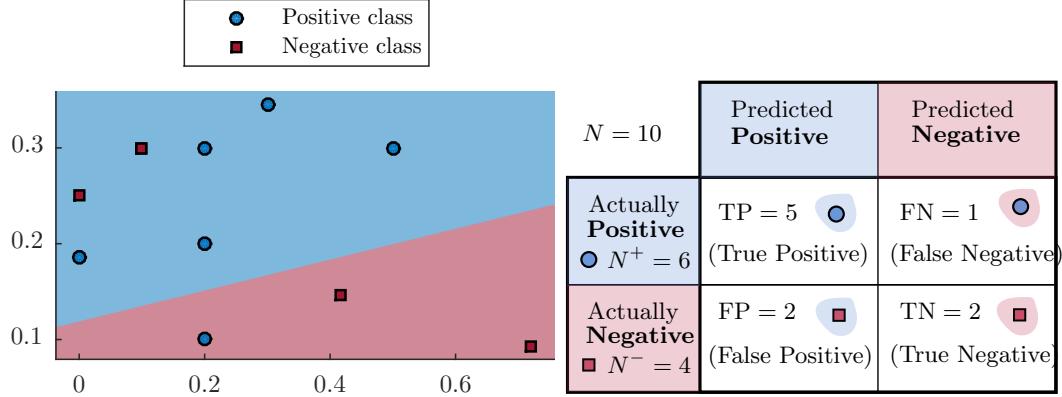


Fig. 8.9. (Left:) A small $N = 10$ observation binary classification problem. The colors indicate the prediction made by a logistic regression classifier obtained by thresholding \hat{y}_i at $\frac{1}{2}$. (Right:) The confusion matrix of the classifier in the left-hand pane. The inserts (ticks on background) indicate which observations counts towards which numbers in the confusion matrix.

it decompose the model into two parts: a *link* function g and a *cost function* d . It then assumes the output of the model is:

$$y = f(\mathbf{x}, \mathbf{w}) = g(\tilde{\mathbf{x}}^\top \mathbf{w})$$

and that the cost function can be written as:

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N d(y_i, g(\tilde{\mathbf{x}}_i^\top \mathbf{w})) \quad (8.23)$$

Parameters are found in the usual way by minimizing $E(\mathbf{w})$. For a summary, see Box 8.3.1.

Method 8.3.1: The general linear model

Given a dataset consisting of N pairs (\mathbf{x}_i, y_i) , we first define $\tilde{\mathbf{x}}_i$ as a feature-transformation of \mathbf{x}_i , in the simplest form obtained by pre-fixing \mathbf{x}_i with a constant intercept term:

$$\tilde{\mathbf{x}}_i = [1 \ \mathbf{x}_i^\top]^\top.$$

The predicted output \hat{y} , cost function E and learned parameters \mathbf{w}^* in a general linear model (GLM) are then defined as:

$$\begin{aligned}\hat{y} &= g(\tilde{\mathbf{x}}^\top \mathbf{w}) \\ E(\mathbf{w}) &= \frac{1}{N} \sum_{i=1}^N d(y_i, g(\tilde{\mathbf{x}}_i^\top \mathbf{w})) \\ \mathbf{w}^* &= \arg \min_{\mathbf{w}} E(\mathbf{w}).\end{aligned}$$

The *linear regression model* can be recovered by defining

$$g(\tilde{\mathbf{x}}^\top \mathbf{w}) = \tilde{\mathbf{x}}^\top \mathbf{w} \quad \text{and} \quad d(y, \hat{y}) = (y - \hat{y})^2$$

and the *logistic regression model* as:

$$g(\tilde{\mathbf{x}}^\top \mathbf{w}) = \sigma(\tilde{\mathbf{x}}^\top \mathbf{w}) \quad \text{and} \quad d(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}).$$

Problems

8.1. Question 1: We fit a linear regression model to the PM10 data shown in Table 8.1. The input attributes are standardized (i.e., we have subtracted the mean of each input attribute, x_1 – x_7 , and divided by their standard deviations) whereas the output logPM10 is kept in its original format. We obtain the following model:

$$f(\mathbf{x}) = 3.27 + 0.36x_1 - 0.01x_2 - 0.19x_3 + 0.01x_4 + 0.05x_7.$$

Which one of the following statements about the model is *incorrect*?

No.	Attribute description	Abbrev.
x_1	Logarithm of number of cars per hour	logCAR
x_2	Temperature 2 meter above ground (degree Celsius)	TEMP
x_3	Wind speed (meters/second)	WIND
x_4	Temperature difference between 25 and 2 meters (degree Celsius)	TEMPDIF
x_5	Wind direction (degrees between 0 and 360)	WINDDIR
x_6	Whole hour of the day	HOUR
x_7	Day number from October 1, 2001	DAY
y	Logarithm of PM10 concentration	logPM10

Table 8.1. The attributes of the PM10 data. The output is given by the hourly values of the logarithm of the concentration of PM10 particles (logPM10).

- A According to the model WINDDIR and HOUR are not relevant for predicting the pollution level.
- B According to the model fewer cars and more wind will result in lower pollution levels.
- C According to the model it seems that pollution is decreasing over time.
- D According to the model higher temperatures will result in lower pollution levels.
- E Don't know.

8.2. Question 2: We consider the Wholesale dataset shown in Table 8.2 and wish to predict whether a consumer is from Lisbon ($y=0$) or Oporto ($y=1$) by discarding observations from the Other region included in the wholesale data. After discarding the observations pertaining to the Other region we standardize the attributes x_1 – x_6 (i.e., for each attribute subtract the mean and divide by the standard deviation) and fit a logistic regression model. We obtain the following model for the prediction of the origin of the consumer:

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = \text{logit}(-0.51 - 0.11x_1 - 0.36x_2 + 0.44x_3 + 0.39x_4 + 0.09x_5 - 0.28x_6),$$

where $\text{logit}(w) = \frac{1}{1+\exp(-w)}$ is the logit function. Which one of the following statements about the model is *correct*?

No.	Attribute description	Abbrev.
x_1	Fresh products	FRESH
x_2	Milk products	MILK
x_3	Grocery products	GROCERY
x_4	Frozen products	FROZEN
x_5	Detergents and paper products	PAPER
x_6	Delicatessen products	DELI
y	Region	REGION

Table 8.2. The six input attributes x_1 – x_6 denoting the annual consumption in monetary units of customers as well as the output y denoting which of the three regions; Lisbon, Oporto, and one additional region denoted Other, the customers came from in the wholesale customer data.

- A According to the model it seems that people in Lisbon buy more FRESH products, MILK products and DELI products than people in Oporto.
- B According to the model if a costumer after the standardization has $x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 0$ the customer is more likely to come from Oporto than Lisbon.
- C The logit function will return the probability a person is from Lisbon.
- D From the model it can be seen that FRESH and PAPER are unimportant and should be removed in order to avoid overfitting.
- E Don't know.

8.3. Question 3: We consider the Galapagos dataset shown in Table 8.3 fit with a linear regression model which predicts the area of an island x_3 based on the remaining attributes, i.e. $x_1, x_2, x_4, x_5, x_6, x_7$. We obtain the following model for the prediction of an islands area (x_3) using the raw untransformed attributes that are plotted in Figure 8.10:

$$f(x_1, x_2, x_4, x_5, x_6, x_7) = 63.4 + 4.3x_1 - 34.7x_2 + 3.0x_4 - 7.2x_5 - 1.4x_6 - 0.5x_7$$

Which one of the following statements about the model is *correct*?

No. Attribute description	Abbrev.
x_1 Number of plant species	Plants
x_2 Number of endemic plant species	E-Plants
x_3 Area of island (in km^2)	Area
x_4 Max. elevation above sea-level (in m)	Elev
x_5 Distance to nearest island (in km)	DistNI
x_6 Distance to Santa Cruz Island (in km)	StCruz
x_7 Area of adjacent island (in km^2)	AreaNI

Table 8.3. The seven attributes of the data on a selection of 29 of the Galápagos islands.

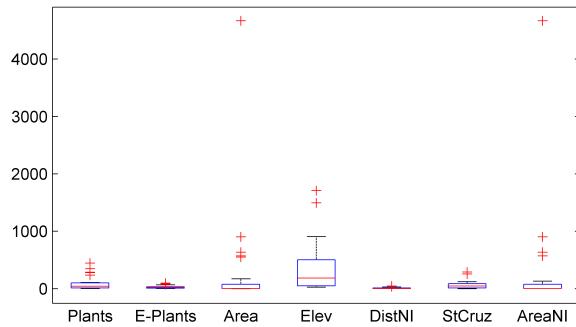


Fig. 8.10. Boxplot of the seven input attributes of the Galápagos data.

- A According to the model AreaNI is irrelevant for predicting the Area of islands.
- B According to the model it seems that the closer the neighboring island is the larger area the island has.
- C According to the model endemic plants is the most important predictor of island area.
- D According to the model an island that is highly elevated and close to Santa Cruz Island will in general be predicted to be relatively small.
- E Don't know.

8.4. Question 4:

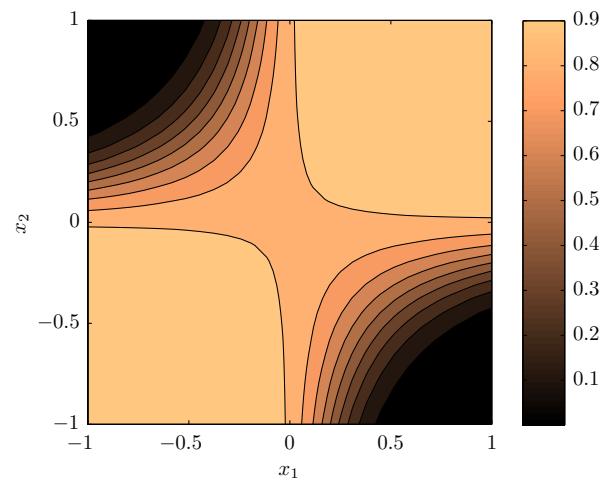
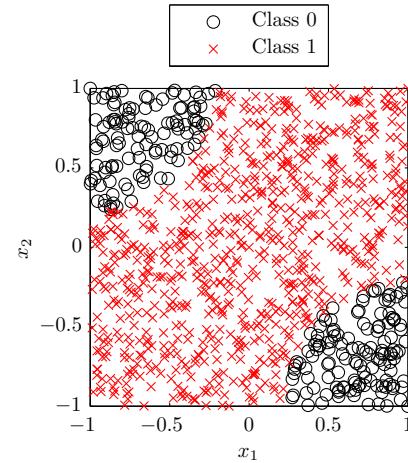


Fig. 8.11. top pane: Observed data. Bottom pane: Estimated probability of belonging to class 1 according to the logistic regression classifier.

Recall the logistic function is defined as

$$\text{logistic}(z) = \frac{1}{1 + e^{-z}}.$$

Suppose logistic regression classifier is trained on observations from two classes as shown in the top pane of fig. 8.11 and the trained classifier produces an estimate of the probability of belonging to class 1 as shown in the bottom pane. If the classifier takes the following form

$$f(x_1, x_2) = \text{logistic}(w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2)$$

what are the values of w_0, w_1, w_2, w_3 ?

- A $w_0 = 2, w_1 = w_2 = 0, w_3 = 10$.
- B $w_0 = -2, w_1 = w_2 = 0, w_3 = -10$.
- C $w_0 = -2, w_1 = w_2 = 1, w_3 = 10$.
- D $w_0 = 2, w_1 = w_2 = 1, w_3 = -10$.
- E Don't know.

Tree-based methods

In this section, we will consider tree-based methods for classification or regression. The goal of tree-based methods is the same as above: In classification, we wish to predict a discrete output label y_i for observation i based on features \mathbf{x}_i , and in regression, we wish to predict a continuous-valued y_i . However it is accomplished quite differently than linear or logistic regression, in that we consider the value of y_i as being determined by asking a series of questions organized as a tree about \mathbf{x}_i and then, based on the answers, assign y_i a constant value. Decision trees were originally developed by Earl B. Hunt (and coauthors) in 1966 in his Concept Learning System where the construction of the sequence of questions was intended to model human concept acquisition [Hunt et al., 1966]. However, today decision trees have grown to be an important yet simply supervised learning model. In the next sections, we will introduce regression and classification trees as well as discuss how they can be learned using *Hunt's algorithm*.

Table 9.1. Animals dataset. A dataset of $N = 15$ observations and $M = 4$ binary features where the goal is to predict if the animal is a Mammal or not.

Name	x_1 : Cold Blooded	x_2 : Has Legs	x_3 : Lay Eggs	x_4 : Has Fur	y : Mammal
Snake	yes	-	yes	-	-
Starfish	yes	-	yes	-	-
Bluebird	-	yes	yes	-	-
Blackbird	-	yes	yes	-	-
Earthworm	yes	-	yes	-	-
Chameleon	yes	-	yes	-	-
Ant	yes	-	yes	-	-
Jellyfish	yes	-	yes	-	-
Snail	yes	-	yes	-	-
Sea Urchin	yes	-	yes	-	-
Dolphin	-	-	-	-	yes
Rat	-	yes	-	yes	yes
Dog	-	yes	-	yes	yes
Monkey	-	yes	-	yes	yes
Lion	-	yes	-	yes	yes

Algorithm 2: Hunt's algorithm for decision trees

Require: Initial tree T only containing the root node
Require: D_r : Dataset associated with the current branch. Initially just the full dataset
if The **stop criterion** is met **then**
 Add a leaf node to the tree which assigns every observation to the most prevalent class in D_r
else
 Try a number of different splits on D_r . For each split, compute the **purity gain** and select the split
 $D_r = \{D_{v_1}, \dots, D_{v_K}\}$ with the highest purity gain
 Recursively call the method on D_{v_1}, \dots, D_{v_K}
end if

9.1 Classification trees

Consider the dataset in table 9.1 comprised of $N = 15$ animals. For each animal, we have recorded $M = 4$ features (cold blooded, has legs, lay eggs, and has fur) and we wish to build a classifier which determines y , if the animal is a mammal or not. Of course this corresponds to our usual situation where we are given a matrix \mathbf{X} and a vector \mathbf{y} , however, for the moment we will limit ourselves to the case where \mathbf{X} is binary and consider the general case later.

The decision tree can then be constructed using what is known as *Hunt's* algorithm and is outlined in fig. 9.1. We first place all 15 animals at the root of the tree (top left pane) and consider a binary yes/no question (we will discuss how these questions are chosen later) for instance “*cold blooded?*”. This question partitions the 15 animals into two new groups (top right pane); since one group (the cold blooded animals) are all non-mammals they are classified as such in a *leaf node* and we say this node is *pure*. The other group consists of a mixture of animals and so we ask a new question: “*lay eggs?*” in the bottom-right pane. This partitions the animals into two new groups and since they only contain mammals or non-mammals (i.e. they are pure) the method terminates.

The general procedure, Hunt's algorithm for decision-tree induction, is a simple recursive application of the same yes/no questioning procedure we just illustrated on the animal dataset. In general we will consider splits which are not just binary but multi-way. In fig. 9.2 we consider 5 example splits for different attribute types. For binary variables, we are limited to simple yes/no split (however there is one such potential split for each attribute type!) and for discrete or continuous values we can potentially consider splits into K branches. We then assume we at every step in the procedure has access to many potential splits and select the best split based on the *purity gain* that we will discuss shortly. The method terminated for the animal dataset when a branch contained only one type of animal, however, in general we will stop when a general *stop criterion* is met. The full method can be found in algorithm 2.

9.1.1 Impurity measures and purity gains

So how do we determine when one question is better than another? In fig. 9.3 we have outlined two potential root-node questions. In the left pane we ask if the animals have legs, and in the right pane we ask if it has fur. There are generally two components to a good question: Firstly, how *balanced* the question is. If the question is very specific, then one branch will only contain very few animals and the question will therefore not be very informative. If we consider the left-pane of fig. 9.3, and we denote the root of the tree by r and the two branches by v_1 and v_2 , the left-most branch of the

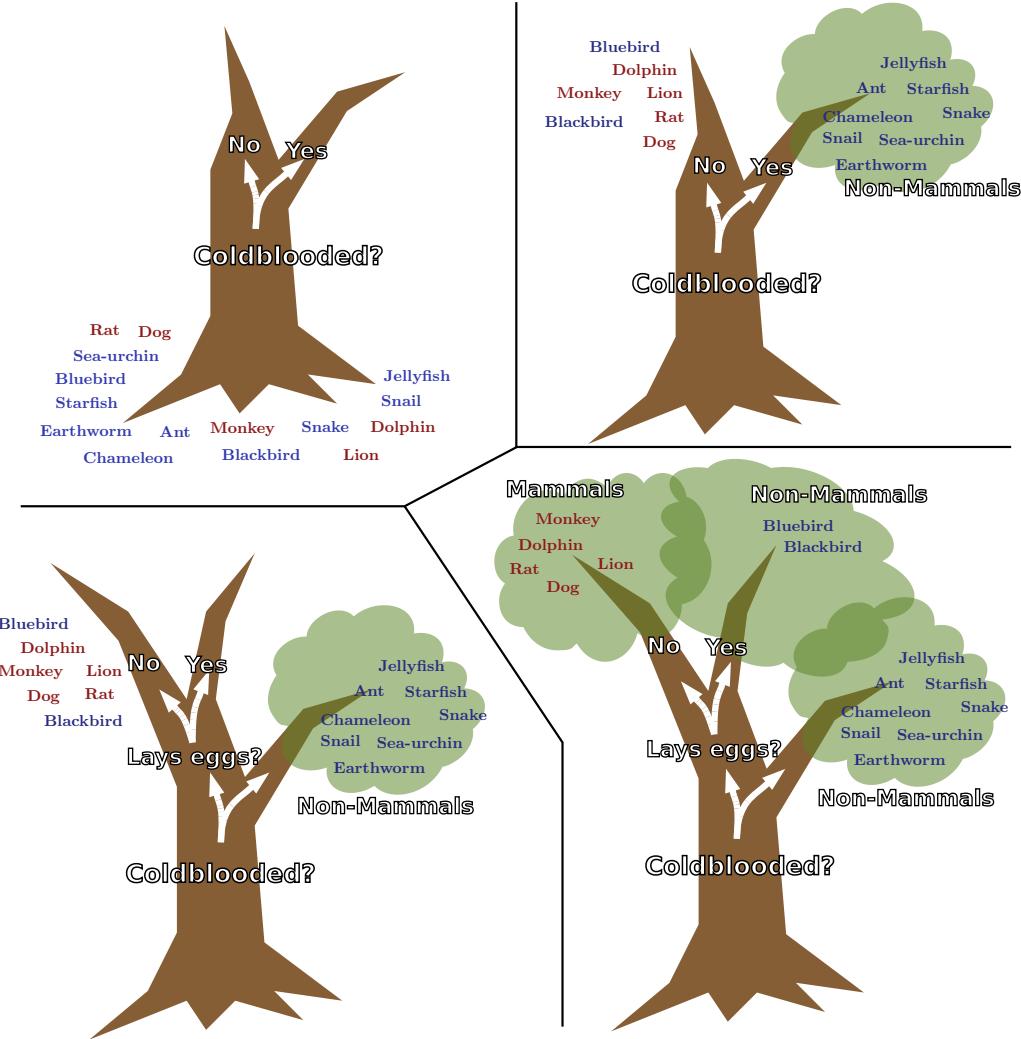


Fig. 9.1. Construction of a decision tree using Hunt's algorithm to classify whether an animal is a mammal or not. Initially, all observations are assigned to the root (top left pane) and we consider a question. The question divides the animals into two sets, if a particular set is pure, i.e. only contains mammals (or non-mammals) the method terminates (top right), else we recursively apply the method (bottom left). The method terminates in the bottom-right pane because all leaf branches are pure. In the general method, we consider many questions at each branch, and select the best according to its *purity gain*.

tree, v_1 , contains $N(v_1) = 7$ animals whereas the right-most branch contains $N(v_2) = 8$ animals. The question is thus fairly *balanced* compared to the right-most branch where we have $N(v_1) = 11$ and $N(v_2) = 4$.

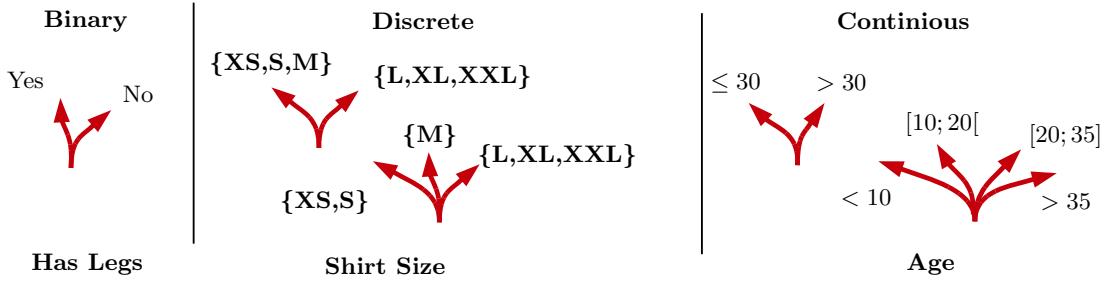


Fig. 9.2. Different types of attributes allow different splits. Binary attributes only allow binary (yes/no) splits, whereas discrete and continuous attributes allow either binary splits or many-way splits.

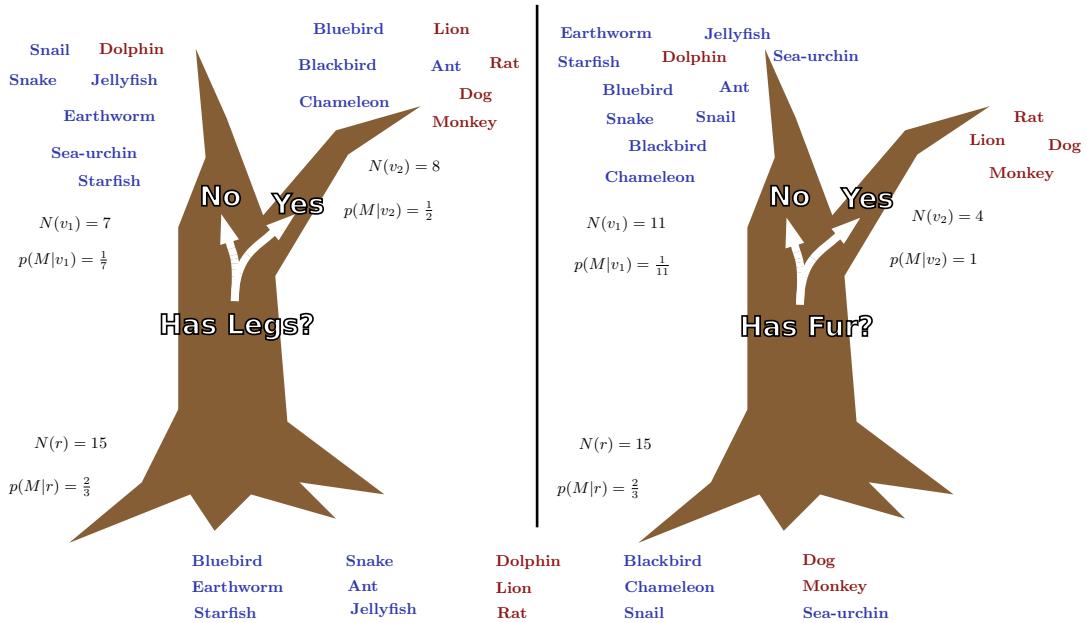


Fig. 9.3. Two different potential splits (left and right pane). The splits divide the animals at the root r into different groups corresponding to the left v_1 and right v_2 branch. When choosing between two splits, we are interested in how *balanced* they are (i.e. if $N(v_1) \approx N(v_2)$) and to what extend it is *pure* i.e. only contain one class as measured with the within-class probabilities $p(M|v_1)$ and $p(M|v_2)$. From these quantities we can compute the *purity gain* Δ .

On the other hand, we also want the split into different groups of animals to be as *pure* as possible, that is to contain (preferably) only one kind of animal. In the left-pane of fig. 9.3 the left-branch v_1 contains only one mammal (i.e. the probability the animal is a mammal in this branch is $p(M|v_1) = \frac{1}{7}$) and in the right-branch $p(M|v_2) = \frac{1}{2}$. However, the fur-split in the right hand pane

is much more pure since the probability of a mammal in the left-branch is $p(M|v_1) = \frac{1}{11}$ and in the right-pane $p(M|v_2) = 1$.

A measure of how good a question is should therefore consider both how balanced the question is and how pure the resulting classes are. This can be accomplished by first quantifying how impure the classes are at the root and at the K potential branches using an *impurity measure*. We will write this as $I(r)$ for the impurity at the root and $I(v_1), I(v_2), \dots, I(v_K)$ for the impurity at the branches. These impurities for each of the branches are weighted by the fraction of observations at the branch and combined to compute the overall impurity after the split. By contrasting the impurity before the split to the overall impurity after the split we obtain the *purity gain* Δ for the question, which is given by the formula

$$\Delta = I(r) - \sum_{k=1}^K \frac{N(v_k)}{N(r)} I(v_k). \quad (9.1)$$

A high purity gain indicates that the impurity of the individual splits, $I(v_1), \dots, I(v_K)$ is low, i.e. the classes have become more pure relative to the root impurity $I(r)$. The weighting by the fraction $\frac{N(v_k)}{N(r)}$ is used to make the measure focus on the larger (important) groups in the split. This only requires us to specify the impurity. The impurity function $I(v)$ only depends on the (relative) size of the classes in the given branch v , i.e. the probabilities $p(c|v)$. If in general we consider there are C classes, we have C such probabilities in each branch $p(c=1|v), \dots, p(c=C|v)$ (in the animals example we had two classes corresponding to $C=2$ and $p(M|v), p(\text{not } M|v)$). The following three are popular choices for the impurity function $I(v)$; entropy, Gini and classification error:

$$\text{Entropy}(v) = - \sum_{c=1}^C p(c|v) \log_2 p(c|v), \quad (9.2)$$

$$\text{Gini}(v) = 1 - \sum_{c=1}^C p(c|v)^2, \quad (9.3)$$

$$\text{ClassError}(v) = 1 - \max_c p(c|v). \quad (9.4)$$

Here $\log_2 p(c|v)$ is the base-2 logarithm. Suppose we consider the example in the right pane of fig. 9.3 and we use the ClassError impurity measure we obtain:

$$I(r) = 1 - \frac{2}{3} = \frac{1}{3}, \quad I(v_1) = 1 - \frac{10}{11} = \frac{1}{11} \quad \text{and} \quad I(v_2) = 1 - 1 = 0. \quad (9.5)$$

We can then compute the purity gain as

$$\Delta = I(r) - \frac{11}{15} I(v_1) - \frac{4}{15} I(v_2) = \frac{1}{3} - \frac{1}{15} = \frac{4}{15}. \quad (9.6)$$

As mentioned, the purity gain can easily be computed for many types of splits and having multiple classes. In fig. 9.4 we consider a three-way split ($K=3$) for $N(r)=12$ objects (the colored balls) corresponding to a total of $C=4$ true classes. In the example, the impurity gain would be

$$\Delta = I(r) - \frac{5}{12} I(v_1) - \frac{1}{4} I(v_2) - \frac{1}{3} I(v_3). \quad (9.7)$$

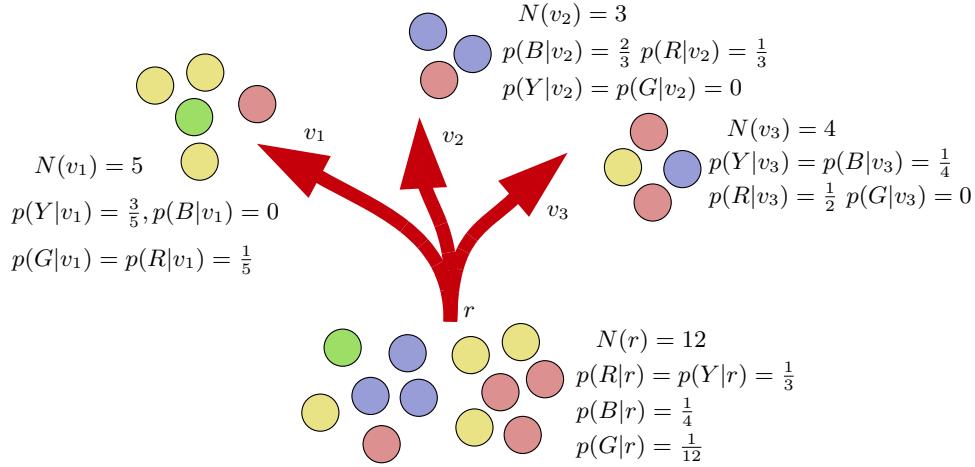


Fig. 9.4. A multi-way split where $N = 12$ observations belonging to $C = 4$ classes are split in a $K = 3$ way split. The classes are indicated by the colors. See text for details on how the purity gain Δ can be computed from the given numbers.

In practice, we are of course primarily interested in applying classification trees to the case where \mathbf{X} contains general continuous features. The splits most often considered are binary, two-way splits obtained by considering each of the M features of \mathbf{X} and then attempting different possible split-values:

$$x_m < x^* \quad (9.8)$$

where x^* is the split-value varied over the range of the observed data points. This gives a very large number of potential splits, each being axis aligned. This is illustrated for a 2D dataset in fig. 9.5. We start by considering all binary splits $x_1 < x^*$ and $x_2 < y^*$ where x^* and y^* are varied. The split with the highest purity gain is selected and indicated by the colors in the top-left pane. The method is applied recursively for each of the two new splits. Again all axis-aligned splits are considered and two splits are selected giving a tree with four leaf nodes (top-right pane). This procedure is continued recursively in the bottom row and the method terminates when it encounters pure classes.

9.1.2 Controlling tree complexity

Hunt's algorithm terminates if it encounters pure splits, i.e. the current set of observations in a leaf only contains one class. However, it is often a good idea to terminate the method earlier. Consider for instance fig. 9.5 bottom-right pane where in order to place *all* observations in a pure leaf node the method creates some rather odd-looking boxes. In general, there are two strategies for ensuring this does not happen.

Early stopping

The simplest way to control the complexity of the tree is to stop Hunt's algorithm before it encounters pure splits. There are several criteria for stopping:

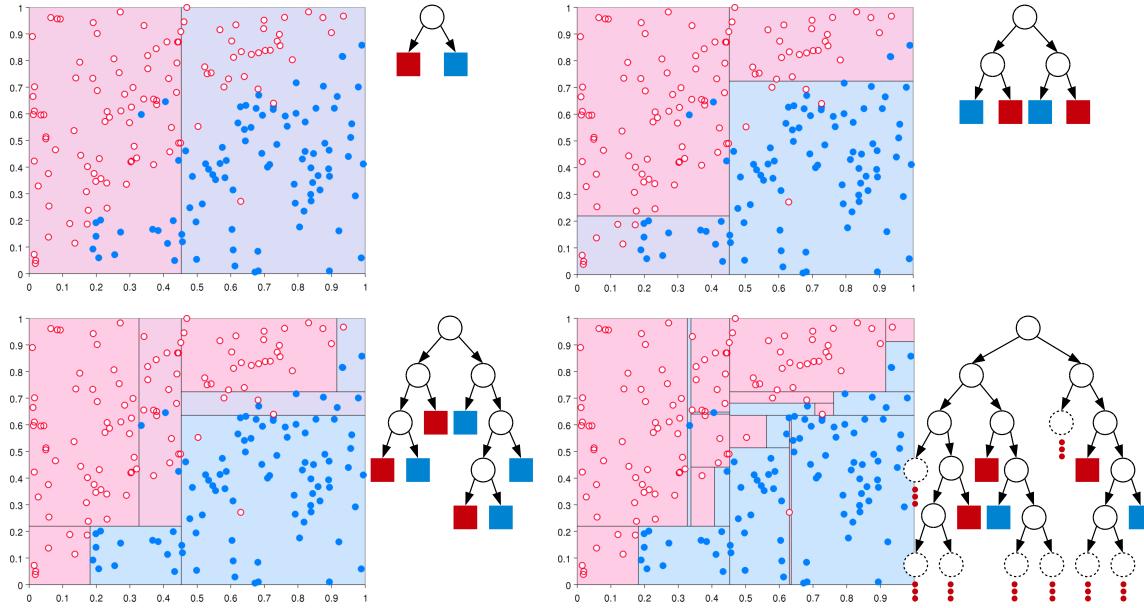


Fig. 9.5. Construction of a decision tree using Hunt's algorithm. We consider binary $K = 2$ splits where we as candidate splits consider if x_1 (and x_2) is less than or greater than a sequence of split-values. In the top-left pane we have selected an initial split based on the value of the x -axis. Hunt's algorithm is then applied recursively (top right pane) to produce two additional splits, then it is applied recursively on the non-pure groups (bottom left) and after several steps produces the final split in the bottom-right pane. Notice the final configuration likely overfits the data.

- Stop splitting when a branch contains less than a specific number of observations.
- Stop splitting if a certain depth of the tree is reached.
- Stop splitting if purity gain Δ for the best split is below a certain value.

This of course leaves open the question of *how* we should select for instance the minimum number of observations in a branch. For now simply assume it is selected manually - in chapter 10 we will consider how cross-validation can be used to solve this problem.

Pruning*

Early stopping is simple to implement, but comes with an important disadvantage known as the horizon effect. Simply put, since early stopping stops growing the tree at some point, we can't know if *continuing* growing the tree by just one node beyond that point would have resulted in a significant reduction in error.

Pruning tries to get around this problem by first growing a full tree with no (or very little) early stopping and then afterwards select which branches in the tree should be replaced by a single leaf (i.e. should be pruned). How the pruned subtrees are selected differ from pruning strategy to pruning strategy but a simple strategy is *cost complexity pruning* [Breiman et al., 1984].

In cost complexity pruning, we construct a series of trees T_0, T_1, \dots, T_m where T_0 is the initial (full) tree produced by Hunt's algorithm and T_m is a tree only consisting of the root. Each tree T_i

Algorithm 3: Cost complexity pruning of decision trees

Require: Initial full tree T_0 produced by Hunt's algorithm
Require: T_0, T_1, \dots, T_m , a sequence of increasingly more pruned trees
for $i = 1, 2, \dots$ and T_i is not only the root **do**
 for each subtree t of T_{i-1} **do**
 Compute the cost-complexity error corresponding to collapsing tree t :

$$C_t = \frac{E(\text{Prune}(T,t)) - E(T)}{|T| - |\text{Prune}(T,t)|}$$

 end for
 Let t be the subtree of T_{i-1} which minimizes C_t
 Set $T_i = \text{Prune}(T,t)$
end for

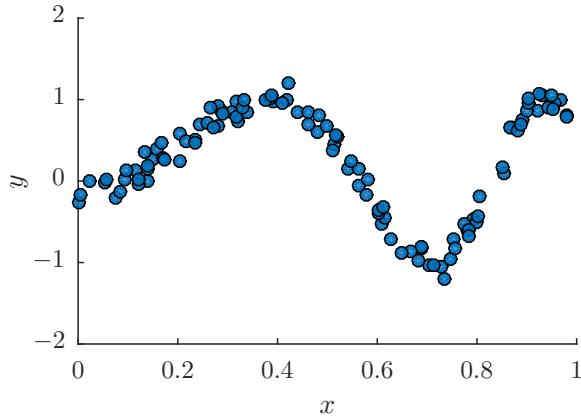


Fig. 9.6. A simple 1D example regression data set containing $N = 100$ observations.

is constructed from T_{i-1} by first trying to collapse each subtree t of T_{i-1} into a single node and for the collapsed tree compute the cost-complexity tradeoff which measures the relative increase in error per removed node; the intuition being that the removal of many nodes should be favored over the removal of a single node. Once the cost-complexity has been computed for each internal branch the branch with the lowest cost-complexity is collapsed producing T_i . Algorithmically the method is shown in algorithm 3. Once the sequence T_0, \dots, T_m has been produced, the tree T_i with the lowest generalization (i.e. test) error is selected. How to estimate the generalization error is discussed in chapter 10 as *two-layer cross-validation*.

9.2 Regression trees

In regression, y_i for an observation i is no longer discrete but continuous. The decision tree method may however very easily be altered to accommodate for this change. Suppose we consider the animal example again, but this time we wish to predict the animals mean life span. We can ask exactly the same questions, but then in order to predict the mean life span y_i for a given branch, say for instance the right-most branch in the right-most pane of fig. 9.3, we would simply predict the mean value of the animals in that branch:

Algorithm 4: Hunt's algorithm for regression trees

Require: Initial tree T only containing the root node
Require: D_r : Dataset associated with the current branch. Initially just the full dataset
if The **stop criterion** is met **then**
 Add a leaf node to the tree which assigns every observation the mean value of the nodes in D_r :
 $y(r) = \frac{1}{N(r)} \sum_{i \in r} y_i$
else
 Try a number of different splits on D_r . For each split, compute the **purity gain** using the sum-of-squares impurity measure and select the split $D_r = \{D_{v_1}, \dots, D_{v_K}\}$ with the highest purity gain
 Recursively call the method on D_{v_1}, \dots, D_{v_K}
end if

$$\text{Predicted } y\text{-value in } v_2 : y(v_2) = \frac{y_{\text{Rat}} + y_{\text{Lion}} + y_{\text{Dog}} + y_{\text{Monkey}}}{4} \quad (9.9)$$

$$= \frac{\sum_{i \in v_2} y_i}{N(v_2)} \quad (9.10)$$

To evaluate the goodness of a new split, we can now simply compute the average sum-of-squares error between the observed y_i 's and the mean value $y(v_2)$. This can be done by simply introducing a new impurity measure:

$$I(v) = \frac{1}{N(v)} \sum_{i \in v} (y_i - y(v))^2 \quad \text{where} \quad y(v) = \frac{1}{N(v)} \sum_{i \in v} y_i, \quad (9.11)$$

and then simply use Hunt's algorithm as already introduced where the stopping criteria may for instance be that the purity gain (or the impurity) falls below a certain value. The algorithm is very similar to algorithm 2 but for completeness it is listed in algorithm 4.

We will consider this method applied to the simple 1-d regression problem in fig. 9.6, the result can be seen in fig. 9.7. We again consider recursive, binary splits. In the first iteration of the algorithm, all observations are assigned to the same branch and we simply predict the mean value of all observations (top left pane). Then, the optimal split is selected as the split which increases the purity gain the most and we split the x -values once at the dotted vertical line to produce two predicted y -values (right pane). This procedure is applied recursively to give two splits at the next level (bottom left) and finally four splits (bottom right). As can be seen, this method very quickly allows for a flexible but piece-wise constant prediction of y .

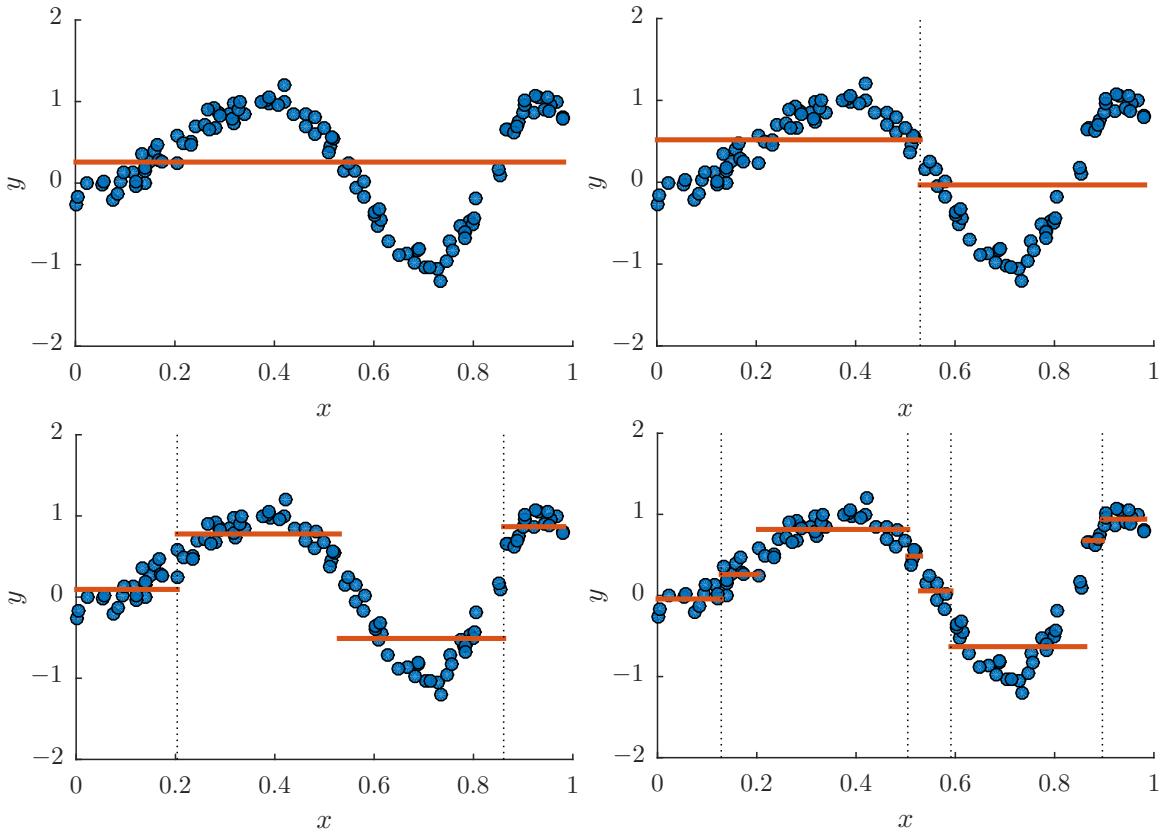


Fig. 9.7. Application of the regression tree method to the 1d example. First, all observations are assigned a constant y -value (top left pane). Then we consider various splits (along the x -axis) and select the one with the highest purity gain computed using the sum-of-squares impurity measure (top right). The method is applied recursively on each section until a stopping criterion is met (bottom row).

Problems

9.1. Question 1: We consider a dataset on survival of breast cancer taken from <http://archive.ics.uci.edu/ml/machine-learning-databases/haberman/haberman.names>. The data has been binarized as outlined in Table 9.2. The dataset contain a total of 306 observations. In the dataset 81 survived after 5 years (i.e., $y = 1$) whereas 225 died (i.e., $y = 0$). We would like to build a decision tree and consider using positive axillary nodes detected (PAN) as an attribute condition at the root of the tree. We thereby split according to whether positive axillary nodes were detected and find:

- For the 170 subjects that had positive axillary nodes detected 62 survived.
- For the 136 subjects that did not have positive axillary nodes 19 survived.

What is the gain, Δ , of splitting according to whether a subject had positive axillary nodes (PAN) using the Gini as impurity measure $I(t)$, (i.e., $I(t) = 1 - \sum_{i=0}^{C-1} p(i|t)^2$)?

No.	Attribute description	Abbrev.
x_1	Young (< 60 years), $x_1 = 0$ or Old (≥ 60 years), $x_1 = 1$	Age
x_2	Operated before, $x_2 = 0$ or after 1960, $x_2 = 1$	OpT
x_3	Positive axillary nodes detected PAN No, $x_3 = 0$ or Yes, $x_3 = 1$	PAN
y	Lived after 5 years No, $y = 0$ or Yes, $y = 1$	Surv

Table 9.2. A modified version of Haberman's Survival Data taken from <http://archive.ics.uci.edu/ml/machine-learning-databases/haberman/haberman.names>. The attributes x_1-x_3 denoting the age, operation time and cancer size as well as the output denoting survival after five years are binary. The data contains a total of $N = 306$ observations.

- A -0.025
 B 0
 C 0.025
 D 0.036
 E Don't know.

9.2. Question 2: We will use the decision tree given in Figure 9.9 to attempt to solve the classification problems given to the right of Figure 9.9 corresponding to the classification problem also considered in Figure 9.8. Which one of the following choices for the two decisions A and B in the decision tree would be the most well suited to separate the two classes?

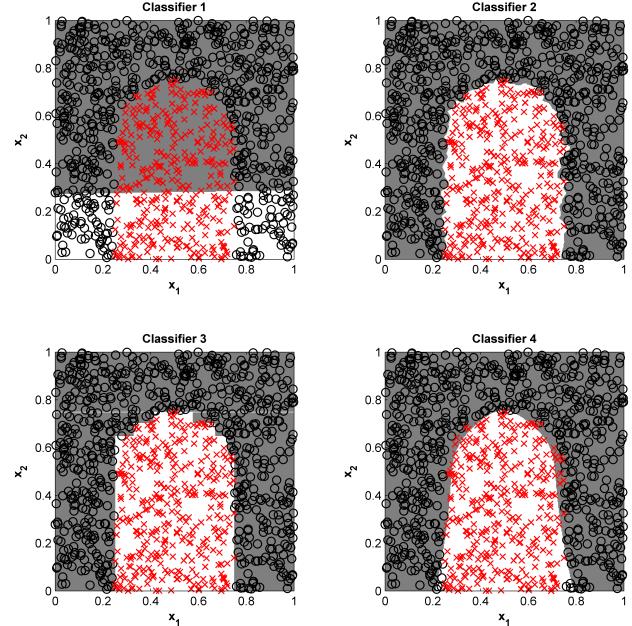


Fig. 9.8. The decision boundaries given in white and gray of four different classifiers used to separate red crosses from black circles.

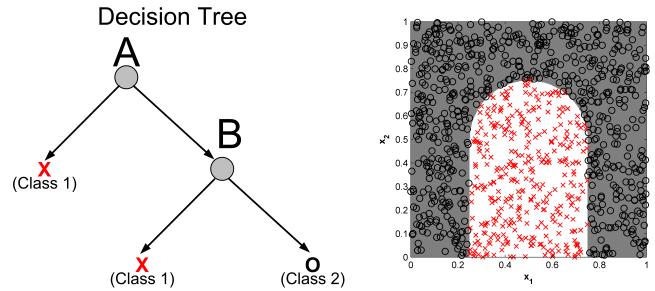


Fig. 9.9. A decision tree with two decisions denoted A and B that if given the right decisions can be used to perfectly separate the red crosses from black circles given in the classification problem to the right that was also considered in Figure 9.8.

$$\begin{aligned}
 A &= \left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix} \right\|_\infty < 0.25 \\
 B &= \left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \right\|_2 < 0.25 \\
 B \quad A &= \left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix} \right\|_1 < 0.25 \\
 B &= \left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \right\|_2 < 0.25 \\
 C \quad A &= \left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix} \right\|_\infty < 0.25 \\
 B &= \left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \right\|_1 < 0.25
 \end{aligned}$$

D A = $\left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix} \right\|_2 < 0.5$
B = $\left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \right\|_\infty < 0.5$
E Don't know.

9.3. Question 3: Consider a one-dimensional data set of features \mathbf{X} and 3-class responses y shown in table 9.3; there are thus $N = 7$ observations.

\mathbf{X}	1	3	1	2	1	4	2
y	2	2	2	0	0	1	0

Table 9.3. Table of data and responses.

Suppose a decision tree is used to classify y on \mathbf{X} . Consider an attempted split at $\mathbf{X} = x_1 > 2.5$. What is the *impurity gain* Δ of this split for the data set if the *classification error* is used as impurity measure?

- A $\Delta = 0.123$
B $\Delta = 0.143$
C $\Delta = 0.239$
D $\Delta = 0.428$
E Don't know.

9.4. Question 4:

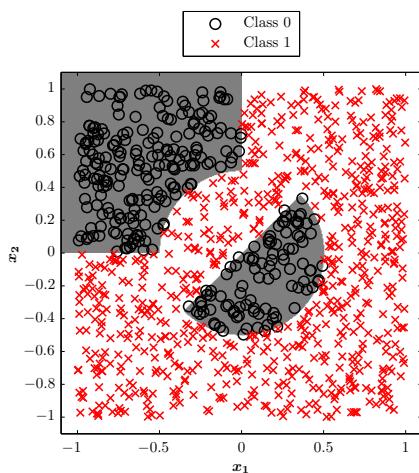


Fig. 9.10. Two-class classification problem

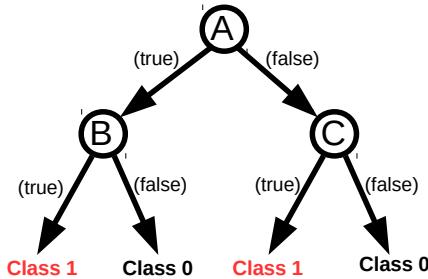


Fig. 9.11. Decision tree with 3 nodes A, B and C

Suppose we wish to solve the two-class classification problem in fig. 9.10 using a classification tree of the form fig. 9.11. What rules, acting on the coordinates $\mathbf{x} = (x_1, x_2)$, should be assigned to the three internal nodes A, B and C for the tree to give rise to the indicated decision boundary?

- A A : $\|\mathbf{x}\|_2 \geq \frac{1}{2}$, B : $\left\| \mathbf{x} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\|_\infty > 1$
C : $\left\| \mathbf{x} - \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\|_1 > 2$
B A : $\|\mathbf{x}\|_2 \geq \frac{1}{2}$, B : $\left\| \mathbf{x} - \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\|_1 > 2$
C : $\left\| \mathbf{x} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\|_\infty > 1$
C A : $\left\| \mathbf{x} - \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\|_1 > 2$, B : $\left\| \mathbf{x} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\|_\infty > 1$
C : $\|\mathbf{x}\|_2 \geq \frac{1}{2}$
D A : $\left\| \mathbf{x} - \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\|_1 > 2$, B : $\|\mathbf{x}\|_2 \geq \frac{1}{2}$
C : $\left\| \mathbf{x} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\|_\infty > 1$
E Don't know.

10

Overfitting and cross-validation

For some practitioners of machine learning the most interesting aspect is devising new and exciting algorithms and words such as “evaluation” or “testing” is likely to be treated as an afterthought. However, testing and quantifying the performance of machine-learning methods is possibly the most important aspect of data modelling.

Suppose we are in a situation where we have S different models $\mathcal{M}_1, \dots, \mathcal{M}_S$ that each tries to solve a particular supervised learning problem. Without an objective way of comparing the models we will not know which to choose. Sure, we might *feel* we should select model \mathcal{M}_S which is the most complicated of the models, but that is not an objective justification. Worse yet, if we are working in a company, it will be impossible to quantify if progress is being made at solving the problem or if there is any benefits for the company to have a machine learning department at all.

Seen in this way quantification (and comparison) of model performance is something a machine-learning practitioner should be obsessively preoccupied with. In this chapter, we will discuss common issues with model performance evaluation and provide the industry standard, cross-validation, for estimating the generalization error which allow us to evaluate a given models performance and thereby select between different models. The chapter will finish with a discussion of how the generalization error provides more qualitative information about the goodness of a given model.

10.1 Cross-validation

The principal way of comparing and validating models is by cross-validation. In this chapter, we will introduce the reasoning behind cross-validation and discuss important applications of cross-validation. We will use the simple linear regression model as a running example.

10.1.1 A simple example, linear regression

To provide a concrete example, consider the simple regression problem in fig. 10.1 where the goal is to predict y from x and we have access to 9 data points collected in a training data set $\mathcal{D}^{\text{train}}$. The data consists of noisy observations of the black curve and we wish to fit a regression model to the data. We assume we have access to three different models

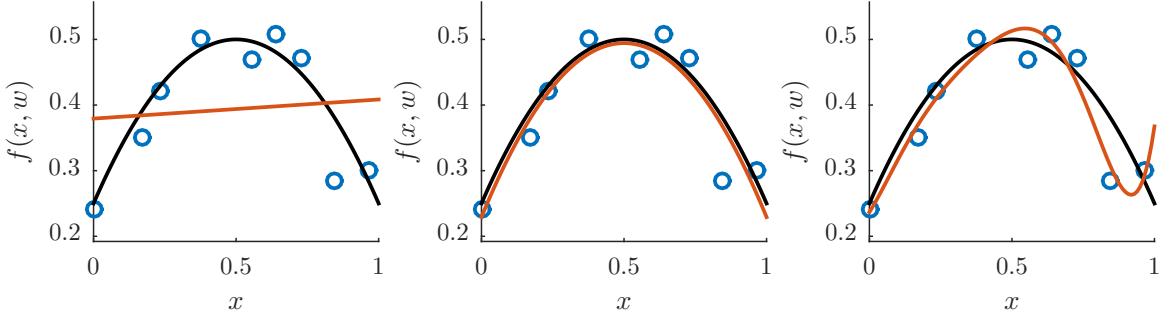


Fig. 10.1. A small dataset of nine observations generated from the true curve shown in the black line. The three red lines are three different linear regression models $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ fitted to the dataset. Clearly the most complicated model \mathcal{M}_3 fits the dataset best, however, the second model \mathcal{M}_2 is better suited to account for the true black curve.

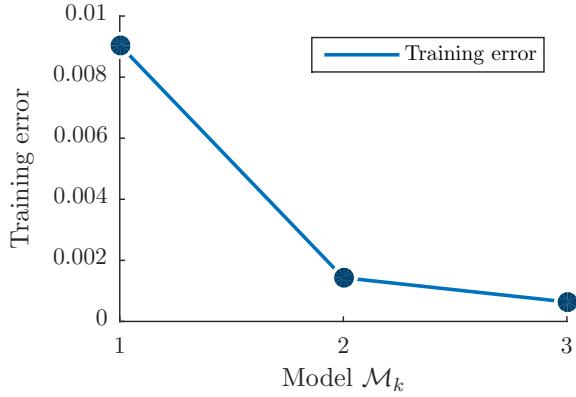


Fig. 10.2. Training error for each of the three models $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ computed on the training data set. The most complicated model has the lowest training error.

$\mathcal{M}_1 = \{1\text{'st order polynomial, i.e. } \} f_{\mathcal{M}_1}(x, \mathbf{w}) = w_0 + w_1 x.$

$\mathcal{M}_2 = \{2\text{'nd order polynomial, i.e. } \} f_{\mathcal{M}_2}(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2.$

$\mathcal{M}_3 = \{6\text{'th order polynomial, i.e. } \} f_{\mathcal{M}_3}(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5 + w_6 x^6.$

The red line indicates the fitted polynomials. For each model we quantify how well the model fits the training data by the *training error* which for model \mathcal{M}_s is

$$E_{\mathcal{M}_s}^{\text{train}} = \frac{1}{N^{\text{train}}} \sum_{i \in \mathcal{D}^{\text{train}}} (y_i - f_{\mathcal{M}_s}(x_i, \mathbf{w}))^2.$$

Here $f_{\mathcal{M}_s}$ is the model \mathcal{M}_s fitted to the training data and $N^{\text{train}} = |\mathcal{D}^{\text{train}}|$ is the number of observations in the training data set. The training error of each of these three models is shown in fig. 10.2. Notice the “most correct” model, \mathcal{M}_2 , fits the data better than the model \mathcal{M}_1 , however, both models fits the data far worse than the complicated model \mathcal{M}_3 .

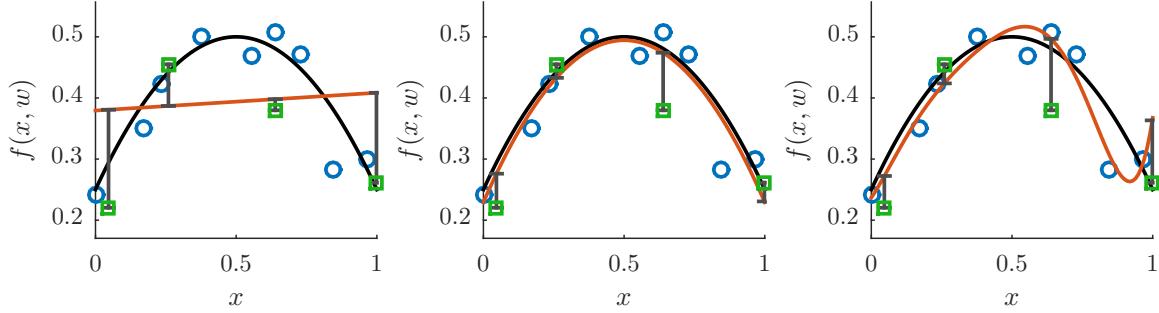


Fig. 10.3. The example from before with a test dataset of three new points. If the models are evaluated in terms of how they predict the *new* points \mathcal{M}_2 is preferred.

Nevertheless, we are not interested in a model like \mathcal{M}_3 as it clearly will not generalize well to new data. We say model \mathcal{M}_3 is *overfitting* the data. Thus, we can't tell the models apart by how well they fit the training data and in fact this can give us an entirely misleading picture of the models performance due to overfitting. This is such an important principle it is worth framing:

Never, ever should you estimate how well a model performs by its predictions on data it was trained upon.

However, let's assume we obtain access to some new data, the test data $\mathcal{D}^{\text{test}}$, indicated by the green squares in fig. 10.3. Testing the models on this new test-dataset gives us the ability to estimate how well the models *generalize* to new data. We can define the test error as

$$E_{\mathcal{M}_s}^{\text{test}} = \frac{1}{N^{\text{test}}} \sum_{i \in \mathcal{D}^{\text{test}}} (y_i - f_{\mathcal{M}_s}(x_i, \mathbf{w}))^2.$$

Notice, $f_{\mathcal{M}_s}$ is the model that was fitted to the *training data*. The test error of each of these three models is shown in fig. 10.4. Notice, the test error allows us to select the correct model, i.e. the model that can be expected to generalize better to new data.

The problem is that as a rule nobody is going to turn up and give us a test dataset when we need it. The basic idea in cross-validation is to overcome this problem by taking our existing fixed data set \mathcal{D} and manually divide it into a training set, $\mathcal{D}^{\text{train}}$, and a testing data set, $\mathcal{D}^{\text{test}}$, and then use these two to select the appropriate model.

10.1.2 The basic setup for cross-validation

The basic setup for cross-validation is as follows: We consider a supervised learning problem with a data set $\mathcal{D} = (\mathbf{X}, \mathbf{y})$. It is important to keep in mind the dataset is finite and *this is all the data we have*. As in the regression example, we consider different models for solving the problem, $\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_S$ and for each of these models we have access to a *loss function* $L(\mathbf{y}_i, \hat{\mathbf{y}}_i)$ which quantifies the error of predicting $\hat{\mathbf{y}}_i$ when the true value is \mathbf{y}_i . In the regression example the loss function was the least square error $L(\mathbf{y}_i, \hat{\mathbf{y}}_i) = \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|^2$, but in general the loss function will be defined according to the specific modeling purpose.

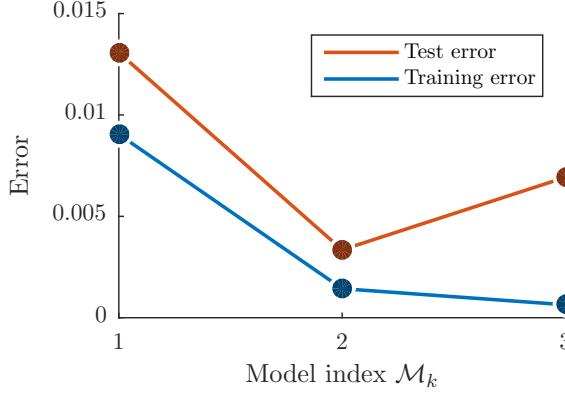


Fig. 10.4. Test error for each of the three models $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ computed on the test data set. The test error correctly singles out model \mathcal{M}_2 as the better model.

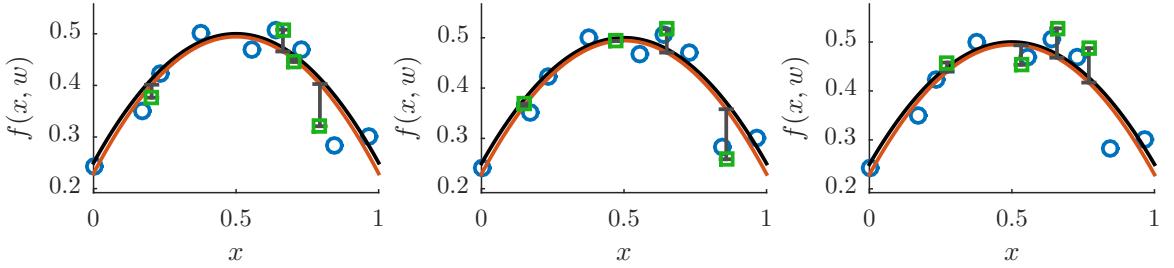


Fig. 10.5. The basic regression problem for model \mathcal{M}_2 and three random test sets. Since the test sets are random, the test error too will vary depending on the particulars of the test set. The generalization error overcomes this by averaging over all test sets.

Training and test error

If the data is divided into a training set and a test set $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{test}}$ and the model \mathcal{M} is fitted on the training set to provide the prediction rule $f_{\mathcal{M}}$ then we define the training and test errors as:

$$E_{\mathcal{M}}^{\text{train}} = \frac{1}{N^{\text{train}}} \sum_{i \in \mathcal{D}^{\text{train}}} L(\mathbf{y}_i, f_{\mathcal{M}}(\mathbf{x}_i)), \quad (10.1)$$

$$E_{\mathcal{M}}^{\text{test}} = \frac{1}{N^{\text{test}}} \sum_{i \in \mathcal{D}^{\text{test}}} L(\mathbf{y}_i, f_{\mathcal{M}}(\mathbf{x}_i)). \quad (10.2)$$

These definitions are similar except $f_{\mathcal{M}}$ is fitted on the training set in both cases.

Generalization error

A problem with the test error is that it depends on the specific test set. Since we have to construct the test set ourselves, this makes the test error slightly random. This is illustrated in fig. 10.5 for the

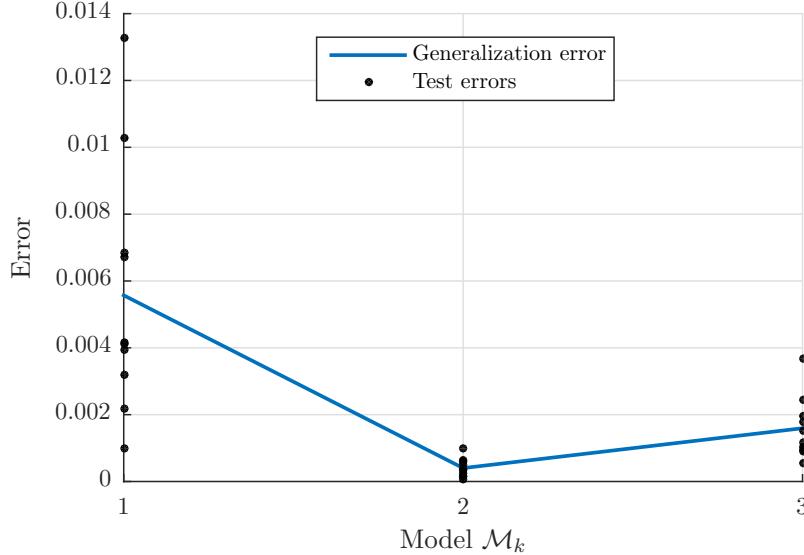


Fig. 10.6. Continuing the example, for the three models different test sets gives different test error as indicated by the black dots. The generalization error is simply the average over all possible test sets according to their probability of occurring.

test error for three different (random) test sets. To alleviate this problem we introduce a new, third error namely the *generalization error*. The generalization error is an idealized quantity indicating how well our model performs on average assuming we had an infinite amount of data to test it on, i.e. the average of the test errors as illustrated in fig. 10.6. *The generalization error is what we truly wish to estimate and the best model is the model with the lowest generalization error.* If we assume the test observations $(\mathbf{x}_i, \mathbf{y}_i)$ come from a distribution $p(\mathbf{x}, \mathbf{y})$ then the generalization error is defined as follows:

- Train the model \mathcal{M} on the full dataset available \mathcal{D} to give a prediction rule $\mathbf{f}_{\mathcal{M}}$.
- The generalization error of model \mathcal{M} is¹

$$E_{\mathcal{M}}^{\text{gen}} = \mathbb{E}_{(\mathbf{x}, \mathbf{y})} [L(\mathbf{y}, \mathbf{f}_{\mathcal{M}}(\mathbf{x}))] \quad (10.3)$$

$$= \int L(\mathbf{y}, \mathbf{f}_{\mathcal{M}}(\mathbf{x})) p(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y}. \quad (10.4)$$

The generalization error is the fairest estimate of how well our model can perform because it assumes we train our model on *all* data we have available and then computes the *average* loss on all future data.

¹ Another popular definition is to consider the training set random as well which we will later call the averaged generalization error. This is however notationally more cumbersome and leads to the same definition of the cross-validation algorithms.

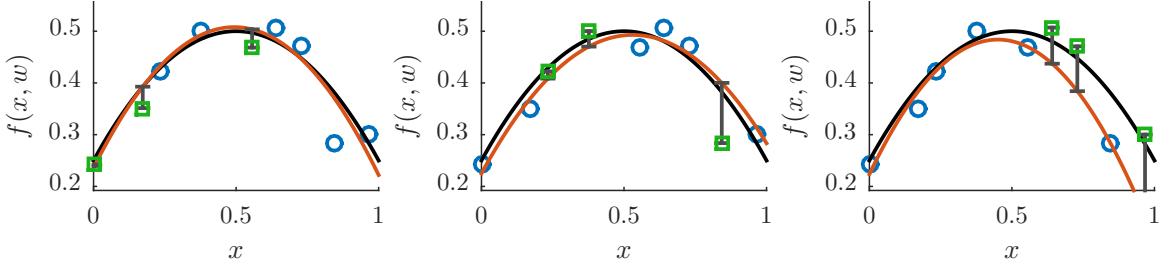


Fig. 10.7. Cross-validation applied to the model \mathcal{M}_2 using 3-fold cross-validation. Errors are estimated from the test data points and is indicated by the gray bars. Averaging all errors produce the estimate of the generalization error $\hat{E}_{\mathcal{M}_2}^{\text{gen}}$

10.1.3 Cross-validation for quantifying generalization

The obvious problem with the generalization error is that we cannot compute it since we don't know the true distribution of the data. Cross-validation, is thus a framework to estimate a model's generalization error typically based on one of the following three approaches:

Hold-out method

In the hold-out method, the full dataset \mathcal{D} is split into a train and a testing set

$$\mathcal{D} = \mathcal{D}^{\text{train}} \cup \mathcal{D}^{\text{test}}.$$

Then, we train a model on $\mathcal{D}^{\text{train}}$ and compute the test error $E_{\mathcal{M}}^{\text{test}}$ using the test data set $\mathcal{D}^{\text{test}}$ and formula eq. (10.2) and simply use the approximation:

$$E_{\mathcal{M}}^{\text{gen}} \approx E_{\mathcal{M}}^{\text{test}}.$$

Why does this work? The test error is different in two ways from the generalization error. Firstly, we only train the model on a subset of the data $\mathcal{D}^{\text{train}}$ and not the full data set \mathcal{D} and secondly we do not compute the true expectation but only the empirical average based on $\mathcal{D}^{\text{test}}$. However, if the training data set is large, we can expect (or rather, hope!) there will be little difference in using $\mathcal{D}^{\text{train}}$ instead of \mathcal{D} and secondly, if we have a lot of test data in $\mathcal{D}^{\text{test}}$, and each element in the test data set is drawn from the true distribution $p(\mathbf{x}, \mathbf{y})$, we can expect the empirical average in eq. (10.2) to be quite close to the true average for the generalization error eq. (10.4). By recognizing these limitations we can provide two alternative methods which generally does better but are also computationally more demanding:

K-fold cross-validation

Ideally, we want each data point to be used in the test set, and one way to accomplish this is with K -fold cross-validation. In K -fold cross-validation the full data set is split into K pieces

$$\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \dots \cup \mathcal{D}_K,$$

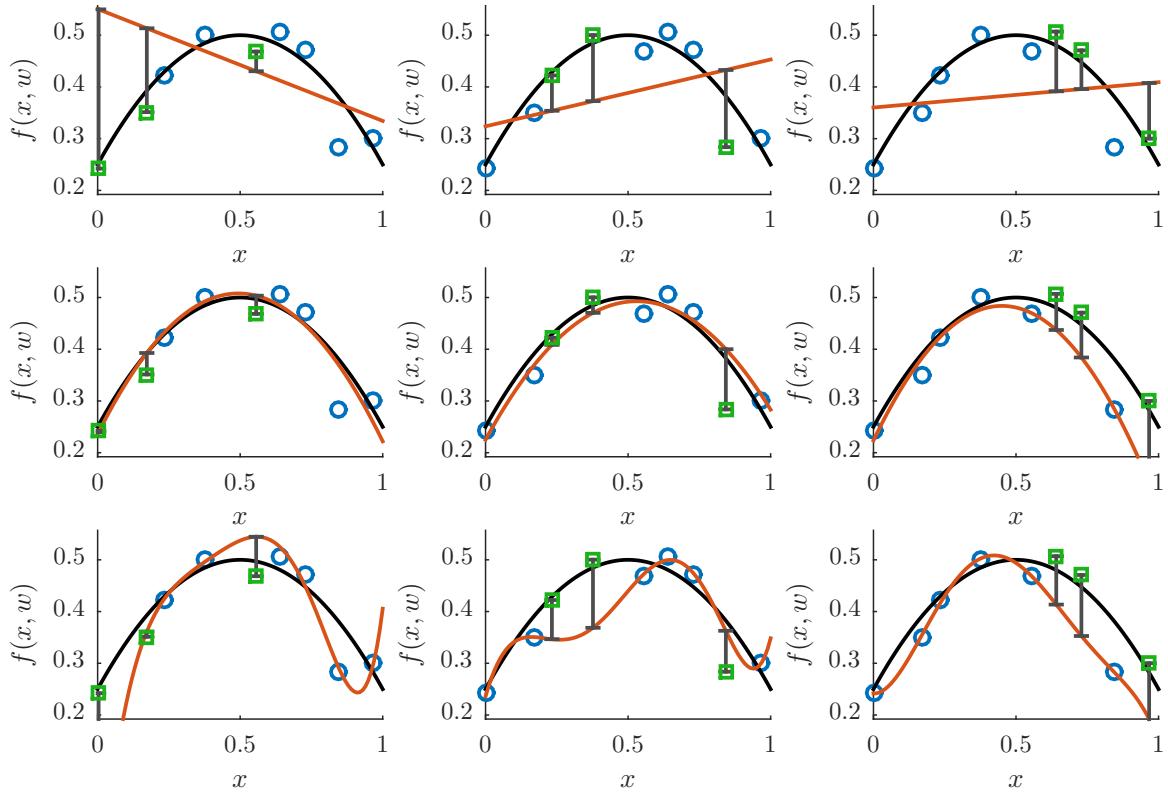


Fig. 10.8. Cross-validation applied to model-selection for the linear regression models. Each row corresponds to one of the three models, and each column to the estimate of the test error on that particular fold.

each containing $\frac{N}{K}$ observations. We then produce K splits into training and test sets by, for each k , treating \mathcal{D}_k as the test set and the other $K - 1$ pieces as the training set. Computing the test error on each of these K splits gives K estimates of the error $E_{\mathcal{M},1}^{\text{test}}, \dots, E_{\mathcal{M},K}^{\text{test}}$ and we now approximate:

$$E_{\mathcal{M}}^{\text{gen}} \approx \sum_{k=1}^K \frac{N_k^{\text{test}}}{N} E_{\mathcal{M},k}^{\text{test}},$$

i.e., as the weighted average of the test errors, weighted by the number of test observations in each fold N_k^{test} relative to the total number of observations used for testing N . Since each data point is used once in the test set this method is generally more precise than the hold-out method, however, it requires K times more training and testing of models than the hold-out method.

Leave-one-out cross-validation

The final method is based on the intuition that we ideally want the training set $\mathcal{D}^{\text{train}}$ to be as close to \mathcal{D} as possible. This can be accomplished by using K -fold cross-validation with $K = N$,

the total number of observations in the full data set. In this way, we train N models and each model is trained on the full data set except a single observation and then tested on that single observation. The benefit of this method is that it uses as much data as possible for training such that each trained model is less prone to overfitting than when larger parts of the data are taken out for testing at a time which is especially a concern when having very limited data. However, the drawback is that it requires N models to be trained which can be very wasteful. While leave-one-out cross-validation gives an almost unbiased estimate of the generalization error, in some cases it can have a larger variance compared with, say, 10-fold cross-validation. Overall, it is recommended to use 10-fold cross-validation [Kohavi, 1995].

In the following we will denote by $\hat{E}_{\mathcal{M}}^{\text{gen}}$ an estimate of the generalization error $E_{\mathcal{M}}^{\text{gen}}$ computed by any of the three above techniques. An illustration where 3-fold cross-validation is applied to the linear-regression example is given in fig. 10.7. Each figure corresponds to a fold and in each fold the test error is computed as the average of the error on the three datapoints that are left out. Notice, all nine data-points are part of the test set exactly once.

10.1.4 Cross-validation for model selection

We will accept that the generalization error eq. (10.4) is the optimal way to measure the performance of a model, and that cross-validation using any of the three techniques (hold-out, K -fold or leave-one-out) is a faithful estimate of the generalization error. Then, an obvious way to select between S models $\mathcal{M}_1, \dots, \mathcal{M}_S$ is to estimate the generalization error of each model using cross-validation and select the model with the lowest cross-validation error. To summarize:

- For each model, compute the estimate of the generalization error $\hat{E}_{\mathcal{M}_1}^{\text{gen}}, \dots, \hat{E}_{\mathcal{M}_S}^{\text{gen}}$ using cross-validation.
- Select the optimal model \mathcal{M}_{s^*} as that with the lowest error:

$$s^* = \arg \min_s \hat{E}_{\mathcal{M}_s}^{\text{gen}}$$

There is nothing more to cross-validation for model selection than this! This technique is most often used in conjunction with K -fold cross-validation. In this case it is strongly recommended that the same data splits (i.e. choices of $\mathcal{D}_1, \dots, \mathcal{D}_K$) is used for all models. Since the resulting method is so important it is provided as an explicit algorithm in algorithm 5.

An illustration of 3-fold cross-validation for model selection in the linear-regression example is given in fig. 10.8. Each of the columns corresponds to a fold and each of the rows to a model. In fig. 10.9 the estimated generalization and training errors (averaged over the cross-validation folds) is plotted. As can be seen the training error drops for the more complicated model, however, the cross-validation estimate of the generalization error allows us to select the right model \mathcal{M}_2 .

10.1.5 Two-layer cross-validation

Let's turn to the following situation: We wish to select the optimal model \mathcal{M}_{s^*} out of S models and estimate the generalization error for this optimal model \mathcal{M}_{s^*} . A tempting way to accomplish this is to apply K -fold cross-validation to estimate the generalization error for each model \mathcal{M}_s using cross-validation and select the model with the lowest generalization error and then use the estimate of the generalization error as an estimate of how well the model performs. This is illustrated in

Algorithm 5: *K*-fold cross-validation for model selection

Require: K , the number of folds in the cross-validation loop
Require: $\mathcal{M}_1, \dots, \mathcal{M}_S$. The S different models to select between
Ensure: \mathcal{M}_{s^*} the optimal model suggested by cross-validation

```

for  $k = 1, \dots, K$  splits do
    Let  $\mathcal{D}_k^{\text{train}}, \mathcal{D}_k^{\text{test}}$  the  $k$ 'th split of  $\mathcal{D}$ 
    for  $s = 1, \dots, S$  models do
        Train model  $\mathcal{M}_s$  on the data  $\mathcal{D}_k^{\text{train}}$ 
        Let  $E_{\mathcal{M}_s, k}^{\text{test}}$  be the test error of the model  $\mathcal{M}_s$  when it is tested on  $\mathcal{D}_k^{\text{test}}$ 
    end for
end for
For each  $s$  compute:  $\hat{E}_{\mathcal{M}_s}^{\text{gen}} = \sum_{k=1}^K \frac{N_k}{N} E_{\mathcal{M}_s, k}^{\text{test}}$ 
Select the optimal model:  $s^* = \arg \min_s \hat{E}_{\mathcal{M}_s}^{\text{gen}}$ 
 $\mathcal{M}_{s^*}$  is now the optimal model suggested by cross-validation

```

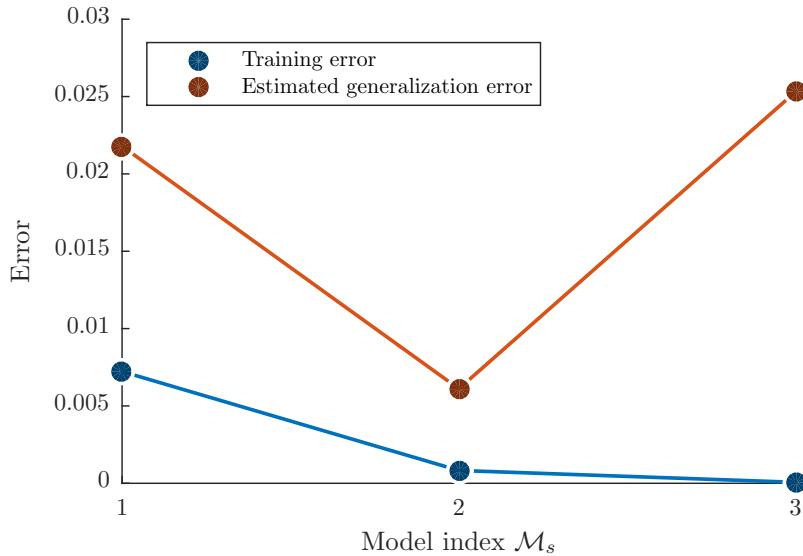


Fig. 10.9. Training error and the cross-validation estimate of the generalization error. The estimate of the generalization error is simply the averages of the errors in fig. 10.8 over all 3 folds as dictated by the cross-validation method for model selection. The model with the lowest estimated generalization error is \mathcal{M}_2 even though it does not have the lowest training error.

fig. 10.10 where we consider 14 different models and for each model the true generalization error is indicated as the black line and the estimated generalization error as the small red dots. The selected model is the model with the lowest (estimated) generalization error indicated with the red circle. The estimates of the generalization error is imprecise due to the randomness in the test set which is why they are not all on the black line.

There is however a problem with this approach. Suppose we had access to additional test sets and use these to estimate the generalization error (indicated in the 3 other panels of fig. 10.10). These too are estimates of the true generalization error (black line), but they are independent of

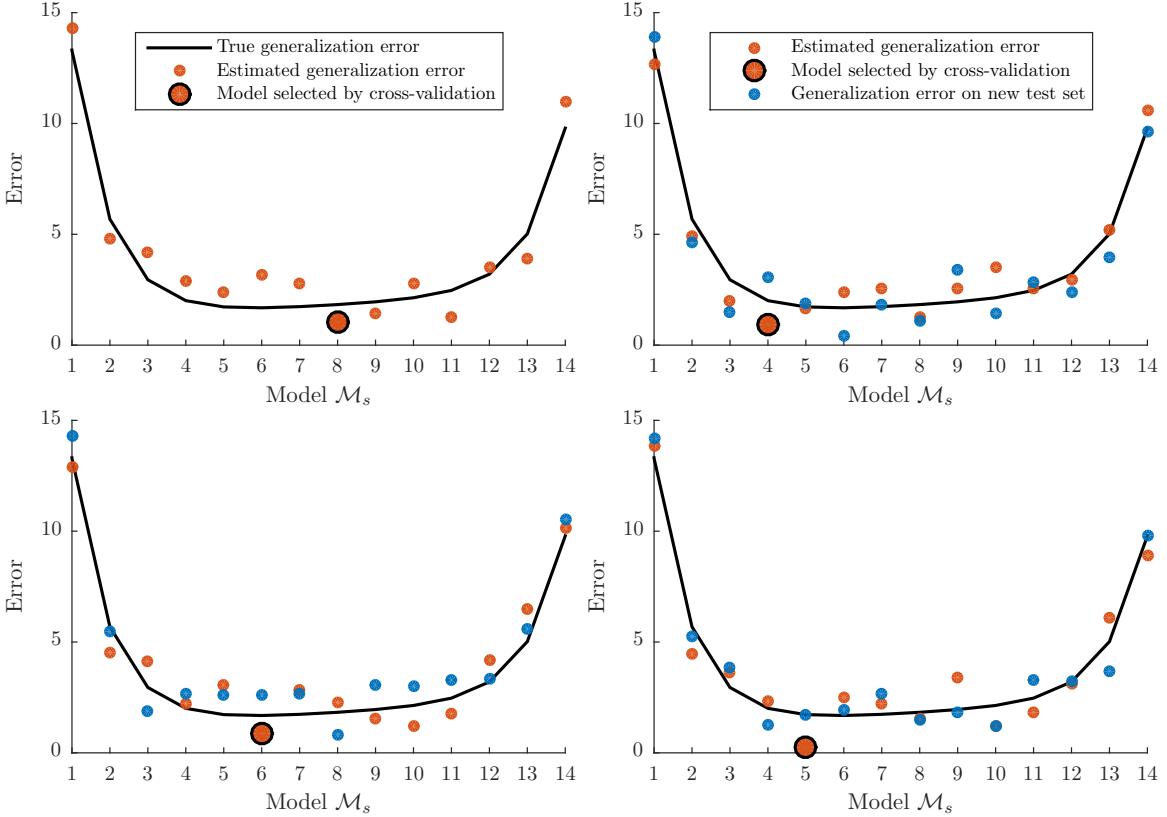


Fig. 10.10. Top right: The true generalization error of 14 models is indicated by the black line and estimates of the generalization error (computed using cross-validation) are indicated by red dots. Standard cross-validation then selects the model with the lowest (estimated) cross-validation error (indicated by red circle). However, this *estimate* of the generalization error is not in general a fair estimate of how the model will generalize to future data because it is selected as a minimum. In subplots 2-4 is shown the same procedure and as seen the estimated generalization error is too optimistic (below the black line) in all instances. A better estimate can be obtained by using a completely new test set, blue dots, which provides a fairer estimate of the generalization error for the selected values. This leads to two-layer cross-validation.

the red dots. However, since we always select the red dot with the *lowest* error, and the blue dots are *random*, we will in general be too optimistic with respect to our estimate of the generalization error. After all, there are many roughly equally good models to choose from, so when we select the *best* of these we will due to the randomness often do exceedingly well. In the figure, this is seen as the selected red point being far lower than the true generalization error in all instances. Obviously, this is cheating! To understand exactly what goes wrong we need to take a step back. By including the step where we select the optimal model $\mathcal{M}^* = M_{s^*}$ based on the data we have actually changed the underlying model being tested. The model the above method produces, \mathcal{M}^* , is now composed of two things:

- Use K_2 -fold cross-validation to estimate \hat{E}_s^{gen} .
- Select \mathcal{M}^* as the optimal model \mathcal{M}_{s^*} where $s^* = \arg \min_s \hat{E}_s^{\text{gen}}$.

Algorithm 6: Two-level cross-validation

Require: K_1, K_2 , folds in outer, and inner cross-validation loop respectively
Require: $\mathcal{M}_1, \dots, \mathcal{M}_S$: The S different models to cross-validate
Ensure: \hat{E}^{gen} , the estimate of the generalization error

```

for  $i = 1, \dots, K_1$  do
    Outer cross-validation loop. First make the outer split into  $K_1$  folds
    Let  $\mathcal{D}_i^{\text{par}}, \mathcal{D}_i^{\text{test}}$  be the  $i$ 'th split of  $\mathcal{D}$ 
    for  $j = 1, \dots, K_2$  do
        Inner cross-validation loop. Use cross-validation to select optimal model
        Let  $\mathcal{D}_j^{\text{train}}, \mathcal{D}_j^{\text{val}}$  be the  $j$ 'th split of  $\mathcal{D}_i^{\text{par}}$ 
        for  $s = 1, \dots, S$  do
            Train  $\mathcal{M}_s$  on  $\mathcal{D}_j^{\text{train}}$ 
            Let  $E_{\mathcal{M}_s, j}^{\text{val}}$  be the validation error of the model  $\mathcal{M}_s$  when it is tested on  $\mathcal{D}_j^{\text{val}}$ 
        end for
    end for
    For each  $s$  compute:  $\hat{E}_s^{\text{gen}} = \sum_{j=1}^{K_2} \frac{|\mathcal{D}_j^{\text{val}}|}{|\mathcal{D}_i^{\text{par}}|} E_{\mathcal{M}_s, j}^{\text{val}}$ 
    Select the optimal model  $\mathcal{M}^* = \mathcal{M}_{s^*}$  where  $s^* = \arg \min_s \hat{E}_s^{\text{gen}}$ 
    Train  $\mathcal{M}^*$  on  $\mathcal{D}_i^{\text{par}}$ 
    Let  $E_i^{\text{test}}$  be the test error of the model  $\mathcal{M}^*$  when it is tested on  $\mathcal{D}_i^{\text{test}}$ 
end for
Compute the estimate of the generalization error:  $\hat{E}^{\text{gen}} = \sum_{i=1}^{K_1} \frac{|\mathcal{D}_i^{\text{test}}|}{N} E_i^{\text{test}}$ 
```

Thus, estimating the generalization error requires estimating the generalization error of the model obtained through this two-step procedure. Fortunately, we know how to estimate the generalization error of a model: Cross-validation. Since the method now makes use of two nested cross-validation procedures, one in selecting \mathcal{M}_{s^*} as above and one for estimating performance, the resulting procedure is known as two-layer cross-validation. The method can be sketched as follows:

- For $i = 1, \dots, K_1$ cross-validation iterations, split the data \mathcal{D} into a training set $\mathcal{D}_i^{\text{par}}$ and a test set $\mathcal{D}_i^{\text{test}}$
- For each iteration, find the optimal value s^* using K_2 -fold cross-validation on $\mathcal{D}_i^{\text{par}}$. (In the j 'th inner fold $\mathcal{D}_i^{\text{par}}$ is split into a training set $\mathcal{D}_j^{\text{train}}$ and a test set (called a validation set) $\mathcal{D}_j^{\text{val}}$).
- Train the model \mathcal{M}^* using the selected model structure \mathcal{M}_{s^*} trained on the full outer fold training set $\mathcal{D}_i^{\text{par}}$
- Let $E_{\mathcal{M}^*, i}^{\text{test}}$ be the test error of \mathcal{M}^* computed on the i 'th test set $\mathcal{D}_i^{\text{test}}$
- Estimate the generalization error as $\hat{E}^{\text{gen}} = \sum_{i=1}^{K_1} \frac{|\mathcal{D}_i^{\text{test}}|}{N} E_{\mathcal{M}^*, i}^{\text{test}}$

Again, since this method is so important it is worth providing it in pseudo code as algorithm 6.

10.2 Sequential feature selection

Consider a dataset where observations \mathbf{x}_i correspond to patients and we wish to predict a patient's survival time after an operation y_i using linear regression. Suppose for each patient we observe three attributes namely: x_1 : Age, x_2 : The room number of the patient and x_3 : Length of hospital stay. Clearly, only the first and last attribute is relevant to our purpose, so rather than considering

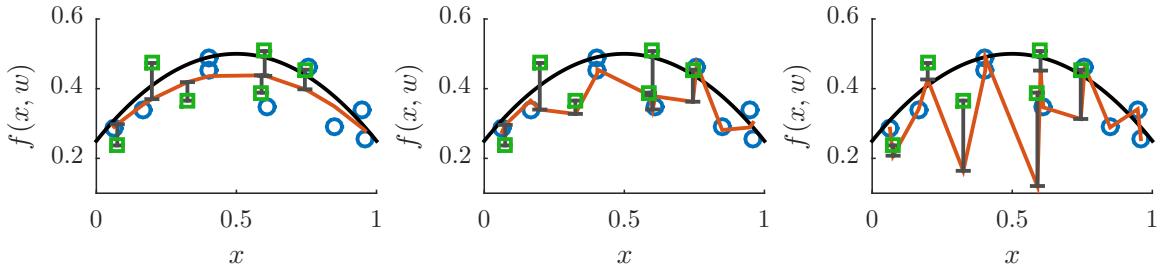


Fig. 10.11. A regularized linear regression model is fitted to the dataset of nine observations and six test observations. The different plots correspond to adding more “junk” attributes, i.e. attributes where the values are just random. As more random attributes are added, the model better fit the training set but does worse on the test set.

the attribute $\mathbf{x} = [x_1 \ x_2 \ x_3]^T$ we could just as well consider the smaller dataset: $\mathbf{x} = [x_1 \ x_3]^T$. So does it matter that we include the room number x_2 in our dataset? Well in general irrelevant attributes matter for three reasons:

- If the number of attributes (in particular the irrelevant ones) is large compared to the total number of observations our model performance will degrade.
- Storing and manipulating irrelevant attributes takes space and makes our models slower.
- A hospital will often wish to know which attributes are important and which are irrelevant. A model with many irrelevant attributes will not tell them that directly.

Let’s examine the first claim first. Suppose we have the simple, linear regression problem which can be fitted well with a second-order polynomial. That is, optimally we should consider:

$$\mathbf{x} = [x_1 \ x_1^2] \quad (10.5)$$

However, we now add “junk” attributes to the dataset and consider

$$\mathbf{x} = [x_1 \ x_1^2 \ x_3 \ x_4 \dots \ x_{S+2}]$$

where S is the number of junk attributes added to the dataset. Thus $S = 0$ will correspond to eq. (10.5) and $S = 3$ will correspond to adding 3 junk attributes. The junk attributes are simply generated as random numbers in the unit interval.

Examples of the predictions on training and test set for $S = 0, 3, 6$ added junk attributes can be seen in fig. 10.11. As can be seen, when more junk attributes are added, the model will begin to overfit the training set. This is easily seen when plotting the training error against the test error as is done in fig. 10.12 for $S = 0, \dots, 7$ and the three specific values shown in fig. 10.11 are indicated by the circles. In a way, we already know how to solve this problem: Each selection of which features to use corresponds to a particular model, so in for instance the hospital example we can consider all eight possible models

$$\begin{aligned} \mathcal{M}_{123} &= [x_1 \ x_2 \ x_3] & \mathcal{M}_{12} &= [x_1 \ x_2] & \mathcal{M}_{13} &= [x_1 \ x_3] & \mathcal{M}_{23} &= [x_2 \ x_3] \\ \mathcal{M}_1 &= [x_1] & \mathcal{M}_2 &= [x_2] & \mathcal{M}_3 &= [x_3] & \mathcal{M}_\cdot &= [\bullet], \end{aligned}$$

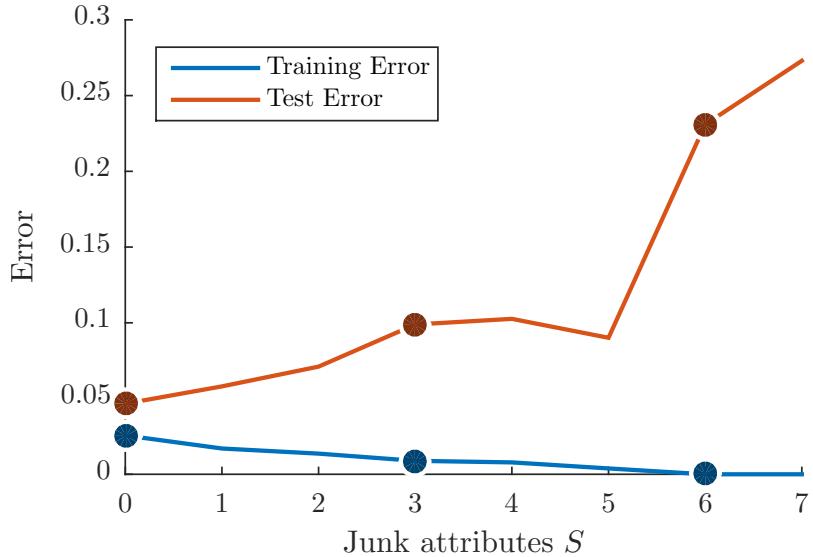


Fig. 10.12. The training and test error for different number of junk attributes in the example of fig. 10.11. The circles indicate the particular values shown in fig. 10.11.

and select the optimal model by the use of cross-validation for model selection as already described. In many ways this is the *best* we can do from a theoretical perspective, however, the problem is that this procedure quickly becomes very costly. In general, if we have M attributes to choose between, we must select between 2^M models, thus if $M = 20$ then this results in having to cross-validate more than a million different models. Clearly this won't do!

Sequential feature selection overcomes this problem by not considering *all* possible models but only a subset. Sequential feature selecting comes in two variation, forward and backward selection, but they are very similar.

10.2.1 Forward Selection

In *forward selection*, we first consider a model with no features

$$\mathcal{M}_\bullet = [\bullet]$$

That is, it predicts y_i as just being constant. Then it considers the models obtained by adding each attribute to the existing (empty) set of selected attributes thereby testing all the models:

$$\mathcal{M}_\bullet = [\bullet], \quad \mathcal{M}_1 = [x_1], \quad \mathcal{M}_2 = [x_2], \dots, \mathcal{M}_K = [x_M].$$

Each of these $M + 1$ models are evaluated by cross-validation for model selection and the optimal model, say \mathcal{M}_i , is selected. If \mathcal{M}_\bullet is selected the process terminates. Else, this procedure is now repeated by evaluating the M models corresponding to

$$\mathcal{M}_i = [x_i], \quad \mathcal{M}_{1i} = [x_1 \ x_i], \dots, \mathcal{M}_{i-1,i} = [x_{i-1} \ x_i], \dots, \mathcal{M}_{iM} = [x_i \ x_M]$$

Model	\hat{E}^{gen}
\mathcal{M}_\bullet	0.91
\mathcal{M}_1	0.86
\mathcal{M}_2	0.92
\mathcal{M}_3	0.88
\mathcal{M}_4	0.83
\mathcal{M}_{12}	0.78
\mathcal{M}_{13}	0.62
\mathcal{M}_{14}	0.78
\mathcal{M}_{23}	0.74
\mathcal{M}_{24}	0.72
\mathcal{M}_{34}	0.76
\mathcal{M}_{123}	0.64
\mathcal{M}_{124}	0.68
\mathcal{M}_{134}	0.73
\mathcal{M}_{234}	0.78
\mathcal{M}_{1234}	0.79

Table 10.1. The estimated generalization error \hat{E}^{gen} as estimated by cross-validation for models trained on different subsets of the features x_1, x_2, x_3 and x_4 .

Again, if \mathcal{M}_i is the optimal model the process terminates, else an optimal model (say model \mathcal{M}_{ij}) is selected and then all $M - 1$ models corresponding to \mathcal{M}_{ij} and the $M - 2$ models obtained by adding all other attributes than x_i, x_j to the set evaluated by cross-validation. If it is found that for instance \mathcal{M}_{ij} is the optimal model, the process terminates, else it continues possibly terminating with the full model: $\mathcal{M}_{12\dots M}$.

Example of forward selection

Let's illustrate this procedure with a concrete example. Suppose we have a dataset of $M = 4$ attributes giving 16 possible models with generalization errors (as estimated by cross-validation) shown in table 10.1. Forward selection now proceeds as follows

- Start with model \mathcal{M}_\bullet with an error of 0.91.
- Compare models \mathcal{M}_\bullet and $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4$.
- Optimal model is \mathcal{M}_4 with error of 0.83.
- Compare models \mathcal{M}_4 and $\mathcal{M}_{14}, \mathcal{M}_{24}, \mathcal{M}_{34}$.
- Optimal model is \mathcal{M}_{24} with error of 0.72.
- Compare models \mathcal{M}_{24} and $\mathcal{M}_{124}, \mathcal{M}_{234}$.
- Optimal model is \mathcal{M}_{124} with error of 0.68.
- Compare models \mathcal{M}_{124} and \mathcal{M}_{1234} .
- Since \mathcal{M}_{124} has lowest error, forward selection terminates and select features 1, 2, 4.

Notice the procedure is completely mechanical, however, it is not guaranteed to select the model with the *lowest* overall generalization error. The benefit of forward selection is naturally that we don't have to compute all the generalization errors beforehand but can compute them as they are required.

10.2.2 Backward Selection

Backward selection builds upon the same idea as forward selection, but instead of starting with the empty model \mathcal{M}_\bullet , we start with the full model $\mathcal{M}_{12\dots M}$ and instead of adding features, features are now *removed* one at a time. To continue the example from before:

- Start with model \mathcal{M}_{1234} with an error of 0.79.
- Compare models \mathcal{M}_{1234} and $\mathcal{M}_{123}, \mathcal{M}_{124}, \mathcal{M}_{134}, \mathcal{M}_{234}$.
- Optimal model is \mathcal{M}_{123} with error of 0.64.
- Compare models \mathcal{M}_{123} and $\mathcal{M}_{12}, \mathcal{M}_{13}, \mathcal{M}_{23}$.
- Optimal model is \mathcal{M}_{13} with error of 0.62.
- Compare models \mathcal{M}_{13} and $\mathcal{M}_1, \mathcal{M}_3$.
- Since \mathcal{M}_{13} has lowest error, backward selection terminates and select features 1, 3.

Notice forward selecting selected model \mathcal{M}_{124} and backward selection selected model \mathcal{M}_{13} . We are thus not guaranteed that these two methods will select the same set of features or that forward selection will select less features than backward selection (or the reverse). In general, compare both methods and see which has the lowest estimated generalization error.

The disadvantage of sequential feature selection is that we are not comparing all models and thus we might miss the model with the lowest generalization error. The advantage is runtime. Comparing all models requires 2^M model evaluations, while in the worst case forward (or backward) selection requires estimating the generalization error of

$$(1) + (M) + (M - 1) + (M - 2) + \dots + (1) = \frac{M(M + 1)}{2} + 1$$

models For $M = 20$ this correspond to only 211 models compared to more than a million models using exhaustive search. As a final note, it is strongly recommended to use the same cross-validation splits when estimating the generalization error of the models to reduce variance in the estimates of the generalization error. In fact, in the calculation above we have used that \mathcal{M}_\bullet does not need to be recalculated if using the same cross-validation splits.

10.3 Cross validation of time-series data★

Our discussion so far has assumed data is atemporal. This choice does not reflect the methods we have considered are irrelevant for time-series data, but rather that we wished to avoid unnecessary complications or caveats when discussing the particular methods.

For completeness, we have nevertheless decided to include a section about validation of time series data. A reader should be aware time-series analysis is a specialized subject, and one in which it is more difficult to provide general advice; we will therefore briefly introduce a few common-sense data-processing suggestions and warn the reader to approach this section with her critical faculties engaged and take note all of the following subsections are marked with a ★.

10.3.1 The setup

Time-series data is data which contains a feature corresponding to time, and where the temporal dependency between observations may be considered vital for the prediction task. In other words,

Table 10.2. The seven entries of the traffic data set

ID	Time t	x_{rain}	$x_{\text{temperature}}$	x_{station}	x_{cars}	y
1	13:20, Sep 6th, 2019	1	23.7	120	236	164
2	13:40, Sep 6th, 2019	1	22.9	141	249	163
3	14:00, Sep 6th, 2019	1	24.6	168	243	186
4	14:20, Sep 6th, 2019	0	25	179	250	184
5	14:40, Sep 6th, 2019	0	26.4	189	254	196
6	15:00, Sep 6th, 2019	0	26.1	215	271	197
7	15:20, Sep 6th, 2019	1	25.3	227	278	211

if we scrambled the time-coordinate, the machine-learning task should seem difficult or impossible. Examples could be the exchange rate of currencies, temperature as recorded throughout a day or a persons future decisions based on recordings of her brain.

The basic setup is as follows: We will assume we have access to a set of sensors which make recordings of a set of features \mathbf{x} , and a predictive target y , over time t . We denote the value at a particular time by:

$$\mathbf{x}(t) = [x_1(t) \ x_2(t) \ \cdots \ x_M(t)]^T, \quad y(t)$$

Then, our goal can be described as predicting the value of y at a future time $t + \Delta_t$ based on whatever data we have recorded, or otherwise have available, up to time t .

Concretely, assume measurements are taken of $\mathbf{x}(t)$ and $y(t)$ at N discrete time steps $t = t_1, t_2, \dots, t_N$. As an example, suppose we use a security camera to estimate the number of people at a specific train station, and it is this quantity we are interested in predicting based on 4 features. All in all:

$y(t)$: Estimated number of people at the station of interest

$x_{\text{rain}}(t)$: Whether it rains or not

$x_{\text{temperature}}(t)$: Temperature in Celsius

$x_{\text{station}}(t)$: Estimated number of people at (another) station

$x_{\text{cars}}(t)$: Car velocity at a major intersection km/hour

We have shown an (unrealistically small) example of this dataset in table 10.2 comprised of just $N = 7$ observations.

The standard format

Our first task is to bring our dataset into the standard format used throughout this course. To simplify the presentation, we will make the assumption that all sensors (values of features $\mathbf{x}(t)$) are recorded *simultaneously* and at *equidistant* time points (i.e., that the interval between when observations are taken is the same). If the later assumption is violated one can do one of three things:

- Ignore the problem; this may lead to useful results if the data is nearly uniformly sampled or the relative spacing between observations is not that important.
- If the data is super-sampled, that is, the intervals between each observation is very small, we can consider subsampling the dataset at regularly spaced intervals.

Table 10.3. The traffic from table 10.2 processed by introducing the hour-of-day and day-of-week features

x_{rain}	$x_{\text{temperature}}$	x_{station}	x_{cars}	x_{hour}	x_{day}	y
1	23.7	120	236	13	4	164
1	22.9	141	249	13	4	163
1	24.6	168	243	14	4	186
0	25	179	250	14	4	184
0	26.4	189	254	14	4	196
0	26.1	215	271	15	4	197
1	25.3	227	278	15	4	211

- One can interpolate the values at regularly spaced intervals (note this is probably inappropriate if the features are categorical).

When the data is divided into equidistant time points we can (linearly) index them by the time point t_i at which the observations was recorded which we call *time steps* or simply *steps*. In the traffic example table 10.2 all observations are made at the same time points and with regular intervals of 20 minutes and the step corresponds to the ID column.

In addition, we will make the assumption the dataset is *stationary*, which we will here loosely define as not being affected by long-running trends (for instance that the number of people at the train station is increasing over time as the train service becomes more popular). If the dataset appears to be obviously non-stationary, one should attempt to standardize it, for instance by regressing out a linear trend.

It is at this point tempting to turn the prediction problem into simply predicting y at step i based on features corresponding to step $i - 1$ and so on. However, in the traffic example this would mean we lost the *most important* pieces of information, namely *when* during the day we wish to predict y . It is therefore (in this particular example) appropriate to introduce an x_{hour} -feature. In a similar vein, weekends implies that the *day* of the week is probably also important (as commuters are likely to change habits during the weekend) and we introduce an x_{day} -feature.

Note these new observations are special, as we can pre-compute them for the future time $y(t)$ which we wish to predict. It makes sense to separate these features from the remaining features which *cannot* be predicted; simply put, if we know we are predicting $y(t)$ for a Monday, we are not provided with additional information to know that at earlier times it was Sunday, Saturday, etc. In the following, we assume that such *special* variables are separated from those that cannot be pre-computed, and we will focus on the non-pre-computed variables in the following.

As a final comment, note that it may make sense to represent the hour and day-variables as categorical variables and perform an one-out-of- K encoding, as representing them with real numbers (artificially) implies the distance between 23:00 and 02:00 is greater than 08:00 and 11:00.

At any rate, after dropping the irrelevant ID feature, the dataset table 10.2 will now be assumed to look like table 10.3.

Embedding size and horizon

The *horizon* $h \geq 1$ is an integer denoting *how far into the future* we are trying to make predictions whereas the *embedding size* $p \geq 1$ refers to *how many past observations* are relevant to make predictions about the future.

Dataset	Padding $p = 1$, horizon $h = 1$	Padding $p = 1$, horizon $h = 2$	Padding $p = 2$, horizon $h = 1$
$\tilde{x}_{11} \tilde{x}_{12}$	\tilde{y}_1	\tilde{y}_2	\tilde{y}_3
$\tilde{x}_{21} \tilde{x}_{22}$	\tilde{y}_2	\tilde{y}_3	\tilde{y}_4
$\tilde{x}_{31} \tilde{x}_{32}$	\tilde{y}_3	\tilde{y}_4	\tilde{y}_5
$\tilde{x}_{41} \tilde{x}_{42}$	\tilde{y}_4	\tilde{y}_5	
$\tilde{x}_{51} \tilde{x}_{52}$	\tilde{y}_5		

Fig. 10.13. Illustration of different choices of embedding size and horizons. A dataset (left) consisting of two feature-attributes (green, blue) and prediction targets (red) in the format defined in eq. (10.6) is converted into the (standard) (\mathbf{X}, \mathbf{y}) format used elsewhere in the book for three different choices of embedding size (number of past time steps) and horizons (number of steps to predict into the future), see eq. (10.7)

To make this concrete, suppose the (non-instantaneous) part of the dataset, i.e. the label values and the first four attributes in table 10.3 are denoted by:

$$\tilde{\mathbf{X}} = \begin{bmatrix} \tilde{\mathbf{x}}_1^T \\ \tilde{\mathbf{x}}_2^T \\ \vdots \\ \tilde{\mathbf{x}}_N^T \end{bmatrix} = \begin{bmatrix} \tilde{x}_{11} & \tilde{x}_{12} & \cdots & \tilde{x}_{1M} \\ \tilde{x}_{21} & \tilde{x}_{22} & \cdots & \tilde{x}_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{x}_{N1} & \tilde{x}_{N2} & \cdots & \tilde{x}_{NM} \end{bmatrix}, \quad \tilde{\mathbf{y}} = \begin{bmatrix} \tilde{y}_1^T \\ \tilde{y}_2^T \\ \vdots \\ \tilde{y}_N^T \end{bmatrix}. \quad (10.6)$$

For an embedding of size p and horizon h , we will therefore predict \tilde{y}_{i+h-1} based on data available at steps $i-p$ to $i-1$, i.e.:

$$\text{predict } \tilde{y}_{i+h-1} \text{ based on } [\tilde{\mathbf{x}}_{i-p} \ \tilde{\mathbf{x}}_{i-p+1} \ \cdots \ \tilde{\mathbf{x}}_{i-2} \ \tilde{\mathbf{x}}_{i-1} \ \tilde{y}_{i-p} \ \tilde{y}_{i-p+1} \ \cdots \ \tilde{y}_{i-2} \ \tilde{y}_{i-1}]. \quad (10.7)$$

Naturally, if we had access to additional, instantaneous features such as x_{hour} and x_{day} these would be added to the vector of attributes in eq. (10.7). This construction likely seem a bit abstract and we have illustrated it in fig. 10.13 for three different choices of horizons and embedding sizes.

In a concrete application, the embedding size should be selected based on a consideration of how far into the past observations remain predictive of the presence, whereas the horizon should be selected based on what our modeling goal is. If the goal is a traffic planner for commuters, a horizon of about an hour ($h = 3$, as each step corresponds to 20 minutes) might be appropriate, whereas if we are trying to plan if extra trains should be included a planning horizon of several hours is probably more appropriate. It is difficult to judge how large the embedding size should be in this example; one could argue an embedding size of 1 (i.e. just the preceding time step) is relevant, or a large embedding size because commuters who took the train in the morning are likely to also use it at the end of their work day. The recommendation is to begin with a small embedding size for simplicity, and use cross-validation to make informed decisions.

What about multiple time series?

In the case of multiple time series what should be done depends slightly on the specifics of the dataset. If for instance the time series represents instances of the same process, but recorded in sequences with breaks in between, we recommend each section is simply processed using the procedure in this section and the data is pooled at the end. An example of this in the context of the traffic example could be if data is only recorded every other week.

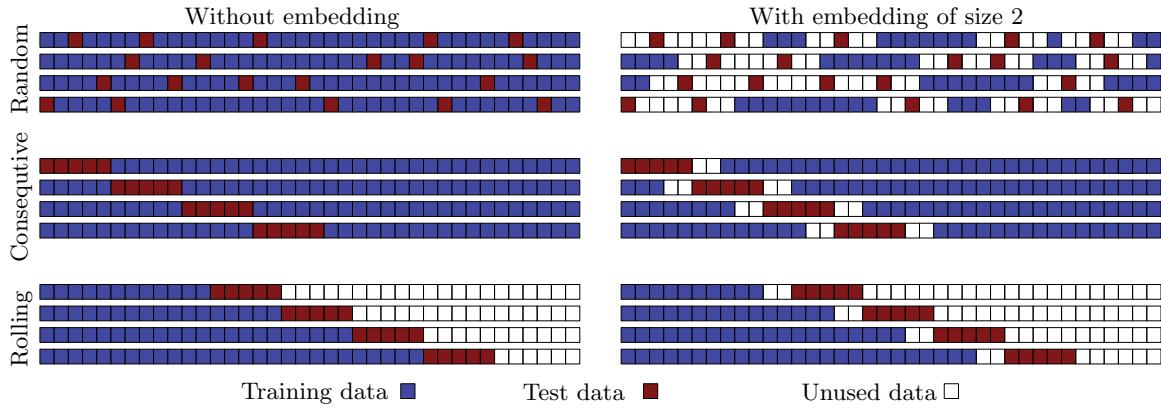


Fig. 10.14. Examples of different cross-validation approaches. Notice each square corresponds to both (\mathbf{x}_i, y_i) values transformed using the approach outlined in section 10.3.1, see fig. 10.13. For each approach, we only show the first 4 of the cross-validation folds.

If on the other hand the different time series represent the same basic phenomena, but can nevertheless be grouped into a fixed number of groups representing a (systematical) difference in the recording procedure, one could consider introducing a categorical variable such that the machine-learning method understands which type of recording is currently under consideration. An example of this in the traffic-example could be if in some weeks we record the number of people at the train station using one method, and in other weeks with another, and we think these may yield systematically different estimates.

10.3.2 Cross-validation

We will limit ourselves to K -fold cross-validation as leave-one-out cross-validation is just a special case and the hold-out method should be trivial. The temporal dependency makes it difficult to provide a single, unique recommendation for doing cross-validation, and we will therefore suggest different approaches and their possible strengths/weaknesses. An interested reader can consult Bergmeir et al. [2018] and references therein for a more comprehensive discussion.

Consider the schemes indicated in fig. 10.14. In the figures, each of the N squares correspond to a training observation and label information (\mathbf{x}_i, y_i) , and it will be assumed these have been converted using the procedure outlined in the previous section (see fig. 10.13).

The rows represent three schemes for selecting training and test splits:

Random splits Training/test splits are selected non-overlapping and at random

Consecutive splits As above, but the K folds consist of consecutive observations

Rolling The test set is always in the future relative to the training set

A difficulty is that training observation \mathbf{x}_i will contain y_{i-1} , and in general for embeddings of size p observation \mathbf{x}_i will contain observations y_{i-p}, \dots, y_{i-1} . This means the training and test sets will overlap which in turn means we are likely to under-estimate the generalization error. For that reason, it is often desirable to pad the test set by removing a set of observations on each side corresponding to the correlation length in the data, here assumed to be roughly the embedding size. This is shown in the right-column of fig. 10.14.

A few remarks are at this point in order for (possible) advantages and disadvantages of the different methods: The random and consecutive splits are likely to behave quite similar without padding, however with padding, and in particular if the dataset is very small, padding will remove proportionally more of the data for random splits (biasing the generalization error estimate upwards). On the other hand, selecting splits consecutively rather than randomly mean the data in the test set in each fold will overlap, thereby increasing the variance of the generalization error estimate.

For both the random and consecutive selections, a problem is we include the future in the training set, whereas in the real world we will train on past data and test on future. The rolling method removes the future data from the splits, however, this (in turn) means we have access to less training data and will possibly bias the generalization error estimate slightly upwards.

10.3.3 Two-layer cross-validation

The extension to two-layer cross-validation is technically straight-forward: Simply choose one of the approaches indicated in fig. 10.14 for the outer fold, and another for the inner fold. Note, however, that the various biases, advantages and disadvantages of the outlined approaches might interact in undesirable ways and degree of appropriateness of a given approach is likely going to be highly problem-dependent and the estimates of the generalization error is likely to differ, especially for datasets of limited size. We therefore recommend a preference is given to simpler models, with a primary focus on demonstrating (a minimum of) past information predicts the (immediate) future *better* than simply using immediately available information. In the case of the traffic example, even assuming we had access to a non-trivial number of observations, it is not immediately clear that past number of passengers during a day improves predictions of the present beyond knowing the day of week and time of day.

10.4 Visualizing learning curves*

The generalization error is such an important quantity it can be used as a qualitative pointer to model problems, here illustrated using *learning curves*. A learning curve refers to how the performance of a given method change as a function of a key quantity or parameter. We have already encountered examples of learning curves, for instance in fig. 7.15 we considered a schematic illustration of learning as a function of computation time and in fig. 1.11 we considered performance as a function of training set size.

In this chapter, we will try to provide an intuitive feeling for how different shapes of learning curves may point to different problems (or lack therefore) with a given method. A word of warning: This discussion should not be taken too literal, and a reader should not expect their curves to re-produce those in this section exactly, as they *will* depend quite strongly on the specifics of the chosen method, evaluation metric and data set. For this reason the section is marked with a ★.

10.4.1 The setup

We will consider a setting where we are trying to solve a supervised learning problem using a somewhat advanced method. Using the techniques considered in this chapter, we can compute the

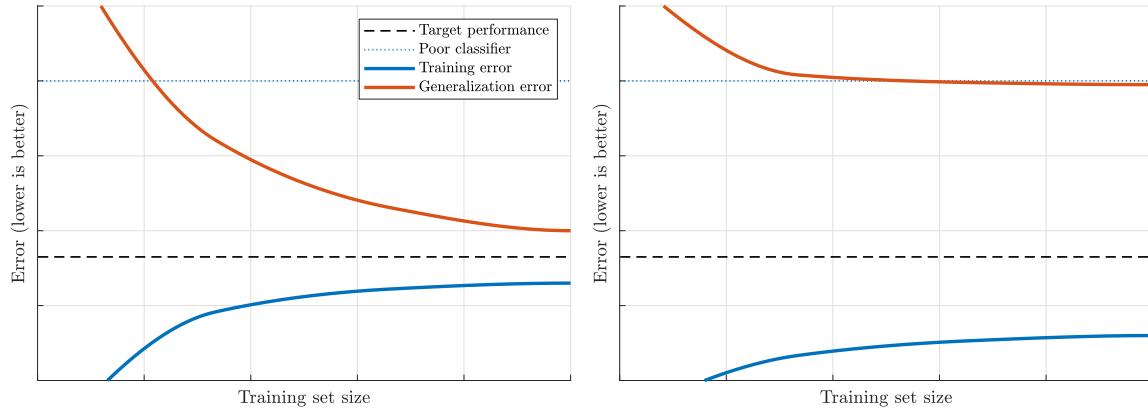


Fig. 10.15. Illustration of good and bad learning outcomes. the left-hand pane shows stereotypical behaviour of a good learning outcome: The training/generalization error is well below baseline performance (dotted blue line), and as more data is added the generalization error drops towards the target performance. That the generalization error is higher than the training error is to be expected in a complex model and should drop with more data. The right-hand pane shows a model which overfit badly. The training error remains very low, but the generalization error does not budge when more data is added. This is probably due to a poor model choice.

training error and *estimate* the generalization error, for instance by simply computing the error on a test set as discussed earlier in this chapter.

In addition to these two quantities, we assume (this may be realistic or not) we have some target performance in mind (i.e. the point where the method is considered useful, good enough or equivalent to state-of-the-art), as well as a baseline of some kind, for instance a linear/logistic regression. Within this setup, we will plot the (estimate of) the generalization and training error as a function of the amount of training data.

Consider the learning curves shown in fig. 10.15. In the left-hand pane we see an example of a learning curve where the training error is fairly low, the generalization error is low (and falling rapidly when more data is available).

This is the expected outcome for a well-applied method: The training error will, within statistical uncertainty, be lower than the generalization error, however the gap should close with more data and the error should approach target performance.

On the other hand, the right-hand pane of fig. 10.15 provides an example of what we do not want to see. The generalization error is very high (nearly corresponding to random guessing), whereas the training error is extremely low. Moreover, none of these appears to budge. This is because the method is overfitting the training set: The method extracts information specific to each training observation, while learning nearly nothing about the general (true) relationship in the test data. This corresponds to a method which learns to identify cars by observing images of cars contain blue sky and therefore learn this relationship. Moreover the curves are nearly flat, indicating adding more data will not change the situation. When one encounters this behavior, the go-to suggestion is to switch to a simpler model where training does not fail.

These two examples provides extremes, however learning curves can, with some care, be used to diagnose other more complex problems. We should here stress our warning from the introduction,

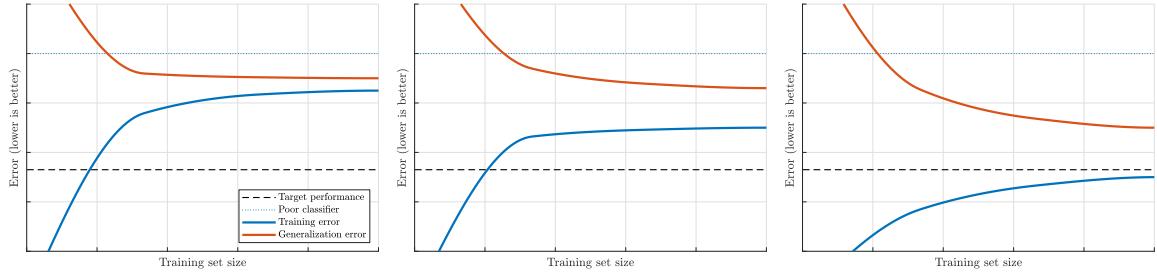


Fig. 10.16. Examples of learning curves. Left: a weak/misapplied model where learning converge, but it is too inflexible to learn the right relationship. Middle: A bad method which is too inflexible to learn the correct relationship in the data (high training error), however the generalization error remains much higher than the training error indicating it is simultaneously overfitting. Right: Example where learning seems to be converging; consider more data/tweaking and see if the generalization error sustains the drop.

namely that the cartoonish examples of learning curves encountered in the happy world of pedagogy, and the ones a reader will likely encounter in practice, will only be approximately similar. For instance, the learning curve in fig. 10.15 (left) should be compared to the (comparable) learning curve in fig. 7.14 (right) obtained using logistic regression.

Method is too weak or misapplied

In the left-hand pane of fig. 10.16 we see another example of a bad outcome: the generalization error seems to have plateaued, but moreover the training error is very high. This method is probably too weak: it is unable to extract enough information about the true relationship in the data to learn it, even on the training set. In this case one should consider if the method is applied correctly (perhaps it is not learning anything?), or consider a more flexible method.

Wrong method

Next consider the center-pane of fig. 10.16. This is another bad outcome where the method is both too weak to capture the true relationship in the data (And thus, as in the previous case, we cannot expect more data will fix our problems), but the generalization error is higher than the training error, indicating some degree of overfitting. The method is therefore learning too little and the little it is learning is the wrong thing. Something is seriously wrong; either with the method, or with the way the training/test set is selected, and more data is unlikely to fix the problems.

More data and tweaking

Finally consider the right-hand pane of fig. 10.16. In this case the training curve seems to have plateaued around the expected level of performance, whereas the generalization error seems to drop as more data is added. In this case we can hope adding more data, and possible tweaking the method a bit to account for hard cases, will get us the rest of the way.

Problems

10.1. Question 1: Alice is considering a linear regression model for a dataset comprised of $N = 1000$ observations. She wishes to both select the optimal regularization strength as well as estimate the generalization error of the model at the optimal regularization strength. To simplify the problem, she only considers the following 6 possible values of the regularization strength λ :

$$\lambda = 10^{-2}, 10^{-1}, 10^0, 10^1, 10^2, 10^3.$$

Alice opts for a two-level strategy in which she uses the hold-out method to estimate the generalization error and cross-validation is used to select the optimal regularization strength, i.e. the dataset is first divided into a validation set $D_{\text{validation}}$, comprised of 20% of the full dataset, and the remainder D_{CV} is used for cross-validation. Alice uses standard $K = 10$ fold cross-validation to select the optimal regularization strength on D_{CV} and, having estimated the optimal regularization strength, uses the hold-out method on D_{CV} and $D_{\text{validation}}$ to estimate the generalization error.

Suppose for any fixed value of the regularization strength, the time taken to *train* the weights of the linear regression model on a dataset of size N_{train} is N_{train}^2 units of time and the time taken to *test* a trained model on a dataset of size N_{test} is $\frac{1}{2}N_{\text{test}}^2$ units of time. Suppose the duration of all other tasks is negligible, what is the total time taken for the entire procedure?

- A $12.78 \cdot 10^6$ units of time.
- B $15.98 \cdot 10^6$ units of time.
- C $31.30 \cdot 10^6$ units of time.
- D $31.96 \cdot 10^6$ units of time.
- E Don't know.

10.2. Question 2: We would like to fit an artificial neural network to the PM10 dataset shown in Table 10.4. It is decided that DAY should not be included in the model as this cannot be influenced by decision makers. We therefore only consider x_1, x_2, x_3 and x_4 corresponding to logCAR, TEMP, WIND and TEMPDIF respectively. An artificial neural network is applied to the data with these four attributes. The neural network has three hidden units and is trained using different combinations of the four attributes x_1, x_2, x_3 and x_4 . Table 10.5 gives the training and test performance of the artificial neural network for different combinations of the four attributes. Which one of the following statements is *correct*?

No.	Attribute description	Abbrev.
x_1	Logarithm of number of cars per hour	logCAR
x_2	Temperature 2 meter above ground (degree Celsius)	TEMP
x_3	Wind speed (meters/second)	WIND
x_4	Temperature difference between 25 and 2 meters (degree Celsius)	TEMPDIF
x_5	Wind direction (degrees between 0 and 360)	WINDDIR
x_6	Whole hour of the day	HOUR
x_7	Day number from October 1. 2001	DAY
y	Logarithm of PM10 concentration	logPM10

Table 10.4. The attributes of the PM10 data. The output is given by the hourly values of the logarithm of the concentration of PM10 particles (logPM10).

Feature(s)	Training		Test
	rmse	rmse	rmse
x_1	0.71	0.75	
x_2	0.58	0.64	
x_3	0.60	0.62	
x_4	0.92	0.94	
x_1 and x_2	0.60	0.69	
x_1 and x_3	0.35	0.44	
x_1 and x_4	0.52	0.66	
x_2 and x_3	0.56	0.69	
x_2 and x_4	0.45	0.52	
x_3 and x_4	0.62	0.64	
x_1 and x_2 and x_3	0.36	0.34	
x_1 and x_2 and x_4	0.28	0.33	
x_1 and x_3 and x_4	0.27	0.45	
x_2 and x_3 and x_4	0.20	0.43	
x_1 and x_2 and x_3 and x_4	0.10	0.35	

Table 10.5. Root mean square error (rmse) for the training and test set when using an artificial neural network with three hidden units to predict the level of pollution (logPM10) based only on the first four attributes (x_1 – x_4) using the hold-out method with 50 % of the observations hold-out for testing.

- A Neither forward nor backward selection will identify the optimal feature combination for this problem.
- B Backward selection will result in a better model being selected than using forward selection.
- C Backward selection will use a model that include all the features x_1, x_2, x_3 , and x_4 .
- D Forward selection will select the features x_1, x_2 and x_4 .
- E Don't know.

10.3. Question 3: Which one of the following statements is *incorrect*?

- A Cross-validation can be used to quantify a models generalization error.
- B K-fold cross-validation requires the fitting of K models such that for K=N where N is the total number of observations K-fold cross-validation is the same as leave-one-out cross-validation.
- C When using cross-validation to estimate model parameters an extra level of cross-validation is needed in order to evaluate how well the model generalizes to new data.
- D For least squares linear regression the test error will always decrease as we include more attributes in the model.
- E Don't know.

10.4. Question 4: We would like to fit an artificial neural network (ANN) model to the Galapagos dataset shown in Table 10.6. To reduce the computational cost of fitting the models it was decided to not include Elev, i.e. x_4 , and AreaNI, i.e. x_7 in the ANN models. We therefore only consider x_1 , x_2 , x_5 and x_6 in order to predict Area, i.e. x_3 . An artificial neural network with three hidden units is applied to the data with these four attributes and trained using different combinations of the four attributes x_1 , x_2 , x_5 and x_6 . Table 10.7 gives the training and test performance in terms of root mean squared error (rmse) of the ANN for different combinations of the considered attributes. Which one of the following statements is *correct*?

No.	Attribute description	Abbrev.
x_1	Number of plant species	Plants
x_2	Number of endemic plant species	E-Plants
x_3	Area of island (in km^2)	Area
x_4	Max. elevation above sea-level (in m)	Elev
x_5	Distance to nearest island (in km)	DistNI
x_6	Distance to Santa Cruz Island (in km)	StCruz
x_7	Area of adjacent island (in km^2)	AreaNI

Table 10.6. The seven attributes of the data on a selection of 29 of the Galápagos islands.

Feature(s)	Training rmse	Test rmse
x_1	81.2	80.1
x_2	62.3	84.3
x_5	68.0	72.9
x_6	98.9	100.5
x_1 and x_2	57.1	69.1
x_1 and x_5	40.2	43.3
x_1 and x_6	55.2	66.4
x_2 and x_5	32.2	36.3
x_2 and x_6	20.3	22.5
x_5 and x_6	48.4	50.3
x_1 and x_2 and x_5	36.6	39.1
x_1 and x_2 and x_6	18.8	23.0
x_1 and x_5 and x_6	33.3	36.7
x_2 and x_5 and x_6	40.4	43.0
x_1 and x_2 and x_5 and x_6	30.0	35.2

Table 10.7. Root mean square error (rmse) for the training and test set when using an artificial neural network with three hidden units to predict the Area of an island based only on the four attributes x_1 , x_2 , x_5 and x_6 using the hold-out method with 40 % of the observations hold-out for testing.

- A Neither forward nor backward selection will identify the optimal feature combination for this problem.
- B Forward selection will result in a better model being selected than using backward selection.
- C Backward selection will terminate at the model that includes the features x_1 , x_2 , and x_6 .
- D Forward selection will select the features x_2 and x_5 .
- E Don't know.

10.5. Question 5: A logistic regression model was trained on Haberman's data shown in Table 10.8 using leave-one-out cross-validation and the confusion matrices given in Figure 10.17 were obtained. Which one of the following statements is *correct*?

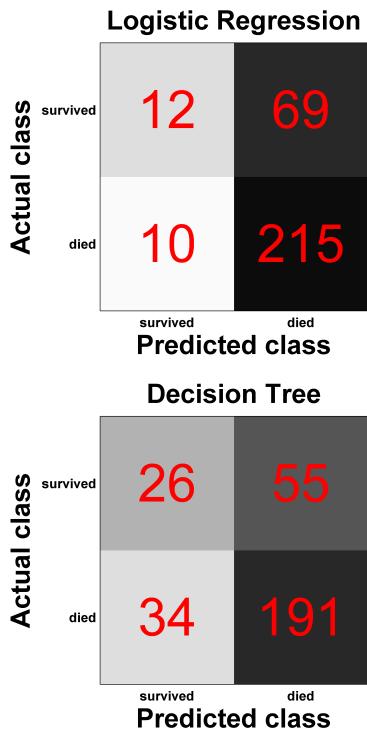


Fig. 10.17. The confusion matrix for the logistic regression and decision tree classifiers used to predict survival based on leave-one-out cross validation.

No.	Attribute description	Abbrev.
x_1	Young (< 60 years), $x_1 = 0$ or Old (≥ 60 years), $x_1 = 1$	Age
x_2	Operated before, $x_2 = 0$ or after 1960, $x_2 = 1$	OpT
x_3	Positive axillary nodes detected PAN No, $x_3 = 0$ or Yes, $x_3 = 1$	
y	Lived after 5 years No, $y = 0$ or Yes, $y = 1$	Surv

Table 10.8. A modified version of Haberman's Survival Data taken from <http://archive.ics.uci.edu/ml/machine-learning-databases/haberman/haberman.names>. The attributes x_1-x_3 denoting the age, operation time and cancer size as well as the output denoting survival after five years are binary. The data contains a total of $N = 306$ observations.

- A The error rate of the logistic regression classifier is larger than the error rate of the decision tree classifier.
- B Predicting every observation to be in the died class would give a better accuracy than the accuracy obtained by the decision tree classifier.
- C The classification problem does not have any issues of imbalanced classes.

- D Using leave-one-out cross validation a total of $306 - 1 = 305$ logistic regression models are trained.
- E Don't know.

10.6. Question 6: Suppose a neural network is trained on input/output pairs (x_i, y_i) on a dataset of $N = 5$ observations. The response of the neural network for input x_i is denoted as \hat{y}_i and the values can be seen in table 10.9.

i	y_i	\hat{y}_i
1	1	0.6
2	0	0.4
3	1	0.5
4	1	0.1
5	0	0.1

Table 10.9. Table desired responses and output of the neural network

We convert the continuous output \hat{y}_i to a proper binary class label by the transformation

$$\hat{y}_i \leftarrow \begin{cases} 1 & \text{if } \hat{y}_i \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

Which of the following values of θ will give the highest accuracy for binarized outputs?

- A $\theta = 0.35$
- B $\theta = 0.45$
- C $\theta = 0.55$
- D $\theta = 0.65$
- E Don't know.

10.7. Question 7: Which one of the following statements pertaining to cross-validation is *correct*?

- A A 2-fold cross-validation is the same as the hold-out method when 50% is hold out.
- B For datasets with very few observations it is in general better to use leave-one-out cross-validation rather than 10-fold cross-validation.
- C Two levels of cross-validation is necessary in order to determine the optimal set of parameters for a model.
- D Leave-one-out cross validation is the most computational efficient procedure as only one observation is in the test set at a time.
- E Don't know.

10.8. Question 8: Consider a system which attempts to classify observations based on the value of four observed features x_1, x_2, x_3, x_4 . Suppose we wish to examine which subset of features gives the least misclassified observations on test data. In table 10.10 is shown how different combinations of features give rise to different number of misclassified observations on a training and test set. Which of the following statements is true?

Feature(s)	C_{train}	C_{test}
None	30	69
x_1	43	70
x_2	14	50
x_3	41	76
x_4	18	81
x_1, x_2	59	73
x_1, x_3	34	59
x_1, x_4	15	32
x_2, x_3	18	58
x_2, x_4	23	36
x_3, x_4	26	33
x_1, x_2, x_3	17	40
x_1, x_2, x_4	37	54
x_1, x_3, x_4	7	15
x_2, x_3, x_4	27	34
x_1, x_2, x_3, x_4	17	25

Table 10.10. Number of misclassified training observations C_{train} and test observations C_{test} for a neural network model trained on different sets of features. Lower is better.

A Forward and backward selection will select the same set of features.

B Forward selection will select a model with higher misclassification rate on the test set than backward selection.

C Forward selection will select a model with lower misclassification rate on the test set than backward selection.

D Forward selection will select the features x_1, x_3, x_4 .

E Don't know.

11

Performance evaluation

In the past sections, we have seen several methods for making predictions on data. To determine which one to use for a particular problem, and therefore to make progress, we need a systematic method for comparing one with another. Statistical tests provides such a method, and this chapter will supply the reader with tools for solving most common model comparison problems.

The main source of difficulty as a user of statistics is how to match a particular problem to a statistical test. The main distinction is whether we are willing to accept our conclusions are conditional on our particular dataset (this will be called **setup I**) or we want our conclusions to remain valid when using another dataset generated by the same mechanism (this will be called **setup II**). We have devoted section 11.1 to clarifying this distinction and providing an overview of the tests as well as concrete advice on when to use which.

We will assume the reader is familiar with basic statistical concepts such as p -values and confidence intervals, and our brief recap in section 11.2 is only intended to introduce our notation.

The chapter, along with the choice of tests and approach, is inspired by Dietterich [1998]. McNemars test, which we will recommend to compare classifiers, is described by Altham [1971] and our treatment of what we describe as **setup II** is based on ideas described in Nadeau and Bengio [2000].

11.1 Statistical testing for machine learning

Consider the simplest setup in which we are given two models \mathcal{M}_A and \mathcal{M}_B and we wish to know if one of the models is better than the other. As we learned in chapter 10, the preferable model is the one which is better at predicting future data, which we measure by the generalization error.

It is at this point necessary to make an important distinction, namely whether our conclusions should be valid for models trained only on the training data we have available, or if our conclusions should generalize to a new training data we might encounter in the future (from the same data population).

To illustrate this distinction, consider a classification problem where the goal is to determine the chance a person will experience post-surgical complications, and we must accomplish this with our available dataset \mathcal{D} consisting of N observations.

Suppose we consider two candidate models for this task: \mathcal{M}_A is a classification tree model and \mathcal{M}_B is an artificial neural network. Accordingly, we train our models on our available datasets \mathcal{D}

and obtain two prediction rules:

$$f_{\mathcal{D},A}, \quad \text{and} \quad f_{\mathcal{D},B}.$$

The subscript is used to clarify that these rules depend on the data they are trained on. The problem of determining which model is better is therefore simply to examine the difference in their generalization error $z_{\mathcal{D}}$ defined as

$$z_{\mathcal{D}} = E_{\mathcal{D},A}^{\text{gen}} - E_{\mathcal{D},B}^{\text{gen}} \quad (11.1)$$

$$\text{where: } E_{\mathcal{D},A}^{\text{gen}} = \int p(\mathbf{x}, y) L(f_{\mathcal{D},A}(\mathbf{x}), y) d\mathbf{x} dy, \quad E_{\mathcal{D},B}^{\text{gen}} = \int p(\mathbf{x}, y) L(f_{\mathcal{D},B}(\mathbf{x}), y) d\mathbf{x} dy. \quad (11.2)$$

In practice, we cannot compute the generalization errors exactly, but can only hope to estimate them using cross validation as described in chapter 10. To avoid complicating things, let's assume we have a test set $\mathcal{D}^{\text{test}}$ available, in which case we can estimate the difference in generalization errors as:

$$\begin{aligned} \hat{z}_{\mathcal{D}} &= \frac{1}{N^{\text{test}}} \sum_{i=1}^{N^{\text{test}}} [L(f_{\mathcal{D},A}(\mathbf{x}_i), y_i) - L(f_{\mathcal{D},B}(\mathbf{x}_i), y_i)] \\ &= \frac{1}{N^{\text{test}}} \sum_{i=1}^{N^{\text{test}}} z_i, \quad \text{where: } z_i = L(f_{\mathcal{D},A}(\mathbf{x}_i), y_i) - L(f_{\mathcal{D},B}(\mathbf{x}_i), y_i). \end{aligned} \quad (11.3)$$

Statistics allow us to use the numbers $z_1, \dots, z_{N^{\text{test}}}$ to draw conclusions about the differences between the true generalization errors defined in eq. (11.1). Crucially, since $z_{\mathcal{D}}$ depends on the *specific* dataset \mathcal{D} , so will our statistical conclusions. We will denote this general problem, where we consider the training set \mathcal{D} as fixed and condition our conclusions on this dataset, by **setup I** in the following:

Setup I Statistical tests of performance considering the *specific* training set \mathcal{D} ?

Returning to the post-surgical example, the conclusions we might arrive at under **setup I** therefore have to be stated conditional on \mathcal{D} . We might (for instance) find model \mathcal{M}_A is significantly better than \mathcal{M}_B , but our conclusion can only be said to have been tested (and therefore, be valid) in the case the models are trained on \mathcal{D} . Section 11.3 addresses typical questions under this heading for both classification and regression:

Section 11.3.2: Estimate plausible values of the generalization error $E_{\mathcal{D}}^{\text{gen}}$ defined in eq. (11.1) for a classification model

Section 11.3.3: If \mathcal{M}_A and \mathcal{M}_B are two classifiers, how to tell if they have the same performance (or not) and if so, by how much

Section 11.3.4: Estimate plausible values of the generalization error $E_{\mathcal{D}}^{\text{gen}}$ for a regression model

Section 11.3.5: If \mathcal{M}_A and \mathcal{M}_B are two regression models, how to tell one is better than another and if so, by how much

It should be obvious there are situations where it is undesirable to state conclusions conditionally on a specific training set. For instance, if a research group at another hospital tried to independently replicate our results, then even though they followed our exact protocols and collected data from the same population of patients, they would end up with another dataset \mathcal{D}' , and they might not reach the same conclusion as us.

Obviously, in many cases we want to know if our conclusions can actually be independently reproduced by other researchers. We can give this a technical formulation by saying that assuming our dataset \mathcal{D} came from some distribution

$$\mathcal{D} \sim p_0(\cdot).$$

Then we want our conclusions to be valid for a comparable dataset \mathcal{D}' of the same size generated according to the same distribution $\mathcal{D}' \sim p_0(\cdot)$:

Setup II Statistical tests of performance considering a dataset of size N

We will outline statistical methods that attempts to address this problem for both classification and regression models in section 11.4.

Which setup should I choose?

As indicated above, **setup II** is a more general conclusion, and typically the conclusion which would be of greater scientific interest: When we say model \mathcal{M}_A is better than model \mathcal{M}_B , what most people understand is that conclusions should be reproducible by another team of researchers, i.e., not depend on the specifics of our training set.

On the other hand, since **setup II** is more general it is also more difficult to confirm. What this means is that it will typically requires more computations, a bigger performance gap between the models, and more data in order for us to demonstrate an effect, assuming there is a difference between the model.

Our overall recommendation is therefore that a reader has a preference towards **setup II** and considers **setup I** in the following cases:

- It might be the case the problem has a clearly defined training/test set other researchers are testing on as well. In this case the training set is fixed and the methods in **setup I** are applicable
- There is so little data the methods for **setup II** are unfeasible
- A company could argue they only care about performance on their specific training set
- It is too computationally expensive to train multiple models

11.2 Statistical primer★

To avoid confusion with machine-learning concepts, we will write the data that enters into our statistical test as D . For instance, in the model-comparison problem eq. (11.3) from the previous section we estimated the difference in model performance as:

$$\hat{\theta} = \frac{1}{n} \sum_{i=1}^n z_i. \quad (11.4)$$

In this case the data is simply the n numbers:

$$D = (z_1, \dots, z_n). \quad (11.5)$$

Generally speaking, the problem statistics is concerned with is how we can use the n numbers in D to draw conclusions about the true value of the difference in generalization error $\theta = E_{A,\mathcal{D}}^{\text{gen}} - E_{B,\mathcal{D}}^{\text{gen}}$. There are two major categories of statistical inference:

Hypothesis testing How to use a finite sample D to answer a binary question such as whether $\theta = 0$. We will use hypothesis testing and p -values for this task.

Estimation How to use a finite sample D to find a plausible range of values the true value of the generalization error is likely to fall within. We will use confidence intervals for this task.

For most tasks, a small performance difference such as 1% between two models is irrelevant, and the emphasis should therefore be on parameter estimation which allows us to distinguish trivial differences from those of practical significance. We therefore recommend the emphasis is placed on computing (and discussing) confidence intervals in a results section, and that p -values are used secondarily.

Parameter

In statistics, the dataset D is assumed to be a random sample from a population. Specifically, we assume each observation z_i in D is a realization of a random variable Z_i , which follows a distribution that depends on the parameter θ and has density

$$p(Z_i = z_i | \theta) = p_\theta(z_i)$$

The goal of statistics is to use a concrete, observed dataset $D = (z_1, \dots, z_n)$ to draw conclusions about θ , which is why we use the abbreviation p_θ to signify the special role of θ . Note that by the product rule the density of the full dataset is simply

$$p_\theta(D) = \prod_{i=1}^n p_\theta(z_i). \quad (11.6)$$

Statistics makes the assumption the dataset was generated from this distribution using a specific value of θ , and the goal of statistics is to make reasonable statements about this value using the dataset D . For instance, we might be interested in knowing if the true value of θ is greater than zero.

What distribution we use in place of p_θ is a crucial choice, and using the wrong distribution is guaranteed to lead to troubles. In other words, statistics assume we know enough about our data to make this choice.

Statistic

A statistic is a function of the data D and will be denoted t . For instance, the mean and variance are both statistics:

$$t_0(D) = \frac{1}{n} \sum_{i=1}^n Z_i, \text{ or } t_1(D) = \frac{1}{n} \sum_{i=1}^n (Z_i - t_0(D))^2.$$

Estimator

Estimation is concerned with finding a statistic t of D such that $t(D)$ is close to θ , and in this case t is called an *estimator* of θ . In the examples we will consider the mean

$$t_0(D) = \frac{1}{n} \sum_{i=1}^n Z_i$$

will be a good estimator for θ . If n is low the estimator is relatively unreliable, whereas if n is large it will converge to the true value.

Confidence interval

An estimator $t(D)$ of the parameter θ is just a single number and provides no information about the relative accuracy. A *confidence interval* (CI) is an attempt to provide this information by finding an interval $[\theta_L, \theta_U]$ where the true value θ is likely to reside.

Obviously, such an interval has to be a function of the data D , in other words, θ_L and θ_U are two statistics and for a concrete dataset the interval is computed to be

$$[\theta_L(D), \theta_U(D)]. \quad (11.7)$$

Specifically, the main property of a confidence interval is that with a probability of $1 - \alpha$, the true value θ should fall within the confidence interval $[\theta_L(D), \theta_U(D)]$ as we randomize over different datasets generated from our distributional assumption $p_\theta(D)$ in eq. (11.6). Symbolically, this is written as

$$P_\theta(\theta \in [\theta_L, \theta_U]) = 1 - \alpha. \quad (11.8)$$

The number α denotes a margin of error we are willing to tolerate: If α is close to 0, the CI will be very wide and very likely to contain the true value θ . On the other hand if α is larger, the CI is more narrow and more likely to not contain θ . Therefore, the functions θ_L and θ_U have to depend on both D and α .

Since this definition is quite technical, we have provided a mechanical description in Box 11.2.1.

Technical note 11.2.1: Understanding confidence intervals

The two statistics θ_L and θ_U is an $1 - \alpha$ confidence interval for θ if for any *specific* value of θ , $\theta = \theta_0$, it is the case that:

- We generate a large number S of datasets D^1, \dots, D^S distributed as $p_{\theta=\theta_0}$
- For each dataset D^k , we compute the confidence interval

$$[\theta_L(D^k), \theta_U(D^k)]$$

- Of all these S intervals, a fraction $1 - \alpha$ will contain θ_0

Hypothesis

The other category of statistical inference is *hypothesis testing*. Hypothesis testing is concerned with whether a specific hypothesis H_0 about the true parameter θ is true or false. Examples could be testing whether θ takes a particular value such as 0

$$H_0 : \theta = 0.$$

For a given hypothesis H_0 , we denote by H_1 the negation of H_0 . In our example, H_1 corresponds to $\theta \neq 0$.

Insofar as the mathematics concerned, we could just as well let H_0 be $\theta \neq 0$ and H_1 be $\theta = 0$, however, it is customary to let H_0 correspond to the case of *no difference* or *no effect*, i.e. what we as scientists would typically want to disprove in order to confirm some hypothesis under question. In this case H_0 is called the *null hypothesis*.

The *p*-value

What is it about a dataset D that can tell us whether a hypothesis H_0 is true or false? One intuition is that if the data seems implausible under our hypothesis H_0 , we should doubt it. Let's make this concrete. Suppose once more we consider a hypothesis of the form

$$H_0 : \theta = \theta_0$$

for some particular value of θ_0 . For our concrete, observed dataset D define

$$t_0 = t(D)$$

where t is some statistic. In the cases we will consider, the statistic of interest is an estimator of θ and accordingly:

$$t_0 = t(D) = \frac{1}{n} \sum_{i=1}^n z_i$$

If H_0 is true, we know $\theta = \theta_0$, and so we know the distribution $p_\theta(D)$. From this, we can work out the probability $t(D)$ takes a particular value to be:

$$p(t(D) = t | H_0) = p_{\theta=\theta_0}(t(D) = t)$$

Although the details will require some algebra. The statistics $t(D)$, when used for null hypothesis testing, is known as the *test statistic*. We can now formalize the notation of observing a *more extreme* value of $t(D)$ to be the chance $t(D) \geq |t_0|$ – the absolute value takes into account extreme can both mean extremely small or extremely large. The probability of this event is the *p*-value:

$$\textit{p-value} : p = P(t(D) > |t_0| | H_0) = P_{\theta=\theta_0}(t(D) \geq |t_0|). \quad (11.9)$$

Example 11.2.1: *p*-value in the simplest case★

Suppose the distribution of our dataset in eq. (11.6) is simply a normal distribution with unknown mean θ and known variance σ_0^2 :

$$p_\theta(z_i) = \mathcal{N}(z_i | \mu = \theta, \sigma^2 = \sigma_0^2). \quad (11.10)$$

In this case, assuming H_0 is true such that $\theta = 0$, it is possible to show that the mean $t(D) = \frac{1}{n} \sum_{i=1}^n z_i$ is distributed as

$$p_\theta(t(D) = t | H_0) = \mathcal{N}\left(t | \mu = 0, \sigma^2 = \frac{\sigma_0^2}{n}\right). \quad (11.11)$$

(c.f. Petersen et al. [2008, section 8.1.4]). We can now compute the *p*-value using the cumulative density function of a normal distribution as:

$$\begin{aligned} P_\theta(t(D) > |t_0| | H_0) &= \int_{-\infty}^{-|t_0|} p(t(D) = t | H_0) dt + \int_{|t_0|}^{\infty} p(t(D) = t | H_0) dt \\ &= 2 \text{cdf}_{\mathcal{N}}\left(-|t_0| \mid \mu = 0, \sigma^2 = \frac{\sigma_0^2}{n}\right). \end{aligned}$$

In other words, the p -value captures how *unlikely* it is to observe a value at least as extreme as the concrete value t_0 given H_0 is true, with the intuition that the more unlikely our observed observations are, the more reason we have to doubt H_0 . Since this definition too can appear convoluted we provide a concrete definition in Technical Note 11.2.2 and a simple example in Example 11.2.1.

Statistical significance

Since the lower the p -value is the more reason there is to doubt H_0 , it is customary to specify a *significance level* α , such as $\alpha = 0.05$, and say that if p falls below this threshold we *reject* H_0 (and say our test showed a *significant result*), and otherwise we *fail to reject* H_0 (and our test did not show a significant result).

The technical language (*reject* and *fail to reject*) is used to emphasize a significant result, or a low p -value in general, is not the same as demonstrating a research hypothesis is true. When we obtain a p -value, what we strictly speaking can conclude is our data is rare or surprising assuming H_0 is true. There can be different reasons for this, for instance that H_0 is actually false, or it could be our data simply violates our statistical assumptions.

Technical note 11.2.2: Understanding p -values

Suppose we consider the hypothesis $H_0 : \theta = 0$ (versus $H_1 : \theta \neq 0$) and assume our concrete, observed dataset is $D = (z_1, \dots, z_n)$. The value of the test statistic t is then:

$$t_0 = \frac{1}{n} \sum_{i=1}^n z_i.$$

If we find this value t_0 has a p -value of $p = 0.03$ it means that:

- If we generate a large number S of datasets D^1, \dots, D^S generated under the assumption H_0 is true, i.e., distributed as $p_{\theta=0}(D)$
- For each dataset D^s , we compute the static $t(D^s)$

$$t_0^s = \frac{1}{n} \sum_{i=1}^n t(D^s) = \frac{1}{n} \sum_{i=1}^n z_i^s$$

- Then out of these S numbers, only a fraction of $p = 0.03$ will be more extreme than t_0 , i.e. satisfy

$$t_0^s \geq |t_0| \quad \text{or} \quad t_0^s \leq -|t_0|.$$

In other words, the p -value expresses the relative rarity of our observed statistic t_0 under the assumption H_0 is true.

11.2.1 Baselines

When assessing the results of a particular machine-learning method, it is often useful to compare it against a very simple model which can be implemented at a minimum of effort to demonstrate the

model is doing something useful at all. Such a model is commonly denoted a *baseline model*. For regression, the simplest baseline model is the model which simply outputs the mean of the training set:

$$f_{\text{baseline}}(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N^{\text{train}}} y_i^{\text{train}}. \quad (11.12)$$

And for classification, a reasonable baseline is one which computes the number of observations $n_c = \sum_{i=1}^{N^{\text{train}}} \delta_{y_i^{\text{train}}, c}$ assigned to class c for $c = 1, \dots, C$ and then always outputs the class with the most members:

$$f_{\text{baseline}}(\mathbf{x}) = c^*, \quad c^* = \arg \max_c \{n_1, \dots, n_C\}. \quad (11.13)$$

As the notation indicate, it is important the baseline models are treated as regular models. I.e. they are trained on the training sets, and later they are evaluated on the test sets.

Note the baseline model does not use any of the features \mathbf{x} . Therefore, if your model is unable to outperform the baseline, it means it does not make meaningful use of the features. This can either be because your model is implemented wrong, it is unsuited for the task, or because there is a serious data quality issue.

11.3 Setup I: the training set is fixed

Recall once more \mathcal{D} is our full dataset of N observations. In this section, we will consider the case where we train one (or two) models on \mathcal{D} , and then ask what the plausible values of the generalization error is for our model $f_{\mathcal{D}}$ trained on the dataset \mathcal{D}

$$E^{\text{gen}} = \int p(\mathbf{x}, y) L(f_{\mathcal{D}}(\mathbf{x}), y) d\mathbf{x} dy \quad (11.14)$$

Where L is the loss. As in eq. (11.3), we will approximate the generalization error using a test set. This can be done using one of the three forms of cross-validation from chapter 10. Since we do not want to confuse this choice with the test, we will first introduce general notation which will allow us to apply our tests to any form of cross-validation.

11.3.1 Translating to a statistical test

Both leave-one-out, K -fold and hold out cross validation can be seen as splitting the dataset \mathcal{D} into training/test pairs:

$$(\mathcal{D}_1^{\text{train}}, \mathcal{D}_1^{\text{test}}), \dots, (\mathcal{D}_K^{\text{train}}, \mathcal{D}_K^{\text{test}}). \quad (11.15)$$

where $\mathcal{D} = \mathcal{D}_k^{\text{train}} \cup \mathcal{D}_k^{\text{test}}$. In case of leave-one-out cross-validation $K = N$, and in case of hold-out $K = 1$ (as there is only one training and test set). The next step is to train the model on each of the K training sets and produce predictions on the K test sets. If we denote the predictions of model k on $\mathcal{D}_k^{\text{test}}$ with $\hat{\mathbf{y}}_k$, we obtain the prediction vector:

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_K \end{bmatrix}. \quad (11.16)$$

We will denote the dimension of $\hat{\mathbf{y}}$ by n , and note that $n = N$ in case of K -fold or leave-one-out and less than N in case of hold-out. Finally, we define:

$$z_i = L(\hat{y}_i, y_i), \quad \text{for } i = 1, \dots, n. \quad (11.17)$$

These n numbers will be the input to our statistical test. The loss L depends on the application. In case we are evaluating a regression model, we might choose either the squared loss or the L_1 loss, i.e. select:

$$L(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2 \quad \text{or} \quad L(\hat{y}_i, y_i) = |\hat{y}_i - y_i|. \quad (11.18)$$

In case the model is a classifier, the loss L will correspond to the error rate

$$L(\hat{y}_i, y_i) = \begin{cases} 0 & \text{if } \hat{y}_i = y_i \\ 1 & \text{if otherwise.} \end{cases} \quad (11.19)$$

Using eq. (11.17), we can estimate the generalization error eq. (11.14) as

$$E^{\text{gen}} = \int p(\mathbf{x}, y) L(f_{\mathcal{D}}(\mathbf{x}), y) d\mathbf{x} dy \approx \frac{1}{n} \sum_{i=1}^n z_i. \quad (11.20)$$

Technical note 11.3.1: What happened to $f_{\mathcal{D}}$?

The observant reader will note a slight point of irritation. The generalization error relevant for **setup I**, see eq. (11.15), is defined for $f_{\mathcal{D}}$ trained on the full dataset \mathcal{D} , whereas when we apply cross-validation we are only training on subsets of \mathcal{D} , and that in the case of K -fold and leave-one-out the estimate of the error is based on multiple models, $f_{\mathcal{D}_1^{\text{train}}}, \dots, f_{\mathcal{D}_K^{\text{train}}}$. This is annoying but not critical. Under normal circumstances, we would expect our model to perform a little poorer when trained on less data and therefore, that our estimates of E^{gen} to overshoot a little. Insofar this has an effect we are erring on the side of caution. This consideration suggests an optimal choice of cross-validation for **setup I**: leave-one-out, as in this case the models we are averaging should be as similar to $f_{\mathcal{D}}$ as possible. Note this conclusion does not hold for **setup II**.

11.3.2 First task: Evaluation of a single classifier

For classifiers, we are interested in estimating the accuracy. Therefore, in this section and section 11.3.3 assume $\hat{\mathbf{y}} = \hat{y}_1, \dots, \hat{y}_n$ has been computed using a form of cross-validation as in eq. (11.16), and then define c_i as whether the prediction for observation i was correct or not:

$$c_i = \begin{cases} 1 & \text{if } \hat{y}_i = y_i \\ 0 & \text{if otherwise.} \end{cases} \quad (11.21)$$

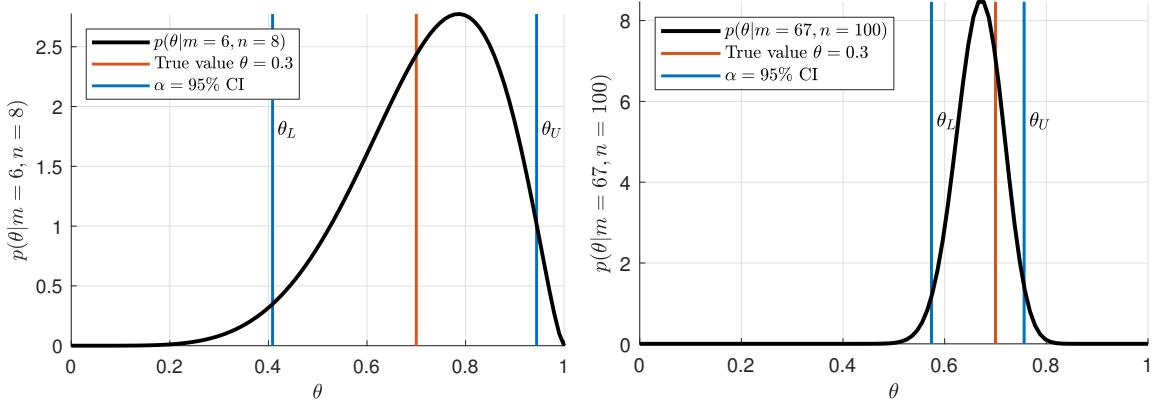


Fig. 11.1. Confidence interval for $\alpha = 0.05$ for a single classifier in two cases, $N = 8, m = 6$ and $N = 100, m = 67$ (left and right pane). The confidence interval corresponds to the area where (under the model assumptions) we can expect θ to be with probability $1 - \alpha = 95\%$.

(note this is closely related to the loss since $z_i = 1 - c_i$). How good a classifier is only depends on n and the number of times it made the correct prediction which can be computed as

$$m = \sum_{i=1}^n c_i.$$

Assuming θ is the (unknown) probability the model correctly classifies each observation, what we are trying to learn is $p(\theta|m, n)$. This problem is exactly equal to the Bernoulli coin we encountered in section 6.4 where θ is the chance a coin comes up heads, m the number of times it came up head and N the number of flips. For technical reasons¹, we will choose a $\text{Beta}(\theta|\alpha = \frac{1}{2}, \beta = \frac{1}{2})$ distribution as prior for θ . This prior is known as the *Jeffrey prior*.

If we simply accept this choice, we can re-use the derivation in Technical Note 6.4.1. Using $\alpha = \beta = \frac{1}{2}$ in eq. (6.38) we obtain:

$$\begin{aligned} p(\theta|m, n) &= \frac{\Gamma(n+1)}{\Gamma(\frac{1}{2} + m)\Gamma(\frac{1}{2} + n - m)} \theta^{\frac{1}{2} + m - 1} (1 - \theta)^{\frac{1}{2} + n - m - 1} \\ &= \text{Beta}(\theta|a, b), \quad a = m + \frac{1}{2}, \text{ and } b = n - m + \frac{1}{2}. \end{aligned} \tag{11.22}$$

Returning to our problem of interest, deriving the confidence interval, we can at this point simply observe that eq. (11.22) provides a formula for the posterior distribution. Using the definition in section 6.3.4 the cdf defined in eq. (6.20) is simply

¹ The motivation for this prior has to do with invariance under re-parameterization, somewhat similar to how a Gaussian with identity covariance matrix looks the same if the coordinate system is rotated. An interested reader can consult Jeffreys [1946] or Jaynes [1968] for further details. An intuitive interpretation of the difference is if we recall our interpretation of α and β as pseudo-counts, we may say this prior is less informed than $\alpha = \beta = 1$ as it corresponds to one less pseudo-observation.

Table 11.1. Intermediate computation and credibility interval for the example in fig. 11.1

	n	m	a	b	θ_L	θ_U
Case 1	8	6	6.5	2.5	0.41	0.94
Case 2	100	67	67.5	33.5	0.57	0.76

$$\text{cdf}_B(\theta|a,b) = \int_0^\theta \text{Beta}(\theta'|a,b)d\theta'$$

and if we let $\text{cdf}_B^{-1}(\theta|a,b)$ be the inverse cumulative distribution function for the beta distribution² the *Jeffrey interval*, defined as $[\theta_L, \theta_U]$ in eq. (6.26), can be computed as:

$$\theta_L = \text{cdf}_B^{-1}\left(\frac{\alpha}{2} \mid a, b\right) \quad (11.23)$$

$$\theta_U = \text{cdf}_B^{-1}\left(1 - \frac{\alpha}{2} \mid a, b\right) \text{ where: } a = m + \frac{1}{2} \text{ and } b = n - m + \frac{1}{2}. \quad (11.24)$$

The method is summarized in Box 11.3.1 and the reader is referred to Example 11.3.1 for an illustration.

Method 11.3.1: Jeffreys interval

- Select a form of cross-validation
- Obtain n predictions $\hat{y}_1, \dots, \hat{y}_n$ using eq. (11.16)
- Let $m = \sum_{i=1}^n c_i$ be the number of times the classifiers prediction is correct
- The $1 - \alpha$ confidence interval $[\theta_L, \theta_U]$ is now obtained by first computing $a = m + \frac{1}{2}$ and $b = N - m + \frac{1}{2}$ and then:

$$\theta_L = \text{cdf}_B^{-1}\left(\frac{\alpha}{2} \mid a, b\right) \text{ if } m > 0 \text{ otherwise } \theta_L = 0 \quad (11.25a)$$

$$\theta_U = \text{cdf}_B^{-1}\left(1 - \frac{\alpha}{2} \mid a, b\right) \text{ if } m < n \text{ otherwise } \theta_U = 1 \quad (11.25b)$$

Report results as follows: A point estimate of the accuracy of the classifier is $\hat{\theta} = \frac{a}{a+b}$, and an $\alpha = 0.95$ Jeffreys interval is given as $[\theta_L, \theta_U]$.

² Special functions to evaluate cdf_B^{-1} are build into all popular mathematical environments. In matlab: $\text{cdf}_B^{-1}(A|a,b) = \text{betainv}(A,a,b)$, in R: $\text{cdf}_B^{-1}(A|a,b) = \text{qbeta}(A,a,b)$ and in Python: $\text{cdf}_B^{-1}(A|a,b) = \text{beta.ppf}(A,a,b)$

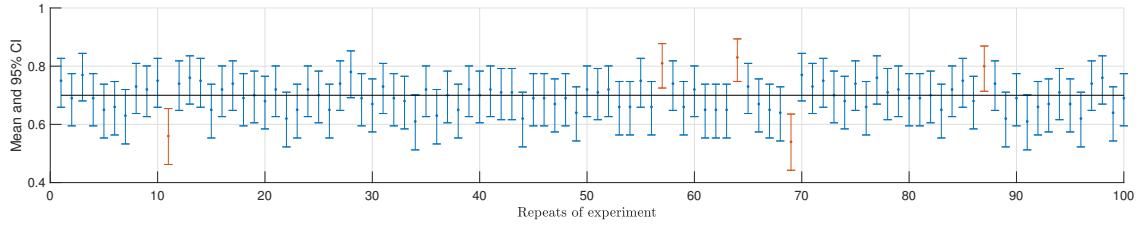


Fig. 11.2. Illustration of the reproducibility of statistical results. We fix the true accuracy of a classifier to $\theta = 0.7$, and then test it on 100 datasets where on each dataset we compute the $\alpha = 0.95$ confidence interval. Therefore, each of these 100 results are independent examples of what we could actually measure for a classifier with an error rate 0.7.

Example 11.3.1: Credibility interval of a single classifier

Let's turn to how this looks in practice. Suppose the true accuracy of a classifier is $\theta = 0.7$ and we consider two situations: In one we observe $n = 8$ and $m = \sum_{i=1}^n z_i = 6$, and in the other $n = 100$ and $m = 67$. In both cases we compute the 95% Jeffreys interval using eq. (11.25).

The intermediate calculations can be found in table 11.1 and the posterior curve of θ is illustrated in fig. 11.1. We see the true value of $\theta = 0.7$ lies within both intervals, as we would expect 95% of the time.

Reproducibility of Jeffreys interval★

We will illustrate the reproducibility of the Jeffrey interval with a simple, simulated example. Suppose we keep the true accuracy of the classifier at $\theta = 0.7$, generate 100 random test datasets of size $n = 100$, and then test it on 100 datasets where on each dataset we compute the $\alpha = 0.95$ confidence interval. The result of this procedure is shown in fig. 11.2. In other words, each of these 100 results are an example of what we could actually measure for a classifier with accuracy rate 0.7. Importantly, note about 5% exclude the true value as we would expect, and there is a significant variability in these results. Note this variability reflect the “best case” scenario because there is no variability in the true error rate θ , and illustrates why a great deal of care must be taken to conclude one model is better than another when using a small test set.

11.3.3 Comparing two classifiers

In this example, we have two classifiers \mathcal{M}_A and \mathcal{M}_B , and we want to know if one is better than another. The first step is to once more use cross-validation (hold-out, K -fold, leave-one-out) to obtain K splits into train/test as in eq. (11.15), then train (and test) both models *on the same splits* (why will be apparent in a moment) and thereby obtain, for each of the models, n predictions as well as the true class labels \mathbf{y} :

$$\hat{\mathbf{y}}^A = \hat{y}_1^A, \dots, \hat{y}_n^A, \quad \hat{\mathbf{y}}^B = \hat{y}_1^B, \dots, \hat{y}_n^B.$$

Given these, define c_i^A and c_i^B as in eq. (11.21)

$$c_i^A = \begin{cases} 1 & \text{if } \hat{y}_i^A = y_i \\ 0 & \text{if otherwise.} \end{cases} \quad \text{and} \quad c_i^B = \begin{cases} 1 & \text{if } \hat{y}_i^B = y_i \\ 0 & \text{if otherwise.} \end{cases} \quad (11.26)$$

The first step is to form the 2×2 matched-pair matrix with entries:

$$n_{11} = \sum_{i=1}^n c_i^A c_i^B = \{\text{Both classifiers are correct}\} \quad (11.27a)$$

$$n_{12} = \sum_{k=1}^n c_k^A (1 - c_k^B) = \{A \text{ is correct, } B \text{ is wrong}\} \quad (11.27b)$$

$$n_{21} = \sum_{k=1}^n (1 - c_k^A) c_k^B = \{A \text{ is wrong, } B \text{ is correct}\} \quad (11.27c)$$

$$n_{22} = \sum_{k=1}^n (1 - c_k^A)(1 - c_k^B) = \{\text{Both classifiers are wrong}\} \quad (11.27d)$$

What matters when comparing two classifiers is not the case where both classifiers are either both correct or incorrect as this might simply indicate an observation is either trivial to classify or impossibly hard. What is of interest is when the two classifiers *differ* in their predictions.

Therefore, we should take it as evidence that A is better than B when $n_{12} > n_{21}$. To test this statistically, we will use what is known as McNemars test [Altham, 1971]. Note that if $n_{12} > n_{21}$, this is equivalent to whether

$$\hat{r} = \frac{n_{12}}{n_{12} + n_{21}} > \frac{1}{2}.$$

This inspired the following idea: First, we let p_{ij} be the probability that a random pair of observations z_k^A, z_k^B will count towards n_{ij} . Note that

$$\sum_{i=1}^2 \sum_{j=1}^2 p_{ij} = 1.$$

What we want to test the null hypothesis which is if $p_{12} = p_{21}$ or equivalently if

$$r = \frac{p_{12}}{p_{12} + p_{21}} = \frac{1}{2}. \quad (11.28)$$

A *p*-value from the McNemars test

To turn this into a *p*-value, we first need an expression for the probability of our data given our parameter of interest r . Letting $s = n_{12} + n_{21}$ this can be shown to be [Altham, 1971]:

$$\begin{aligned} p(n_{12}, n_{21} | s) &= \binom{s}{n_{12}} r^{n_{12}} (1 - r)^{n_{21}} \\ &= p_{\text{Binom}}(n_{12} | \theta = r, N = s) \end{aligned} \quad (11.29)$$

where we have used the binomial distribution from eq. (5.35).

We can use this result to test whether model \mathcal{M}_A is better than model \mathcal{M}_B by defining the null hypothesis H_0 and alternative hypothesis H_1 as follows:

$$H_0 : \text{Model } \mathcal{M}_A \text{ has the same performance as model } \mathcal{M}_B \quad (11.30a)$$

$$H_1 : \text{Model } \mathcal{M}_A \text{ and } \mathcal{M}_B \text{ have different performance.} \quad (11.30b)$$

Recall the definition of p -value from eq. (11.9) is defined as the probability of observing a value of n_{12} and n_{21} more extreme, i.e. unlikely, than the one actually observed assuming H_0 is true, that is, $r = \frac{1}{2}$. In this case, since $p_{\text{Binom}}(n_{12} | \theta = \frac{1}{2}, N = n_{12} + n_{21}) = p_{\text{Binom}}(n_{21} | \theta = \frac{1}{2}, N = n_{12} + n_{21})$, we can consider $m = \min\{n_{12}, n_{21}\}$ as the more extreme value and find the p -value to be:

$$p = P(N_{12} \leq m | H_0) + P(N_{21} \leq m | H_0) = 2\text{cdf}_{\text{binom}}(m | \theta = \frac{1}{2}, N = n_{12} + n_{21}) \quad (11.31)$$

Where we have used the binomial cumulative distribution function:

$$\text{cdf}_{\text{binom}}(m | \theta, N) = \sum_{k=0}^m p_{\text{binom}}(k | \theta, N). \quad (11.32)$$

A confidence interval for the McNemar test

While p -value are useful to get an indication if one classifier is better than another they are less useful for determining a plausible interval of their performance difference. In other words, let θ_A denote the (true) chance classifier \mathcal{M}_A is correct and θ_B the (true) chance classifier \mathcal{M}_B is correct. We hope to derive a confidence interval for the difference in performance:

$$\theta = \theta_A - \theta_B$$

with the interpretation that if $\theta > 0$ model A is preferable over B . One idea for creating a confidence interval is to derive a (reasonable) expression for the posterior similar to the Jeffreys interval. An exact expression is given by Bloch et al. [1967], however this expression is rather complicated and a reasonably accurate approximation is

$$p(\theta | \mathbf{n}) = \frac{1}{2} \text{Beta} \left(\frac{\theta + 1}{2} \mid \alpha = p, \beta = q \right) \quad (11.33a)$$

$$p = \frac{E_\theta + 1}{2} (Q - 1) \quad (11.33b)$$

$$q = \frac{1 - E_\theta}{2} (Q - 1) \quad (11.33c)$$

$$E_\theta = \frac{n_{12} - n_{21}}{n}, \quad Q = \frac{n^2(n+1)(E_\theta + 1)(1 - E_\theta)}{n(n_{12} + n_{21}) - (n_{12} - n_{21})^2}. \quad (11.33d)$$

For this approximation to be sensible we recommend that $n_{12} + n_{21} \geq 5$. From this, we can obtain a $1 - \alpha$ confidence interval for $\theta' = \frac{\theta+1}{2}$ using $\text{cdf}_B^{-1}(\frac{\alpha}{2} | p, q)$, and transform this into a $1 - \alpha$ confidence interval for θ by observing $\theta = 2\theta' - 1$. The test is summarized in Box 11.3.2.

Method 11.3.2: The McNemar test for comparing classifiers

Our goal is to compare two classifiers A and B . Specifically, we want to know if A is better than B

- Select a form of cross-validation
- Obtain n predictions $\hat{y}_1^A, \dots, \hat{y}_n^A$ and $\hat{y}_1^B, \dots, \hat{y}_n^B$ from the classifiers by evaluating them on the *same* cross-validation splits.
- Compute the matrix \mathbf{n} using eq. (11.27)

To estimate the performance difference:

- The estimated difference in accuracy of A and B , $\theta = \theta_A - \theta_B$, is given as

$$\hat{\theta} = \frac{n_{12} - n_{21}}{n} \quad (11.34)$$

- An approximate $1 - \alpha$ confidence interval $[\theta_L, \theta_U]$ for θ can be obtained as

$$\theta_L = 2\text{cdf}_B^{-1}\left(\frac{\alpha}{2} \mid \alpha = p, \beta = q\right) - 1 \quad (11.35a)$$

$$\theta_U = 2\text{cdf}_B^{-1}\left(1 - \frac{\alpha}{2} \mid \alpha = p, \beta = q\right) - 1 \quad (11.35b)$$

where p, q are computed using eq. (11.33). For this interval to be useful we require $n_{12} + n_{21} \geq 5$.

To test if they have different performance:

- Let H_0 be the null hypothesis they have the same performance.
- Compute the p -value using the cumulative distribution function of the binomial distribution as in eq. (11.31):

$$p = 2\text{cdf}_{\text{binom}}\left(m = \min\{n_{12}, n_{21}\} \mid \theta = \frac{1}{2}, N = n_{12} + n_{21}\right) \quad (11.36)$$

- The interpretation is that the lower p is, the more evidence there is A is better than B , but only interpret the p -value together with the estimate $\hat{\theta}$ and ideally the confidence interval computed above.

11.3.4 Estimating the generalization error of a regression model

For a regression model, we once more use cross-validation to compute n predictions $\hat{\mathbf{y}}$ as in eq. (11.16), and then use either the L_1 or squared loss to compute z_1, \dots, z_n as in eq. (11.18):

$$z_i = |\hat{y}_i - y_i| \quad \text{or} \quad z_i = (\hat{y}_i - y_i)^2 \quad (11.37)$$

As usual, we are interested in a confidence interval of the estimated generalization error

$$\hat{z} = \frac{1}{n} \sum_{i=1}^n z_i. \quad (11.38)$$

To accomplish this, we treat the true value of the mean of the observations $Z = \frac{1}{n} \sum_{i=1}^n Z_i$ as a random quantity. To proceed, we assume that given $Z = u$, each observation is identically and independently normally distributed with variance σ^2 . Hence:

$$p(z_i|Z = u, \sigma^2) = \mathcal{N}(z_i|\mu = u, \sigma^2) \quad (11.39)$$

It follows the distribution of the entire dataset is:

$$p(D|u, \sigma^2) = \prod_{i=1}^n \mathcal{N}(z_i|u, \sigma^2)$$

We will at this point omit some steps, but it is possible to show under natural assumptions that u is distributed as a student- t distribution:

$$p(u|D) = p_T(u|\nu = n - 1, \mu = \hat{z}, \sigma = \tilde{\sigma}) \quad (11.40)$$

with parameters $\hat{z} = \frac{1}{n} \sum_{i=1}^n z_i$ and $\tilde{\sigma} = \sqrt{\sum_{i=1}^n \frac{(z_i - \hat{z})^2}{n(n-1)}}$. The Student's t -distribution has density

$$\text{Student } t\text{-distribution} \quad p_T(x|\nu, \mu, \sigma) = \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2}) \sqrt{\pi\nu\sigma^2}} \left(1 + \frac{1}{\nu} \left[\frac{x - \mu}{\sigma}\right]^2\right)^{-\frac{\nu+1}{2}}. \quad (11.41)$$

Since we have found an expression for $p(u|D)$, we can use the cumulative distribution function of the student's t -distribution to obtain a confidence interval. If we denote this by $\text{cdf}_{st}^{-1}(A|\nu, \hat{z}, \tilde{\sigma})$ ³ we can write up the confidence interval corresponding to eq. (6.26) as $[z_L, z_U]$ where

$$z_L = \text{cdf}_T^{-1}\left(\frac{\alpha}{2} \mid \nu, \hat{z}, \tilde{\sigma}\right), \quad z_U = \text{cdf}_T^{-1}\left(1 - \frac{\alpha}{2} \mid \nu, \hat{z}, \tilde{\sigma}\right). \quad (11.42)$$

Obviously, we should at this point be quite worried about the normality assumption in eq. (11.39). What happens if this assumption is violated? In this case, the test is inaccurate for small values of n , but will in general hold when n is appreciably large, see comments in Technical Note 11.3.2. The full procedure is described in Box 11.3.3.

³ Special functions to evaluate $\text{cdf}_{st}^{-1}(A|\nu, \hat{z}, \tilde{\sigma})$ are build into all popular mathematical environments. In matlab: $\text{cdf}_T^{-1}(A|\nu, \hat{z}, \tilde{\sigma}) = \hat{z} + \tilde{\sigma} \text{tinv}(A, \nu)$, in R: $\text{cdf}_T^{-1}(A|\nu, \hat{z}, \tilde{\sigma}) = \hat{z} + \tilde{\sigma} \text{qt}(A, \nu)$ and in Python: $\text{cdf}_T^{-1}(A|\nu, \hat{z}, \tilde{\sigma}) = \hat{z} + \tilde{\sigma} \text{t.ppf}(A, \nu)$

Method 11.3.3: A central limit theorem-based interval for a regression model

- Select a form of cross-validation
- Compute z_1, \dots, z_n as in eq. (11.37)
- The $1 - \alpha$ confidence interval $[z_L, z_U]$ for E^{gen} is now obtained as

$$z_L = \text{cdf}_{\mathcal{T}}^{-1} \left(\frac{\alpha}{2} \mid \nu = n - 1, \mu = \hat{z}, \sigma = \tilde{\sigma} \right) \quad (11.43a)$$

$$z_U = \text{cdf}_{\mathcal{T}}^{-1} \left(1 - \frac{\alpha}{2} \mid \nu = n - 1, \mu = \hat{z}, \sigma = \tilde{\sigma} \right) \quad (11.43b)$$

$$\hat{z} = \frac{1}{n} \sum_{i=1}^n z_i, \quad \tilde{\sigma}^2 = \frac{1}{n(n-1)} \sum_{i=1}^n (z_i - \hat{z})^2. \quad (11.43c)$$

- Unless it is known the errors z_i are normally distributed (and this cannot be assumed), this approach depends on the central limit theorem. It is therefore only be applied when n is appreciably large, but will in this case be fairly accurate. We recommend $n \geq 30$.

If these conditions are satisfied report the results as follows: The estimated generalization error, computed using the L_1 (or L_2) loss, is given as \hat{z} and since n is large we computed a confidence interval based on the student- t distribution with parameters $\mu = \hat{z}$ and variance $\sigma^2 = \tilde{\sigma}^2$. Based on this approximation, a $1 - \alpha$ confidence interval is $[z_L, z_U]$.

Technical note 11.3.2: Comment on normality assumption

If the values z_i in eq. (11.38) are not normally distributed as assumed in eq. (11.39), we can still use that

$$Z = \frac{1}{n} \sum_{i=1}^n Z_i. \quad (11.44)$$

is an average of n independent random variables. We therefore know according to the central limit theorem (see eq. (6.30)) that for high n , the mean $Z = u$ is approximately normally distributed with parameters defined from the mean, variance of any one of the Z_i 's :

$$u \sim \mathcal{N}\left(\mu = \mathbb{E}[Z_i], \sigma^2 = \frac{1}{n} \text{Var}[Z_i]\right). \quad (11.45)$$

While we do not know the true value of the mean/variance of Z_i , a reasonable guess is they are equal to the empirical mean/variances if n is large. Therefore

$$u \sim \mathcal{N}\left(\mu = \hat{z}, \sigma^2 = \frac{\hat{s}_0^2}{n}\right), \quad \hat{s}_0^2 = \frac{1}{n-1} \sum_{i=1}^n (z_i - \hat{z})^2. \quad (11.46)$$

This result might appear different from the student- t distribution we obtained in eq. (11.40), but note that when ν is large, the student- t distribution converge to the normal distribution:

$$\lim_{\nu \rightarrow \infty} p_T(x|\nu, \mu, \sigma) = \mathcal{N}(x|\mu, \sigma^2).$$

Hence, if n is appreciably large (and usually $n \geq 30$ is sufficient) the student- t result will be accurate even when the observations are not normally distributed.

11.3.5 Comparing two regression models

To compare two regression models \mathcal{M}_A and \mathcal{M}_B , we once more assume a form of cross-validation has been applied to obtain K train/test splits, and the two regression models are trained on the *same* splits to produce n paired predictions as in eq. (11.16)

$$\hat{y}_1^A, \dots, \hat{y}_n^A, \quad \text{and} \quad \hat{y}_1^B, \dots, \hat{y}_n^B. \quad (11.47)$$

Given these, and the true values y_1, \dots, y_n , we once more select a loss-function to compute the per-observation losses as in eq. (11.18)

$$z_1^A, \dots, z_n^A, \quad \text{and} \quad z_1^B, \dots, z_n^B.$$

Note the estimated difference in generalization error can be written as

$$\begin{aligned}
z &= E_A^{\text{gen}} - E_B^{\text{gen}} \approx \\
\hat{z} &= \left(\frac{1}{n} \sum_{i=1}^n z_i^A \right) - \left(\frac{1}{n} \sum_{i=1}^n z_i^B \right) \\
&= \frac{1}{n} \sum_{i=1}^n z_i, \quad \text{where } z_i = z_i^A - z_i^B
\end{aligned} \tag{11.48}$$

Therefore, with z_1, \dots, z_n defined as above, we can re-use the derivations from eq. (11.38) to eq. (11.40) to find that the distribution of the mean $Z = \frac{1}{n} \sum_{i=1}^n Z_i$ has density given by a student- t distribution

$$p(Z = u | D) = p_{\mathcal{T}}(u | \nu = n - 1, \mu = \hat{z}, \sigma = \tilde{\sigma}) \tag{11.49a}$$

$$\hat{z} = \frac{1}{n} \sum_{i=1}^n z_i, \quad \tilde{\sigma}^2 = \sum_{i=1}^n \frac{(z_i - \hat{z})^2}{n(n-1)}. \tag{11.49b}$$

The credibility interval of the performance difference is therefore simply eq. (11.42), and a p -value of the null-hypothesis versus the alternative hypothesis

$$H_0 : \text{Model } \mathcal{M}_A \text{ and } \mathcal{M}_B \text{ have the same performance, } Z = 0 \tag{11.50a}$$

$$H_1 : \text{Model } \mathcal{M}_A \text{ and } \mathcal{M}_B \text{ have different performance, } Z \neq 0. \tag{11.50b}$$

can be computed as an integral of eq. (11.49a)

$$\begin{aligned}
p &= P(Z \geq |\hat{z}| \mid H_0) = 2 \int_{-\infty}^{-|\hat{z}|} p_{\mathcal{T}}(z \mid \nu = n - 1, \mu = 0, \sigma = \tilde{\sigma}) dz \\
&= 2 \text{cdf}_{\mathcal{T}}(-|\hat{z}| \mid \nu = n - 1, \mu = 0, \sigma = \tilde{\sigma}).
\end{aligned} \tag{11.51}$$

The procedure is summarized in Box 11.3.4.

Method 11.3.4: Comparing two regression models

- Select a form of cross-validation
- Let z_i^A and z_i^B be the loss for observation i for the two models for $i = 1, \dots, n$ as computed using either absolute or squared loss.
- Define $z_i = z_i^A - z_i^B$
- The $1 - \alpha$ confidence interval $[z_L, z_U]$ for E^{gen} is now obtained as

$$z_L = \text{cdf}_{\mathcal{T}}^{-1} \left(\frac{\alpha}{2} \mid \nu = n - 1, \mu = \hat{z}, \sigma = \tilde{\sigma} \right) \quad (11.52a)$$

$$z_U = \text{cdf}_{\mathcal{T}}^{-1} \left(1 - \frac{\alpha}{2} \mid \nu = n - 1, \mu = \hat{z}, \sigma = \tilde{\sigma} \right) \quad (11.52b)$$

$$\hat{z} = \frac{1}{n} \sum_{i=1}^n z_i, \quad \tilde{\sigma}^2 = \frac{1}{n(n-1)} \sum_{i=1}^n (z_i - \hat{z})^2. \quad (11.52c)$$

- A p -value for the null hypothesis the two models have the same performance, $Z = 0$, is obtained as

$$p = 2\text{cdf}_{\mathcal{T}}(-|\hat{z}| \mid \nu = n - 1, \mu = 0, \sigma = \tilde{\sigma}). \quad (11.53)$$

- Unless the z_i 's are known to be normally distributed, and they usually are not, this approach depends on the central limit theorem and should only be applied when n is appreciably large. We recommend $n \geq 30$.

If these conditions are satisfied report the results as follows: The estimated difference in generalization error, computed using the L_1 (or L_2) loss, is given as \hat{z} and since n is large we computed a confidence interval based on the student- t distribution with parameters $\mu = \hat{z}$ and variance $\sigma^2 = \tilde{\sigma}^2$. Based on this approximation, a $1 - \alpha$ confidence interval is $[z_L, z_U]$.

11.4 Setup II: The training set is random★

The previous methods have all been concerned with estimating the generalization error of a model when trained on a specific training set. This is most notably true with the hold-out method, in which we split our data into $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{test}}$ and then evaluate the model trained on $\mathcal{D}^{\text{train}}$ on the $n = N^{\text{test}}$ observations in $\mathcal{D}^{\text{test}}$. Note however that even if we could evaluate the models performance exactly, for instance if we had an unlimited amount of test-data, we would still fail to capture a significant source of variability, namely what happens when we train a model on a *different* training set. In other words, we now consider the generalization error

$$E^{\text{gen}} = \int \left[\int L(f_{\mathcal{D}^{\text{train}}}(\mathbf{x}), y) d\mathbf{x} dy \right] d\mathcal{D}^{\text{train}} \quad (11.54)$$

Where the outer integral should be understood as integrating over training sets of size N and $f_{\mathcal{D}^{\text{train}}}$ is the prediction rule obtained when training \mathcal{M} on $\mathcal{D}^{\text{train}}$.

In some circumstances this distinction is less important, for instance, a company which really have a single training set and are not interested in sharing their methods. However, in many applications we want to know how well our method can be re-produced by another group who have their

own training set. In other words, we are in **setup II**, where we also wish to include the variability in model performance from different training sets in our statistical estimates.

11.4.1 A problem

This is in a sense quite simple: Suppose we have a large number J of independent training/test splits

$$(\mathcal{D}_1^{\text{train}}, \mathcal{D}_1^{\text{test}}), \dots, (\mathcal{D}_J^{\text{train}}, \mathcal{D}_J^{\text{test}}) \quad (11.55)$$

If we train J models on these training sets, we can estimate each of the models generalization error as usual

$$E_j^{\text{gen}} = \int L(f_{\mathcal{D}_j^{\text{train}}}(\mathbf{x}), y) d\mathbf{x} dy \approx \frac{1}{n_j} \sum_{i=1}^{n_j} L(f_{\mathcal{D}_j^{\text{train}}}(\mathbf{x}_i^j), y_i^j), \quad n_j = |\mathcal{D}_j^{\text{train}}| \quad (11.56)$$

and finally obtain an estimate of eq. (11.54) by averaging these over training sets:

$$E^{\text{gen}} \approx \frac{1}{J} \sum_{j=1}^J E_j^{\text{gen}}. \quad (11.57)$$

This procedure works and produces an unbiased estimate of the generalization error, and we could easily use it for statistical analysis since (i) each of the generalization errors E_j^{gen} can be assumed to be normally distributed by virtue of the central limit theorem (ii) the estimate eq. (11.57) of the generalization error eq. (11.54) is therefore a simple average of normally distributed variables.

There is, however, an important problem as pointed out by Bengio and Grandvalet [2004]. Normally, we would obtain the training/test datasets eq. (11.55) using a cross-validation procedure, however if we do so the training set $\mathcal{D}_j^{\text{train}}$ will in general overlap, and by a sizable fraction $\frac{K-2}{K}$, with any other training set $\mathcal{D}_i^{\text{train}}$, $i \neq j$. Therefore, the trained models will not be independent, and we do not have J independent estimates of eq. (11.57).

To re-iterate what the issue is: There are three sources of variance/covariance in the problem:

1. The variance on a prediction on a single observation when we randomize over the observation and the training set.
2. The covariance between losses of two observations due to being trained on the same set, and the final is the covariance induced by
3. The above-mentioned covariance due to the overlap of training sets

Bengio and Grandvalet [2004] showed it is not possible to estimate these at the same time from a single training set. While this result might preclude us from choosing an *optimal* solution, it does not prevent us from choosing a solution better than simply ignoring the problem. One such approach was given by Nadeau and Bengio [2000] but our presentation will follow Benavoli et al. [2017].

11.4.2 The correlation heuristic

We are still going to estimate the generalization error using an average of cross-validation predictions as in eq. (11.57), and obviously this estimate will be better the more estimates we average. Therefore,

while we will still obtain the train/test splits using cross-validation, but we will assume the K -fold cross-validation procedure is repeated one or more times (randomizing the assignment to folds each time) to obtain $J = K, J = 2K, \dots$ splits into training/test sets as shown in eq. (11.55). For each of these J pairs of training/test data, train the model on the training set and test on the test set. Denoting:

$$\mathcal{D}_j^{test} = ((\mathbf{x}_1^j, y_1^j), \dots, (\mathbf{x}_{n_j}^j, y_{n_j}^j))$$

This result in an estimated generalization error:

$$r_j = \frac{1}{n_j} \sum_{i=1}^{n_j} L(f_{\mathcal{D}_j^{train}}(\mathbf{x}_i^j), y_i^j). \quad (11.58)$$

We then assume these error estimates have a mean value \bar{z} , which is what we hope to estimate, and errors v_j which are correlated. Specifically we assume:

$$r_j = \bar{z} + v_j \quad (11.59a)$$

$$\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}) \quad (11.59b)$$

here \mathbf{v} is the $J \times 1$ vector of noise terms and, crucially, we assume they have a covariance

$$\Sigma_{ii} = \sigma^2, \quad \Sigma_{ij} = \rho\sigma^2. \quad (11.60)$$

It is this covariance matrix which allows us to account for the (unknown) correlation which arises due to overlap between training sets: The scale term σ^2 is the true variance due to natural variation that arises from the use of finite training and test sets, and ρ is the (unknown) correlation which arises from overlapping training sets.

With these assumptions, and assuming $p(\sigma) \propto \frac{1}{\sigma}$, we can derive the posterior distribution of \bar{z} to follow a (non-standardized) student- t distribution [Benavoli et al., 2017]:

$$p(\bar{z} | \mathbf{r}) = p_{\mathcal{T}}(\bar{z} | \nu = J - 1, \mu = \hat{r}, \sigma = \tilde{\sigma}) \quad (11.61a)$$

$$\hat{r} = \frac{1}{J} \sum_{j=1}^J r_j, \quad \tilde{\sigma}^2 = \left(\frac{1}{J} + \frac{\rho}{1-\rho} \right) \hat{s}^2, \quad \text{and} \quad \hat{s}^2 = \frac{1}{J-1} \sum_{j=1}^J (r_j - \bar{r})^2 \quad (11.61b)$$

which we previously encountered in eq. (11.41). This is in some sense quite natural, since if $\rho = 0$, implying no correlation, the assumed model eq. (11.59) corresponds to the normal model that lead to the student t distribution in eq. (11.40).

Inspecting the above, it should be clear that all terms are at this point computable except for ρ which accounts for the correlation induced due to overlapping training sets and, as previously discussed, cannot be estimated from data. The suggestion by Nadeau and Bengio [2000] is to select:

$$\rho = \frac{|\mathcal{D}_j^{test}|}{|\mathcal{D}_j^{train}| + |\mathcal{D}_j^{test}|} = \frac{1}{K} \quad (11.62)$$

which is known as the *correlation heuristic*. To understand this choice, note the variance term in the test eq. (11.61) becomes:

$$\tilde{\sigma}^2 = \left(\frac{1}{J} + \frac{1}{K-1} \right) \hat{\sigma}^2.$$

So if K becomes large, the term $\frac{1}{J} + \frac{1}{K-1}$ will generally tend towards zero, however, in this case the variance \hat{s}^2 on the test data should be expected to be larger as we are estimating the r_j 's with fewer observations. On the other hand this also shows that just repeating cross validation (increasing J) many times in eq. (11.61) will have diminishing returns. All in all, it suggest we should select a moderate value of K , for instance $K = 5$ or $K = 10$, and repeat the procedure one or more times to increase J , as our computation budget allows.

Given this, we can compute an approximate confidence interval using the inverse of the cumulative density function of the student- t distribution and similarly obtain a p -value exactly as in Box 11.3.3. The method is summarized in Box 11.4.1.

Method 11.4.1: Correlated t -test for cross-validation

Our goal is to compare two models \mathcal{M}_A and \mathcal{M}_B by estimating the difference in generalization error, where we are interested in the more general form of the generalization error described in **setup II** where the randomness of the training sets are also taken into account.

- Select as many cross-validation splits as your computational budget allows. We recommend as minimum $K = 5$ and $J = K$, and better $K = 10$ repeated a few times to get $J = 20$ or 30. Remembering to randomize your cross-validation splits between different runs, obtain J splits as in eq. (11.55)
- For each of the J splits $(\mathcal{D}_j^{\text{train}}, \mathcal{D}_j^{\text{test}})$, train both models on $\mathcal{D}_j^{\text{train}}$ and obtain two sets of n_j predictions on $\mathcal{D}_j^{\text{test}}$, $\hat{\mathbf{y}}^{A,j}$ and $\hat{\mathbf{y}}^{B,j}$
- Estimate the difference in generalization error for split j

$$r_j = \frac{1}{n_j} \sum_{i=1}^{n_j} \left[L(\hat{y}_i^{A,j}, y_i^j) - L(\hat{y}_i^{B,j}, y_i^j) \right] \quad (11.63)$$

where y_i^j are the true y -values and the loss can either be L_1 or squared loss (in case of regression) or the error rate loss in case of classification. Do this for each $j = 1, \dots, J$

- A $1 - \alpha$ confidence interval for the difference in generalization error is

$$z_L = \text{cdf}_{\mathcal{T}}^{-1} \left(\frac{\alpha}{2} \mid \nu, \hat{r}, \tilde{\sigma} \right), \quad z_U = \text{cdf}_{\mathcal{T}}^{-1} \left(1 - \frac{\alpha}{2} \mid \nu, \hat{r}, \tilde{\sigma} \right). \quad (11.64)$$

- A p -value for the null hypothesis the two models have the same performance can be computed as

$$\begin{aligned} p &= 2 \text{cdf}_{\mathcal{T}} (-|\hat{t}| \mid \nu = J - 1, \mu = 0, \sigma = 1) \\ \hat{t} &= \frac{\hat{r}}{\hat{\sigma} \sqrt{\frac{1}{J} + \frac{\rho}{1-\rho}}}. \end{aligned} \quad (11.65)$$

Note that since we have to assume each r_j are normally distributed, the CLM has to apply. In other words, select K so low the test sets contain at least 30 observations.

12

Nearest neighbor methods

Classifying observations based on the labels of their nearest neighbours is a simple idea and has been re-invented several times, however the first discussion of a nearest-neighbour classification rule was by statisticians Evelyn Fix and Joseph Hodges in an (unpublished!) technical report in 1951 [Fix and Hodges, 1951].

12.1 K-nearest neighbour classification

We will introduce the K -nearest neighbour classifier with an example. In fig. 12.1 is shown a subset of the Fisher Iris data where only the two first attributes are plotted. Suppose we ask a human to guess the name of a flower at the black square. Most humans would properly say *Setosa* (the blue class) because *the nearby flowers are predominantly blue*. For similar reasons, labelling an observation at the black cross would be more difficult. Two things seem to play a role

- The closeness of the nearby points.
- How many there are of a particular color.

This intuition is the idea behind the K -nearest neighbour method. Consider again the data in fig. 12.1 but focus on a test point at the black cross. Imagine we draw a circle around the black cross and slowly increases its radius. At some point the circle will contain one point which (as it happens) is yellow in our case, see the upper-left pane of fig. 12.2 where the selected point is highlighted with red. If we increases the radius of the circle it will at some point contain two and then three points, see the upper-right pane of fig. 12.2 where these correspond to a yellow and two green points. In general, we can define the K -neighbourhood of a point \mathbf{x} as:¹

$$N_{\mathbf{X}}(\mathbf{x}, K) = \{\text{The } K \text{ observations in } \mathbf{X} \text{ which are nearest to } \mathbf{x}\}. \quad (12.1)$$

Notice we have included the dataset \mathbf{X} in the above definition, however, sometimes the K -neighbourhood is simply written as $N(\mathbf{x}, K)$ if it is clear from the context what \mathbf{X} is. The lower-row of fig. 12.2 shows $N_{\mathbf{X}}(\mathbf{x}, K = 5)$ and $N_{\mathbf{X}}(\mathbf{x}, K = 7)$ respectively. A simple classification method

¹ What if several points have the same distance to \mathbf{x} ? This borderline case might appear in the case where there are duplicate observations in the dataset and can be handled in several ways, for instance by (randomly) selecting between the tied points. We will assume such a scheme exist and therefore the K -neighbourhood always consist of K observations.

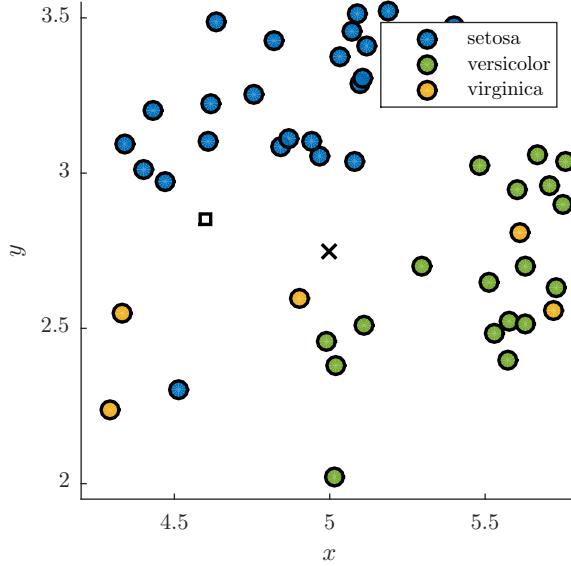


Fig. 12.1. A subset of the Fisher Iris dataset where we only consider two features. Which class and why would you assign the test points at the black square and cross if you were to just make a qualified guess?

is then to fix K (for instance $K = 5$) and just assign \mathbf{x} to the class which has the most elements in $N_{\mathbf{X}}(\mathbf{x}, K)$. In the case where two classes has equally many members (we say they are tied), for instance the lower-right pane of fig. 12.2, \mathbf{x} is assigned to the class which has a *closest* member to \mathbf{x} in $N_{\mathbf{X}}(\mathbf{x}, K)$, in this case the green class. This is simply the KNN classification rule for an observation \mathbf{x} :

- Compute $N_{\mathbf{X}}(\mathbf{x}, K)$.
- Classify \mathbf{x} to the class k which has the most members in $N_{\mathbf{X}}(\mathbf{x}, K)$.
- In the case of ties, simply classify \mathbf{x} to the class which has a member nearest to \mathbf{x} .

An alternative tie-breaking rule is simply to select a random of the tied classes. Notice in particular the case where $K = 1$, here we simply assign \mathbf{x} to the class of the *nearest* observation in the training set. This is known as the *nearest neighbour* classification rule. In fig. 12.3 is shown the classification boundary for the full problem, i.e. the colors indicate what class a point at that given location will be classified to for $K = 1, 3, 5, 7$.

12.1.1 A Bayesian view of the KNN classifier★

As presented, the KNN classifier is simply a heuristic. However, it is possible to give the KNN classifier a Bayesian interpretation [Bishop, 2013]. Suppose we denote by K_c the number of elements in $N_{\mathbf{X}}(\mathbf{x}, K)$ (we will suppress \mathbf{X} in this section) which belong to class c and N_c the number of observations in the *entire* dataset which belongs to class c :

$$K_c = \text{Number of observations } \mathbf{x}_i \in N_{\mathbf{X}}(\mathbf{x}, K) \text{ where } y_i = c. \quad (12.2)$$

$$N_c = \text{Number of observations } \mathbf{x}_i \text{ where } y_i = c. \quad (12.3)$$

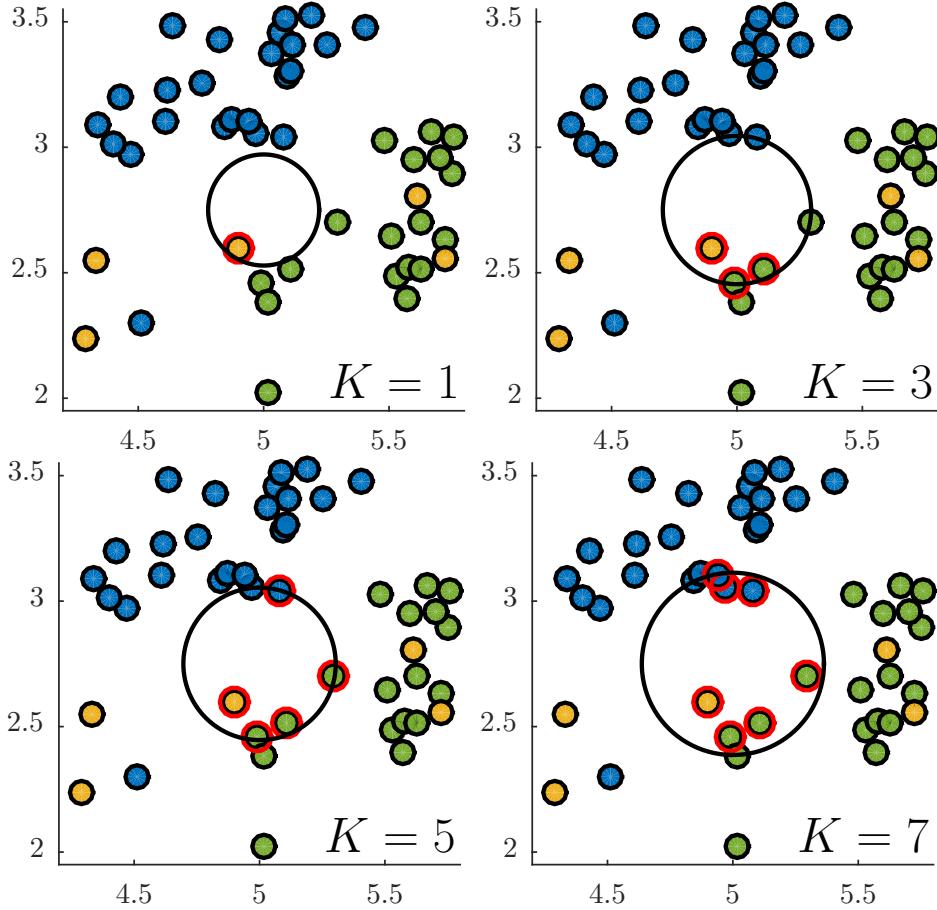


Fig. 12.2. Illustration of the K -nearest neighbourhood $N_{\mathbf{X}}(\mathbf{x}, K)$ of the black cross \mathbf{x} for $K = 1, 3, 5, 7$, observations within the neighbourhood is highlighted with the red circles. The KNN classifier simply assigns the observation \mathbf{x} to the class with the most observations within the circle.

Then clearly $K = \sum_{c=1}^C K_c$ and $N = \sum_{c=1}^C N_c$. If we select a random observation, the probability it belongs to class c is:

$$p(y = c) = \frac{N_c}{N}$$

Then, notice for any volume V by the definition of probability:

$$\int_V p(\mathbf{x}|y = c)d\mathbf{x} = \{\text{Probability an observation of class } c \text{ is in } V\}$$

If we now consider the volume V to be the size of the K -nearest neighbourhood of \mathbf{x} , i.e. the area of the discs in fig. 12.2) around \mathbf{x} , the left-hand side and right-hand side of the above becomes:

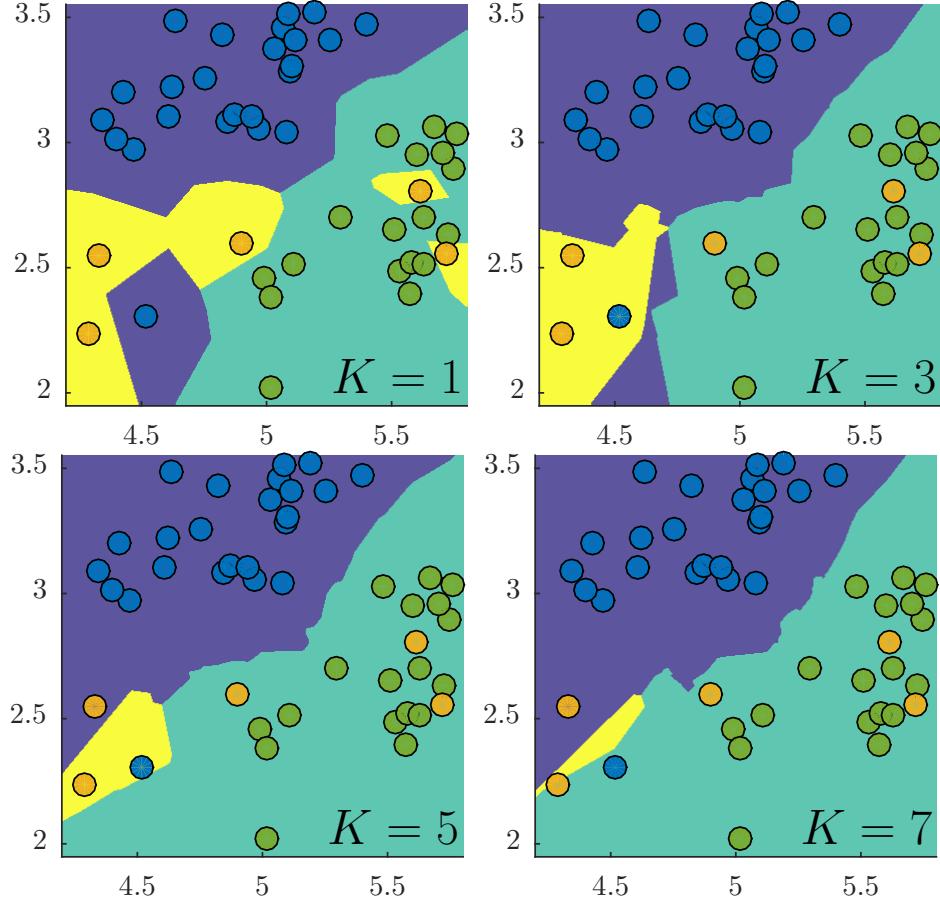


Fig. 12.3. KNN classification boundary for the problem in fig. 12.1 for $K = 1, 3, 5, 7$. Notice as K increases, the boundary becomes more smooth. For $K = 3$ (upper-right corner), the blue point in the lower left corner is able to induce a small blue area due to the tie-breaking rule.

$$\begin{aligned} \{\text{lhs.}\} &= \int_V p(\mathbf{x}|y=c)d\mathbf{x} && \approx V p(\mathbf{x}|y=c) \\ \{\text{rhs.}\} &= \frac{\text{Number of observations of class } y=c \text{ in } V_{\mathbf{x}}}{\text{Total number of observations of class } c} \approx \frac{K_c}{N_c} \end{aligned}$$

If we put this together we obtain $p(\mathbf{x}|y=c) = \frac{K_c}{N_c V}$. Then simply applying Bayes theorem we obtain:

$$p(y=c|\mathbf{x}) = \frac{p(\mathbf{x}|y=c)p(y=c)}{\sum_{c'=1}^C p(\mathbf{x}|y=c')p(y=c')} = \frac{K_c}{K} \quad (12.4)$$

So when the KNN classification method selects the class c where K_c is the highest it corresponds to selecting the most *probable* class according to Bayes theorem and the above approximations.

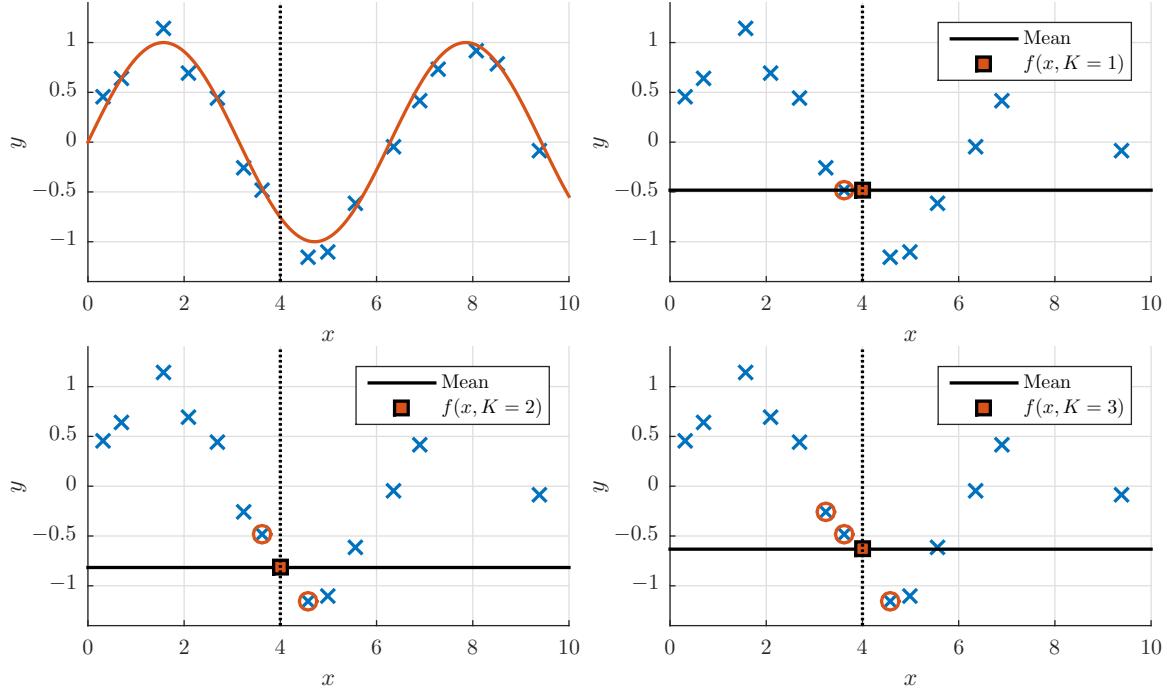


Fig. 12.4. 1D example dataset (upper-right pane) generated as noisy observations of the sinusoidal signal. KNN regression for an observation x indicated by the vertical dotted line first finds the K -nearest neighbourhood of x , $N_{\mathbf{X}}(x, K)$, and then simply outputs the mean of the observations y_i in $N_{\mathbf{X}}(x, K)$. The three panes illustrates $K = 1, 2, 3$.

12.2 K-nearest neighbour regression

The K -nearest neighbour classification rule is easily modified for regression. Suppose we have the dataset shown in fig. 12.4 (top left pane) which consists of $N = 16$ noisy observations of a sinusoidal signal. If we wish to make predictions around $x = 4$ (the vertical bar), this can be accomplished finding the K closest elements to x in the dataset, $N_{\mathbf{X}}(x, K)$, (shown as the circles) and simply predicting the mean value of the elements in $N_{\mathbf{X}}(x, K)$ (shown as the horizontal bar). The prediction at $x = 4$ is then just the red square. In general, the prediction rule is:

$$f(\mathbf{x}, K) = \frac{1}{K} \sum_{i \in N_{\mathbf{X}}(\mathbf{x}, K)} y_i. \quad (12.5)$$

which of course works for arbitrary dimensions. Notice in particular the $K = 1$ prediction rule simply corresponds to finding the observation \mathbf{x}_i closest to a test point \mathbf{x} and predicting $f(\mathbf{x}, K = 1) = y_i$. In fig. 12.5 the prediction rule is visualized for $K = 1, 2, 3, 4$. Notice, as K increases the rule becomes less driven by an error in any single value (less variation), however, it also becomes more biased towards predictions near the mean.

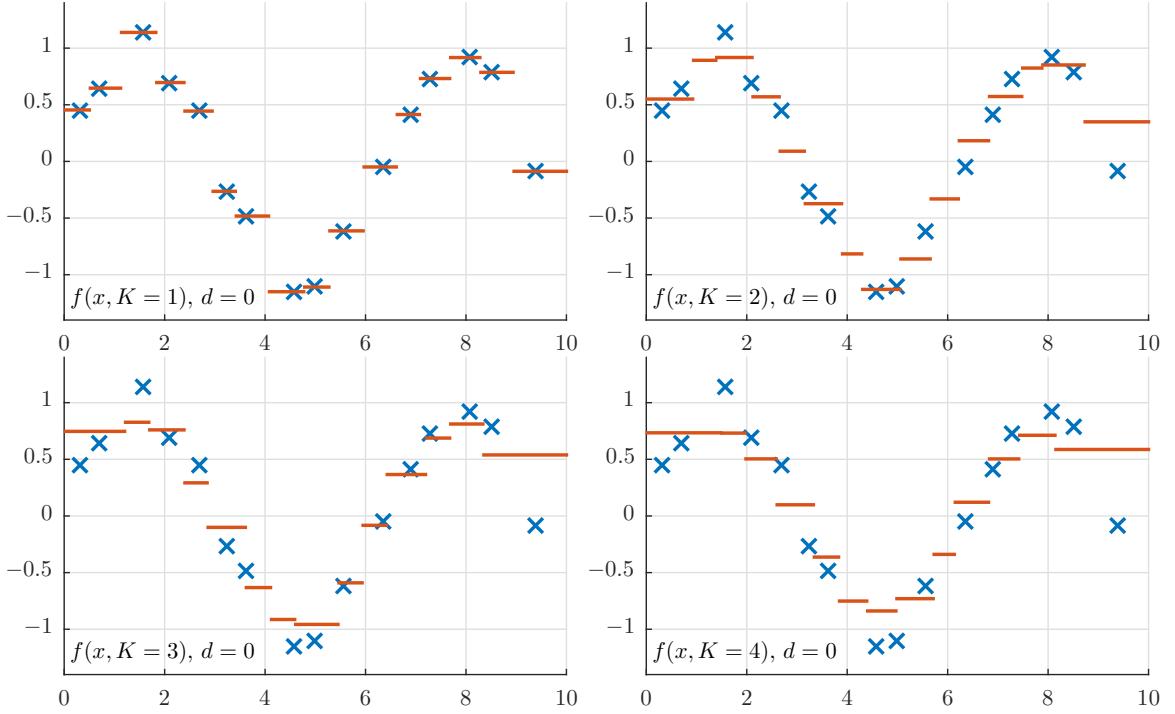


Fig. 12.5. Illustration of the prediction rule for KNN regression from fig. 12.4 (red line) when computed over the entire dataset for $K = 1, 2, 3, 4$. Notice the prediction rule is piece-wise linear corresponding to different neighbourhoods.

12.2.1 Higher-order KNN regression★

If we return to the KNN regression dataset in fig. 12.4 and the prediction curves in fig. 12.5 notice that the curve is piece-wise constant. It might be better if the curve within each neighbourhood is fitted to the dataset with a more powerful model. A simple way to obtain this is to rather than predicting the mean within each local neighbourhood, fitting a polynomial of degree d . The piece-wise linear model then corresponds to $d = 0$ (the constant polynomial). This is illustrated in fig. 12.6 for $K = 3, 5$ and $d = 1, 2$.

The corresponding prediction curve is shown in fig. 12.7. Compared to the piece-wise linear case in fig. 12.5, the high-order polynomials allow much smoother interpolation of the underlying curve, however, in general they also require higher values of K in order not to overfit locally.

12.3 Cross-validation and nearest-neighbour methods

Selecting K in nearest neighbour methods can easily be accomplished using cross-validation. Simply let $\mathcal{M}_1, \dots, \mathcal{M}_S$ in algorithm 5 correspond to different values of K (for instance $K = 1, \dots, S$) and let the error measure be (in case of classification) the classification error, or for regression the

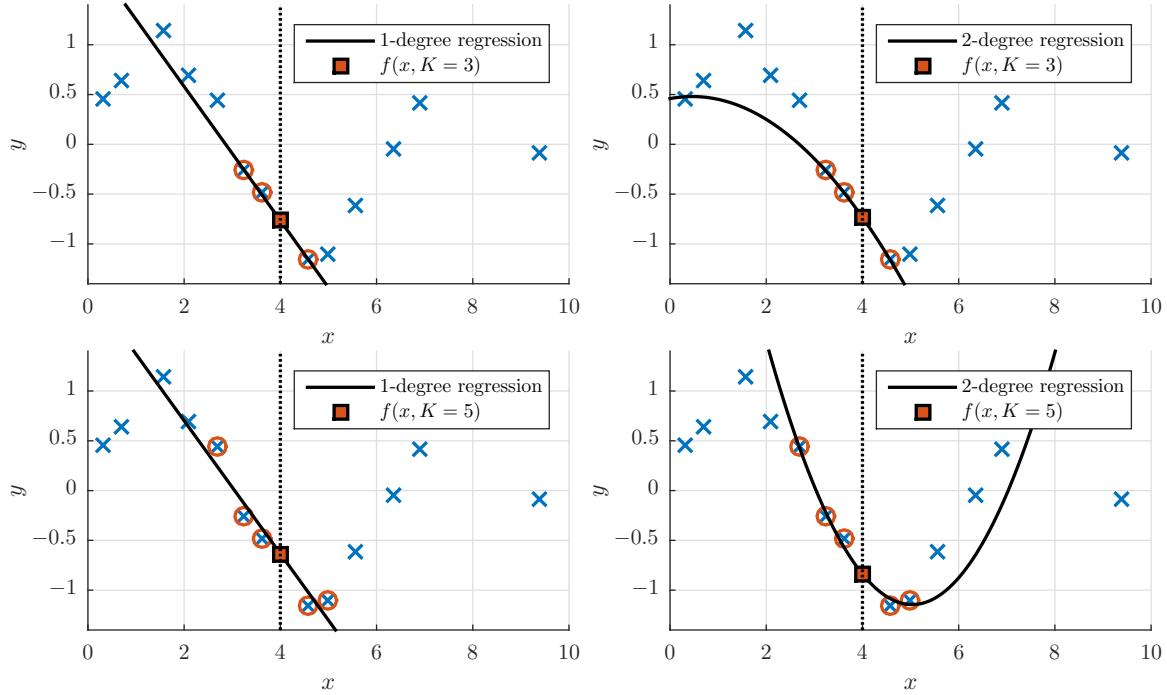


Fig. 12.6. KNN regression can be extended by instead of computing the mean within each region, we carry out a polynomial regression (similar to the example encountered in the previous section on linear regression) for the observations in each neighbourhood $N_{\mathbf{X}}(\mathbf{x}, K)$. For different degree $d = 2, 3$ this allows smoother regression curves, but higher d also requires a larger neighbourhood.

sum of square error. One-layer cross-validation for model selection can be used to select K and two-layer cross-validation used for selecting K and estimating the generalization error. A particularly useful simplification is when we apply leave-one-out cross-validation. Recall in leave-one-out cross-validation we have to iterate over all observations in our data set, train a model on $N - 1$ observations and test on the left-out observation. In the case of nearest-neighbour methods this can be accomplished by first defining $\mathbf{X}_{\setminus i}$ as \mathbf{X} with observation \mathbf{x}_i removed:

$$\mathbf{X}_{\setminus i}^T = [\mathbf{x}_1 \ \mathbf{x}_2 \ \cdots \ \mathbf{x}_{i-2} \ \mathbf{x}_{i-1} \ \mathbf{x}_{i+1} \ \mathbf{x}_{i+2} \ \cdots \ \mathbf{x}_N] \quad (12.6)$$

then $N_{\mathbf{X}_{\setminus i}}(\mathbf{x}_i, K)$ is the K -neighbourhood of \mathbf{x}_i when \mathbf{x}_i has been left out of the dataset \mathbf{X} . The generalization error for a given value of K can then be estimated as:

$$\tilde{E}_K^{\text{gen}} = \frac{1}{N} \left[\sum_{i=1}^N \text{Error of observation } \mathbf{x}_i \text{ when predicted using } N_{\mathbf{X}_{\setminus i}}(\mathbf{x}_i, K) \right]$$

Where the error in question could be either classification error or the sum-of-square error in case of regression. As usual the generalization error is estimated for each K and the K with the lowest generalization error is selected.

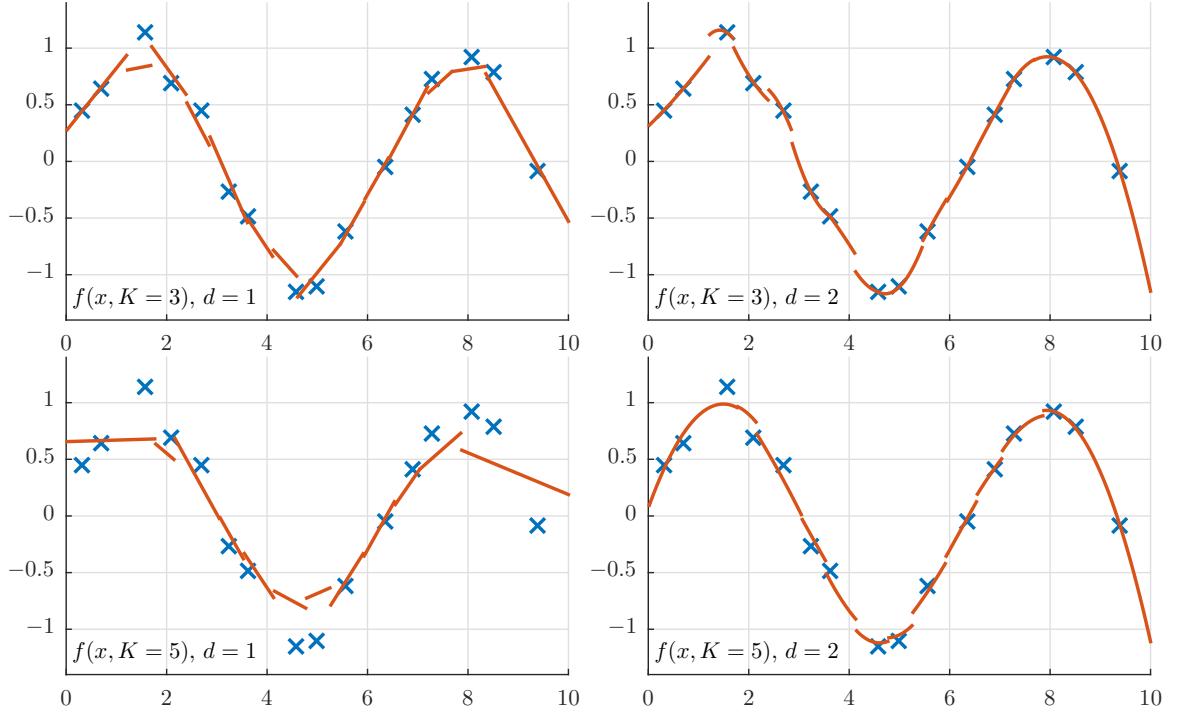


Fig. 12.7. The prediction curves for KNN regression with polynomials introduced in fig. 12.6, here shown for linear polynomials and second-degree polynomials for different neighborhood sizes.

Problems

12.1. Question 1: In order to predict if an observation corresponds to a relatively small or large island we will use a k-nearest neighbor (KNN) classifier based on the Euclidean distance between the eight observations given in Table 12.1. We will use leave-one-out cross-validation for the KNN in order to classify whether the eight considered observations constitute small islands (given in red, i.e. observation O1, O2, O3, O4) or large island (given in blue, i.e. observation O5, O6, O7, O8) using a three-nearest neighbor classifier, i.e. $K = 3$. The analysis will be based only on the data given in Table 12.1. Which one of the following statements is *correct*?

	O1	O2	O3	O4	O5	O6	O7	O8
O1	0	2.39	1.73	0.96	3.46	4.07	4.27	5.11
O2	2.39	0	1.15	1.76	2.66	5.36	3.54	4.79
O3	1.73	1.15	0	1.52	3.01	4.66	3.77	4.90
O4	0.96	1.76	1.52	0	2.84	4.25	3.80	4.74
O5	3.46	2.66	3.01	2.84	0	4.88	1.41	2.96
O6	4.07	5.36	4.66	4.25	4.88	0	5.47	5.16
O7	4.27	3.54	3.77	3.80	1.41	5.47	0	2.88
O8	5.11	4.79	4.90	4.74	2.96	5.16	2.88	0

Table 12.1. Pairwise Euclidean distance, i.e. $d(Oa, Ob) = \|\mathbf{x}_a - \mathbf{x}_b\|_2 = \sqrt{\sum_m (x_{am} - x_{bm})^2}$, between eight observations of the Galápagos data. Red observations (i.e., O1, O2, O3, and O4) correspond to the four smallest islands whereas blue observations (i.e., O5, O6, O7, and O8) correspond to the four largest islands.

- A The error rate of the classifier will be 1/8
- B The error rate of the classifier will be 1/4
- C The error rate of the classifier will be 3/8
- D The error rate of the classifier will be 1/2
- E Don't know.

12.2. Question 2: Consider a two-dimensional data set consisting of $N = 7$ observations as shown in fig. 12.8. The dataset consist of two classes indicated by the black crosses (class 1) and red circles (class 2). In the figure, the decision boundary for four K -nearest neighbor classifier (KNN) is shown such that the lighter brown color indicates Class 2 (red circles) and the darker brown color indicates Class 1 (black crosses). Suppose K is restricted to the values 1, 3, 5, 7, which of the following statements are true about values of k_1, k_2, k_3 and k_4 ?

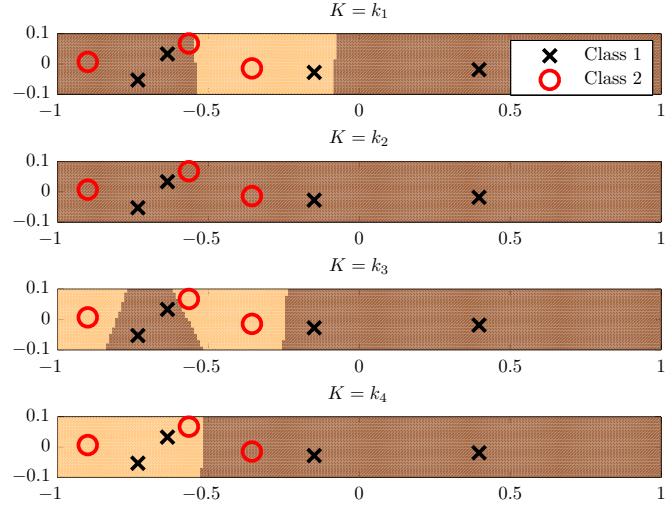


Fig. 12.8. Decision boundaries for four KNN classifiers.

- A $k_1 = 1, k_2 = 7, k_3 = 3, k_4 = 5$
- B $k_1 = 1, k_2 = 5, k_3 = 7, k_4 = 3$
- C $k_1 = 5, k_2 = 7, k_3 = 1, k_4 = 3$
- D $k_1 = 3, k_2 = 7, k_3 = 1, k_4 = 5$
- E Don't know.

12.3. Question 3: Consider again the distances in table 12.2. We will predict the label indicated by the blue color, i.e. observations o_1, \dots, o_4 belong to class C_1 and observations o_5, \dots, o_8 to class C_2 . This will be done using a k -nearest neighbor (KNN) classifier based on the cityblock distance measure indicated in table 12.2. We will use leave-one-out cross validation (i.e. the observation that is being predicted is left out) using one-nearest classifier, i.e. $k = 1$. What is the accuracy if all $N = 8$ observations are classified?

	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8
o_1	0	4	7	9	5	5	5	6
o_2	4	0	7	7	7	3	7	8
o_3	7	7	0	10	6	6	4	9
o_4	9	7	10	0	8	6	10	9
o_5	5	7	6	8	0	8	6	7
o_6	5	3	6	6	8	0	8	11
o_7	5	7	4	10	6	8	0	7
o_8	6	8	9	9	7	11	7	0

Table 12.2. Pairwise Cityblock distance, i.e. $d(o_i, o_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_1 = \sum_{k=1}^M |x_{ik} - x_{jk}|$, between 8 observations. Each observation o_i corresponds to a $M = 15$ dimensional binary vector, $x_{ik} \in \{0, 1\}$. The blue observations $\{o_1, o_2, o_3, o_4\}$ belong to class C_1 and the black observations $\{o_5, o_6, o_7, o_8\}$ belong to class C_2 .

- A accuracy = $\frac{1}{8}$
- B accuracy = $\frac{1}{4}$
- C accuracy = $\frac{1}{2}$
- D accuracy = $\frac{5}{8}$
- E Don't know.

Bayesian methods

Bayesian methods is the application of the basic rules of probability, in particular Bayes' theorem

$$p(y|x) = \frac{p(x|y)p(y)}{\sum_{y'=1}^C p(x|y')p(y')}$$

to machine learning. We have already seen several such application, for instance the analysis of the coin in ?? and as an element of the credibility intervals in chapter 10. However, in this chapter we will consider the problem more heads on and discuss additional terminology particular to Bayesian methods, namely Bayesian networks also called Bayesian belief networks, which is a graphical way of representing a distribution of many variables. Before this we will consider the distinction between discriminative and generative models.

13.1 Discriminative and generative modeling

Consider a standard classification problem in which we try to determine what class y_i an observation \mathbf{x}_i belongs to. For instance, consider trying to learn to distinguish between cats ($y_i = 0$) and dogs ($y_i = 1$) based on features \mathbf{x}_i of each animal. Logistic regression essentially tries to fit a straight line – a decision boundary– that separates the cats from the dogs. A new instance \mathbf{x}_i is then classified by observing which side of the decision boundary it lies on. This can be seen as *directly* coming up with a mapping of

$$p(y|\mathbf{x}, \mathbf{w}). \quad (13.1)$$

This is known as *discriminative* analysis. Bayesian inference also tries to determine this mapping, but considers a very different approach. First, we look at all instances of cats, and then we build a model of what cats look like. Then we look at all instances of dogs, and we build a model of what dogs look like. Then to classify a new instance, we consider how well it corresponds to what we expect a cat or a dog will look like according to respectively the cat model and the dog model, and we make our decision accordingly. This way of first coming up with models of what respectively dogs and cats look like is known as *generative* modelling and Bayesian inference naturally corresponds to generative modelling.

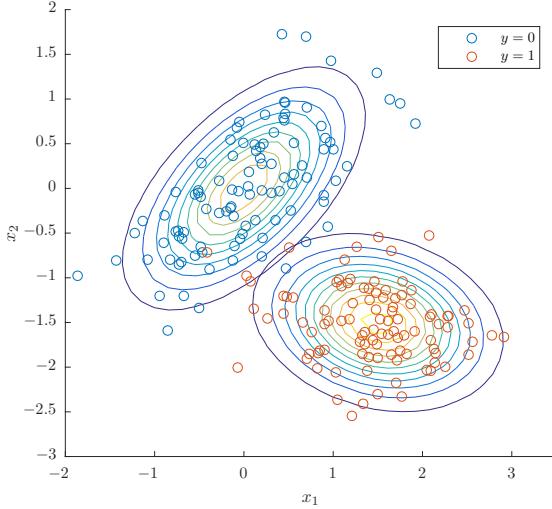


Fig. 13.1. Example of a Bayes classifier fitted to a two-class example dataset. The contours indicate the multivariate Gaussians fitted to each of the two classes separately.

13.1.1 Bayes classifier

Continuing the above discussion, consider Bayes' theorem in the two-class setting:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x}|y=0)p(y=0) + p(\mathbf{x}|y=1)p(y=1)} \quad (13.2)$$

So when this model considers if a new animal should be classified as a cat, $y = 0$, it considers how much the animal looks like a cat

$$p(\mathbf{x}|y=0),$$

multiply by the prior probability the animal is a cat $p(y=0)$ and divide this by the same quantity including the similar expression for dogs, $p(\mathbf{x}|y=1)$ and $p(y=1)$. To make this more concrete, let's suppose we observe n_0 instances of cats, \mathbf{X}^{Cats} , and n_1 instances of dogs, \mathbf{X}^{Dogs} , each observation consisting of two features. The labelled dataset is plotted in fig. 13.1. Then we can model the observations for instance as two multivariate normal distributions

$$p(\mathbf{x}|y=0) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \quad (13.3)$$

$$p(\mathbf{x}|y=1) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1), \quad (13.4)$$

where the parameters $\boldsymbol{\mu}_0$, $\boldsymbol{\Sigma}_0$ and $\boldsymbol{\mu}_1$, $\boldsymbol{\Sigma}_1$ can be estimated from the data as (here given for Cats):

$$\boldsymbol{\mu}_0 = \frac{1}{n_0} \sum_{i=1}^{n_0} \mathbf{x}_i^{\text{Cats}}, \quad \text{and} \quad \boldsymbol{\Sigma}_0 = \frac{1}{n_0 - 1} \sum_{i=1}^{n_0} (\mathbf{x}_i^{\text{Cats}} - \boldsymbol{\mu}_0)(\mathbf{x}_i^{\text{Cats}} - \boldsymbol{\mu}_0)^T, \quad (13.5)$$

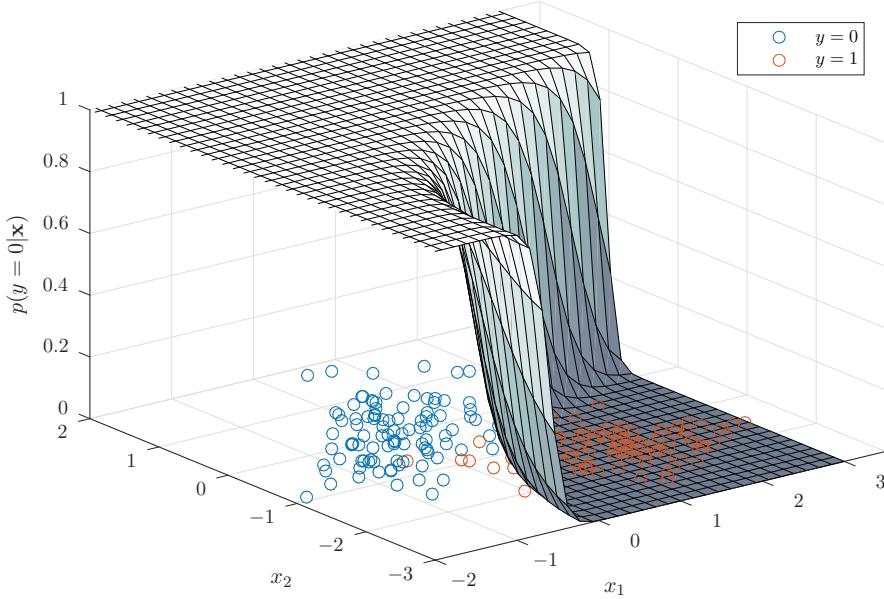


Fig. 13.2. Decision rule, i.e. the probability $p(y = 0|\mathbf{x})$, of the Bayes classifier when trained on the two-class dataset from fig. 13.1. Notice, the decision rule is quite steep at the boundary.

corresponding to the two contour plots in fig. 13.1. Using this together with the priors $p(y = 0) = \frac{n_0}{n_0+n_1}$, $p(y = 1) = \frac{n_1}{n_0+n_1}$ allows us to compute the probability the animal belongs to either of the two classes as:

$$p(y = c|\mathbf{x}) = \frac{p(\mathbf{x}|y = c)p(y = c)}{p(\mathbf{x}|y = 0)p(y = 0) + p(\mathbf{x}|y = 1)p(y = 1)}. \quad (13.6)$$

The decision boundary, i.e. $p(y = 0|\mathbf{x})$, is plotted in fig. 13.2 as the gray surface.

13.2 Naïve-Bayes classifier

Naïve-Bayes is simply the standard Bayesian approach with a particular simplification. Consider the Bayes classifier for C classes using a dataset with M features:

$$p(y|x_1, x_2, \dots, x_M) = \frac{p(x_1, x_2, \dots, x_M|y)p(y)}{\sum_{c=1}^C p(x_1, \dots, x_M|y = c)p(y = c)} \quad (13.7)$$

y	x_1	x_2
1	1	0
1	0	1
1	1	1
2	1	1
2	1	0
2	0	0
3	1	1
3	0	1

Table 13.1. A dataset consisting of $N = 8$ students and for each student we record two binary features x_1 and x_2 corresponding to the sex of the student and if the student is typically going out in the evening or not. The first column $y = 1, 2, 3$ correspond to the grade of the student, where $y = 1$ means a low grade, $y = 2$ a medium grade and $y = 3$ a high grade.

A problem with the Bayes classifier is if M is very large, representing the conditional distribution

$$p(x_1, x_2, \dots, x_M | y),$$

may be very expensive. For instance, for the multivariate normal distribution this requires storing a symmetric covariance matrix Σ and the mean vector μ , in total $M + \frac{1}{2}M(M + 1)$ numbers. This is not only costly, but if we do not have much data, estimating this many parameters may not be possible to do reliably. The Naïve-Bayes assumption is simply that we *assume* that the conditional distribution factorizes:

$$p(x_1, x_2, \dots, x_M | y) = p(x_1 | y)p(x_2 | y) \dots p(x_M | y).$$

If we still represent each factor as a 1D normal distribution this only requires $2M$ numbers (i.e., the mean value and variance for each of the attributes x_1, x_2, \dots, x_M). Plugging this into eq. (13.7) we obtain the Naïve Bayes approximation:

$$p(y|x_1, x_2, \dots, x_M) = \frac{p(x_1|y) \times \dots \times p(x_M|y)p(y)}{\sum_{c=1}^C p(x_1|y=c) \times \dots \times p(x_M|y=c)p(y=c)}. \quad (13.8)$$

Example:

To illustrate the basics of the procedure, we will consider a simple example based on the data shown in table 13.1. The dataset consists of $N = 8$ students and for each student we record two binary features x_1 and x_2 corresponding to the sex of the student and if the student is typically going out in the evening or not. The first column $y = 1, 2, 3$ corresponds to the grade of the student, where $y = 1$ means a low grade, $y = 2$ means a medium grade and $y = 3$ a high grade. Suppose we want to train a naïve-Bayes classifier on the dataset and use it to determine the probability a new observation $x_1 = 0$ and $x_2 = 1$ belong to any of the three classes. We first compute the class-priors to be

$$p(y = 1) = p(y = 2) = \frac{3}{8} \quad \text{and} \quad p(y = 3) = \frac{2}{8} = \frac{1}{4}$$

Then we can compute the probability of $p(x_j = 0 | y = c)$ as:

$$p(x_j = 0|y = c) = \frac{\{\text{Number of times where } x_j = 0 \text{ and } y = c\}}{\{\text{Total number of times where } y = c\}}. \quad (13.9)$$

In particular, we obtain:

$$\begin{aligned} p(x_1 = 0|y = 1) &= \frac{1}{3}, & p(x_1 = 0|y = 2) &= \frac{1}{3}, & p(x_1 = 0|y = 3) &= \frac{1}{2}, \\ p(x_2 = 0|y = 1) &= \frac{1}{3}, & p(x_2 = 0|y = 2) &= \frac{2}{3}, & p(x_2 = 0|y = 3) &= \frac{0}{2}. \end{aligned}$$

Using that $p(x_2 = 1|y = c) = 1 - p(x_2 = 0|y = c)$ we then compute the probability of the new observation as

$$\begin{aligned} p(x_1 = 0, x_2 = 1|y = 1) &= p(x_1 = 0|y = 1)p(x_2 = 1|y = 1) = \frac{1}{3} \times (1 - \frac{1}{3}) = \frac{2}{9}, \\ p(x_1 = 0, x_2 = 1|y = 2) &= p(x_1 = 0|y = 2)p(x_2 = 1|y = 2) = \frac{1}{3} \times (1 - \frac{2}{3}) = \frac{1}{9}, \\ p(x_1 = 0, x_2 = 1|y = 3) &= p(x_1 = 0|y = 3)p(x_2 = 1|y = 3) = \frac{1}{2} \times (1 - 0) = \frac{1}{2}. \end{aligned}$$

In our case $p(y=c|x_1=0, x_2=1)$ can then be computed using eq. (13.8) to be:

$$\frac{p(x_1=0, x_2=1|y=c)p(y=c)}{p(x_1=0, x_2=1|y=1)p(y=1) + p(x_1=0, x_2=1|y=2)p(y=2) + p(x_1=0, x_2=1|y=3)p(y=3)}$$

and simply plugging in the above numbers we obtain:

$$\begin{aligned} p(y = 1|x_1 = 0, x_2 = 1) &= \frac{\frac{2}{9} \times \frac{3}{8}}{\frac{2}{9} \times \frac{3}{8} + \frac{1}{9} \times \frac{3}{8} + \frac{1}{2} \times \frac{1}{4}} = \frac{1}{3}, \\ p(y = 2|x_1 = 0, x_2 = 1) &= \frac{\frac{1}{9} \times \frac{3}{8}}{\frac{2}{9} \times \frac{3}{8} + \frac{1}{9} \times \frac{3}{8} + \frac{1}{2} \times \frac{1}{4}} = \frac{1}{6}, \\ p(y = 3|x_1 = 0, x_2 = 1) &= \frac{\frac{1}{2} \times \frac{1}{4}}{\frac{2}{9} \times \frac{3}{8} + \frac{1}{9} \times \frac{3}{8} + \frac{1}{2} \times \frac{1}{4}} = \frac{1}{2}. \end{aligned}$$

The naïve-Bayes assumption is often used when the number of features M is very large; a popular application is spam-filtering where each of the binary features correspond to the presence or absence of a word in the email.

13.2.1 Naïve-Bayes for non-binary data and robust estimation

The Naïve-Bayes method, as introduced in the above example, has two shortcomings. The first shortcoming is that in most applications, the attributes x_1, \dots, x_M will be of mixed types (binary, categorical, continuous, etc.). This is relatively simple to overcome as it simply amounts to selecting appropriate densities for the terms $p(x_j = x|y = c)$ in eq. (13.8).

The second shortcoming is that when we estimate the individual probabilities directly $p(x_1|y), \dots, p(x_M|y)$ from the data as in eq. (13.9), one of these probabilities may easily be zero for a new test point which will cause the entire expression eq. (13.8) to be zero, and the method will be overly confident on new input (in fact, we can easily end up dividing by zero in eq. (13.8)). This limitation can be overcome by using *robust estimation* of the probabilities $p(x_j|y = c)$.

Technical note 13.2.1: Where does the regularization constants come from?

In the binary case, the parameter α arises by treating the probability $\theta = p(x_j = 1|y)$ as an unknown quantity which we estimate from data using the Beta-Bernoulli calculation section 6.4, but using a symmetric $\text{Beta}(\theta|\alpha, \alpha)$ prior. Doing this we obtain the posterior distribution using eq. (6.38) as:

$$\text{Beta}(\theta|n_1 + \alpha, N_c - n_1 + \alpha).$$

This is the distribution of the unknown probability, and a reasonable estimate is simply the mean value: $\mathbb{E}[\theta] = \frac{n_1 + \alpha}{N_c + 2\alpha}$, which is exactly our probability estimate in eq. (13.10).

Robust estimation, binary case

If we let $N_c = \sum_{i=1}^N \delta_{y_i, c}$ be the number of observations belonging to class c , we can then count the number of these observations for which feature j was k , i.e. were $X_{ij} = k$, as

$$n_k = \sum_{i=1}^N \delta_{X_{ij}, k} \delta_{y_i, c}, \quad (13.10)$$

Using this, our new robust estimate of the marginal probabilities in the case x_j is binary is:

$$\text{Binary case: } p(x_j = 1|y = c) = \frac{n_1 + \alpha}{N_c + 2\alpha}. \quad (13.11)$$

Where $\alpha \geq 0$ denotes the amount of robustness. Notice this is equivalent to simply adding a factor of α to the nominator and a factor 2α to the denominator in eq. (13.9). For instance, we would get $p(x_2 = 0|y = 1) = \frac{1+\alpha}{3+2\alpha}$ in our example. In other words, if $\alpha = 0$ we have no robustness, and if α is very large the probabilities will all be near $\frac{1}{2}$. This might seem familiar, and there is a connection to the robustness parameter and the priors in the Beta-Bernoulli derivation from section 6.4, see Technical Note 13.2.1.

Robust estimation, categorical case

The above trick generalizes to the categorical case with nearly no modifications. Assume attribute j can take K different values, and denote the number of observations in class $y = c$ such that attribute j takes a value of $x_j = k$ as n_k , $k = 1, \dots, K$ just as in eq. (13.10) and note that $\sum_{k=1}^K n_k = N_c$. We can then define the robust estimate as simply:

$$\text{Categorical case: } p(x_j = k|y = c) = \frac{n_k + \alpha}{N_c + K\alpha}. \quad (13.12)$$

Note this simplifies to the binary case when $K = 2$, assuming the classes are re-labeled to 0 and 1.

Robust estimation, normal case

If attribute x_j is continuous, the natural choice is to model it as a normal distribution where the mean and variances are estimated from the observations belonging to class $y = c$. We can obtain a robust estimate by adding a factor α to the estimate of the standard deviations to ensure they do not collapse to singular values. Specifically:

$$\text{Continious case: } p(x_j = x|y = c) = \mathcal{N}(x|\mu = \mu_c, \sigma^2 = (\sigma_c + \alpha)^2) \quad (13.13)$$

$$\mu_c = \mathbb{E}_{y=c}[x_j] = \frac{1}{N_c} \sum_{i=1}^N \delta_{y,c} X_{ij}, \quad \text{and} \quad \sigma_c = \text{std}_{y=c}[x_j] = \sqrt{\frac{1}{N_c - 1} \sum_{i=1}^N \delta_{y,c} (X_{ij} - \mu_c)^2}$$

Selecting the parameters

The robust estimation parameter α should be selected by using cross-validation for parameter selection (algorithm 5) to choose between a handful of reasonable values of α . Include $\alpha = 0$ if it does not cause underflow. Note different computational environments might implement different strategies for parameter estimation and robust estimation than described above. For instance, it is also customary to robustly estimate the standard deviation in the normal case by adding a value corresponding to a fraction of the *maximal* per-class standard deviation rather than an absolute numerical value as above.

13.3 Bayesian networks★

A Bayesian network also called a belief network or Bayesian belief network is not as such adding a new method to our toolbox, but it provides a convenient and often-used notation for presenting existing probabilities. Consider the following example adapted from Pearl [2014], MacKay [2003]

Fred lives in Los Angeles and commutes 60 miles to work. Whilst at work, he receives a phone-call from his neighbour saying that Fred's burglar alarm is ringing. What is the probability that there was a burglar in his house today? While driving home to investigate, Fred hears on the radio that there was a small earthquake that day near his home. 'Oh', he says, feeling relieved, 'it was probably the earthquake that set off the alarm'. What is the probability that there was a burglar in his house?

To analyse this story we first introduce the variables:

- a : The alarm is ringing.*
- b : A burglar was in Fred's house.*
- c : Fred received a phone-call reporting the alarm.*
- e : A small earthquake took place today near Fred's house.*
- r : The radio report of the earthquake is heard by Fred.*

In a case like this, we know (from our experience) that some of these events must be independent. That there is a burglar or a minor earthquake is presumably unrelated events, so $p(b, e) = p(b)p(e)$. In general, the probability of these variables will factorize as follows:

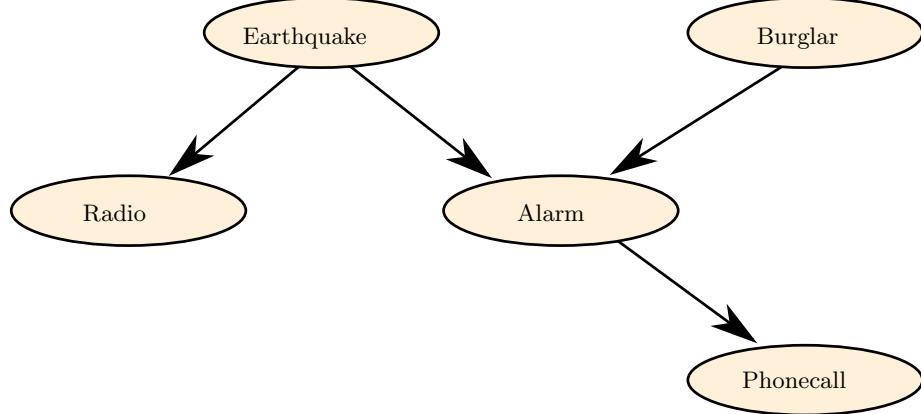


Fig. 13.3. Bayesian network of the burglar example. Each vertex corresponds to a variable, and incident edges correspond to conditional dependence.

$$p(a, b, c, e, r) = p(b)p(e)p(a|b, e)p(c|a)p(r|e). \quad (13.14)$$

This factorization of the probability has important consequences. Firstly, as for the naïve-Bayes assumption, it makes the probability density much less costly to store on a computer and reliable to estimate as there are fewer parameters than the full (un-factorized) joint distribution. Secondly, it allows faster computation by exploiting the factorization structure and finally it allows us easier to see what quantities are independent of each other. It is common to represent the factorization as a network where the vertices correspond to the variables and the edges correspond to statistical dependence, see fig. 13.3 for an illustration. So for instance, if there is an edge from B to A , that means that in the joint distribution, then A must be conditional on B and possible other variables connected to A . To solve the Burglar problem, assume the probability of there being a burglar is $p(b = 1) = 0.1\%$, earthquake $p(e = 1) = 0.1\%$ (corresponding to roughly one burglar and earthquake every four years) and that the alarm is triggered either by (1) false alarms (very low probability), (2) if an earthquake takes place (low probability) and finally if a burglar enters the home (high probability). In our example these probabilities are:¹

¹ For instance, suppose we let $f = 0.1\%$ denote the chance a false alarm triggers a , $\alpha_e = 1\%$ the chance an earthquake triggers a and finally $\alpha_b = 99\%$ the chance a burglar triggers a . The probabilities can then be obtained as:

$$\begin{aligned} p(a = 1|b = 0, e = 0) &= f, \\ p(a = 1|b = 0, e = 1) &= 1 - (1 - f)(1 - \alpha_e), \\ p(a = 1|b = 1, e = 0) &= 1 - (1 - f)(1 - \alpha_b), \\ p(a = 1|b = 1, e = 1) &= 1 - (1 - f)(1 - \alpha_b)(1 - \alpha_e). \end{aligned}$$

$$\begin{aligned} p(a = 1|b = 0, e = 0) &= 0.1\%, \\ p(a = 1|b = 0, e = 1) &= 1.099\%, \\ p(a = 1|b = 1, e = 0) &= 99.001\%, \\ p(a = 1|b = 1, e = 1) &= 99.011\%. \end{aligned}$$

Finally assume the neighbour would never phone if the alarm is not ringing ($p(c = 1|a = 0) = 0$) and that the radio reported is also trustworthy ($p(r = 1|e = 0) = 0$) and let's return to the problem: Suppose first the phone calls $c = 1$; then we know the alarm is ringing $a = 1$ and so the posterior probability of b, e (burglary and earthquake) becomes:

$$p(b, e|a = 1) = \frac{p(a = 1|b, e)p(b, e)}{p(a = 1)}.$$

We can use the Bayes network to compute these probabilities. For instance when computing $p(a = 1)$, we must compute this by summing over all other variables than a :

$$p(a = 1) = \sum_{b \in \{0,1\}} \sum_{c \in \{0,1\}} \sum_{e \in \{0,1\}} \sum_{r \in \{0,1\}} p(a = 1, b, c, e, r), \quad (13.15)$$

However, if we plug in the expression of the likelihood (13.14) we see that variables c and r can trivially be summed out (i.e., marginalized):

$$\begin{aligned} p(a = 1) &= \sum_{b \in \{0,1\}} \sum_{c \in \{0,1\}} \sum_{e \in \{0,1\}} \sum_{r \in \{0,1\}} p(b)p(e)p(a = 1|b, e)p(c|a = 1)p(r|e) \\ &= \sum_{b \in \{0,1\}} \sum_{e \in \{0,1\}} \left[p(b)p(e)p(a = 1|b, e) \left(\sum_{c \in \{0,1\}} p(c|a = 1) \sum_{r \in \{0,1\}} p(r|e) \right) \right] \\ &= \sum_{b \in \{0,1\}} \sum_{e \in \{0,1\}} p(b)p(e)p(a = 1|b, e) \end{aligned} \quad (13.16)$$

Comparing to the Bayesian network in fig. 13.3 we see that to determine what variables remain in the sum when computing $p(a)$, we look at all other vertices in a network where we can move to a by going in the *direction* of the edges. See fig. 13.4 where we have illustrated the two nodes that remain, e, b , with red. Using the above numbers we obtain:

$$\begin{aligned} p(a = 1|b = 0, e = 0)p(b = 0)p(e = 0) &= 0.000998, \\ p(a = 1|b = 1, e = 0)p(b = 1)p(e = 0) &= 0.0000989, \\ p(a = 1|b = 0, e = 1)p(b = 0)p(e = 1) &= 0.000010979, \\ p(a = 1|b = 1, e = 1)p(b = 1)p(e = 1) &= 9.9 \times 10^{-7}. \end{aligned}$$

By inserting these four numbers into eq. (13.16) and summing we obtain $p(a = 1) = 0.002$ and so from eq. (13.15)

$$p(b = 0, e = 0|a = 1) = 0.4993, \quad (13.17)$$

$$p(b = 1, e = 0|a = 1) = 0.4947, \quad (13.18)$$

$$p(b = 0, e = 1|a = 1) = 0.0055, \quad (13.19)$$

$$p(b = 1, e = 1|a = 1) = 0.0005. \quad (13.20)$$

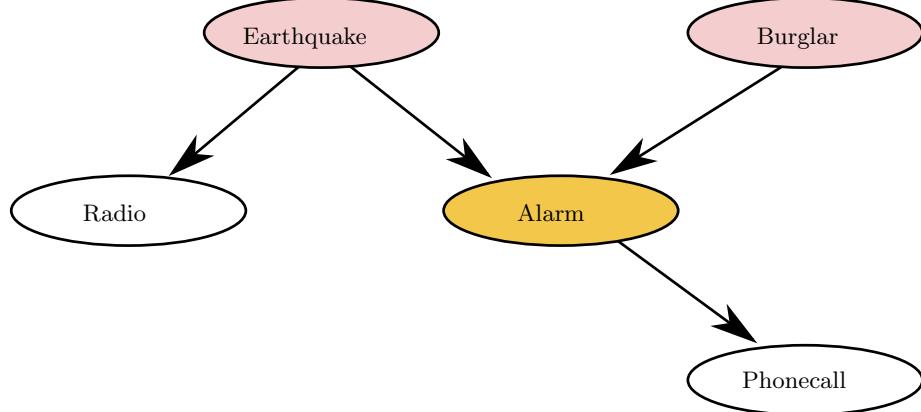


Fig. 13.4. To determine what variables must be summed out when computing the marginal of a variable such as a , we look at all variables such that one can move in the direction of the arrows from those variables to a . This gives $p(a, e, b)$

So returning to the initial question, when we determine if there was a burglar at the house we must compute $p(b = 1|a = 1)$ which can be accomplished by marginalizing over the burglar-variable:

$$\begin{aligned} p(b = 0|a = 1) &= p(b = 0, e = 0|a = 1) + p(b = 0, e = 1|a = 1) &= 0.505 \\ p(b = 1|a = 1) &= p(b = 1, e = 0|a = 1) + p(b = 1, e = 1|a = 1) &= 0.494 \end{aligned}$$

so after receiving the call, we believe there to be a 50% chance there was a burglar in the house. An important point to take away from this example is that b and e , which were initially independent: $p(e, b) = p(e)p(b)$, are made *dependent* by the information a . Now consider the final part of the example. Suppose we also learn that $e = 1$ (i.e. there was an earthquake). The probability there was a burglar can now be computed as:

$$p(b|e, a) = \frac{p(b, e|a)}{p(e|a)} = \frac{p(b, e|a)}{p(e, b = 0|a) + p(e, b = 1|a)}.$$

If we plug in numbers we obtain $p(e = 1|a = 1) = 0.006$ and so

$$\begin{aligned} p(b = 0|e = 1, a = 1) &= \frac{p(b = 0, e = 1|a = 1)}{p(e = 1|a = 1)} &= 0.92 \\ p(b = 1|e = 1, a = 1) &= \frac{p(b = 1, e = 1|a = 1)}{p(e = 1|a = 1)} &= 0.08 \end{aligned}$$

So after learning the alarm was triggered, this lowers our probability there was a burglar in the house from about 50% to about 8%. This is in accordance to everyday intuition: when we learn about the earthquake, we consider *that* to be the more plausible explanation of the alarm.

13.3.1 A brief comment on causality

Using the implied rules any factorization of a joint distribution is easily translated into a network: Vertices implies variables and there is an edge from variable x to y if there is a term $p(y|x, \dots)$ in

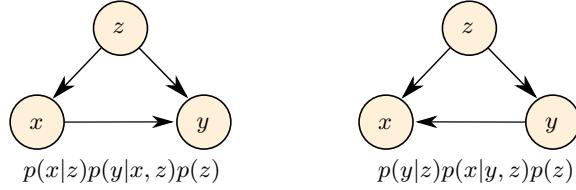


Fig. 13.5. Two bayesian networks which both represent the same distribution $p(x, y, z)$. Since the two networks are not similar this shows a Bayesian network cannot be interpreted as a causal graph.

the factorization of the joint distribution. A point that is sometimes confused is to interpret the Bayesian network as having a causal meaning. Consider a general distribution $p(x, y, z)$. We are always allowed to write this distribution as:

$$p(x, y, z) = p(x|z)p(y|x, z)p(z), \quad p(x, y, z) = p(y|z)p(x|y, z)p(z),$$

since the two distributions are the same, but clearly give rise to different Bayesian networks as shown in fig. 13.5, this shows we cannot interpret a Bayesian network as a causal graph.

Problems

13.1. Question 1: Nine of the fifteen observations in Table 13.2 have chronic kidney disease (i.e., O_1-O_9 given in red) whereas six of the observations do not have chronic kidney disease (i.e., $O_{10}-O_{15}$ given in black). We would like to predict whether a subject has chronic kidney disease or not using the data in Table 13.2 and the attributes RBC , PC , DM , and CAD . We will apply a Naïve Bayes classifier that assumes independence between the four attributes. Given that a subject has these four attributes (i.e., $RBC = 1$, $PC = 1$, $DM = 1$, and $CAD = 1$) what is the probability that the person has chronic kidney disease, i.e., what is

$P(CKD = 1|RBC = 1, PC = 1, DM = 1, CAD = 1)$ according to the Naïve Bayes classifier?

	RBC	PC	PCC	HTN	DM	CAD	PE
O_1	0	0	0	0	1	0	0
O_2	0	1	1	1	0	0	1
O_3	0	0	0	0	0	0	0
O_4	0	1	0	0	1	0	1
O_5	0	1	1	1	1	0	0
O_6	1	1	1	1	1	0	0
O_7	1	1	1	1	1	0	1
O_8	0	1	1	1	1	1	1
O_9	0	1	0	1	0	0	0
O_{10}	1	1	0	0	0	1	0
O_{11}	0	0	0	0	1	0	0
O_{12}	0	0	0	0	0	0	0
O_{13}	0	0	0	0	0	0	0
O_{14}	0	0	0	0	0	0	0
O_{15}	0	0	0	0	0	0	0

Table 13.2. For each observation there are $M = 7$ binary features and $N = 15$ observations O_1, \dots, O_{15} belonging to two categories (i.e., CKD=1 for O_1, \dots, O_9 and CKD=0 for O_{10}, \dots, O_{15}).

- A 2.56 %
- B 96.14 %
- C 98.03 %
- D 100 %
- E Don't know.

13.2. Question 2: We will consider a Bayes classifier using the attributes RBC , PC , and DM in Table 13.2 (i.e., we no longer consider the attribute CAD). What is $P(CKD = 1|RBC = 1, PC = 1, DM = 1)$ according to a Bayes classifier (i.e. we are no longer imposing independence as in the Naïve Bayes classifier)?

- A 26.67 %
- B 97.07 %
- C 98.03 %
- D 100 %
- E Don't know.

13.3. Question 3: Five of the ten considered subjects in Table 13.3 survived after five years ($S_1 - S_5$) given in black whereas five subjects died within five years ($NS_1 - NS_5$) given in red. We would like to predict

whether a subject survived using the data in Table 13.3 and the attributes YAY , OAY , PAY . We will apply a Naïve Bayes classifier that assumes independence between the three attributes. Given that a subject had these three attributes (i.e., $YAY = 1$, $OAY = 1$, $PAY = 1$) what is the probability that the subject survived according to the Naïve Bayes classifier. I.e., what is $P(S|YAY = 1, OAY = 1, PAY = 1)$ according to the Naïve Bayes classifier?

	YAY	YAN	OAY	OAN	PAY	PAN
S_1	1	0	1	0	1	0
S_2	1	0	1	0	0	1
S_3	0	1	0	1	1	0
S_4	0	1	1	0	1	0
S_5	0	1	1	0	1	0
NS_1	0	1	1	0	1	0
NS_2	0	1	0	1	1	0
NS_3	1	0	0	1	0	1
NS_4	0	1	1	0	1	0
NS_5	0	1	1	0	1	0

Table 13.3. Given are five subjects that survived in Haberman's study (denoted S_1, S_2, \dots, S_5) as well as the five subjects that did not survive in Haberman's study (denoted NS_1, NS_2, \dots, NS_5) including whether these subjects are young or old (YAY , YAN), were operated after 1960 or not (OAY , OAN), and had positive axillary nodes or not (PAY , PAN).

- A $\frac{16}{125}$
- B $\frac{3}{11}$
- C $\frac{1}{2}$
- D $\frac{8}{11}$
- E Don't know.

13.4. Question 4: Consider the observations in table 13.4. Suppose we only consider the first two features f_1, f_2 and train a Naive-Bayes classifier to classify between class C_1 (black) and C_2 (blue) based on these two features alone. Suppose an observation has $f_1 = 0, f_2 = 1$, what is the probability this observation belongs to class C_1 according to the Naive-Bayes classifier?

	f_1	f_2	f_3	f_4	f_5	f_6
s_1	0	1	1	0	1	0
s_2	0	1	1	1	0	1
s_3	1	1	1	0	1	0
s_4	1	1	1	0	1	0
s_5	0	1	1	0	1	1
s_6	0	0	1	1	1	1
s_7	1	1	0	1	1	1
s_8	1	1	1	0	0	0
s_9	1	0	1	1	0	0
s_{10}	1	1	1	0	0	1

- A $p_{NB}(C_1|f_1 = 0, f_2 = 1) = 0.83$
 B $p_{NB}(C_1|f_1 = 0, f_2 = 1) = 0.70$
 C $p_{NB}(C_1|f_1 = 0, f_2 = 1) = 0.67$
 D $p_{NB}(C_1|f_1 = 0, f_2 = 1) = 0.75$
 E Don't know.

Table 13.4. $N = 10$ observations s_1, \dots, s_{10} belonging to two categories. The black category C_1 (observations s_1, \dots, s_5) and the blue category C_2 (observations s_6, \dots, s_{10}). For each observation there are $M = 6$ binary features f_1, \dots, f_6 .

14

Regularization and the bias-variance decomposition

As we already saw in chapter 10, “Overfitting and cross-validation”, a too flexible model can easily overfit the dataset leading to a high generalization error. In this chapter we will consider a general technique for controlling model complexity known as *regularization*, which is useful in many supervised learning settings but it is particularly apt for linear and logistic regression as well as neural network modelling. We will then consider the problem (and need) to control the model complexity in a more general setting and analyse the tradeoff between having a very flexible model that may overfit and a very stable model that might underfit in what is known as the bias-variance decomposition of the generalization error. Regularization has been re-invented many times, but was first considered by Andrey Nikolayevich Tikhonov in 1943 [Tikhonov, 1943], meanwhile a good introduction to the tradeoff between bias and variance can be found in the discussion by James et al. [2014].

14.1 Least squares regularization

In this section, we will look at a general approach for managing model complexity known as *regularization*. Just as in the polynomial example, regularization allows us to make different models (by adding different degrees of regularization), and the most appropriate choice of regularization is then made using cross-validation for model selection. We illustrate the technique using least-squares regression. Consider the simple linear regression model we previously encountered in with prediction rule:

$$\mathbf{y}_i = f(\mathbf{x}_i, \mathbf{w}) = \tilde{\mathbf{x}}_i^T \mathbf{w},$$

as the reader may recall from section 8.1.1, the linear regression model was trained by minimizing the sum-of-squares error term:

$$E(\mathbf{w}) = \left\| \mathbf{y} - \tilde{\mathbf{X}} \mathbf{w} \right\|^2. \quad (14.1)$$

The optimal weights \mathbf{w}^* can be found by minimizing the error and are given by:

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} E(\mathbf{w}) = (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} (\tilde{\mathbf{X}}^T \mathbf{y}). \quad (14.2)$$

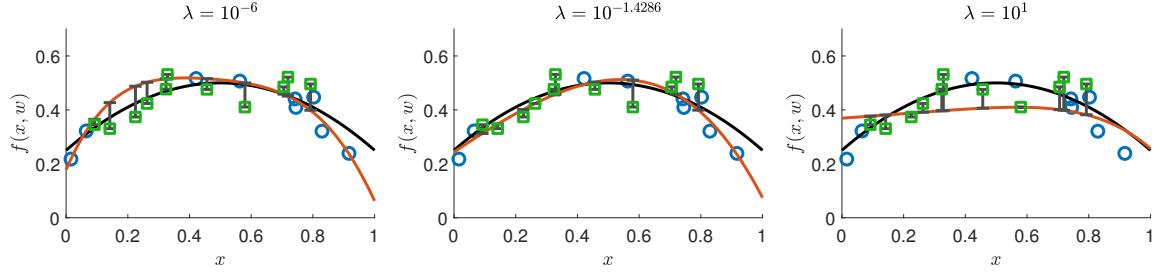


Fig. 14.1. A regularized linear regression model is fitted to the dataset of 9 observations and 10 test observations. The solutions, corresponding to three different values of λ , are shown in the three panes. Notice for larger values of λ , the solution is dragged towards the x -axis because the solution for the weights w^* becomes smaller according to eq. (14.3). The left-most pane has high variance but low bias, the right-most pane has high bias but low variance.

The way we arrived at this formulation was a simple application of the general likelihood framework discussed in section 6.5, see in particular eq. (6.47).

There are two potential issues in linear regression. Firstly, the matrix $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$ might not be invertible, which will happen if $N \leq M$ or if $\tilde{\mathbf{X}}$ contains many linearly dependent rows, and secondly, if M is large relative to N the linear regression model can overfit.

Regularization attempt to solve these problems, by simply altering the cost function to have a stronger preference small weights. However, note the magnitude of the weights are affected by the relative scaling of the columns of \mathbf{X} , and we therefore transform \mathbf{X} by subtracting the mean and dividing by the standard deviation of the columns:

$$\hat{X}_{ij} = \frac{X_{ij} - \mu_j}{\hat{s}_j}, \quad \mu_j = \frac{1}{N} \sum_{i=1}^N X_{kj}, \quad \hat{s}_j = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (X_{ij} - \mu_j)^2}$$

Next, we don't want the constant term in the regression to be affected by regularization, and we therefore consider a cost function of the form:

$$E_\lambda(\mathbf{w}, w_0) = \left\| \mathbf{y} - w_0 \mathbf{1} - \hat{\mathbf{X}} \mathbf{w} \right\|^2 + \lambda \|\mathbf{w}\|^2, \quad \lambda \geq 0. \quad (14.3)$$

The last term, $\lambda \|\mathbf{w}\|^2$, is called the *regularization term*, and the constant λ is called the *regularization constant*. The regularization constant influence the relative importance of the regularization term, starting with $\lambda = 0$ which correspond to the ordinary least-squares cost function eq. (14.1) aside the standardization. This form of regularization term is commonly referred to as L_2 regularization.

Solving regularized linear regression \star

Note our new objective still only depends on terms which are linear or quadratic in \mathbf{w} , and can therefore still be solved. To do so, first note that for any \mathbf{w} , the minimal value of the intercept term w_0 is

$$\frac{dE_\lambda(\mathbf{w}, w_0)}{dw_0} = \sum_{i=1}^N -2(y_i - w_0 \mathbf{1} - \hat{\mathbf{x}}_i^\top \mathbf{w}) = -2N\mathbb{E}[y] - 2Nw_0 - N \left(\frac{1}{N} \sum_{i=1}^N \hat{\mathbf{x}}_i^\top \right) \mathbf{w}$$

Since we have subtracted the column-wise mean from $\hat{\mathbf{X}}$, the term involving \mathbf{w} disappears. Setting the derivative equal to zero and solving gives:

$$w_0 = \mathbb{E}[y] = \frac{1}{N} \sum_{i=1}^N y_i$$

which, retrospectively, might seem fairly obvious. Therefore, suppose we define $\hat{y}_i = y_i - \mathbb{E}[y]$, we can then re-write the objective as:

$$E_\lambda = \|\hat{\mathbf{y}} - \hat{\mathbf{X}}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|^2, \quad \lambda \geq 0.$$

This objective can be solved by computing the derivative and setting it equal to zero. We get:

$$\begin{aligned} \frac{dE_\lambda}{d\mathbf{w}} &= -\hat{\mathbf{X}}^\top (\hat{\mathbf{y}} - \hat{\mathbf{X}}\mathbf{w})^2 + 2\mathbf{w} \\ \Rightarrow \mathbf{w}^* &= \arg \min_{\mathbf{w}} E(\mathbf{w}) = (\hat{\mathbf{X}}^\top \hat{\mathbf{X}} + \lambda \mathbf{I}) \backslash (\hat{\mathbf{X}}^\top \hat{\mathbf{y}}) \end{aligned} \quad (14.4)$$

This is very nearly the linear regression solution, except for the diagonal term $\lambda \mathbf{I}$ and that matrices have been transformed. To make a prediction, we have to be careful to apply the right feature transformations. Specifically, to make predictions for a test observation \mathbf{x} compute:

$$\left[\frac{x_1 - \mu_1}{\sigma_1} \quad \frac{x_2 - \mu_2}{\sigma_2} \quad \dots \quad \frac{x_M - \mu_M}{\sigma_M} \right] \mathbf{w}^* + \mathbb{E}[y]$$

where μ_i , σ_i , and $\mathbb{E}[y]$ are all computed on the *training* data. Note that aside from analytical convenience, the L_2 regularization can be motivated using Bayes' theorem, see Technical Note 14.1.1.

14.1.1 The effect of regularization

If we simply look at the expression for $E_\lambda(\mathbf{w})$ in eq. (14.3) then if $\lambda = 0$ we get ordinary linear regression. If on the other hand λ is large, the error term prefers each coordinate of \mathbf{w} , w_i , to be as small as possible. This is also evident from the expression for the solution eq. (14.4): If we for a moment naively ignore the standardization and assume \mathbf{X} and \mathbf{y} are scalars we get:

$$w^* = \frac{Xy}{X^2 + \lambda},$$

so the larger λ is, the smaller w^* becomes and in the limit $\lambda \rightarrow \infty$ then $w^* = 0$. In fig. 14.1 is shown a small dataset with 9 observations (blue dots) and 10 test data points (green dots) and the solution for three different values of λ . The linear regression model is in this case a 6'th degree polynomial. We see that for the larger λ , since w^* is smaller the fitted polynomial is dragged (biased) towards the x -axis. If on the other hand λ is very small, the polynomial wiggles quite a lot (high variance) as can be expected for a 6-degree polynomial on such a small dataset. The full evolution of the size of each coordinate of the weights w_i^* for many values of λ is shown in fig. 14.2.

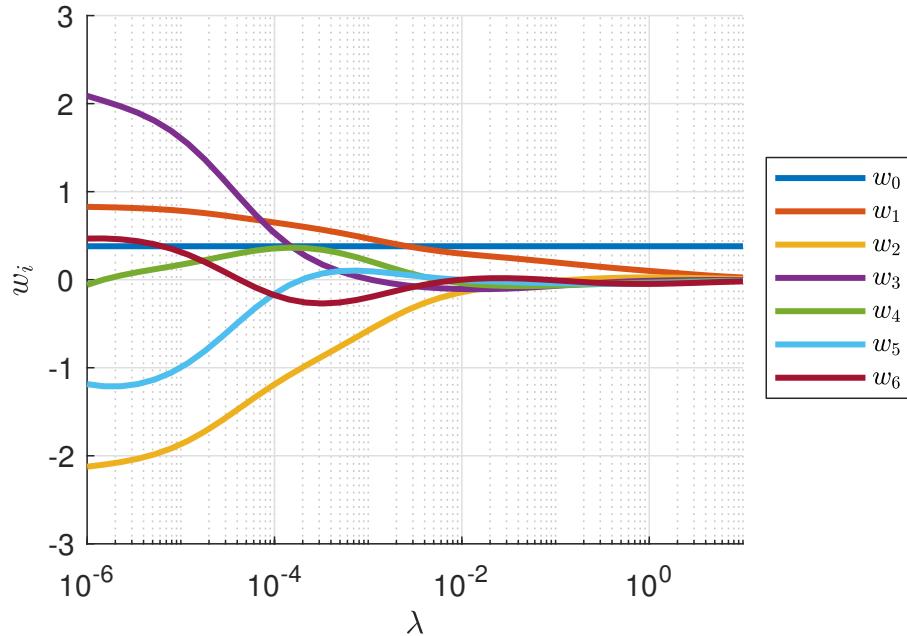


Fig. 14.2. A regularized linear regression model is fitted to the dataset shown in fig. 14.1 and the coordinates of the optimal weights are plotted. When λ is small, the weights are large indicating high variance but low bias. When λ is larger, the weights become smaller indicating lower variance but higher bias in the solutions.

This also holds in general: When the regularization λ is small, the models have high variance and low bias. When λ is large, the models have low variance (they are all dragged towards the x -axis) but high bias. As a rule, varying λ to search for an optimal value of the generalization error will therefore lead to better models. In fig. 14.3 the variable λ is tweaked from a very small value of $\lambda = 10^{-6}$ to a higher value $\lambda = 10^0$ and the training and test error (normalized by the number of observations) of the small dataset in fig. 14.1 displayed. The three particular values plotted in fig. 14.1 are plotted as circles. We see that the training error generally increases as λ increases (after all, for small λ the model will overfit the training data set), however, the test error has an optimum when $\lambda \approx 10^{-2}$. In practice when we search for the optimal value of λ , we test S different values of λ , $\lambda_1, \dots, \lambda_S$ selected beforehand and then compare each of the corresponding linear regression models using cross-validation for model selection.

Other choices of regularization \star

A reader may wonder why we chose the particular L_2 square-loss regularization. An alternative is the L_1 -norm regularization term: $\lambda (\sum_i |w_i|) = \lambda \|\mathbf{w}\|_1$. The advantage of the L_1 regularization term is that it prefers *sparse* solutions where many of the coordinates of w_i becomes equal to zero, as opposed to L_2 regularization where they in general only become *approximately* equal to 0. This is useful when the data set is known to contain many irrelevant attributes we wish to disregard. A disadvantage of pure L_1 regularization is small changes in regularization may mean very different weights are selected.

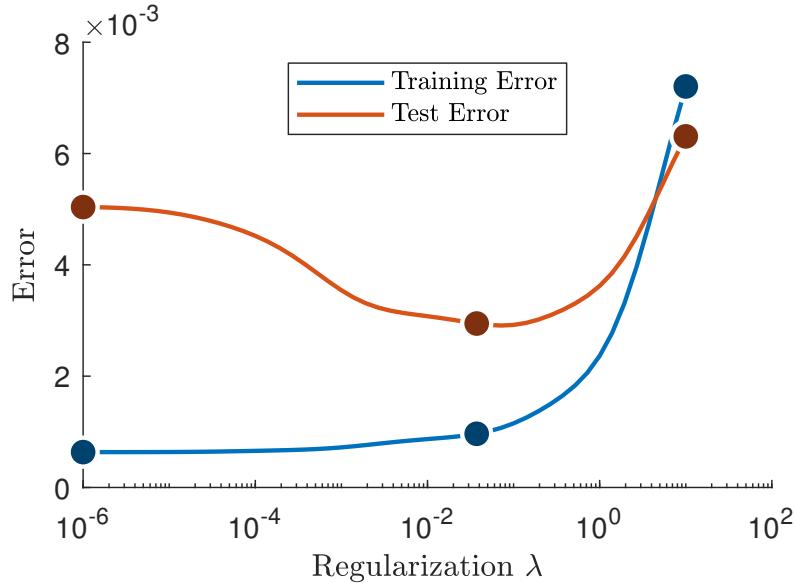


Fig. 14.3. Effect of varying the regularization parameter λ on the training and test error. The colored dots indicate three models shown in fig. 14.1.

Technical note 14.1.1: Why use L_2 regularization?

Regularization, as explained here, is simply adding a factor $\lambda \mathbf{w}^T \mathbf{w}$ to our error which may appear rather arbitrary. It is however possible to give regularization a natural Bayesian interpretation using our general likelihood learning framework discussed in section 6.5. To simplify the discussion, we will assume the bias is treated similar to the other parameters, and assume that just as in our original discussion of linear regression in chapter 8, the likelihood of the data is

$$p(\mathbf{y} | \mathbf{X}, \mathbf{w}) = \prod_{i=1}^N \mathcal{N}(y_i | \mathbf{x}_i^\top \mathbf{w}, \sigma^2).$$

Our discussion then proceeded by assuming the prior term $p(\mathbf{w})$ could be ignored. However, let's make the assumption the prior term cannot be ignored. The simplest case is to assume the prior is normally distributed with diagonal covariance matrix $\delta \mathbf{I}$:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \delta^2 \mathbf{I})$$

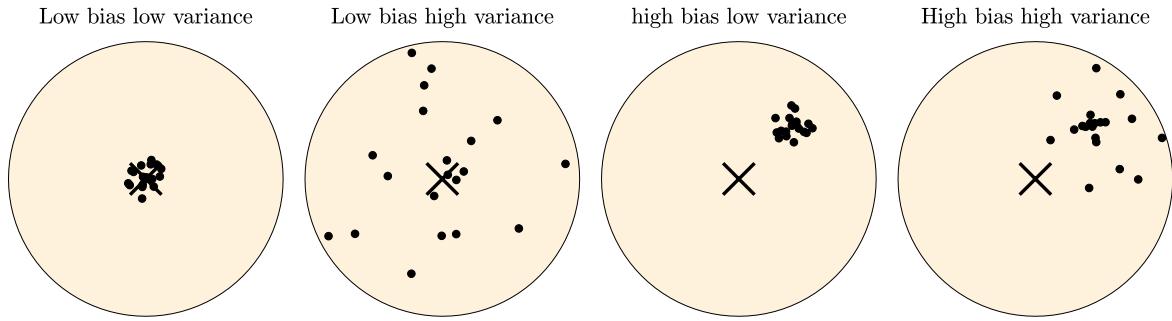
The maximum-likelihood formulation (see eq. (6.46) in summary box 6.5.1) consist of *maximizing*:

$$p(\mathbf{w}) + \sum_{i=1}^N \log p(y_i | \mathbf{x}_i, \mathbf{w}) = -\frac{1}{2\sigma^2} \sum_{i=1}^N \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 - \frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{2\delta^2} \mathbf{w}^T \mathbf{w} - \frac{M}{2} \log(2\pi\delta^2)$$

If we drop constant terms and re-scale the expression by a factor $-2\sigma^2$

$$\sum_{i=1}^N \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \frac{\sigma^2}{\delta^2} \mathbf{w}^T \mathbf{w}$$

Therefore, if we define $\lambda = \frac{\sigma^2}{\delta^2}$, we nearly recover eq. (14.3) aside the standardization and difference in how the bias is treated (this discrepancy can be solved by assuming a flat prior for w_0). Since the key assumption that lead to the particular form of the regularization term was the distribution of the prior, other choices would lead to other forms of regularization.

**Fig. 14.4.** Illustration of bias and variance

14.2 Bias-variance decomposition

In this section, we will analyse the generalization error from a more theoretical perspective using what is known as the *bias-variance decomposition*. Recall bias is how far away from the true mean we are on average and variance measures how spread out our observations are, see fig. 14.4 for an intuitive illustration. It turns out that the generalization error can in general be decomposed into a systematic error known as the bias term and a term depending on how much our trained models vary known as the variance term.

Showing this is not too difficult but requires some math. We will therefore first illustrate the result with a linear regression example and leave the proof as optional reading. Suppose we are in a standard, supervised situation with a square loss where we predict y from observations \mathbf{x} . If \mathcal{D} denotes our training data, a given model learns a function f on the training data to accomplish this task. In fig. 14.5 this is illustrated for two different (random) training data sets and the model \mathcal{M}_2 corresponding to second-degree polynomials. Notice the learned function f depends on the training sets and to keep track of this dependency we will write it as $f_{\mathcal{D}}$.

Suppose we want to know how the generalization error behaves on average. Recall from chapter 10 the generalization error was defined as how well our model performed on a test set on average when trained on the training set \mathcal{D} (see eq. (10.4))

$$\begin{aligned} E_{\mathcal{M}}^{\text{gen}} &= \mathbb{E}_{(\mathbf{x}, \mathbf{y})} [L(\mathbf{y}, \mathbf{f}_{\mathcal{M}}(\mathbf{x}))] \\ &= \int L(\mathbf{y}, \mathbf{f}_{\mathcal{M}}(\mathbf{x})) p(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y}. \end{aligned} \quad (14.5)$$

Since the generalization error E^{gen} depends on the training data set \mathcal{D} , we will indicate this by the notation $E^{\text{gen}}(\mathcal{D})$, and we will in this section consider the expectation of the generalization error over all training data set:

$$\mathbb{E}_{\mathcal{D}} [E^{\text{gen}}] = \int E^{\text{gen}}(\mathcal{D}) p(\mathcal{D}) d\mathcal{D}.$$

The above, i.e., the *averaged generalization error*, is what we in this section consider our true objective estimate of how well our model performs: How well it generalizes based on averaging over all training data sets.

To get insight into the average generalization error let's consider the average behavior of the by now well-known linear regression model when trained on different training sets. In fig. 14.6 the three

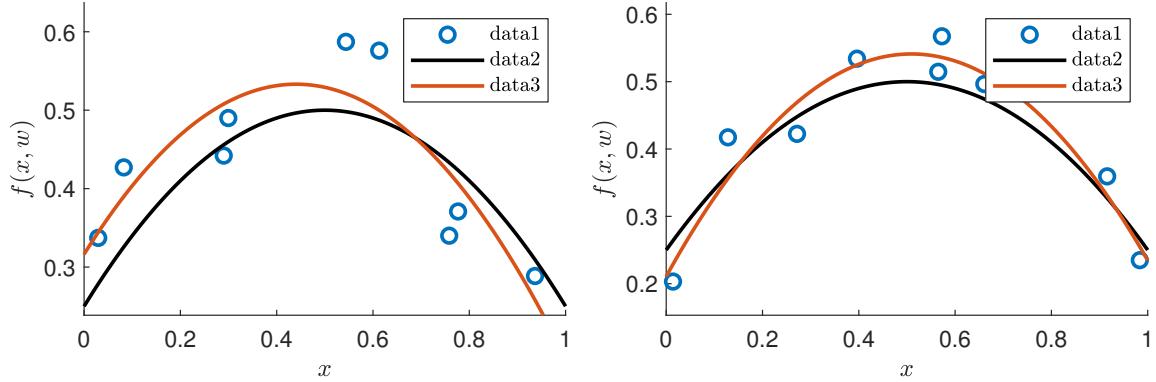


Fig. 14.5. A linear regression model corresponding to a second-order polynomial trained on two different training data sets. The learned model (red line) depends on the training sets. The training set is distributed around the black line.

linear regression models are each trained on 10 different training sets and the prediction curves, $f_{\mathcal{D}}$, are plotted as the thin red lines. Of particular importance will be the average of all these curves shown as the thick red line in fig. 14.6. Formally, this is written as

$$\bar{f}(\mathbf{x}) = \mathbb{E}_{\mathcal{D}} [f_{\mathcal{D}}(\mathbf{x})]$$

The black line is the true average of the training sets, i.e. the training points (which are not shown in fig. 14.6) are distributed around this curve. Formally, it is defined as the average value of y given \mathbf{x} , i.e.:

$$\bar{y}(\mathbf{x}) = \mathbb{E}_{y|\mathbf{x}} [y].$$

Considering fig. 14.6 we can make some general observations. Firstly, the thin red curves (each of the 10 models trained on different training sets) are quite close together in the first two plots, perhaps even the closest together in the first plot: They are said to have low *variance*. Meanwhile, in the third plot the curves are spread out quite a lot because the model is too flexible and we say this model has a high variance. If we turn to the average behaviour of the curves (the thick red line), in the second and third plot the average of all the models is quite similar to the thick black line (the true average of the training data) and we say the curves have a low bias. Meanwhile, the first model, which is too inflexible, has a high bias because the average of the model $\bar{f}(\mathbf{x})$ and the average of the data $\bar{y}(\mathbf{x})$ is quite different.

In the following, we will show these two effects –bias and variance– is all we need to describe the average generalization error for any model.

Derivation of the bias-variance decomposition*

The average generalization error for a training set \mathcal{D} for a square loss is:

$$\mathbb{E}_{\mathcal{D}} [E^{\text{gen}}] = \mathbb{E}_{\mathcal{D}, (\mathbf{x}, y)} \left[(y - f_{\mathcal{D}}(\mathbf{x}))^2 \right],$$

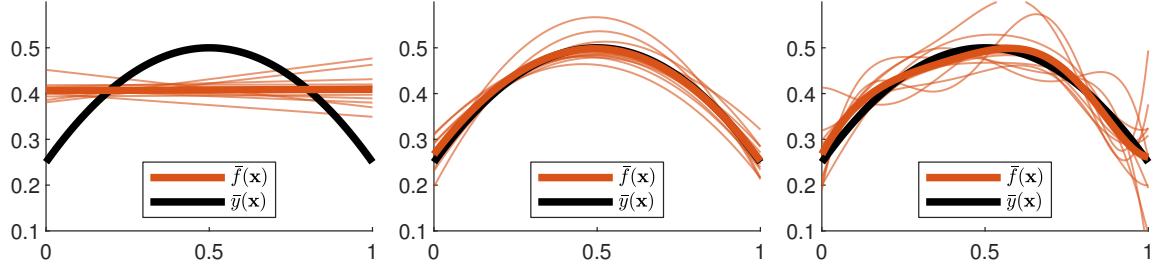


Fig. 14.6. A linear regression model corresponding to a second-order polynomial trained on two different training data sets. The learned models (the thinner red lines) depends on the training sets. The training set is distributed around the black line. The average of all models is shown as the thicker red line.

where the expectation can be written out as $\mathbb{E}_{\mathcal{D},(\mathbf{x},y)} [\cdot] = \int [\cdot] p(\mathbf{x},y, \mathcal{D}) d\mathbf{x} dy d\mathcal{D}$. We first assume \mathbf{x} to be fixed and consider the average:

$$\begin{aligned} & \mathbb{E}_{\mathcal{D},y|\mathbf{x}} [(y - f_{\mathcal{D}}(\mathbf{x}))^2] \\ &= \mathbb{E}_{\mathcal{D},y|\mathbf{x}} [(y - \bar{y}(\mathbf{x}) + \bar{y}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x}))^2] \\ &= \mathbb{E}_{y|\mathbf{x}} [(y - \bar{y}(\mathbf{x}))^2] + \mathbb{E}_{\mathcal{D}} [(\bar{y}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x}))^2] + 2\mathbb{E}_{\mathcal{D},y|\mathbf{x}} [(y - \bar{y}(\mathbf{x})) (\bar{y}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x}))]. \end{aligned}$$

The last term is zero since

$$\mathbb{E}_{\mathcal{D},y|\mathbf{x}} [(y - \bar{y}(\mathbf{x})) (\bar{y}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x}))] = \mathbb{E}_{y|\mathbf{x}} [y - \bar{y}(\mathbf{x})] \mathbb{E}_{\mathcal{D}} [\bar{y}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x})],$$

and $\mathbb{E}_{y|\mathbf{x}} [y - \bar{y}(\mathbf{x})] = 0$. If we look at the second term we can do the same trick once again:

$$\begin{aligned} & \mathbb{E}_{\mathcal{D}} [(\bar{y}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x}))^2] \\ &= \mathbb{E}_{\mathcal{D}} [(\bar{y}(\mathbf{x}) - \bar{f}(\mathbf{x}) + \bar{f}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x}))^2] \\ &= \mathbb{E}_{\mathcal{D}} [(\bar{y}(\mathbf{x}) - \bar{f}(\mathbf{x}))^2] + \mathbb{E}_{\mathcal{D}} [(\bar{f}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x}))^2] + 2\mathbb{E}_{\mathcal{D}} [(\bar{y}(\mathbf{x}) - \bar{f}(\mathbf{x})) (\bar{f}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x}))]. \end{aligned}$$

Again, since $\mathbb{E}_{\mathcal{D}} [\bar{f}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x})] = 0$, the third term is zero in the above. Putting all these things together, we obtain:

$$\mathbb{E}_{\mathcal{D},y|\mathbf{x}} [(y - f_{\mathcal{D}}(\mathbf{x}))^2] \tag{14.6}$$

$$= \mathbb{E}_{y|\mathbf{x}} [(y - \bar{y}(\mathbf{x}))^2] + (\bar{y}(\mathbf{x}) - \bar{f}(\mathbf{x}))^2 + \mathbb{E}_{\mathcal{D}} [(\bar{f}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x}))^2]. \tag{14.7}$$

The last term is simply the variance of $f_{\mathcal{D}}(\mathbf{x})$ computed with respect to \mathbf{x} and the first term too is just the variance of y conditional on \mathbf{x} . Using this the above can be written as:

$$\mathbb{E}_{\mathcal{D},y|\mathbf{x}} [(y - f_{\mathcal{D}}(\mathbf{x}))^2] = \text{Var}_{y|\mathbf{x}} [y] + \text{Var}_{\mathcal{D}} [f_{\mathcal{D}}(\mathbf{x})] + (\bar{y}(\mathbf{x}) - \bar{f}(\mathbf{x}))^2. \tag{14.8}$$

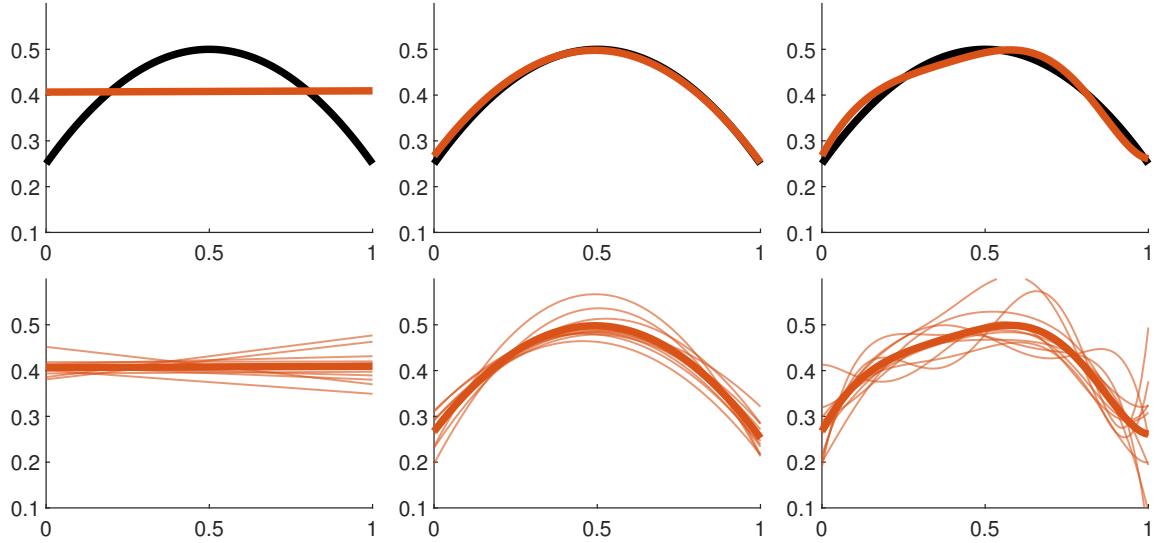


Fig. 14.7. Bias-variance decomposition for the three linear regression models. In the top row is shown the bias term. Namely, how much the average values of the models trained on different random data sets (illustrated with the thick red line) differ from the true mean values of the data illustrated by the black line. In the bottom row is shown the variance term. Namely, how much each model wiggles around the mean of all models.

Taking the expectation with respect to \mathbf{x} and rearranging we finally obtain the result:

$$\mathbb{E}_{\mathcal{D}} [E^{\text{gen}}] = \mathbb{E}_{\mathbf{x}} \left[\text{Var}_{y|\mathbf{x}} [y] + (\bar{y}(\mathbf{x}) - \bar{f}(\mathbf{x}))^2 + \text{Var}_{\mathcal{D}} [f_{\mathcal{D}}(\mathbf{x})] \right]. \quad (14.9)$$

Interpreting the bias-variance decomposition

The last equation in the previous section is known as the *bias-variance* decomposition. The term on the left-hand side of the equality sign is how well we expect the model to generalize to new data: This is *the* objective measure for how well our model performs. The terms on the right-hand side of the equality sign tells us that the error of any model is decomposed into the following three parts

- The first term $\text{Var}_{y|\mathbf{x}} [y]$ is just a constant. It does not depend at all upon our choice of model but simply represents the intrinsic difficulty of the problem. We cannot make this term any larger or smaller by selecting one model over another.
- The second term $(\bar{y}(\mathbf{x}) - \bar{f}(\mathbf{x}))^2$ is the *bias* term. It tells us how much the average values of models trained on different training datasets differ compared to the true mean of the data $\bar{y}(\mathbf{x})$.
- The third term $\text{Var}_{\mathcal{D}} [f_{\mathcal{D}}(\mathbf{x})]$ is the *variance* term. It tells us how much the model wiggles when trained on different sets of training data. That is, when you train the models on N different (random) sets of training data and the models (the prediction curves) are nearly the same this term is small.

To illustrate the variance and bias terms for the linear regression example, in fig. 14.7 we have shown what contributes to the two terms. In the top-row are given the bias terms (the difference between

the models mean values and the data mean values) and in the bottom row are given the variance terms (the spread of the models). The generalization error is the sum of these two contributions, plus the third contribution we cannot do anything about. We can thus see the first model does badly because it has a high bias (but low variance), the third model does also poorly because it has a high variance (but low bias) and the second model does well because both of these terms are low.

When we think about how well different models perform, each model has different values of the bias and variance terms which explains their generalization error. Often it is possible to construct models such that e.g. the bias or variance term is low, but at the expense of a larger value of the other term. This is known as the bias-variance tradeoff. The purpose of regularization in the context of this bias-variance tradeoff is to substantially reduce the variance without introducing too much bias.

Problems

14.1. Question 1: Which one of the following statements pertaining to regression is *incorrect*?

A In regularized least squares regression the aim is to reduce the model's variance without introducing too much bias.

B Linear regression where the inputs are transformed can only model linear relations between the original untransformed inputs and the outputs.

C To investigate what attribute transformations may be relevant to consider it is useful to plot each attribute versus the residuals.

D Forward selection can be used both for regression and classification problems.

E Don't know.

Neural Networks

Artificial neural networks (ANNs) were originally invented as mathematical models of the information processing by neurons McCulloch and Pitts [1943]. Today it is clear there are important differences between ANNs and biological neurons, however, the basic structure is very similar. In this chapter, we will consider the most simple forms of ANNs, the feedforward network, which is nevertheless an extremely powerful approach to both classification and regression.

15.1 The feedforward neural network

An average adult human brain consists of about 86 billion neurons. Each neuron (a neuron is simply a special type of cell) is connected to up to 10 000 other neurons by synapses. Each neuron has an electric activity (for simplicity this can be considered as a real number) which depends on how many of the neurons connected to the neuron are active. That is, if sufficiently many of the neurons connected to a given neuron becomes active, the neuron itself becomes active and may then in turn excite other neurons connected to it. It is surprising how such a simple mechanism can give rise to interesting information processing and how intelligence arise from neuronal activity remains the greatest open problem in neuroscience.

15.1.1 Artificial neural networks

In ANNs we consider a set of information processing units also called *neurons* and each neuron is connected to other neurons by weighted connections. The neurons are organized in layers with connections from one layer feeding into the next. In this way information is processed sequentially (layer-wise) in the network: First, the input pattern (which is just a vector $\mathbf{x} = (x_1, \dots, x_M)$) is presented to the *input layer* such that neuron i in the input layer is given an activation equal to x_i . The activation is then propagated to one or several *hidden layers* and finally to the *output layer* consisting of one or more neurons corresponding to the coordinates of the output vector.

This process is known as a *forward pass* through the network. In fig. 15.1¹ is illustrated a simple neural network with one hidden layer. The input layer consists of three neurons, the hidden layer of four neurons and the output of a single neuron.

¹ By Glosser.ca [CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0>)], via Wikimedia Commons

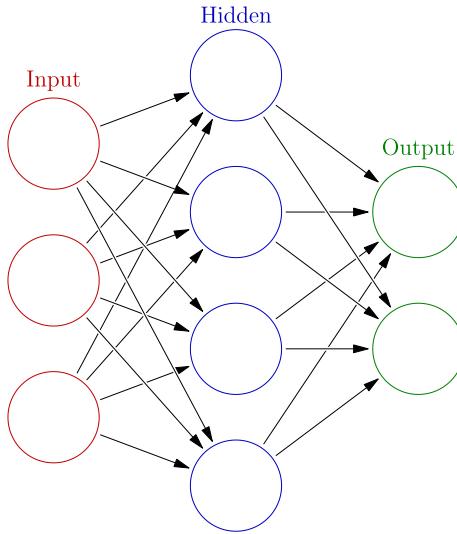


Fig. 15.1. Simple artificial neural network (ANN) consisting of three input units in the input layer, a single hidden layer with four hidden units and two output units in the output layer. This neural network would implement a mapping $f : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ and would be suitable for regression or classification in the case of two output variables.

If M is the number of neurons in the input layer and D is the number of neurons in the output layer a neural network is then simply a mapping:

$$f : \mathbb{R}^M \rightarrow \mathbb{R}^D$$

which maps from \mathbf{x} to \mathbf{y} : $\mathbf{y} = f(\mathbf{x})$; thus the neural network is useful for solving a (multi-dimensional) regression or classification problem.

15.1.2 The forward pass in details

Recall the basic linear regression model in which the output y is predicted from the rule

$$f(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^M x_i w_i + w_0$$

If we let

$$\tilde{\mathbf{x}} = [1 \ x_1 \ x_2 \ \dots \ x_M]^T$$

we can write this in a more condensed form

$$f(\mathbf{x}, \mathbf{w}) = \tilde{\mathbf{x}}^T \mathbf{w}.$$

The forward pass in a neural network now proceeds as follows for vector \mathbf{x} :

- Each neuron i in the input layer is initialized to have activity x_i .

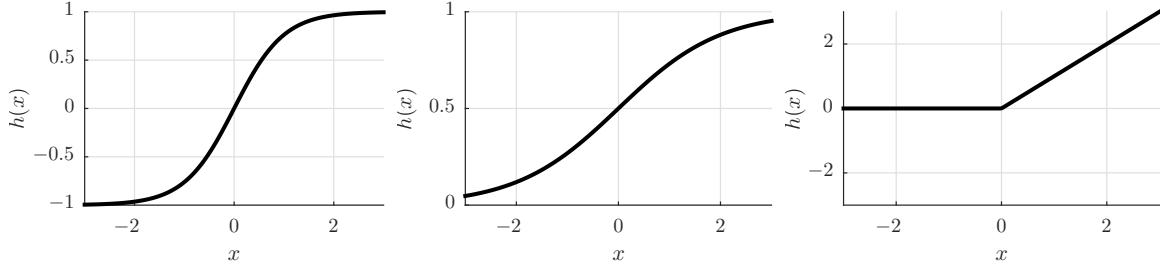


Fig. 15.2. Different choices of activation function. (Left:) hyperbolic tangent: $h(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, (middle:) logistic sigmoid $h(x) = (1 + e^{-x})^{-1}$ and (right:) rectified linear unit: $h(x) = 0$ if $x < 0$ and otherwise $h(x) = x$. The basic information-processing ability is similar for all activation functions, but during training the choice may be important as the gradients will differ in magnitude. For this reason it is also common to apply a fixed transformations such as $h(x) = b \tanh(ax)$.

- Neuron j in the hidden layer is given activity $a_j^{(1)} = \tilde{\mathbf{x}}^T \mathbf{w}_j^{(1)}$. Notice this is just a real number.
- Each of the H hidden unit are transformed using a nonlinear *activation function* h to give $z_j^{(1)} = h(a_j^{(1)})$. We then define

$$\mathbf{z}^{(1)} = [z_1^{(1)} \ z_2^{(1)} \ \dots \ z_H^{(1)}]^T$$

- Output neuron k is given an activation of $a_k^{(2)} = (\tilde{\mathbf{z}}^{(1)})^T \mathbf{w}_k^{(2)}$
- The output neurons are transformed using a function $h^{(2)}$ to give $z_j^{(2)} = h^{(2)}(a_k^{(2)})$
- The value of the neural network (output) is simply

$$\mathbf{f}(\mathbf{x}) = [z_1^{(2)} \ z_2^{(2)} \ \dots \ z_D^{(2)}]^T.$$

These steps may look daunting and it is perhaps useful to consider what they concretely mean. Suppose we collect the various weight-terms into matrices and define

$$W^{(1)} = [\mathbf{w}_1^{(1)} \ \mathbf{w}_2^{(1)} \ \dots \ \mathbf{w}_H^{(1)}] \quad \text{and} \quad W^{(2)} = [\mathbf{w}_1^{(2)} \ \mathbf{w}_2^{(2)} \ \dots \ \mathbf{w}_D^{(2)}].$$

The activation of the k th output neuron is simply:

$$f_k(\mathbf{x}, \mathbf{w}) = h^{(2)} \left(\sum_{j=1}^H W_{kj}^{(2)} z_j^{(1)} \right) \tag{15.1}$$

$$= h^{(2)} \left(\sum_{j=1}^H W_{kj}^{(2)} h^{(1)} \left(\tilde{\mathbf{x}}^T \mathbf{w}_j^{(1)} \right) \right). \tag{15.2}$$

The activation function $h^{(1)}$ of the hidden units could be chosen as the hyperbolic tangent

$$h^{(1)}(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

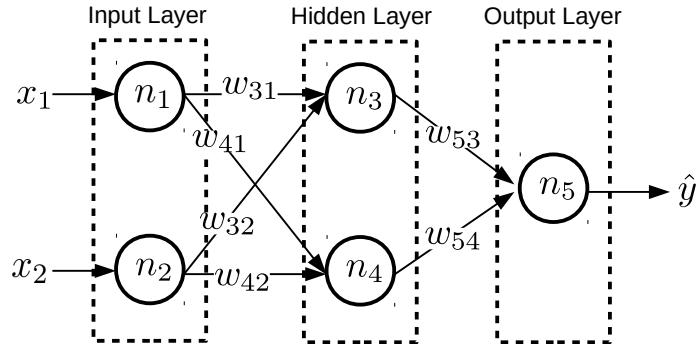


Fig. 15.3. Simple neural network of 6 weights and with one hidden layer with 2 neurons.

Many choices of activation function can be found in the literature all with roughly the same basic information-processing abilities but with different characteristics under training. A few common examples can be found in fig. 15.2.

Example 15.1.1: Forward pass of a neural network

Consider the feedforward neural network shown in fig. 15.3. The network has no bias weights. Suppose the weights of the neural network after training are

$$\begin{aligned} w_{31} &= 0.05, & w_{41} &= 0, & w_{32} &= 0.1, \\ w_{42} &= -0.05, & w_{53} &= 0.1, & w_{54} &= -10 \end{aligned}$$

and the activation functions of the neurons in the hidden layer and output layer, i.e., \$n_3\$, \$n_4\$, and \$n_5\$ all are given by the following leaky rectified linear unit

$$h(x) = \begin{cases} x & \text{if } x > 0 \\ \frac{1}{10}x & \text{otherwise.} \end{cases}$$

Suppose the network is evaluated on input \$x_1 = 0.5\$, \$x_2 = 1\$, the output is then computed by first evaluating the hidden layer:

$$\begin{aligned} x_3 &= h(x_1 0.05 + x_2 0.1) = h(1/8) = \frac{1}{8}, \\ x_4 &= h(x_1 0 + x_2 (-0.05)) = h(-1/20) = \frac{-1}{200}. \end{aligned}$$

and then the output layer:

$$x_5 = h(x_3 0.1 + x_4 (-10)) = h(1/16) = 1/16.$$

The general L -layer neural network

The neural network discussed in the previous section is said to have two layers (the hidden layer and the output layer; the input layer is not counted). The construction can be immediately generalized to L layers by simply repeating the two steps in the hidden layer. Written in a more condensed fashion we proceed as follow:

- We define $\mathbf{z}^{(0)} = \mathbf{x}$ as the input activation
- For each layer $l = 1, \dots, L$ set $\mathbf{z}^{(l)} = h^{(l)}((\mathbf{W}^{(l)})^T \tilde{\mathbf{z}}^{(l-1)})$.
- Return as output $\mathbf{f}(\mathbf{x}, \mathbf{w}) = \mathbf{z}^{(L)}$.

In general each hidden unit also contains a bias term corresponding to an additional input to each neuron of 1 with the corresponding weight term accounting for the bias (i.e., just as we appended a column of 1 to our input data \mathbf{x} in regression to form $\tilde{\mathbf{x}}$ we add a bias term as input to the neuron of the l 'th layer using as input to the neuron $\tilde{\mathbf{z}}^{(l-1)} = [1 \ \mathbf{z}^{(l-1)}]$).

15.2 Training neural networks

Regardless if one choose a two-layer neural network with a single hidden layer, or a general L layer neural network, one simply obtains a parametric function $\mathbf{f}(\mathbf{x}, \mathbf{w})$. For a fixed \mathbf{w} this function maps \mathbf{x} values to \mathbf{y} values such that the vector \mathbf{w} contains all the weight-matrices in the network $\mathbf{w} = (\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \dots)$.

We are thus faced with a standard supervised learning problem where we are given instances of observations $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ and corresponding targets $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N$, and our solution will be very similar to what we already considered for linear and logistic regression: Since the neural network cannot be expected to perfectly map from \mathbf{x}_i to the corresponding \mathbf{y}_i , we will assume \mathbf{y}_i is normally distributed around the prediction of the neural network. If \mathbf{y} has dimension K , the probability density of observation \mathbf{y}_i is then:

$$p(\mathbf{y}_i | \mathbf{x}_i, \mathbf{w}) = \mathcal{N}(\mathbf{y}_i | \mathbf{f}(\mathbf{x}_i, \mathbf{w}), I\sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{K}{2}}} e^{-\frac{\|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i, \mathbf{w})\|^2}{2\sigma^2}}. \quad (15.3)$$

Applying the maximum likelihood framework from section 6.5, and for generality including a regularization term as in chapter 14, we see once more that the value of \mathbf{w} that maximize $p(\mathbf{w} | \mathbf{X}, \mathbf{y})$ can be found by *minimizing* the cost function E defined as

$$\begin{aligned} \mathbf{w}^* &= \arg \min_{\mathbf{w}} E_{\lambda}(\mathbf{w}) \\ E_{\lambda}(\mathbf{w}) &= \frac{1}{N} \sum_{i=1}^N \|\mathbf{f}(\mathbf{x}_i, \mathbf{w}) - \mathbf{y}_i\|^2 + \lambda \mathbf{w}^T \mathbf{w} \end{aligned} \quad (15.4)$$

where λ is the regularization strength we have to specify. Training a neural network is therefore reduced to searching for a minimum of the function E . In arriving at this formulation we considered the simple feed-forward neural network for regression, however, we stress that in nearly all applications of neural networks, whether they are used to translate from French to English, recognize images or play Atari videogames, depend on specifying an appropriate function E and searching for the minimizing \mathbf{w}^* . Thus, headway on solving the problem eq. (15.4) can be used in a variety of contexts.

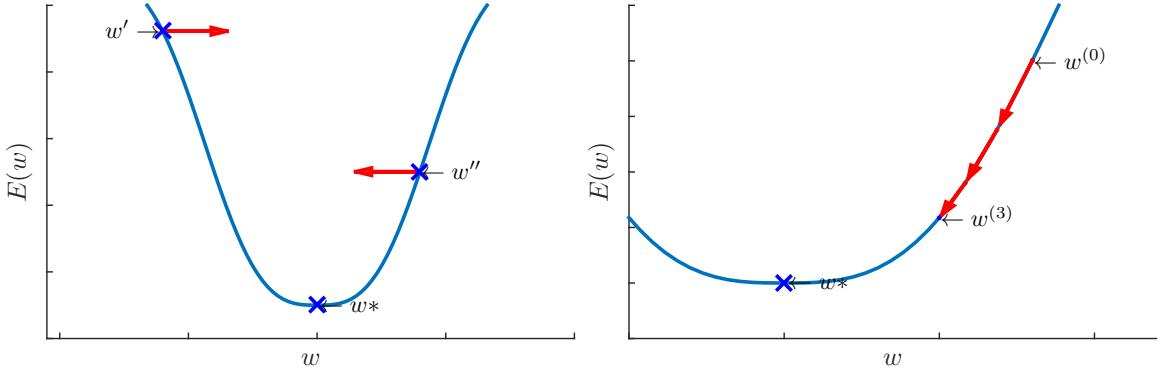


Fig. 15.4. (left:) Value of error function in a one-dimensional example. Weights at w' should move right and weights at w'' should move left in order to approach the minimum point w^* of $E(w)$. (right:) Gradient descent algorithm applied for three steps starting at $w^{(0)}$

15.2.1 Gradient Descent*

The problem is that it is impossible to analytically solve for \mathbf{w}^* . Instead, the following iterative algorithm is proposed:

- Start from an initial guess at \mathbf{w}^* , $\mathbf{w}^{(0)}$.
- At step t , modify $\mathbf{w}^{(t-1)}$ by a small amount $d\mathbf{w}$ to produce a better guess $\mathbf{w}^{(t)}$:

$$\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} + d\mathbf{w},$$

where we leave it for later how to compute $d\mathbf{w}$.

- Do this for a large number T of iterations to produce (hopefully!) better and better guesses, i.e., $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \dots, \mathbf{w}^{(T)}$.

After T iterations $\mathbf{w}^{(T)}$ is then used as the “best” available guess of \mathbf{w}^* . This algorithm is very simple if not for the unspecified step 2. To solve this we will use *gradient descent* which only requires E to be differentiable.

The one-dimensional case

To introduce gradient descent, suppose \mathbf{w} is one dimensional (the neural network only contains a single “weight”) and suppose E as a function of w looks like fig. 15.4. If we suppose at step t of the algorithm $w^{(t-1)}$ is located at position w' in the figure, a “better” guess at w^* can be obtained by moving $w^{(t-1)}$ slightly to the right by a positive amount dw' :

$$w^{(t)} = w' + dw', \quad dw' > 0,$$

on the other hand if $w^{(t-1)}$ equals w'' then a “better” guess at w^* can be obtained by moving $w^{(t-1)}$ slightly to the left by a negative amount dw''

$$w^{(t)} = w'' + dw'', \quad dw'' < 0.$$

This obviously leave the question of how we compute dw' or dw'' . Notice, if we compute the gradient of E at w' or w'' we have:

$$\frac{dE}{dw}(w') < 0 \quad \text{and} \quad \frac{dE}{dw}(w'') > 0.$$

Thus, if we let $dw = -\epsilon \frac{dE}{dw}(w^{(t-1)})$ be the gradient of E evaluated at $w^{(t-1)}$ multiplied by $\epsilon > 0$ which is called the *learning rate* of the method (usually set somewhere in the interval $[0, 1]$ for instance $\epsilon = 1/5$), we can consider the simple update rule:

$$\theta^{(t)} = \theta^{(t-1)} + dw.$$

It is easy to check this indeed works – in fig. 15.4 is plotted $w^{(t)}$ and $E(w^{(t)})$ as a function of t when this rule is applied for 3 steps. Notice that the method “slows down” when $w^{(t)}$ is closer to w^* as the magnitude of the gradient $\frac{dE}{dw}(w^{(t-1)})$ becomes smaller; this is useful to prevent overshooting the target, however, it also potentially slows down the algorithm.

So why does this work? We can formalize the above argument as follows. Suppose for simplicity we define $w' = w^{(t)}$. Then we can Taylor expand² E around w' to obtain:

$$E(w' + dw) \approx E(w') + dw \frac{dE}{dw}(w') \quad (15.5)$$

$$\approx E(w') + dwg \quad (15.6)$$

$$\text{where } g = \frac{dE}{dw}(w'). \quad (15.7)$$

Thus, if we select $dw = -\epsilon g$ in the above we get:

$$E(w' + dw) = E(w') + dwg = E(w') - \epsilon g^2.$$

In other words we are guaranteed that if we let $w^{(t)}$ be equal to $w' + dw = w^{(t-1)} - \epsilon g$ then

$$E(w^{(t)}) \leq E(w^{(t-1)}).$$

This decreases the error with roughly an amount ϵg^2 (this also explains why the error changes less and less in fig. 15.4). In this view, it is surprising why we don't select ϵ to be very large – perhaps $\epsilon = 1000$. The reason is that the Taylor expansion is only accurate for *small* values of dw , thus we can't trust the above result when ϵ is very large.

Multiple dimensions

We have spent some time on the one-dimensional case, however, the multi-dimensional case can be treated very similar. In this case we can consider a small, perturbation dw of w' . The multivariate Taylor expansion now gives:

$$E(\mathbf{w}' + \mathbf{dw}) \approx E(\mathbf{w}') + \mathbf{dw}^T \mathbf{g} \quad (15.8)$$

$$\approx E(\mathbf{w}') + \mathbf{dw}^T \mathbf{g} \quad (15.9)$$

$$\text{where } \mathbf{g} = \nabla E(\mathbf{w}') \quad (15.10)$$

² See also https://en.wikipedia.org/wiki/Taylor_series and appendix A.

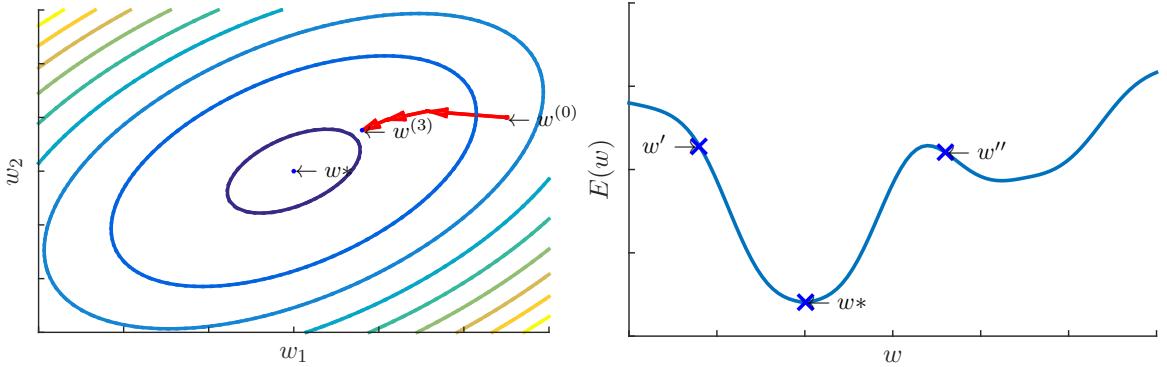


Fig. 15.5. (left:) Value of error function in a two dimensional example as a contour plot along with three steps of the gradient descent algorithm. Notice the step size slows down when moving towards the minimum. (right:) An example with two local minima. If the gradient descent method is initialized at w' it will converge to the global minima w^* , whereas if it is initialized at w'' it will converge to a local minima at the bottom of the right-most valley.

The multi-dimensional Taylor expansion is briefly reviewed in appendix A. Thus, if we select $d\mathbf{w} = -\epsilon \mathbf{g}$ we again get

$$E(\mathbf{w}^{(t)}) = E(\mathbf{w}^{(t-1)} + d\mathbf{w}) \approx E(\mathbf{w}^{(t-1)}) - \epsilon \|\mathbf{g}\|^2 \leq E(\mathbf{w}^{(t-1)}),$$

which again is seen to decrease the error assuming the Taylor expansion is fairly accurate. This allows us to define the Gradient-descent algorithm as:

- Start from an initial guess at $\mathbf{w}^*, \mathbf{w}^{(0)}$
- For each $t = 1, \dots, T$, compute the divergence $\mathbf{g}^{(t-1)} = \nabla E(\mathbf{w}^{(t-1)})$
- Compute $\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \epsilon \mathbf{g}$.
- Do this for a large number T of iterations to produce a sequence of (hopefully!) better and better guesses of \mathbf{w}^* : $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \dots, \mathbf{w}^{(T)}$.

In fig. 15.5 we have illustrated how \mathbf{w} is updated for three iterations in an example where \mathbf{w} is two-dimensional.

Training neural networks in practice

Gradient descent is the prototypical training algorithm for neural networks. Most advanced applications of neural networks use either plain gradient descent, or gradient descent with very simple modifications. A serious omission of the preceding discussion is how to compute the gradient \mathbf{g} efficiently. If we consider the i 'th coordinate of \mathbf{g} , g_i , this can be computed as:

$$g_i = \frac{\partial E(\mathbf{w})}{\partial w_i} \quad (15.11)$$

$$= \frac{\partial}{\partial w_i} \left(\frac{1}{2} \sum_{i=1}^N \|f(\mathbf{x}_i, \mathbf{w}) - \mathbf{y}_i\|^2 + \frac{1}{2} \lambda \mathbf{w}^T \mathbf{w} \right) \quad (15.12)$$

$$= \sum_{i=1}^N \left[\sum_{k=1}^D (f_k(\mathbf{x}_i, \mathbf{w}) - y_{ik}) \frac{\partial f_k(\mathbf{x}_i, \mathbf{w})}{\partial w_i} \right] + \lambda w_i. \quad (15.13)$$

It should be stressed these computations are *in principle* just simply algebra: computing the derivative of a function with respect to a single variable w_i . In practice there is a simple trick for how to organize the derivatives layer-wise to re-use computations which can lead to dramatic speedup compared to an naive computation, the resulting algorithm, which compute the same derivative but in an intelligent manner, is known as *back-propagation*. A further issue which should be mentioned is when E has different local minima. In fig. 15.5 is shown a function E with two local minima. If $w^{(0)}$ is initially selected to be at either w' or w'' it will find different solutions as indicated by the arrows and no amount of training will cause a move from the suboptimal solution (the first valley) to the optimal solution (the second valley). This is a difficulty of considerable practical interest as in higher dimensions there will typically be many local minima and so the solution $\mathbf{w}^{(T)}$, as well as the training error $E(\mathbf{w}^{(T)})$, will depend on how the model is initialized $\mathbf{w}^{(0)}$ as well as other parameters of the training.

15.3 Neural networks for classification

Making a neural network suitable for regression useful for classification is very similar to how we changed linear regression into logistic regression by the use of the Bernoulli distribution. As neural networks are nearly always applied to situations with multiple classes, the multi-class setting is the more relevant, however for completeness, and a warm up exercise, we have included the binary classification setting as a special case.

15.3.1 Neural networks for binary classification

In the case of a *binary* classification problem, where $y = 0$ and $y = 1$, the procedure is entirely similar to how we derived the logistic regression model using the re-parameterization trick in section 5.4.3. Specifically, we assume:

$$p(y|\mathbf{x}, \mathbf{w}) = \text{Bernouilli}(y|\hat{y}) = \hat{y}^y (1 - \hat{y})^{1-y}, \quad \hat{y} = \sigma(f(\mathbf{x}, \mathbf{w})) \quad (15.14)$$

Using this probability density in place of eq. (15.3), the cost function to be minimized becomes:

$$E_\lambda(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] + \lambda \mathbf{w}^T \mathbf{w}, \quad \hat{y}_i = f(\mathbf{x}_i, \mathbf{w}). \quad (15.15)$$

As usual, the minimization is done using gradient descend. Note the particular case where the neural network network is linear (i.e., the activation function is just the identity function), we have $f(\mathbf{x}, \mathbf{w}) = \tilde{\mathbf{x}}^T \mathbf{w}$ and the neural network is simply implementing standard logistic regression.

15.3.2 Neural networks for multi-class classification

Suppose \mathbf{y} corresponds to a classification problem with C classes $1, 2, \dots, C$. As an example, suppose $C = 3$ corresponding to “dog”, “cat”, and “cow”.

We will assume \mathbf{y} is one-of- K encoded in the usual manner:

$$\mathbf{y} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ \vdots \end{bmatrix},$$

implying that the first observation is in class 2 (a cat), the second and third observations are in class 1 (a dog) and the fourth observation is in class 3, a cow. As indicated, this is one-of- K encoded in the matrix \mathbf{y} such that $y_{ik} = 1$ if observation i is in class k and otherwise $y_{ik} = 0$. Notice $\sum_{k=1}^C y_{ik} = 1$ because each observation is in exactly one class.

What we need in order to apply the familiar maximum-likelihood machinery of section 6.5 is a way to express the probability

$$p(y_i = [y_{i1} \ y_{i2} \ \cdots \ y_{iC}] | \mathbf{x}_i, \mathbf{w})$$

To do this, we will use the parameter transformation trick previously discussed in section 5.4.3 applied to the categorical distribution eq. (5.28). Specifically, we will assume $\mathbf{f}(\mathbf{x}, \mathbf{w})$ outputs a C -dimensional vector, and apply the softmax function to transform this into a probability vector:

$$[\hat{y}_{i1} \ \hat{y}_{i2} \ \cdots \ \hat{y}_{iC}] = \text{softmax}(\mathbf{f}(\mathbf{x}_i, \mathbf{w})) \quad (15.16)$$

where $f_k(\mathbf{x}, \mathbf{w})$ is the value of the k 'th output neural of the neural network. For an example of how the softmax function works, see Example 15.3.1. The probability of a given observation can then be expressed using the categorical distribution eq. (5.28)

$$p(\mathbf{y}_i | \mathbf{x}_i, \mathbf{w}) = \text{Catagorical}(\mathbf{y}_i | \hat{\mathbf{y}}_i) = \prod_{k=1}^C \hat{y}_{ik}^{y_{ik}} \quad (15.17)$$

Applying the maximum-likelihood framework to this cost function, and including a regularization term, we obtain the multi-class equivalent of eq. (15.15)

$$E_\lambda(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N \left[\sum_{k=1}^C y_{ik} \log \hat{y}_{ik} \right] + \lambda \mathbf{w}^\top \mathbf{w}. \quad (15.18)$$

where: $\hat{y}_{ik} = \frac{e^{f_k(\mathbf{x}_i, \mathbf{w})}}{\sum_{c=1}^C e^{f_c(\mathbf{x}_i, \mathbf{w})}}$

As usual, the optimal weights should be found using gradient descend.

Example 15.3.1: Softmax function

Let's consider a concrete example. Suppose the output of a neural network is $f_1 = 2$, $f_2 = 1$ and $f_3 = -1$. We then have that

$$e^{f_1} \approx 7.39, \quad e^{f_2} \approx 2.72, \quad e^{f_3} \approx 0.37$$

and so

$$\hat{y} = \text{softmax}(\mathbf{f}) \approx \left[\frac{7.39}{10.5} \quad \frac{2.72}{10.5} \quad \frac{0.37}{10.5} \right] = [0.7 \quad 0.26 \quad 0.04].$$

Therefore, the neural network indicate this observation should be classified as belonging to class 1 with probability 0.7.

15.3.3 Multinomial regression

Since logistic regression corresponds to a linear neural network with no activation function, it should be apparent the multi-class neural network allows us to extend logistic regression to the multi-class setting.

One way to accomplish this is to simply replace $\mathbf{f}(\mathbf{x}, \mathbf{w})$ in eq. (15.18) with a linear function; while this is certainly a valid way to proceed, there is one slightly annoying side-effect. Recall from section 15.3.1 that in the case where we applied neural networks to a two-class classification task, the neural network had a single output neuron. However, in the multi-class setting considered in section 15.3.2, the neural network had as many outputs C as there was classes. This means that this approach to multi-class regression would not directly generalize the binary classification case. To get around this, it is customary to implement *linear* multi-class classification using the (modified) softmax with $C - 1$ inputs which we encountered in eq. (5.31). Specifically, assume $\tilde{\mathbf{X}}$ is our dataset transformed in the usual manner by pre-fixing it with 1, and $\tilde{\mathbf{X}}$ has dimensions $N \times M$. We then have:

$$\mathbf{f}(\mathbf{x}_i, \mathbf{W}) = \mathbf{W}\tilde{\mathbf{x}}_i$$

where \mathbf{W} is a general $C - 1 \times M$ -dimensional matrix of the form

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \vdots \\ \mathbf{w}_{C-1}^\top \end{bmatrix}$$

and each \mathbf{w}_k is a $M \times 1$ vector of weights. We then define the objective using the modified softmax eq. (5.31)

$$E(\mathbf{W}) = -\sum_{i=1}^N \left[\sum_{k=1}^C y_{ik} \log \tilde{y}_{ik} \right] + \lambda \mathbf{w}^\top \mathbf{w}, \quad \text{where: } \tilde{y}_{ik} = \begin{cases} \frac{e^{\mathbf{w}_k^\top \mathbf{x}_i}}{1 + \sum_{c=1}^{C-1} e^{\mathbf{w}_c^\top \mathbf{x}_i}} & \text{if } k \leq C - 1 \\ \frac{1}{1 + \sum_{c=1}^{C-1} e^{\mathbf{w}_c^\top \mathbf{x}_i}} & \text{if } k = C. \end{cases}$$

This raises the obvious question why we didn't use this parameterization of the softmax (which, after all, contains fewer parameters) for the multi-class neural network. One answer is that the alternative parameterization creates an asymmetry between the classes, in that class $C = 1$ is

special. This does not matter as much the simple multinomial regression model which provides more robust parameter estimates, and where it is often considered important to be able to interpret the parameters. However, for neural networks, nobody expects to interpret the parameters anyway, and the asymmetry is considered to be undesirable.

15.3.4 Flexibility and cross-validation

The strength of neural networks derives from their great flexibility. If we consider the sigmoid activation function, the first layer of the neural network can be considered as performing as many logistic regressions as there are internal neurons; to draw a parallel to the decision tree, each neuron in the first hidden layer corresponds to asking one “question” about the input observation but with the added flexibility that the output can be graduated (rather than binary) and will involve a combination of features rather than asking if one feature is greater than another. However it is what happens at the subsequent layers that really sets neural networks aside from decision trees: A decision tree would use the output of a *single* question to ask further questions, however a neural network *combines* the output of many other questions. It is this ability that allows neural networks, especially deep neural networks (i.e. neural networks with several/many hidden layers), to be extremely flexible.

The downside of this flexibility is that neural networks are prone to overfitting the data and it is therefore important to use cross-validation in conjunction with neural network training. Neural networks provide many knobs to limit overfitting, most importantly the regularization parameters λ which should be tuned in most settings. In addition to λ , it is worth experimenting with other parameters in the neural network, for instance the number of hidden layers, the number of units in each hidden layer and the choice of activation function. Starting with the simplest settings (for instance a single hidden layer), it is important to tune the parameters using cross-validation and use two-layer cross-validation to estimate the generalization error in a fair manner as discussed in chapter 10.

15.4 Advanced topics★

In this section we will briefly sketch upon some advanced topics of neural network training

15.4.1 Mini-batching

Gradient descent requires computing the divergence of the error $\nabla E(\mathbf{w})$ which in turn requires iterating over all observations in the data set. If the data set contains millions of images (or billions of words) this would be completely infeasible. Mini-batching is a simple yet very widely used approach to overcome this problem. In mini-batching with a batch size of B the observations in the data set is divided into $m = \frac{N}{B}$ smaller data sets $\mathcal{D}_1, \dots, \mathcal{D}_m$ each containing B observations. Instead of using the gradient:

$$\mathbf{g} = \nabla E(\mathbf{w}) = \nabla \left(\sum_{i=1}^N \|\mathbf{f}(\mathbf{x}_i, \mathbf{w}) - \mathbf{y}_i\|^2 \right)$$

we use the approximate gradients computed for the observations in each batch k :

$$\mathbf{g}_k = \nabla \tilde{E}(\mathbf{w}) = \frac{N}{B} \nabla \left(\sum_{i \in \mathcal{D}_k} \|\mathbf{f}(\mathbf{x}_i, \mathbf{w}) - \mathbf{y}_i\|^2 \right).$$

The gradient-descent method is thus simply

- Start at $\mathbf{w}^{(0)}$
- For each iteration t :
- For each batch $k = 1, \dots, m$:
- Update $\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \epsilon \mathbf{g}_k^{(t)}$

In realistic applications, we might have that N range from thousands to billions whereas B is usually selected at around 100 to 1000. So why does this work? From a theoretical point of view, we want the learning rate ϵ to be as high as possible. However the neural network function is highly non-linear meaning that when the weights are changed even just slightly by $-\epsilon g$ the local Taylor expansion becomes inaccurate and we have to re-compute the gradients. This implies we have to select ϵ fairly small for the gradient descent method to work.

In mini-batching, we replace the true gradient \mathbf{g} at a point $\mathbf{w}^{(t)}$ with an approximate gradient \mathbf{g}_k . Even though this (as a rule) introduces more uncertainty in the algorithm, this uncertainty is relatively small comparable to the uncertainty already present due to the Taylor expansion not being very exact. And since computing the approximate gradient is extremely inexpensive (scales with B and not N) taking *many* smaller steps in mini-batching becomes better than taking one step in ordinary gradient descent which is only slightly more exact than the smaller steps in mini-batching.

15.4.2 Convolutional neural networks

Suppose we wish to apply a neural network to classify semi large (for instance 999×999) images. If we use 1000 neurons in the first hidden layer, the first layer of the neural network alone would contain $999 \times 999 \times 1000 \approx 10^9$ weights (here we haven't included the bias term for each neuron, which would add 1000 additional parameters). Not only is this a considerable computational burden, it is doubtful we have enough images to tune this many parameters in a meaningful manner. A way to significantly cut down on the computational cost is using convolutions which can be sketched as follows: Suppose we consider a very small neural network with no hidden layer which takes an 11×11 input image and maps it onto a single neuron. We can then "translate" this small neural network over the entire image by moving it in *strides* of $F = 4$. That is, if we let \mathbf{A} be the matrix representing the image, we first apply the neural network to pixels $\mathbf{A}_{[1:11] \times [1:11]}$, then $\mathbf{A}_{[5:16] \times [1:11]}$, then $\mathbf{A}_{[9:20] \times [1:11]}$ and so on in both the horizontal and vertical direction until we apply the neural network to $\mathbf{A}_{[989:999] \times [989:999]}$.

If we keep track of the output of the small neural network over all these patches, this leads to a new "hidden layer" of dimensions 247×247 where $247 = \frac{999-11}{F}$, however only about $121 = 11^2$ weights were used to produce this output. Including D such convolutional filters we obtain a hidden layer of dimensions $247 \times 247 \times D$ using only about $121 \times D = 11^2 \times D$ weights. The process can (and should) be made more elaborate by using several such convolutional layers to allow greater flexibility and the process is typically repeated on the second hidden layer to produce an even smaller set of neurons, however these details need not concern us at this stage: The important point is that the same set of weights is "re-used" over the entire image which both cuts down on the number of weights and allow each weight to be trained using much more data. At some point the number of neurons becomes manageable and the neural network can proceed using one or more

fully connected layers. This kind of architecture is known as a *convolutional* neural network and forms the basis of the best image-recognition systems.

15.4.3 Autoencoders

Neural networks can be used as a powerful dimensionality reduction method known as an *autoencoder*. Take the completely standard feed-forward neural network considered in this chapter and suppose we have access to MNIST handwritten digit dataset. However instead of predicting the identity of the digits y_i from \mathbf{x}_i , we simply predict \mathbf{x}_i from \mathbf{x}_i . That is we model

$$\mathbf{x}_i = \mathbf{f}(\mathbf{x}_i, \mathbf{w}) + \epsilon,$$

where ϵ is noise. Notice, this is entirely trivial when one has a working neural network implementation – simply replace y_i with \mathbf{x}_i . The benefit of this approach is if one of the hidden layers contains *less* dimensions than there are pixels in the image, for instance $H = 100$, then the neural network will effectively learn a 100-dimensional representation of handwritten digits. This can be seen as a variant of PCA in that it also finds a lower-dimensional representation of the digits, however, it allows a highly non-linear mapping.

15.4.4 Recurrent neural networks

In the brain information clearly does not simply flow in one direction as in the feedforward neural network. An attempt to create more realistic neural networks, where information is processed multiple times by the same neural network, is a *recurrent neural network*. Suppose we wish to train a neural network to read parts of a DNA sequence (a DNA sequence is simply a sequence of four letters, *ACGT*, repeated a varying number of times) and determine if the sequence is coding for a protein or not. We assume we have access to example sequences x_i as well as if they express genes or not, $y_i = 0, 1$.

This is a standard classification problem were it not for the fact the DNA sequences can have varying length. One attempt to overcome this is as follows: Suppose each gene \mathbf{x} is a sequence of letters in a one-of- K coding i.e. $\mathbf{x} = (\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_S)$ for an S -long sequence. We then introduce a new vector \mathbf{h} which is initially zero. The idea is to train a neural network which takes \mathbf{h} and a letter \mathbf{b} as inputs and returns an output \mathbf{y} consisting of both the label y and a new state \mathbf{h}' concatenated as a vector $\begin{bmatrix} y \\ \mathbf{h}' \end{bmatrix}$. I.e.

$$\begin{bmatrix} y \\ \mathbf{h}' \end{bmatrix} = \mathbf{f} \left(\begin{bmatrix} \mathbf{b} \\ \mathbf{h} \end{bmatrix}, \mathbf{w} \right).$$

This is just a standard feed-forward neural network. We can then apply it to an arbitrary long sequence by first initializing $\mathbf{h}^{(0)} = \mathbf{0}$ and evaluating

$$\begin{bmatrix} y^{(1)} \\ \mathbf{h}^{(1)} \end{bmatrix} = \mathbf{f} \left(\begin{bmatrix} \mathbf{b}_1 \\ \mathbf{h}^{(0)} \end{bmatrix}, \mathbf{w} \right)$$

and again for the second digit

$$\begin{bmatrix} y^{(2)} \\ \mathbf{h}^{(2)} \end{bmatrix} = \mathbf{f} \left(\begin{bmatrix} \mathbf{b}_2 \\ \mathbf{h}^{(1)} \end{bmatrix}, \mathbf{w} \right)$$

and so on until for the n th digit:

$$\begin{bmatrix} y^{(n)} \\ \mathbf{h}^{(n)} \end{bmatrix} = \mathbf{f} \left(\begin{bmatrix} \mathbf{b}_n \\ \mathbf{h}^{(n-1)} \end{bmatrix}, \mathbf{w} \right).$$

Continuing in this manner for S iterations produces a output $y^{(S)}$ which can then be compared against the ground truth. This model is quite complicated, but writing out the function evaluation one can see that the final output $y^{(S)}$ is simply a function of \mathbf{w} and the input string \mathbf{x} :

$$y = F(\mathbf{x}, \mathbf{w}) = f_1 \left(\left[\mathbf{b}_S \mathbf{f} \left(\left[\mathbf{b}_{S-1} \mathbf{f} \left(\left[\mathbf{b}_{S-2} \cdots \right]^T, \mathbf{w} \right] \right]^T, \mathbf{w} \right) \right]^T, \mathbf{w} \right).$$

Thus, we can train the neural network using gradient descent on the combined function F . The network is called recurrent since it (recursively) updates the intermediate variable \mathbf{h} which allows it to “remember” information found in the beginning of the gene. Many popular architectures for working with text is based on recurrent neural networks.

15.4.5 Serious neural network modelling

The recent success in neural network modelling is partly due to the creation of powerful computational environments which can automate much of the construction of neural network algorithms. Two of the most popular frameworks are the open-source framework Theano <http://deeplearning.net/software/theano/> and Tensorflow <https://www.tensorflow.org/> by google. Both of these frameworks rely on python and powerful GPU-implementations of the underlying operations. Students who has a serious interest in neural networks should try to learn one of these frameworks and not try to build the neural networks from the ground up. The benefits of the framework include

- Automatic computation of derivations and building of inference code.
- Automatic tuning of relevant parameters.
- Automatic validation.
- Automatic GPU-implementations and (more recently) automatic parallelization of code to run on many CPUs and GPUs.

In addition to this, model validation play a central role in testing different neural network architectures. It is highly recommended to keep a log book to track the performance of different neural architectures to see if progress is being made towards solving the problem.

Problems

15.1. Question 1: Which one of the following statements pertaining to regression is *correct*?

- A In regularized least squares regression the aim is to introduce more variance by reducing substantially the model's bias.
- B In least squares regularized regression the regularization strength λ is chosen to be the value of λ that minimizes the term $\lambda \mathbf{w}^\top \mathbf{w}$.
- C An artificial neural network with linear transfer functions ($q(t) = t$) can be written in terms of a linear regression model.
- D For regression problems backward or forward selection can be used to define which part of the output that is relevant for modeling.
- E Don't know.

15.2. Question 2: Consider a feedforward neural network shown in fig. 15.6. The network has no bias weights.

Suppose the weights of the neural network are trained to be $w_{31} = 0.5$, $w_{41} = 0.4$, $w_{32} = -0.4$, $w_{42} = 0$, $w_{53} = -0.4$, $w_{54} = 0.1$ and the activation function of all five n_1, \dots, n_5 nodes is the thresholded linear function

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Suppose the network is called evaluated on input $x_1 = 1, x_2 = 2$, what is the output?

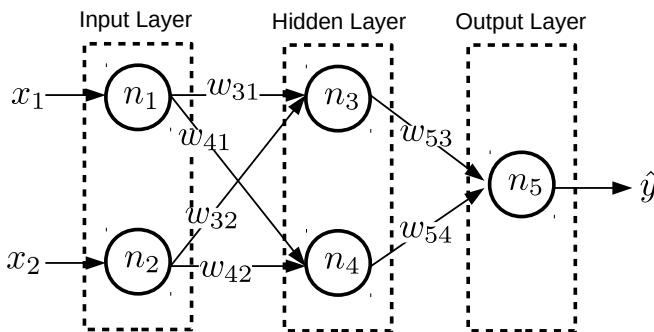


Fig. 15.6. Simple neural network of 6 weights

- A $\hat{y} = 0.04$
- B $\hat{y} = 0.0$
- C $\hat{y} = 1.0$
- D $\hat{y} = 0.16$
- E Don't know.

15.3. Question 3: Consider the classification problem given in Figure 15.7. The problem is solved using a 1-nearest neighbor classifier, a decision tree, an artificial

neural network with four hidden units and a logistic regression model. All the classifiers are only using the attributes x_1 and x_2 . The decision boundaries are indicated in gray and white. We would like to know which classifier each of the four decision boundaries in Figure 15.7 correspond to. Which one of the following statements is *correct*?

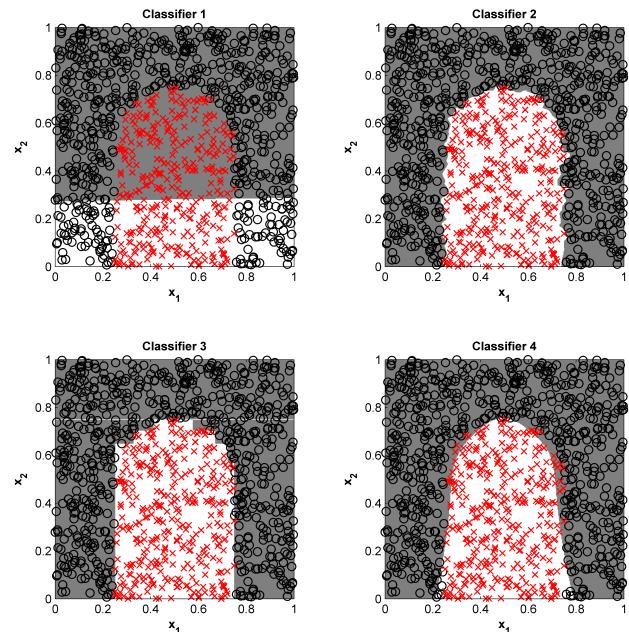


Fig. 15.7. The decision boundaries given in white and gray of four different classifiers used to separate red crosses from black circles.

A Classifier 1 is the decision tree, Classifier 2 is the artificial neural network, Classifier 3 is the logistic regression model, and classifier 4 is the 1-nearest neighbor classifier.

B Classifier 1 is the artificial neural network, Classifier 2 is the 1-nearest neighbor, Classifier 3 is the decision tree, and classifier 4 is the logistic regression model.

C classifier 1 is the logistic regression model, Classifier 2 is the decision tree, Classifier 3 is the 1-nearest neighbor classifier, and classifier 4 is the artificial neural network classifier.

D Classifier 1 is the logistic regression model, Classifier 2 is the 1-nearest neighbor, Classifier 3 is the decision tree, and classifier 4 is the artificial neural network.

E Don't know.

15.4. Question 4: Consider the classification problem given in fig. 15.9. Suppose the problem is solved using the following four classifiers

- (1NN) A 1-nearest neighbour classifier
- (TREE) A decision tree
- (LREG) Logistic regression

(NNET) An artificial neural network with four hidden units

All classifiers are using only the two attributes x_1, x_2 , corresponding to the position of each observation, as well as the class label. Which of the descriptions (1NN), (TREE), (LREG), (NNET) matches the boundaries of the four plots (P_1, P_2, P_3, P_4) indicated in fig. 15.8?

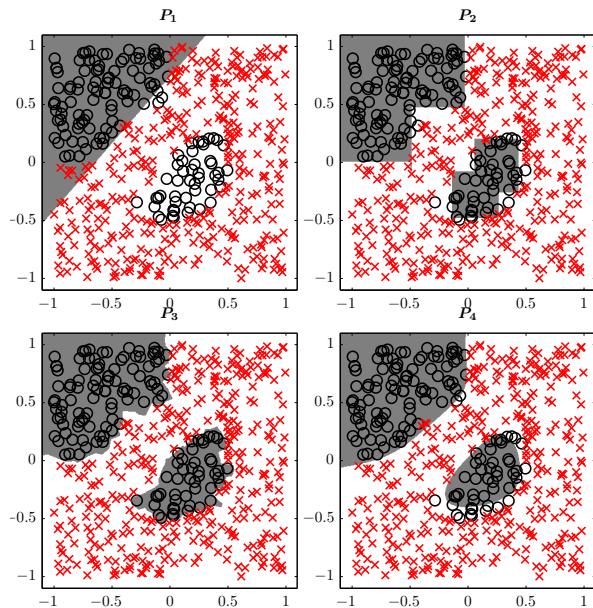


Fig. 15.8. Two-class classification problem

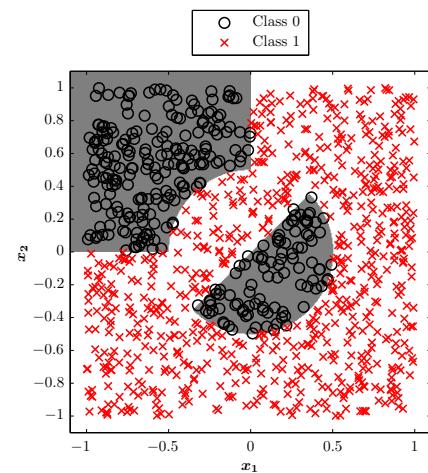


Fig. 15.9. Two-class classification problem

- A P_1 is LREG, P_2 is 1NN, P_3 is TREE, P_4 is NNET.
- B P_1 is LREG, P_2 is TREE, P_3 is NNET, P_4 is 1NN.
- C P_1 is LREG, P_2 is TREE, P_3 is 1NN, P_4 is NNET.
- D P_1 is TREE, P_2 is LREG, P_3 is NNET, P_4 is 1NN.
- E Don't know.

Class imbalance

Class imbalance refers to the situation where the classes in a dataset are not represented equally. The problem with class imbalance is that it confounds our ability to fairly assess the performance of our model using for instance accuracy. Consider the following example: Suppose Ken is devising a test for Ebola. Ken is very impressed by his test accuracy of 0.99999999, however, suppose you learn Ken's test *actually* just consists of a card which says: "*Ebola negative*", but since there are so few people with Ebola it still obtains an accuracy of roughly:

$$\text{Accuracy of Kens Ebola test} = 1 - \frac{\#\text{Cases of Ebola}}{\#\text{Number of people}} \quad (16.1)$$

$$\approx 1 - \frac{80}{8\,000\,000\,000} = 1 - 10^{-8}. \quad (16.2)$$

This is probably *the worst* Ebola test imaginable – but nobody is ever going to discover it by looking at the accuracy.

In this section, we will consider strategies for evaluating models in the presence of class imbalance for a binary classifier. While class imbalance can certainly be present in the multiclass setting, the binary setting is simpler and many of the same comments apply. Because class imbalance is a so frequently occurring feature of many datasets, it has a long history in a variety of fields, see Chawla [2005] for an overview. The main measure we will consider in this chapter, the area under curve (AUC) of the receiver operating characteristic (ROC), was originally invented by British radar engineers around the beginning of the world war II to analyse radar signals [Collinson, 1998].

16.1 Dealing with class imbalance

As the example with the Ebola test illustrates class imbalance can make ordinary measures of performance such as accuracy highly misleading because if we just put everything in the largest class our method will seem to have a high accuracy. Furthermore, in many situations class imbalance is not just common but expected, for instance if we are trying to detect fraud in a set of credit card transactions or build a system to recognize obstacles on the road. In this chapter, we will consider a few ways to combat class imbalance in increasing degree of sophistication:

Resampling: where the dataset is changed.

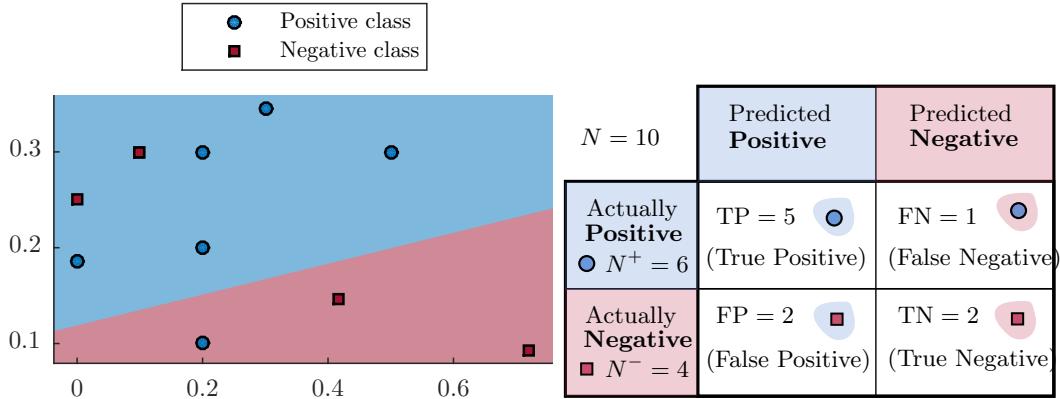


Fig. 16.1. (Left:) A small $N = 10$ observation binary classification problem and the classification boundary. (Right:) The confusion matrix of the classifier in the left-hand pane. The inserts (ticks on background) indicate which observations counts towards which numbers in the confusion matrix.

Penalization: where the relative importance of the classes are changed.

Change performance measure: where we use a performance measure such as area-under-curve (AUC) of the receiver operating characteristic (ROC) which is invariant to class imbalance.

16.1.1 Resampling

The simplest way to handle class imbalance is to change the dataset. There are two variants: If the dataset is very large, we can consider *under-sampling* where we simply remove (by random) observations of the over-represented class until the two classes have the same size. Alternatively, we can try *over-sampling* where we add copies from the under-sampled class until the two classes have the same size. These approaches are very simple to implement and therefore provide an excellent starting point, however, they also have obvious drawbacks: In the first case we loose information, in the second case we must take into account some methods can be very influenced by duplicated observations.

16.1.2 Penalization

Penalization works by scaling the relative importance of the two classes. Recall the definition of the confusion matrix which is reproduced for convenience in fig. 16.1 and which we encountered earlier in section 8.2.1 of chapter 8. In the notation of the confusion matrix the accuracy of the classifier can be written as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{N}.$$

Let's assume that it is the positive class which is over-represented. We can then consider a "scaled" accuracy measure of the form:

$$\text{Accuracy-scaled} = \frac{\text{TP}}{2N^+} + \frac{\text{TN}}{2N^-}.$$

Let's assume we are in the imbalanced setting where $N^+ = 1000$ and $N^- = 10$. A classifier that puts everything in the positive class would have an accuracy of $\frac{1000}{1010} \approx 99\%$, but a scaled accuracy of only $\frac{1000}{2 \times 1000} + \frac{0}{2 \times 10} = 50\%$ corresponding to random guessing (also notice the scaled and true accuracy are both 1 if the classifier is perfect). A disadvantage of the scaled accuracy is that it is also 50% if everything is classified as belonging to the negative class which might seem counterintuitive because all but 10 observations are then classified incorrectly!

In general, the errors of the classifier are not equally important. Suppose we have credit-card transaction system where the positive class corresponds to a fraudulent transaction and the negative to a non-fraudulent (normal) transaction. In this case labelling a few good transactions as fraudulent, FP, (transactions that are actually negative labelled positive) is not so bad, but labelling fraudulent transactions as good, FN, correspond to a loss of money. We can therefore consider a general measure of the quality of the classifier of the form

$$w_1 \text{TP} + w_2 \text{FN} + w_3 \text{FP} + w_4 \text{TN}, \quad (16.3)$$

where w_1, \dots, w_4 are constants. As a crude example, in the credit-card system we could choose $w_1 = 2, w_2 = -1000, w_3 = -1$ and $w_4 = 0.01$ to signify that classifying non-fraudulent transactions as non-fraudulent (which happen very often) is good (weight 0.01), labelling fraudulent transactions as fraudulent is even better (weight 2; keep in mind this happens rarely) but *incorrectly* labelling a fraudulent transaction as non-fraudulent is very bad (weight -1000). The drawback of this method is that the user has to specify w_1, w_2, w_3 and w_4 .

If we consider the credit card transaction problem, we should ask ourselves *why* it is so bad to classify all transactions as being without fraud. The obvious answer is that it is bad because we lose customers and, ultimately, money. A way around the problem could therefore be to figure out the expected loss of money for each classification outcome: How much do we expect to lose by (incorrectly) closing a credit card and annoy a customer and how much do we expect to lose by not closing a credit card in time? This information could in turn be used to select w_1, \dots, w_4 in the penalization scheme eq. (16.3). Keep in mind that especially in medical applications this may lead to fairly uninviting utilitarianism when bad medical decisions are balanced against monetary concerns.

Precision and recall

Two terms relating to the performance of a classifier which roughly falls within the above category is the precision and recall. They are simply defined as:

$$\begin{aligned} \text{Recall: } & \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\#\{\text{Observations in the positive class}\}}, \\ \text{Precision: } & \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\#\{\text{Observations predicted as positive}\}}. \end{aligned}$$

The recall is also known as the *true positive rate* which we will see again in a moment. The recall can trivially be improved by labelling all observations as positive, however the precision will suffer if all observations are labelled positive. Both of these measures are different from for instance the accuracy in that they place more emphasis on the positive class. For instance in a credit-card fraud detection system, where fraud corresponds to the positive class, high recall is the measure of how many actually cases of fraud are caught. Meanwhile precision might be appropriate in a case where false positive comes at a significant cost, for instance medical screening.

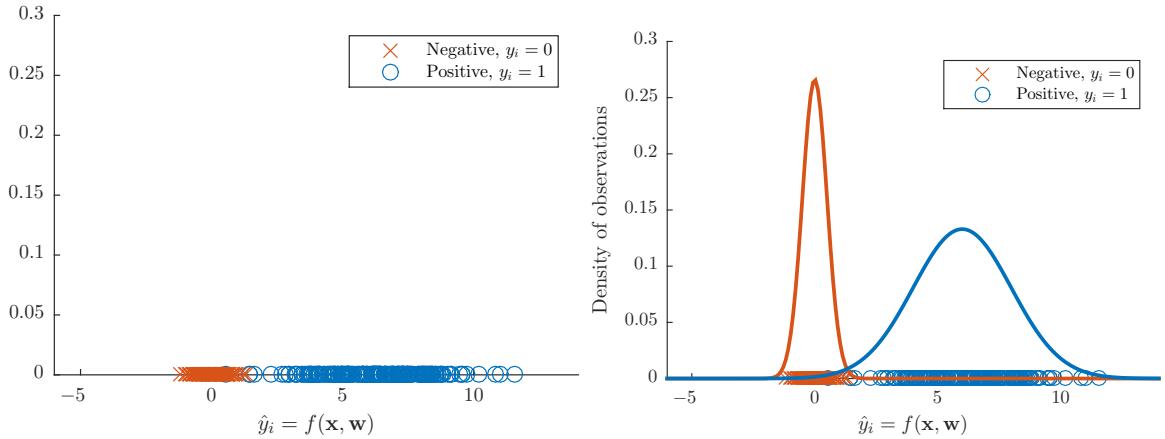


Fig. 16.2. A dataset consisting of a positive $y_i = 1$ class and a negative $y_i = 0$ class. The x -position indicates the predicted y -value by a classification model. In the right plane we have plotted the same dataset but with the density of each class which is easier to visualize and which will be used in the following.

16.2 Area-under-curve (AUC)

The final strategy we will consider for the class-imbalance problem is to change the performance measure to implicitly take class imbalance into account. In order to do so we need to take a step back and consider what a classifier actually does. Consider therefore a standard two-class classification problem with observations x_i and output y_i where $y_i = 0$ means observations i belongs to the negative class and $y_i = 1$ means observations i belongs to the positive class. Suppose we build a classifier that assigns to each observation i a number \hat{y}_i

$$\hat{y}_i = f(\mathbf{x}_i, \mathbf{w}).$$

In many practical situations, the number \hat{y}_i will be continuous and only indicate a relative “propensity” for i to belong to a given class according to the classifier, see fig. 16.2. For instance in logistic regression, \hat{y}_i is a (continuous) probability in the interval $[0, 1]$ such that the higher \hat{y}_i is the more likely it is to belong to the positive class.

How do we evaluate such a classifier? The first step is to translate the continuous numbers \hat{y}_i into binary class-predictions. The simplest way is to introduce a parameter θ and simply assign all i where $\hat{y}_i > \theta$ to the positive class and all i where $\hat{y}_i \leq \theta$ to the negative class. In fig. 16.3 is shown two different thresholds. Notice, the different thresholds have a large influence on the behaviour of the classifier: If we use the high (shown in the left plot) threshold, it is very unlikely to ever say an observation that in fact is negative ($y_i = 0$) belongs to the positive class, whereas it will be slightly prone to falsely saying an observation which is in fact positive ($y_i = 1$) is negative. The other threshold has the opposing effect. Since the threshold is chosen arbitrarily, when we wish to discuss the performance of a classifier f , we must take into account the different threshold values. This requires some terminology.

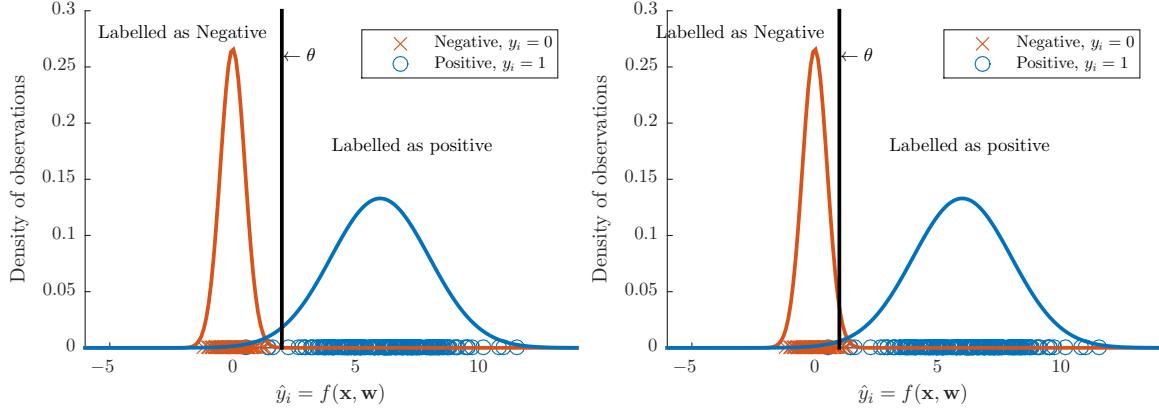


Fig. 16.3. The dataset considered in fig. 16.2 with a positive $y_i = 1$ class and a negative $y_i = 0$ class and where the x -position indicates the predicted y -value by a classification model. If we threshold the classifier at the value θ the choice of this threshold value has a large influence on what will be labelled as positive and negative.

16.2.1 The confusion matrix and thresholding

Each thresholding level θ produce a confusion matrix. For convenience this is illustrated in fig. 16.4 and are:

True Positives, TP_θ : Number of observations which are in fact positive $y_i = 1$ which the classifier correctly labels as positive $\hat{y}_i > \theta$.

False Positives, FP_θ : Number of observations which are in fact negative $y_i = 0$ which the classifier incorrectly labels as positive $\hat{y}_i > \theta$.

False Negatives, FN_θ : Number of observations which are in fact positive $y_i = 1$ which the classifier incorrectly labels as negative $\hat{y}_i < \theta$.

True Negatives, TN_θ : Number of observations which are in fact negative $y_i = 0$ which the classifier correctly labels as negative $\hat{y}_i < \theta$.

Notice, these numbers depend on θ . Since the class sizes (the total number of positive examples and negative examples) may differ significantly it is common to normalize with the class sizes. We thus define the true positive rate and false positive rate as:

$$FPR_\theta = \frac{FP_\theta}{FP_\theta + TN_\theta}, \quad (16.4)$$

$$TPR_\theta = \frac{TP_\theta}{TP_\theta + FN_\theta}, \quad (16.5)$$

where by definition $TP_\theta + FN_\theta$ is the total number of positive examples and $FP_\theta + TN_\theta$ is the total number of negative examples. An illustration of how these numbers depend on θ is probably in order. In fig. 16.5 is illustrated three values of the threshold θ . In fig. 16.6 we have plotted the TPR and FPR for each of these values as solid dots whereas the line indicate all other values of θ . Notice, the colors agree with fig. 16.4. Let's try to make some sense of why the curves look the way they look. When θ is very low, everything is labelled as positive, and so the true positives has to

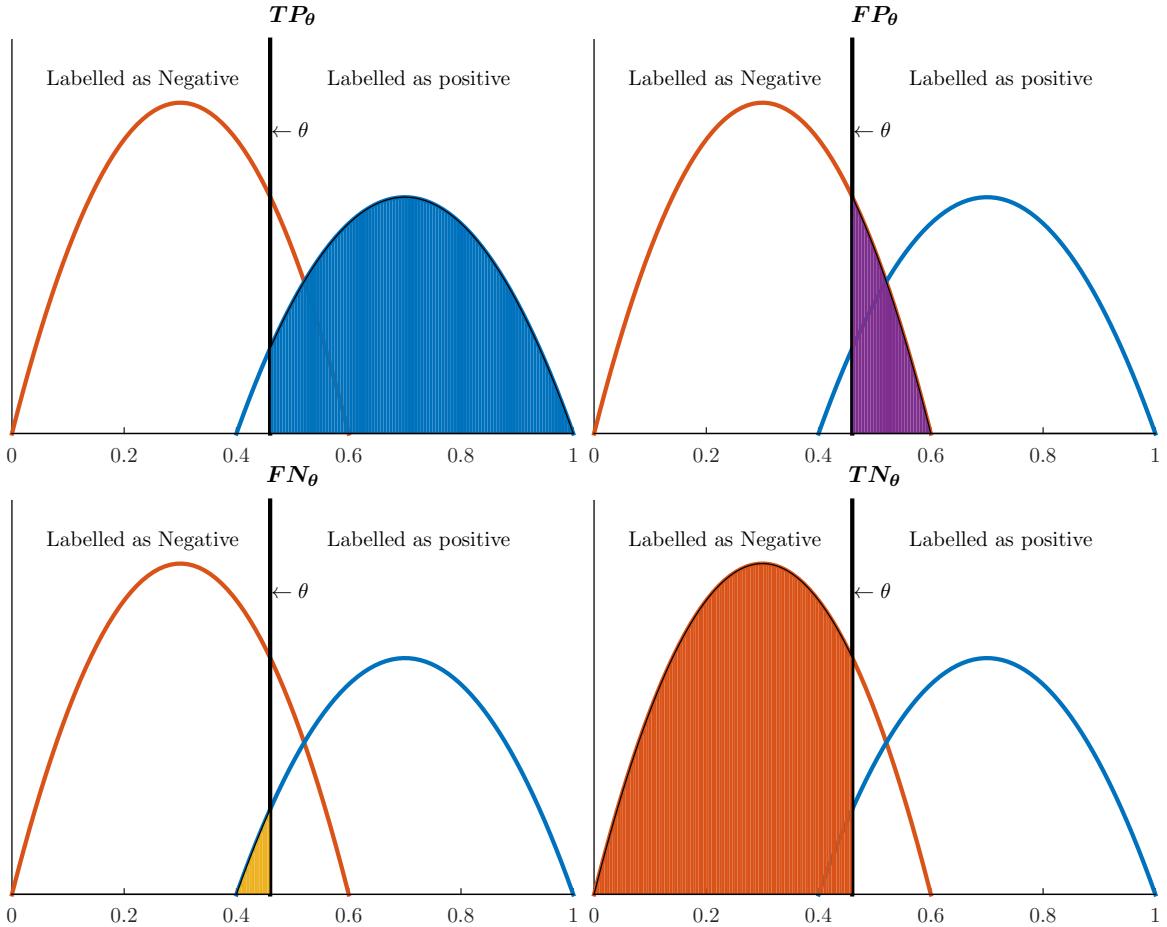


Fig. 16.4. Everything left of the black line θ is labelled to belong to the negative class and everything to the right of the black line is labelled as belonging to the positive class. The areas then respectively indicate the number of true positives, false positives, false negatives and true negatives. See text for details.

be all the positives and therefore the true positive rate (TPR) is 1. When θ increases, eventually everything is labelled negative and so the TPR becomes 0. Roughly, the same goes for the false positives. First, all elements in the negative class are (incorrectly) labelled as positive, and therefore the false positive rate is 1. However, as θ increases, the false positive rate will drop faster than the true positive rate simply because the red hump (of the negatives) is to the left of the blue hump of the positives. Eventually, everything is labelled as negative and so there are no false positives. This is why the curves start at 1 and ends at 0 and the TPR is normally above the FPR.

With these definitions, we can simply plot values of (FPR_θ, TPR_θ) for all conceivable values of θ forming the receiver operating characteristic (ROC) curve given in the right-pane of fig. 16.6. Since the TPR is normally higher than the FPR, the curve will generally be above the diagonal indicated by the dotted line. This allows us to define the *Area Under Curve* (AUC) as simply the

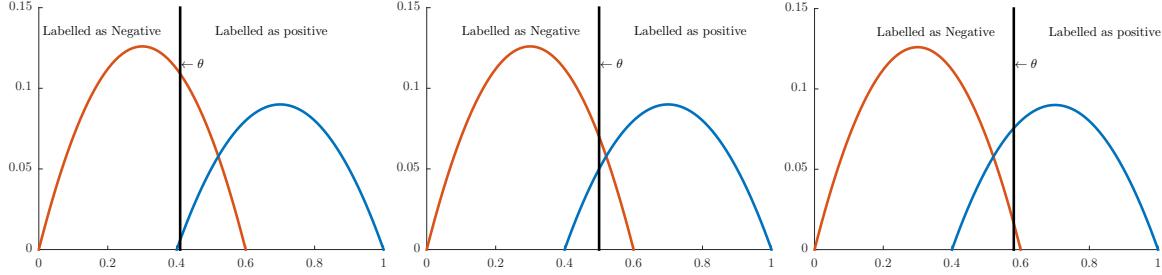


Fig. 16.5. Illustration of our two classifiers with three different threshold values.

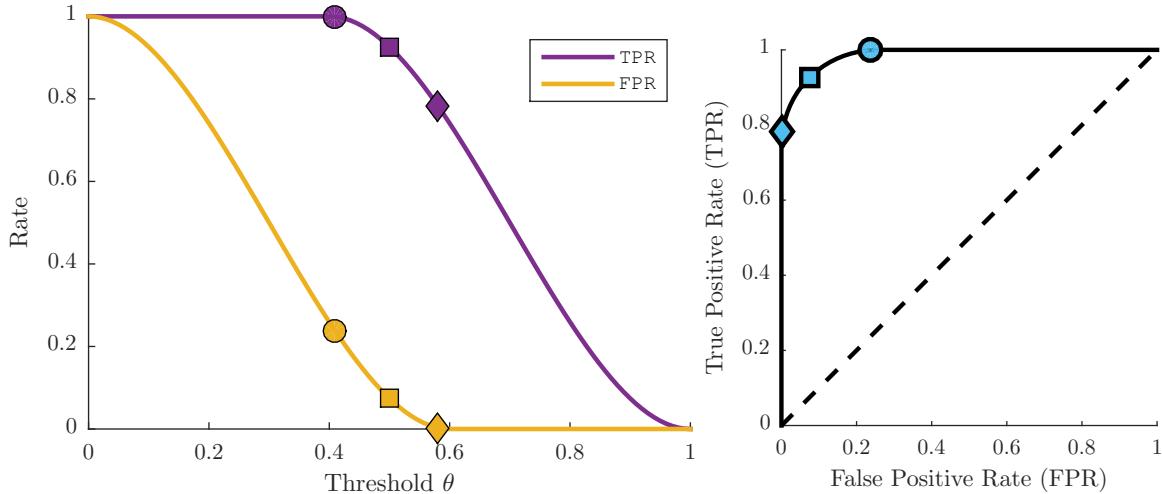


Fig. 16.6. (left:) Illustration of the TPR and FPR curves for different thresholds for the classifier indicated in fig. 16.5. The solid points corresponds to the three specific threshold values indicated. (right:) Illustration of the receiver operating characteristic (ROC) curve. The AUC is simply the area under this curve obtained by plotting TPR_θ against FPR_θ for different values of θ . High AUC is good.

area under the curve of the ROC curve shown in the right-hand side of fig. 16.6 (which can be obtained by numerical integration). Since the curves are generally above the dotted line, this value will be between 0.5 and 1.

Let's re-assure ourselves the AUC really behaves like a performance evaluator. If the AUC is 1, this means it goes through the point $(0, 1)$ meaning that for some θ the number of false positives is 0 and the number of true positives is equal to the total number of positives – i.e. the classifier works perfectly. On the other hand, let's suppose we have an inferior classifier as indicated in fig. 16.7, again with three values of θ selected. The corresponding plot of the TPR and FPR and AUC is given in fig. 16.8. This curve is much closer to the dotted line, indicating a lower value of the AUC. Hopefully, these examples are sufficient persuasion the AUC evaluates the performance of the classifier, but the reader is encouraged to investigate what an AUC of 0.5 would correspond to.

In conclusion, the area under curve (AUC) is a performance measure for classifiers, which has two desirable properties: First, it allows us to get rid of the dependence of θ by integrating over all

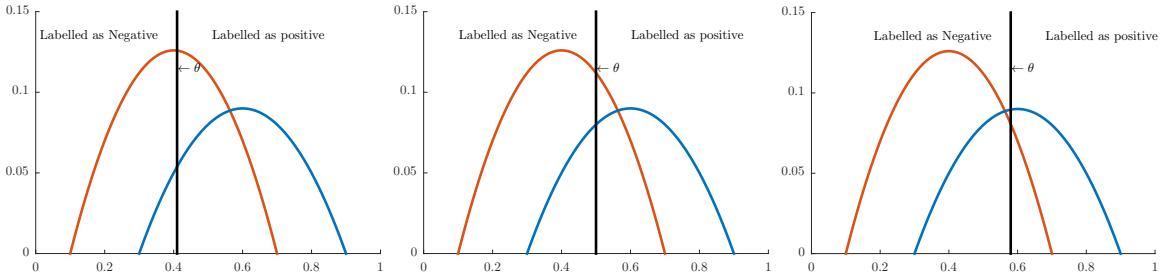


Fig. 16.7. Illustration of three different threshold values for an inferior classifier. The classifier is inferior since the two predicted classes overlap such that no single threshold can tell them apart.

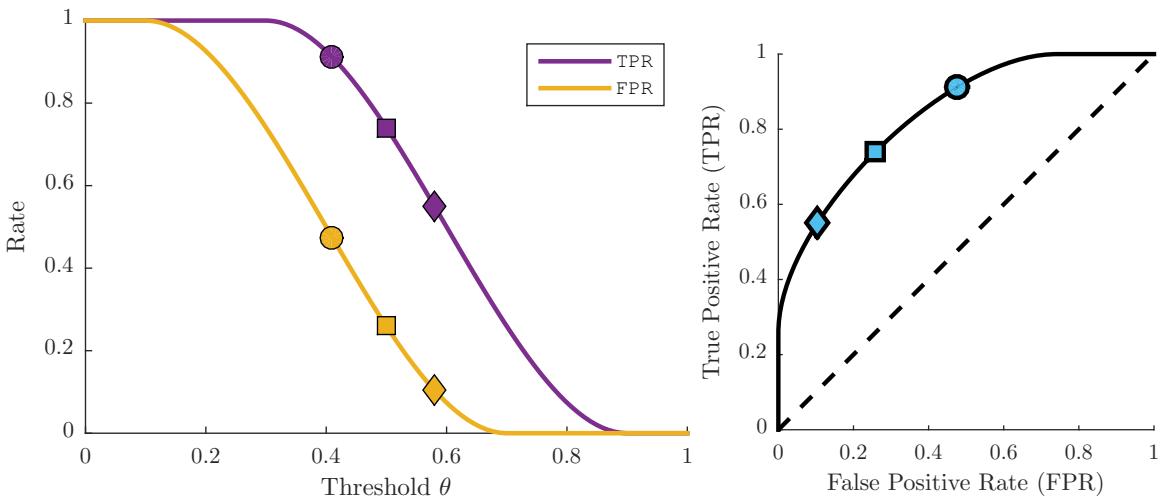


Fig. 16.8. (left:) Illustration of the TPR and FPR curves for different thresholds for the inferior classifier indicated in fig. 16.7. The solid points corresponds to the three specific threshold values indicated. (right:) Illustration of the AUC; the AUC is simply the area under the curve obtained by plotting TPR_θ against FPR_θ for different values of θ . High AUC is good, i.e. this classifier is worse than that indicated in fig. 16.5

the conceivable values of θ . Secondly, it accounts for imbalanced classes due to the normalization of the TPR and FPR by the number of observations in the true and false class respectively. A drawback of the AUC is that it only allows for two classes.

Problems

16.1. Question 1: Suppose a small network is trained on a binary classification dataset containing $N = 1000$ points. The output of the neural network is continuous and restricted to $\hat{y} \in [0, 1]$. To evaluate the performance on the network, the network makes predictions on a test set of $N_{\text{test}} = 1000$ points. By thresholding the output of the neural network at different levels, i.e. for each level θ assign an output \hat{y}_i to class 0 if $\hat{y} \leq \theta$ and otherwise to class 1, one obtains different values of the TP, TN, FP and FN. This allows us to draw the ROC curve shown in fig. 16.10. Which of the true positive rate (TPR), and false positive rate (FPR) curves A,B,C or D shown in fig. 16.9 corresponds to the ROC curve in fig. 16.10?

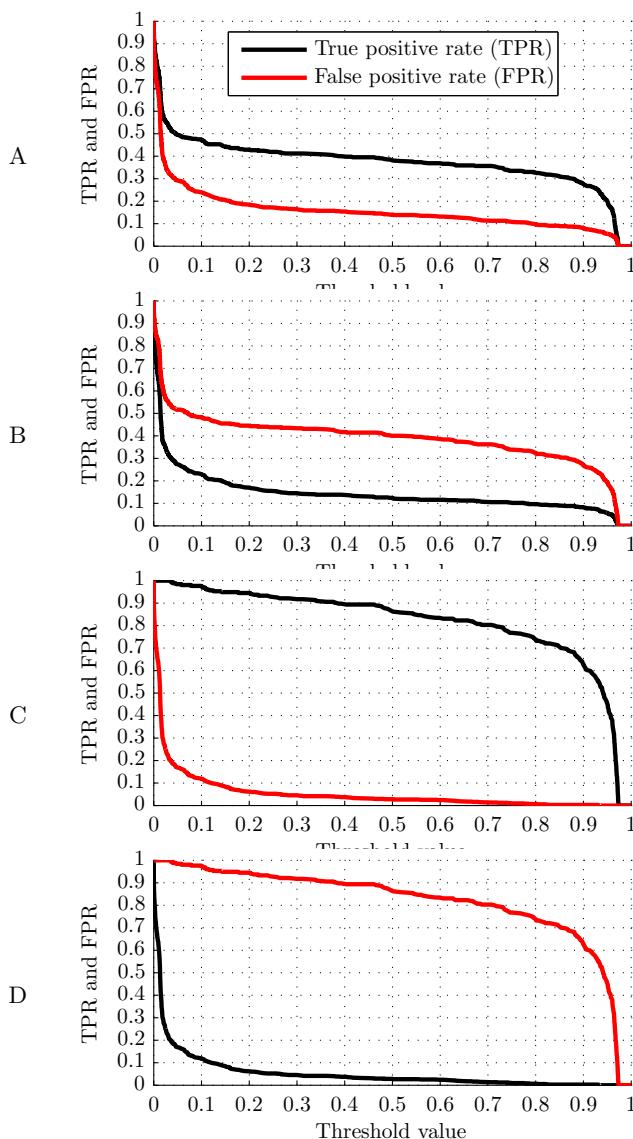


Fig. 16.9. Proposed values of the TPR and FPR, as calculated for $N = 1000$ predicted values, for different values of the threshold θ .

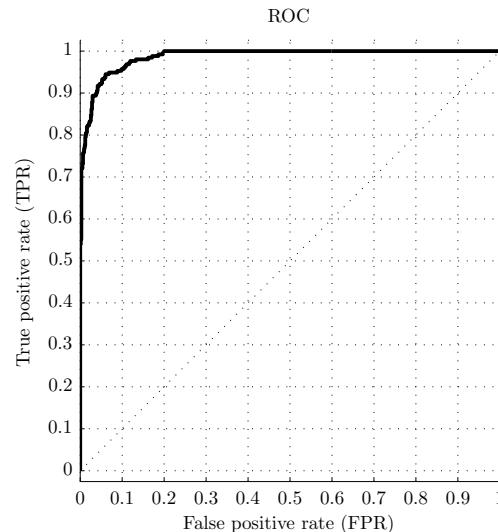


Fig. 16.10. ROC curve of a neural network.

- A Figure A
- B Figure B
- C Figure C
- D Figure D
- E Don't know.

16.2. Question 2: Consider the true positive rate (TPR), and false positive rate (FPR) as a function of the threshold θ for the problem of $N = 1000$ observations shown in fig. 16.9(A). Suppose we consider predictions made at a threshold of $\theta = 0.3$, and suppose we are told that the number of true negatives at $\theta = 0.3$ is $TN = 489$, the TPR at this threshold is $TPR = 0.412$ and the FPR at this threshold is $FPR = 0.164$. What is the (approximate) number of true positives (TP) of the model at this threshold?

- A $TP = 93$
- B $TP = 171$
- C $TP = 275$
- D $TP = 381$
- E Don't know.

16.3. Question 3: We will consider survived as the positive class (i.e., $y = 1$) and died as the negative class (i.e., $y = 0$) in Haberman's survival dataset shown in Table 16.1. Considering again the confusion matrices for the logistic regression and decision tree classifiers given in Figure 16.11, which one of the following statements is *correct*?

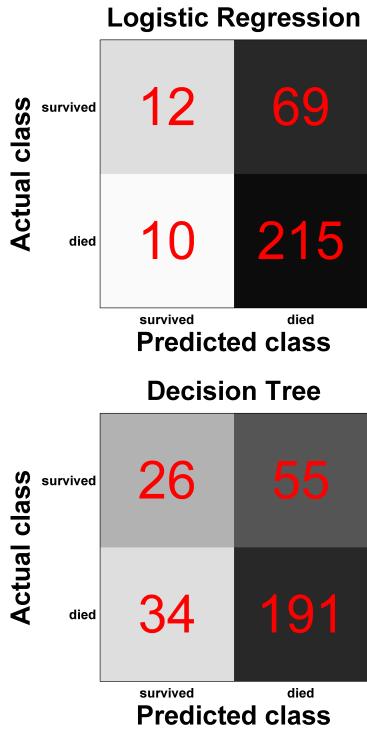


Fig. 16.11. The confusion matrix for the logistic regression and decision tree classifiers used to predict survival based on leave-one-out cross validation.

No.	Attribute description	Abbrev.
x_1	Young (< 60 years), $x_1 = 0$ or Old (≥ 60 years), $x_1 = 1$	Age
x_2	Operated before, $x_2 = 0$ or after 1960, $x_2 = 1$	OpT
x_3	Positive axillary nodes detected PAN No, $x_3 = 0$ or Yes, $x_3 = 1$	PAN
y	Lived after 5 years No, $y = 0$ or Yes, $y = 1$	Surv

Table 16.1. A modified version of Haberman's Survival Data taken from <http://archive.ics.uci.edu/ml/machine-learning-databases/haberman/haberman.names>. The attributes x_1-x_3 denoting the age, operation time and cancer size as well as the output denoting survival after five years are binary. The data contains a total of $N = 306$ observations.

- A The precision of the logistic regression classifier is higher than the precision of the decision tree classifier.
- B The recall of the logistic regression classifier is higher than the recall of the decision tree classifier.
- C The true negative rate of the logistic regression classifier is lower than the true negative rate of the decision tree classifier.
- D The false positive rate of the logistic regression classifier is higher than the false positive rate of the decision tree classifier.
- E Don't know.

Ensemble methods

Ensemble methods are methods where multiple models are trained and their outputs are then combined to obtain better predictive performance than each of the methods on their own. The procedure can be compared to how we might ask the opinion of many doctors and use an average of their opinions to form our opinion or consider multiple reviews before buying a cinema ticket. The benefits of this procedure in machine learning is twofold: Firstly, averaging many classifiers produces a classifier with less variance than each of the individual classifiers. This allows us to use classifiers with larger variance and therefore better ability to tell observations apart while being less sensitive to the occasional overfitting of each individual classifier, much like using an aggregate of several doctors opinion can be used to rule out the occasional crank doctor. Secondly, while most classifiers may assign two hard-to-tell-apart observations to the same class, it may be that a few classifiers in the ensemble can tell the two observations apart, much as how many doctors may have difficulties telling two diagnosis apart where but a few specialist doctors can. (The specialists may then not be able to tell other observations apart but hopefully other specialists in the ensemble can).

In this chapter, we will consider two methods for ensemble methods, bagging and boosting. Bagging is the simpler method where model predictions are simply averaged whereas boosting actively seek out hard-to-classify observations and build specialized models for these observations. The idea behind bagging has roots going back to [Dasarathy and Sheela, 1979] and was applied to neural networks by [Hansen and Salamon, 1990]. The most famous application of bagging, random forests where bagging is applied to decision trees, was initially developed by Ho [1995] and later developed into the method of random forest by Breiman [2001]. Boosting was developed in the very early 90s by Schapire [1990] and the particular method we will consider here, AdaBoost, was developed about 10 years later by [Freund and Schapire, 1997].

17.1 Introduction to ensemble methods

The basic idea of ensemble methods is very simple: train multiple models and combine their outputs into a single model as illustrated in fig. 17.1. Let's consider how this combination takes place before we discuss how the ensemble of models is produced: Suppose $\mathcal{M}_1, \dots, \mathcal{M}_T$ are T regression models and model \mathcal{M}_t is trained on some data \mathcal{D} to learn a regression function $y = f_t(\mathbf{x})$, we can then define a new model \mathcal{M}^* as simply the average:

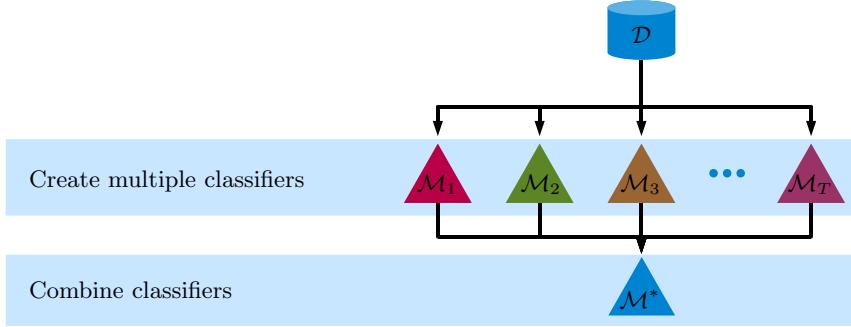


Fig. 17.1. Combining T models $\mathcal{M}_1, \dots, \mathcal{M}_T$ to a single classifier \mathcal{M}^* using majority voting (classification) or averaging (regression) is often a useful strategy to come up with a classifier which outperforms each individual model.

$$(Regression:) \quad y = \frac{1}{T} \sum_{t=1}^T f_t(\mathbf{x}). \quad (17.1)$$

Alternatively, suppose $\mathcal{M}_1, \dots, \mathcal{M}_T$ are classifiers, i.e. $f(\mathbf{x}) = y = 1, \dots, C$. We can then combine their outputs by letting each classifier “vote” for an output and then select the class which most classifiers agree is the correct one.

$$(Classification:) \quad f(\mathbf{x}) = \arg \max_{c=1, \dots, C} \{\text{Number of classifiers which output } f_t(\mathbf{x}) = c\}, \quad (17.2)$$

this is known as *majority voting*. In case of ties, the classifier can just select at random from the tied classes.

So why might ensemble methods work? Suppose we consider a binary classification problem with T independent classifiers. If each classifier is correct with probability p , the chance the majority voting scheme will classify a new point correctly is,

$$\begin{aligned} P(\text{Majority voting is correct}) &= \sum_{t=(T+1)/2}^T \{t \text{ of the classifiers are correct}\} \\ &= \sum_{t=\lceil T/2 \rceil}^T \binom{T}{t} p^t (1-p)^{T-t}, \end{aligned}$$

where $\lceil a \rceil$ denotes rounding a up to the closest integer value larger than or equal to a . The graph as a function of T is plotted in fig. 17.2. In reality we do not have access to independent classifiers even if we are using quite different methods, however, in practice combining different classifiers, especially when they rely on different assumptions, often performs better than simply using the best classifier, and for machine-learning competitions this is a strategy which is often used by the winner.

A problem with combining T classifiers is that it requires about T times as much work to create T classifiers as it takes to create one. A strategy which is therefore often used is to use the same classifier, but train it on different training datasets, see fig. 17.3. There are essentially two strategies:

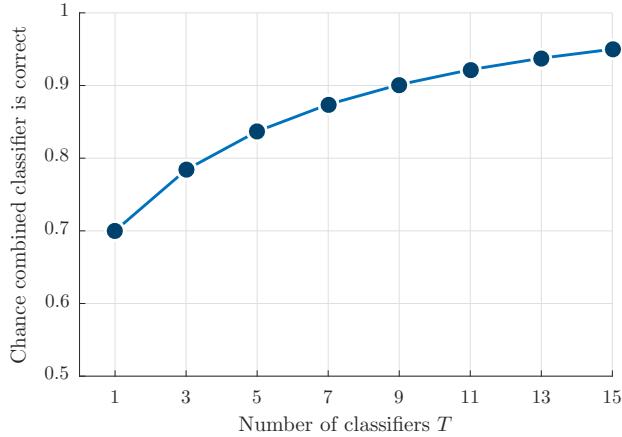


Fig. 17.2. If T independent classifiers is combined using majority voting, each with an accuracy of only $p = 0.7$, the accuracy of the resulting classifier quickly approaches 1. In practice, it is difficult to find independent classifiers, however, the picture still holds approximately for dependent classifiers.

- Apply different transformations to the training set, for instance images can be rotated or translated.
- Select a subset of features.
- Re-sample subsets of the training set.

The first technique is specific to the application, however, it is very often used in Neural-network applications to images. The second technique is very popular for decision trees and is used in *random forests* which we will consider in section 17.3. The third technique, resampling the dataset, and depending on how the dataset is resampled we either obtained *bagging* or *boosting* which we will consider in the following sections.

17.2 Bagging

Bagging begins with a dataset \mathcal{D} of size N , and then randomly selects T new datasets $\mathcal{D}_1, \dots, \mathcal{D}_T$ of size $N' \leq N$ by randomly subsampling \mathcal{D} . The simplest strategy is to set $N' = N$ and sample each \mathcal{D}_t by randomly selecting N points from \mathcal{D} *with replacement*. That is, the same points may occur many times in each \mathcal{D}_t and some points may be omitted. The same classification (or regression) model is then trained on each of the T datasets to produce T different classifiers which are then combined into a single classifier using eq. (17.1) or eq. (17.2). This procedure is illustrated in fig. 17.3 and the number of classifiers T can either be selected as a high (but tractable) number or selected using cross-validation. Typically, about 100-1000 classifiers are used.

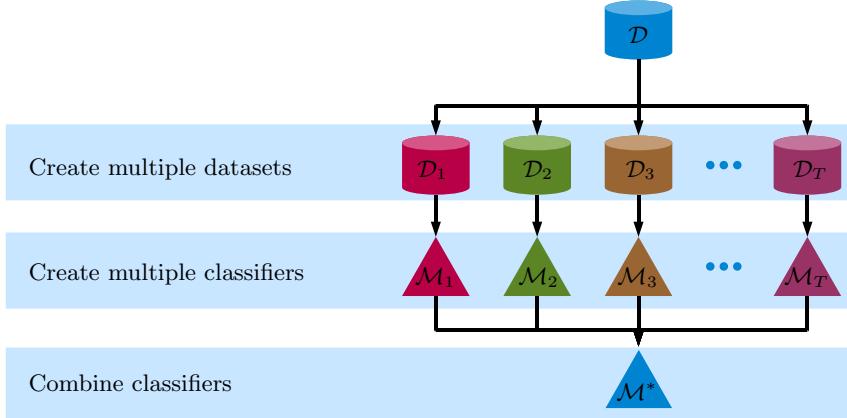


Fig. 17.3. A different strategy for obtaining multiple classifiers is to create T new datasets $\mathcal{D}_1, \dots, \mathcal{D}_T$ from the training dataset \mathcal{D} and train classifiers to each of the dataset. The T obtained classifiers can then be combined as in fig. 17.1.

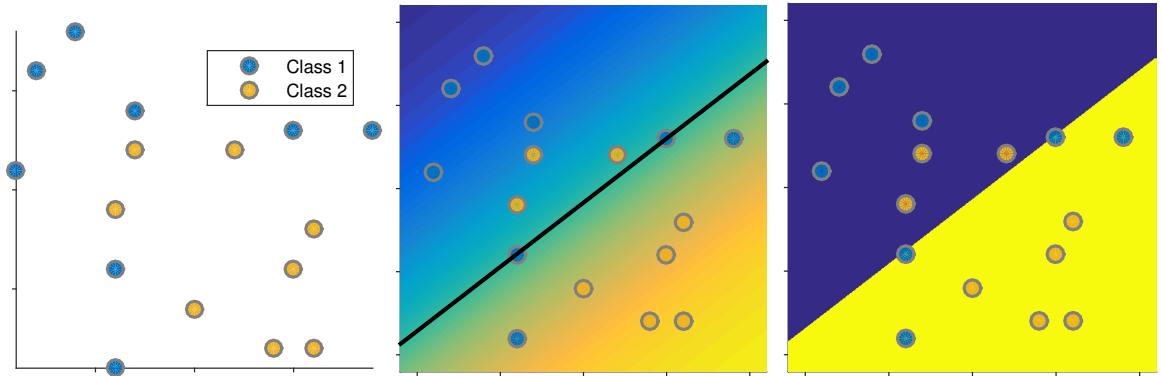


Fig. 17.4. (left:) A simple 2D classification problem with two classes and two features. (middle:) A logistic regression model is fitted to the data to give the class-probability indicated with the colors. (right:) thresholding the logistic regression output at 0.5 gives the classification boundary indicated by the colors. This is the decision rule of the classifier.

Bagging applied to logistic regression

We will illustrate the bagging procedure with a small 2-class classification problem with $N = 16$ points shown in fig. 17.4 (left). The dataset is fitted with a standard logistic regression model giving the linear decision boundary $p(y|\mathbf{x}, \mathcal{D})$ shown in the middle pane. Since we are only interested in the class labels for the majority-voting scheme eq. (17.2) we will assume the predictions of the logistic regression model is thresholded at 0.5 to produce the decision boundary shown in the right pane.

In fig. 17.5 bagging is illustrated for $T = 8$. In each pane, a subset of the datasets are selected at random and the points not selected are shown as hollow circles. As can be seen, there is quite a lot of variability in the decision surfaces since the datasets are random and consist of few observations.

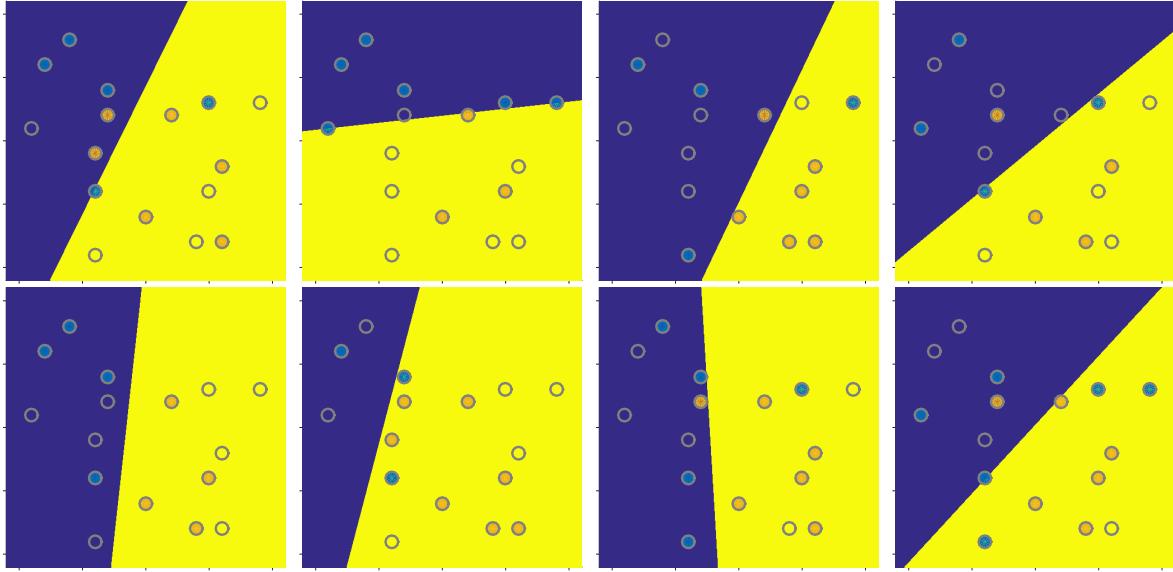


Fig. 17.5. Example of bagging for the classification problem in fig. 17.4. In each pane, a new training set \mathcal{D}_t is obtained by sampling N points with replacement from \mathcal{D} and a logistic regression model is fitted. Notice, not all observations are selected and some points may be selected multiple times. Observations not selected are indicated by hollow circles.

In fig. 17.6 the bagging classifiers are combined, i.e., for each point \mathbf{x} we plot the bagged classifiers' predictions:

$$y = \frac{1}{T} \sum_{t=1}^T f_t(\mathbf{x}), \quad (17.3)$$

and the black line corresponds to $y > \frac{1}{2}$ corresponding to majority voting eq. (17.2). As seen from the figure, each single classifier is worse than the classifier which used all data shown in fig. 17.4 (middle and left pane), however, the errors average out and produce a decision boundary which follows the dataset slightly better than any single logistic regression model. In the right-pane of fig. 17.6 is shown the same bagging setup but using $T = 100$ classifiers. Again, we see the use of many classifiers average out the errors and produces (some) non-linearity in the classification rule which (slightly) better follows the data. The reason why bagging does not affect the classification accuracy very much in this example is because the classifiers are still highly correlated: If all classifiers are the same, clearly bagging will have no effect at all, and as a rule a diverse pool of classifiers as possible is desirable. A diverse pool of classifiers can be obtained by for instance including extra features using feature transformations (for instance high-order Taylor expansions such as x_i^2) or varying the parameters in each of the models in the bagging ensemble. When we consider random forests in section 17.3 we will look at a technique for creating a diverse class of classifiers by manipulating the tree-learning method.

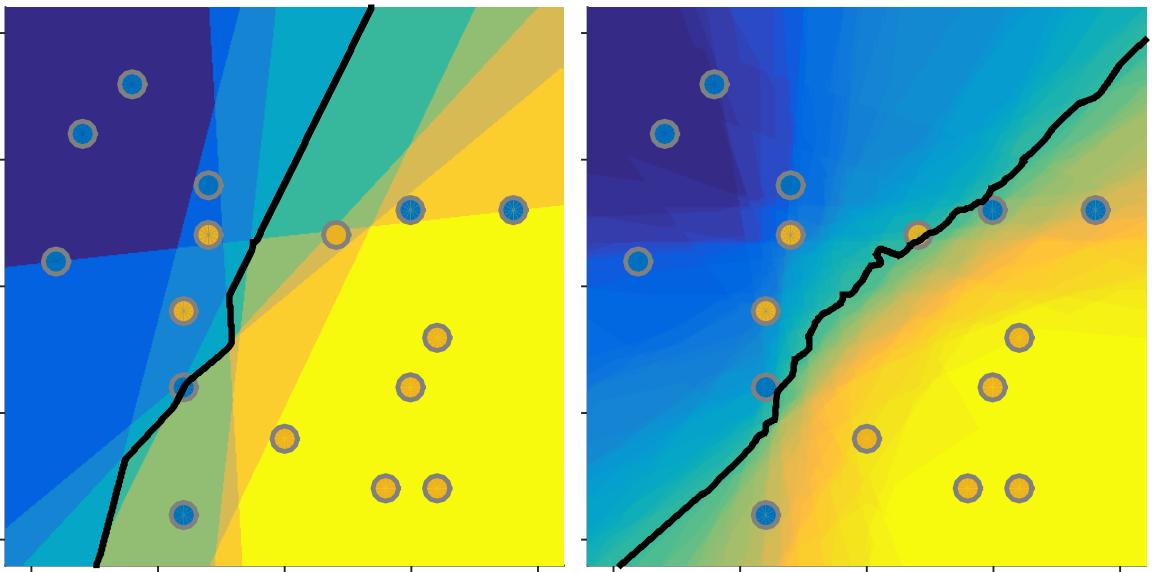


Fig. 17.6. (left:) By averaging the individual prediction boundaries from fig. 17.5 using eq. (17.3) we can define the majority voting rule. The resulting classification boundary (i.e. thresholding at 0.5) is indicated by the black line. In the right pane the same construction is shown but for $T = 100$ datasets. Notice, the resulting rule is smoother and still slightly non-linear.

17.3 Random Forests

Random forests is simply an application of bagging to decision or regression trees. Bagging of decision trees were first developed in a basic version by Ho [1995], which was later extended into the random forest method by Breiman [2001], a paper which has garnered more than 23 000 citations. In order to introduce random forests let's first discuss the simple bagging procedure applied to decision or regression trees: Bagging first produces T datasets (by sampling with replacement) from the original dataset (\mathbf{X}, \mathbf{y}) , then train the standard decision tree algorithm on each sampled dataset to produce a predictor $f_t(\mathbf{x})$ for $t = 1, \dots, T$ and finally combine the predictors using either eq. (17.1) (regression) or eq. (17.2) (classification). As for the logistic regression example, a problem is that the decision trees will often select the same splits over and over again at the root and directly adjacent branches creating very correlated trees. To overcome this, Breiman [2001] proposed that when generating tree T_t , at each step of Hunt's algorithm, Hunt's algorithm should only consider splits from $m < M$ of the features selected at random from all M features (new sets are considered for each new node of the tree). Since the root split will (often) not have the same features available this produces less correlated trees.

There are a few other ingredients to the method found in Breiman [2001], however, the randomness at the feature-selecting step and bagging are the main ones. Typically, T is taken to be of the order 100-1000 and $m = \sqrt{M}$ for classification and $\frac{1}{3}M$ for regression [Hastie et al., 2009, Chapter 15].

17.4 Boosting

As we saw bagging produced some non-linearity in the decision surface of a linear classifier (i.e., logistic regression), however, at least for the considered problem it was quite slight and it still had difficulty with the island of orange points. A message to take away from the problem is that most of the observations are easy to classify, however some are very hard. An alternative strategy would therefore be to select the *hard-to-classify* observations more often than those which are easy to classify and thereby create classifiers which are better suited to solve the hard part of the classification problem. This is basically the idea in boosting.

To make the above idea more concrete, suppose we introduce a parameter w_i for each observation \mathbf{x}_i in the training data set. w_i is the probability of selecting this particular observation when creating the bagging data set, i.e. $w_i > 0$ and $\sum_{i=1}^N w_i = 1$. In the bagging algorithm $w_i = \frac{1}{N}$, however, in boosting the idea is to iteratively adjust w_i depending on how difficult observation i is to classify.

The basic boosting algorithm is illustrated in fig. 17.7 and consists of the following steps:

- We first select a training set \mathcal{D}_1 by sampling N observations with replacement with probability w_i of selecting an observation i ; the dataset \mathcal{D}_1 with a fitted logistic regression model is shown in the left-most pane. Notice, usually not all points are selected.
- In the next step the decision boundary is used to see which of *all* points in the training dataset are classified incorrectly marked with red in the second pane from the left.
- This information is used to *update* the weights in a way we will specify later but such that weights of the wrongly classified points are increased and the weights of the correctly classified points are decreased. The weights still sum to 1; this is indicated by the size of the points in the third pane of fig. 17.7.
- Finally, a new dataset \mathcal{D}_2 is selected by randomly sampling according to the new weights and the procedure is repeated for this new dataset, i.e. a new classifier trained on \mathcal{D}_2 , weights updated a new dataset sampled and so on.

Obviously, we still need to specify how the weights are updated. One can try to come up with a reasonable scheme based on one's intuition, however the weight-updating problem can be analyzed using decision theory which has led to the AdaBoost algorithm [Freund and Schapire, 1997].

17.4.1 AdaBoost

Suppose we denote by $\mathbf{w}(t)$ the weight of the observations at step t which determines how likely that observation is to be included in the training set. The AdaBoost algorithm then produces T classifiers $f_1(\mathbf{x}), \dots, f_T(\mathbf{x})$ and *importance weights* $\alpha_1, \dots, \alpha_T$ (which determines how important each classifier is) which are combined to produce the output of the method:¹

$$f^*(\mathbf{x}) = \arg \max_{y=1,2} \sum_{t=1}^T \alpha_t \delta_{f_t(\mathbf{x}), y}. \quad (17.4)$$

¹ To get a feeling for this definition, recall the delta-function $\delta_{a,b}$ is defined as $\delta_{a,b} = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases}$.

The combined classifier f^* can therefore be understood to first compute the number of "votes" for the positive class: $\sum_{i:y_t(\mathbf{x}_i)=1} \alpha_i$ (and similarly for the negative class, $\alpha^- = \sum_{i:y_t(\mathbf{x}_i)=0} \alpha_i$) and then output 1 if $\alpha^+ > \alpha^-$ and otherwise 0

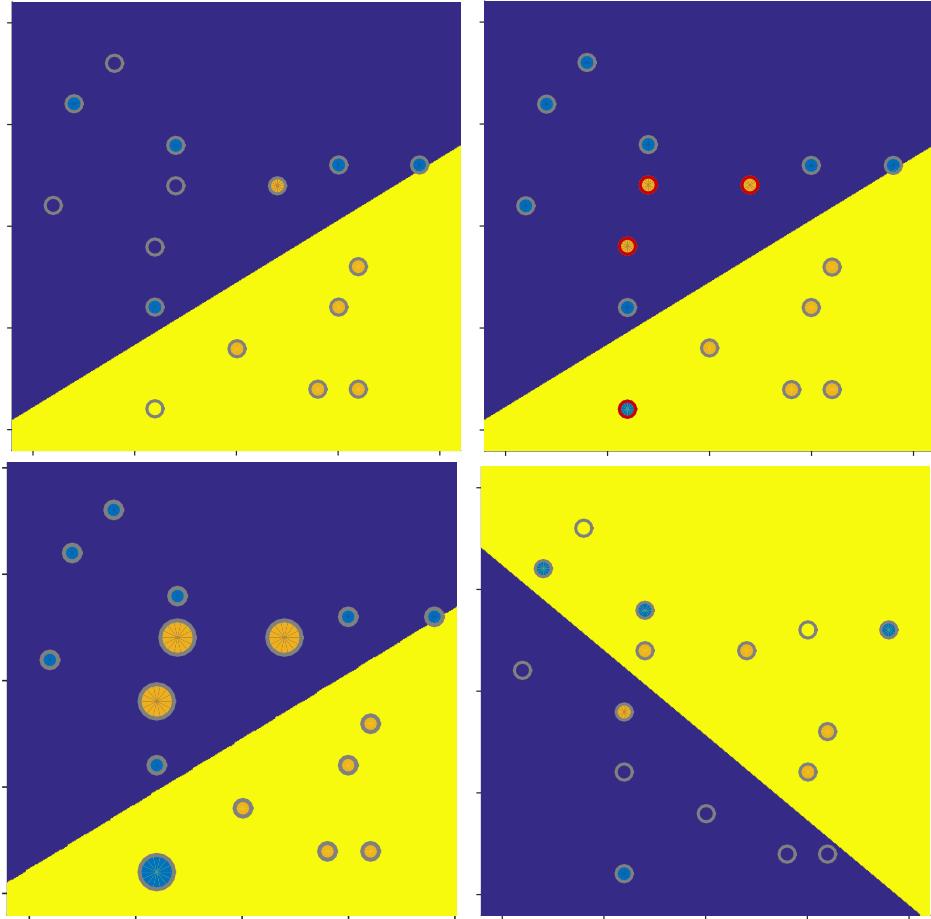


Fig. 17.7. Illustration of a boosting sweep. (Top left:) a dataset \mathcal{D}_t is selected by random sampling from \mathcal{D} with replacement but with probability w_i of selecting observation i and a logistic regression model is fitted to the dataset. Points not selected are hollow. (Top right:) all points are classified using the trained classifier and the misclassified observations are shown in red. (Bottom left:) the weights w_i corresponding to the red misclassified points are increases and the rest are decreased (indicated by the size). (Bottom right:) The next dataset is selected by random sampling using the *new* weights and the procedure is repeated.

The AdaBoost algorithm updates $\mathbf{w}(t)$ and α_t by first computing the weighted error:

$$\epsilon_t = \sum_{i=1}^N w_i(t) (1 - \delta_{f_t(\mathbf{x}_i), y_i}) \quad (17.5)$$

The importance of the classifier at step t is then computed as:

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t} \quad (17.6)$$

and finally the new weights $\mathbf{w}(t + 1)$ are updated by computing:

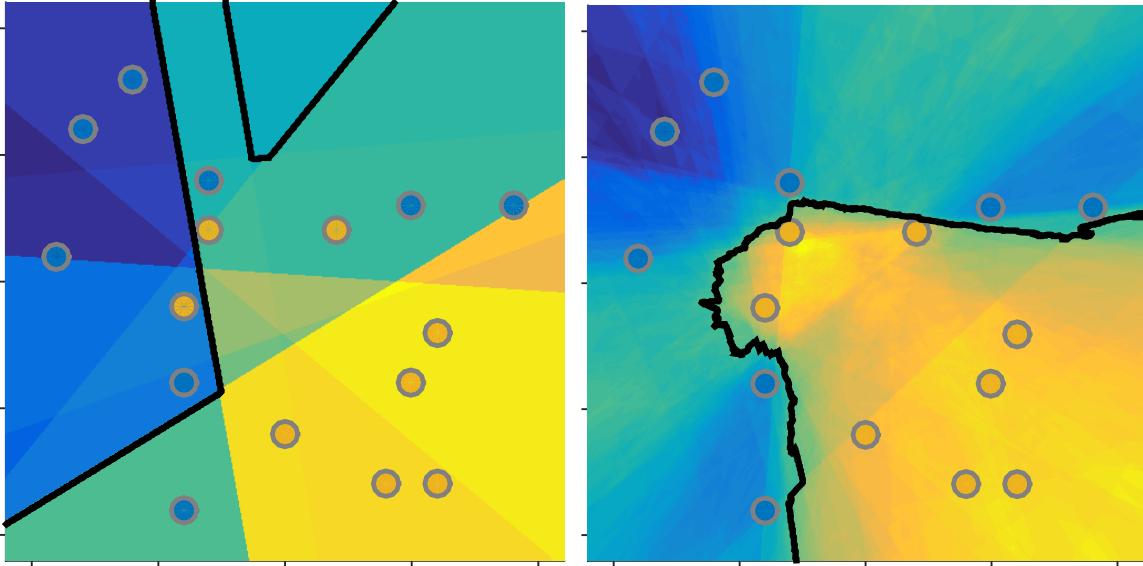


Fig. 17.8. The (importance-weighted) decision function eq. (17.4) when Boosting is applied for $T = 10$ (left) and $T = 500$ (right) rounds. The decision boundary is indicated by the black line. Notice, the decision boundary is highly non-linear and for $T = 500$ (right) perfectly fits the training data even though each classifier is linear.

$$w_i(t+1) = \frac{\tilde{w}_i(t+1)}{\sum_{j=1}^N \tilde{w}_j(t+1)} \quad (17.7)$$

where $\tilde{w}_j(t+1) = \begin{cases} w_j(t)e^{-\alpha_t} & \text{if } f_t(\mathbf{x}_j) = y_j \\ w_j(t)e^{\alpha_t} & \text{if } f_t(\mathbf{x}_j) \neq y_j. \end{cases}$

Thus, this mechanism either up- or down-scales the weights with a factor depending on the importance parameter at the current round, α_t . Finally the majority voting classifier \mathcal{M}^* is found by averaging the vote of each classifier with the importance parameters and selecting the most popular output:

$$f^*(\mathbf{x}) = \arg \max_{y=1,2} \sum_{t=1}^T \alpha_t \delta_{f_t(\mathbf{x}), y}.$$

The full AdaBoost procedure can be seen in algorithm 7 and in fig. 17.8 we have plotted importance-weighted decision functions.

When AdaBoost is applied to the $N = 16$ -observations example considered previously for $T = 10$ or $T = 500$ boosting rounds the individual AdaBoost classifiers are much more extreme since they are trained on fewer datapoints, however, when many AdaBoost classifiers are averaged the decision boundary becomes highly non-linear and is able to separate the two classes.

Algorithm 7: AdaBoost algorithm

```

1: Initialize  $w_i(1) = \frac{1}{N}$  for  $i = 1, \dots, N$ 
2: for  $t = 1, \dots, T$  do
3:   Create  $\mathcal{D}_t$  by sampling (with replacement) from  $\mathcal{D}$  according to  $\mathbf{w}(t)$ 
4:   Let  $f_t$  be the classifier trained on  $\mathcal{D}_t$ 
5:    $\epsilon_t = \sum_{i=1}^N w_i(t) (1 - \delta_{f_t(\mathbf{x}_i), y_i})$  (weighted error of  $f_t$  on all data)
6:    $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$ 
7:   For each  $i$  update weights using eq. (17.7):

```

$$w_i(t+1) = \frac{\tilde{w}_i(t+1)}{\sum_{j=1}^N \tilde{w}_j(t+1)}, \quad \tilde{w}_i(t+1) = \begin{cases} w_i(t)e^{-\alpha_t} & \text{if } f_t(\mathbf{x}_i) = y_i \\ w_i(t)e^{\alpha_t} & \text{if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

```

8: end for
9:  $f^*(\mathbf{x}) = \arg \max_{y=1,2} \sum_{t=1}^T \alpha_t \delta_{f_t(\mathbf{x}), y}$  (Majority voting classifier)

```

17.4.2 Properties of the AdaBoost algorithm★

As mentioned, the peculiar form of the update rules for α_t and $\mathbf{w}(t)$ in the AdaBoost is due to a decision-theoretical analysis in [Freund and Schapire, 1997]. It can be shown the training error of the ensemble classifier f^* is bounded by

$$\epsilon^* \leq \prod_{t=1}^T 2\sqrt{\epsilon_t(1-\epsilon_t)},$$

where ϵ_t are the error rates of each of the classifiers as described in algorithm 7. Suppose each error rate is less than 50%, we can then write $\epsilon_t = \frac{1}{2} - \gamma_t$. Then γ_t measures how much better the classifier is than random guessing and by standard inequalities:

$$\epsilon^* \leq \prod_{t=1}^T 2\sqrt{\epsilon_t(1-\epsilon_t)} = \prod_{t=1}^T \sqrt{1 - 4\gamma_t^2} \leq e^{-2\sum_{t=1}^T \gamma_t^2}.$$

Consequently, if all $\gamma_t \geq \gamma_0$ then the training error of the ensemble is bounded as $\epsilon^* \leq e^{-2\gamma_0^2 T}$ and thus decreases exponentially in T . This may sound like great news, however, recall from chapter 10 a low *training* error is not in itself a good sign. Theoretical analysis of AdaBoost reveals that with high probability: [Freund and Schapire, 1997]

$$\text{Test error} \leq \text{Train error} + \mathcal{O}\left(\sqrt{\frac{dT}{N}}\right),$$

where d is a term dependent of the complexity of our classification model and $\mathcal{O}(\cdot)$ means a term that scale no faster than what is in the parenthesis.

So this is a slightly more negative picture, since when T increases the test error may go towards zero, however, the second term will grow as \sqrt{T} . In addition, we do not know the scaling factor of the second term so the above result should not be taken as predicting the test error is lower than the training error which it will almost certainly never be. From an intuitive perspective, when

we only select a very small subset of training points in each round t (the *difficult* points), each classifier is very prone to overfitting which is why the combined classifier can fit the training data perfectly. When the classifiers are combined, this average out some of the overfitting, however, the combined classifier may still be overfitting the data which plausibly is already happening in fig. 17.8. In practice, AdaBoost often turns out to work very well and increases performance, however, as always it is important to *test* if that is actually the case using for instance cross-validation.

Problems

17.1. Question 1: Suppose Jane wishes to apply a decision tree classifier to a binary classification problem of only $N = 4$ observations. Training and applying the decision tree to the full dataset \mathbf{X} and y_1, \dots, y_4 gives predictions $\hat{y}_1, \dots, \hat{y}_4$ shown in table 17.1.

y	\hat{y}
1	1
1	0
0	0
0	0

Table 17.1. True values y_j and predictions \hat{y}_j for a decision tree classifier trained on the full data set with observed values y_1, \dots, y_4 .

To improve performance Jane decides to apply AdaBoost, however Jane implements AdaBoost such that instead of sampling the N elements of the training sets D_i with replacement, Jane samples the training sets *with-*

out replacement, i.e. the training set D_i is simply the full dataset. Suppose Jane applies AdaBoost for $k = 1$ round of boosting, what is the resulting (approximate) value for the weights w ?

- A $w = [0.123 \ 0.630 \ 0.123 \ 0.123]$
- B $w = [0.167 \ 0.5 \ 0.167 \ 0.167]$
- C $w = [0.081 \ 0.756 \ 0.081 \ 0.081]$
- D $w = [0.077 \ 0.769 \ 0.077 \ 0.077]$
- E Don't know.

17.2. Question 2: Which one of the following statements pertaining to bagging or boosting is *correct*?

- A In boosting miss-classified observations are given less importance in the next round.
- B For each round of bagging it is expected that only a subset of the observations are used for training.
- C Boosting uses leave-one-out cross-validation to learn which observations to sample in the next round.
- D When combining multiple classifiers using bagging the classifier with the best performance is selected.
- E Don't know.

Part III

Unsupervised learning

Distance-based clustering techniques

In this and the following chapters we will consider *unsupervised learning* techniques. In the previous sections we considered the dataset as being composed of a data matrix \mathbf{X} and a set of target values \mathbf{y} . In unsupervised learning we assume we only have \mathbf{X} and our goal is to infer structure in \mathbf{X} such as a clustering (which observations naturally group together), outlier detection (which observations are anomalous), density estimation (how typical is a given observation), and association mining (what are prominent patterns of binary feature co-occurrence). All these tasks depend on how one *defines* a clustering or an outlier and are thus not nearly as well defined as supervised learning where we *know* what the target \mathbf{y} is.

In this chapter, we will consider the particular unsupervised learning problem of identifying groups, or clusters, of data points in a space of arbitrary dimension. Recall a clustering of a set of observations is simply a division of the set of observations into non-overlapping sets, often illustrated as a coloring of the observations. We will consider two methods, K -means and agglomerative hierarchical clustering. Both of these methods are similar in that they are distance-based. However, they differ in that K -means attempts to identify K clusters, whereas hierarchical clustering identifies a nested clustering.

K -means clustering was first discovered by the polish mathematician Hugo Steinhaus in 1956 [Steinhaus, 1956] but given its name and popularized by MacQueen et al. [1967]. The basic hierarchical clustering algorithm was discovered by Johnson [1967].

18.1 Types of clusters

As already mentioned, what constitutes a clustering of a set of observations is somewhat in the eye of the beholder, and different clustering methods are suitable for producing clusters with different properties. We will therefore begin by discussing some general categories of clustering described in Tan et al. [2013].

18.1.1 The distance-based cluster types

The simplest cluster types are the simple, distance-based types illustrated in fig. 18.1 and which are all defined by the distance between observations and (possible) the center of clusters. They are:

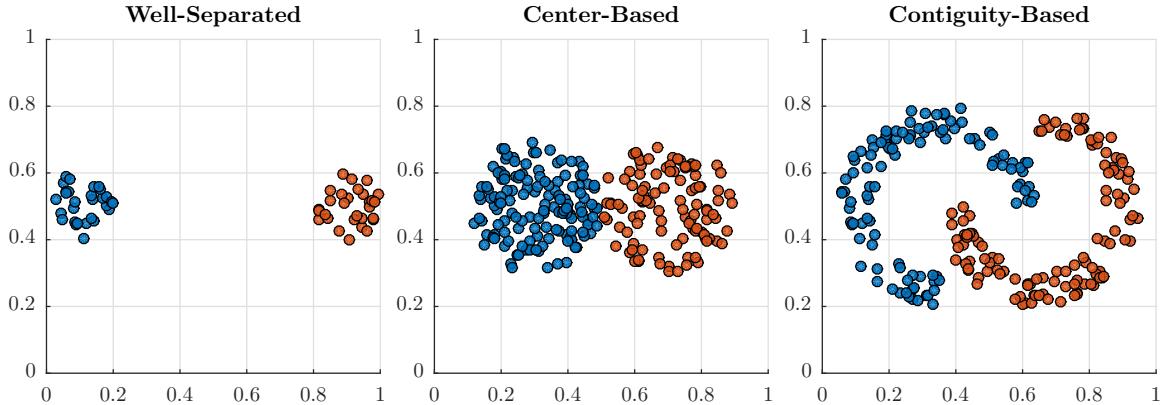


Fig. 18.1. Illustration of the three simple cluster types. The colors indicate the clusters.

Well-separated Each point is closer to all points in its cluster than any point in another cluster. As we will see, hierarchical clustering with max-linkage assumes clusters are well-separated when identifying clusters.

Center-based Each point is closer to the center of its cluster than to the center of any other cluster. As we will see, K -means and Ward clustering (and arguably also average linkage hierarchical clustering) takes a center-based approach to finding clusters.

Contiguity-based Each point is closer to at least one point in its cluster than to any point in another cluster. Hierarchical clustering with min-linkage takes a contiguity-based approach to finding clusters.

18.1.2 More elaborate cluster types

The above three basic types of clustering are the simplest, but it is possible to consider methods that rely on more elaborate (or specific) definitions of what constitutes the clustering. A particular important example are density-based clustering (where a cluster is a group of observations that lie unusually close to each other), however we have also included conceptual clusters as a separate category for cluster-definitions that does not fit any of the other descriptions, see fig. 18.2.

Density-based Clusters are regions of high density separated by regions of low density. The Gaussian mixture-model, which we will consider in chapter 19 takes a density-based approach to finding clusters.

Conceptual clusters Points in a cluster share some general property that is derived from the entire set of points.

18.2 K -means clustering

The goal of K -means clustering is to take as input an arbitrary data set \mathbf{X} comprised of N observations $\mathbf{x}_1, \dots, \mathbf{x}_N$ in a D -dimensional space and then partition (or cluster) the data observations into

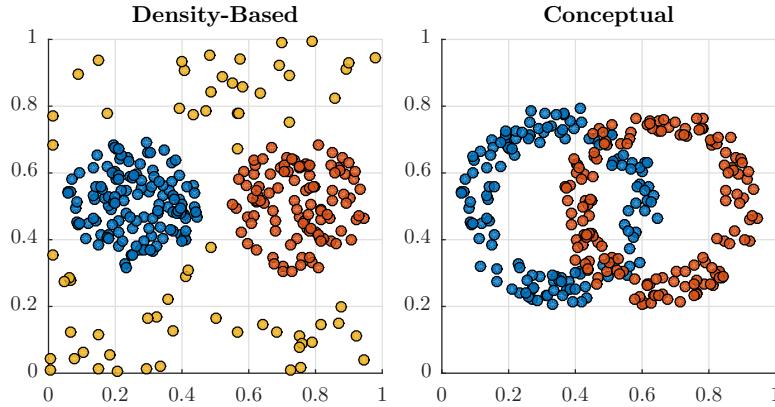


Fig. 18.2. Illustration of two more elaborate cluster types. Note we do not have any general methods for finding conceptual clusters.

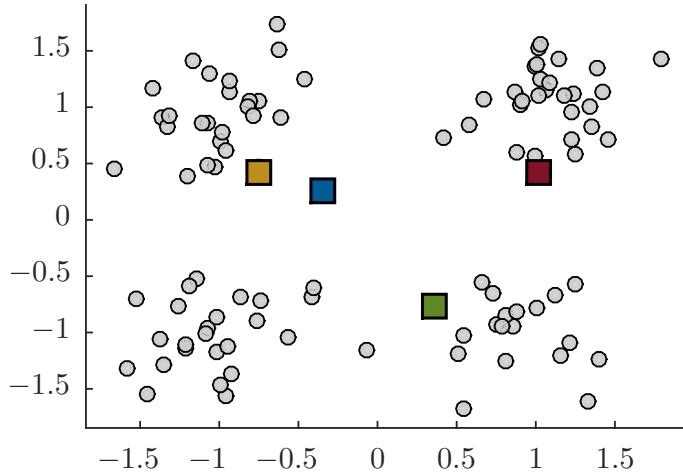


Fig. 18.3. 2D K -means example dataset. Observations are indicated by gray points and the initial location of the K -means cluster locations are indicated as the colored squares. In the example, the location of the clusters are initialized at random.

K groups. A natural way to represent such a partition is as a coloring, where each of the K groups corresponds to one of K colors and the clustering then corresponds to coloring the observations. In K -means clustering, a *cluster* is considered a group of observations where the distance between observations within the group is small relative to the distance between observations outside the group. This notation can be formalized by introducing a vector μ_k for each group $k = 1, \dots, K$ which represents the typical location (or prototypical element) of the group. An observation x_i then belongs to the cluster k where the distance (typically based on the Euclidean distance $\|x_i - \mu_k\|$) is the smallest and as we will see in a moment the μ_k 's represent the centers of the clusters. However, before explaining *why* the K -means algorithm is the way it is, it is easier to explain *what* it does since it is such a simple algorithm.

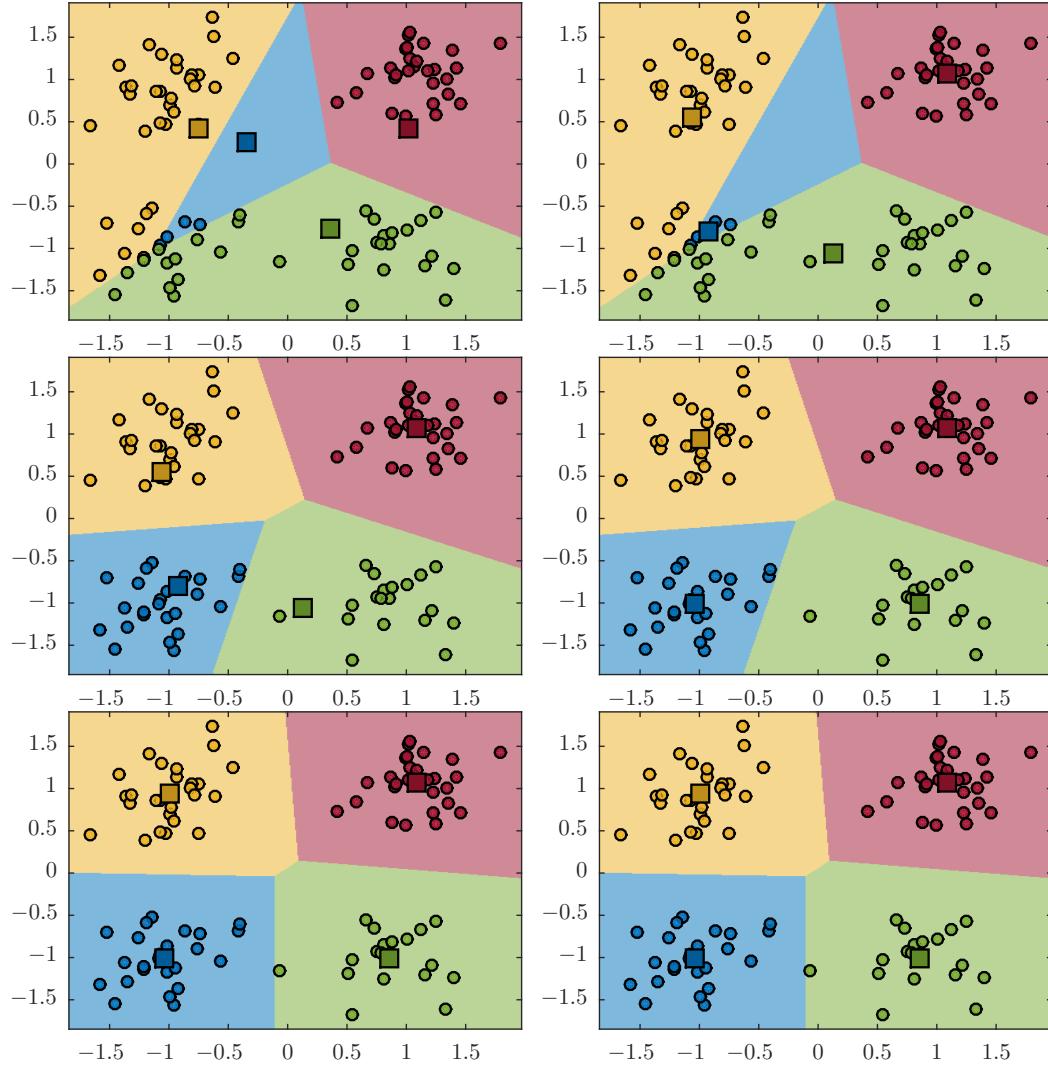


Fig. 18.4. Example of running the K -means algorithm for three iterations and $K = 4$ clusters. Starting with initial location of the cluster vectors indicated by the colored squares, the observations are first assigned to the *closest* mean vector μ_k (top-left). Then in the next step (top-left) the cluster location vectors μ_k are updated to correspond to the mean of the assigned points. This procedure is repeated in the second row, however, in the third row (bottom-left) the cluster assignments do not change and the method has therefore converged.

Suppose we wish to apply the K -means algorithm (based on Euclidean distance) to the 2D dataset shown in fig. 18.3 (gray circles) for $K = 4$ clusters. The location of each of the four μ_k cluster locations are indicated by the colored squares. This is accomplished by the following steps:

- First, initialize each μ_k at a random location as shown in fig. 18.3.

- Assign each of the gray points to the *nearest* μ_k . It now belongs to this cluster. In fig. 18.4 (top-left pane) the region belonging to each cluster is indicated by the colors.
- Update the location of each μ_k to be the *mean* of the points assigned to it. In fig. 18.4 this is shown in the top-right pane.
- Repeat the two previous steps until the location of μ_k does not change. In fig. 18.4 this is shown in row two and three.

18.2.1 A closer look at the *K*-means algorithm

So why does the *K*-means algorithm look the way it does? The objective of the *K*-means algorithm is to find a set of K vectors μ_1, \dots, μ_K as well as an assignment of observations to clusters such that the sum-of-squares of each observation to the nearest vector μ_k is minimized. If we for each point i introduce a binary variable z_{ik} describing which cluster $k = 1, \dots, K$ observation i belongs to, such that if \mathbf{x}_i belongs to k then $z_{ik} = 1$ and $z_{ih} = 0$ for $h \neq k$. Considering Euclidean distance we can define the sum-of-squares between each point to the cluster it is assigned to as:

$$E = \sum_{i=1}^N \sum_{k=1}^K z_{ik} \|\mathbf{x}_i - \mu_k\|_2^2. \quad (18.1)$$

Our goal is then to find values of z_{ik} and μ_k to minimize E . The two steps in the *K*-means algorithm accomplish exactly this. In the first step, the upper-left pane of fig. 18.4, we keep the μ_k 's fixed and minimize z_{ik} . Since this expression is independent for each observation i , we can just for each i choose z_{ik} to minimize $\sum_{k=1}^K z_{ik} \|\mathbf{x}_i - \mu_k\|_2^2$. Obviously, this corresponds to selecting $z_{ik} = 1$ for the k where μ_k is the closest (in Euclidean distance) to \mathbf{x}_i . In other words:

$$z_{ik} = \begin{cases} 1 & \text{if } k = \arg \min_h \|\mathbf{x}_i - \mu_h\|_2^2 \\ 0 & \text{otherwise.} \end{cases} \quad (18.2)$$

Now consider the second step, the upper-left pane of fig. 18.4, where the location μ_k are updated and z_{ik} are kept fixed. If we consider the derivative of the objective function with respect to μ_k we obtain:

$$\nabla_{\mu_k} E = 2 \sum_{i=1}^N z_{ik} (\mathbf{x}_i - \mu_k). \quad (18.3)$$

Setting this equal to zero and solving we obtain:

$$\mu_k = \frac{\sum_{i=1}^N z_{ik} \mathbf{x}_i}{\sum_{i=1}^N z_{ik}}. \quad (18.4)$$

However, the nominator is simply the sum of those observations assigned to cluster k , and the denominator is simply the number of observations assigned to k , so the expression is simply the mean of the observations assigned to cluster k (notice, the update for the μ_k depends on the distance measure and a change in distance measures may also lead to a change in the updates for the cluster locations). We can then see the two steps in the *K*-means algorithm simply corresponds to minimizing E with respect to z_{ik} or μ_k respectively while keeping the other quantity fixed. This is also why the *K*-means algorithm converges; since each step makes the error E smaller, and $E \geq 0$, the algorithm must converge.

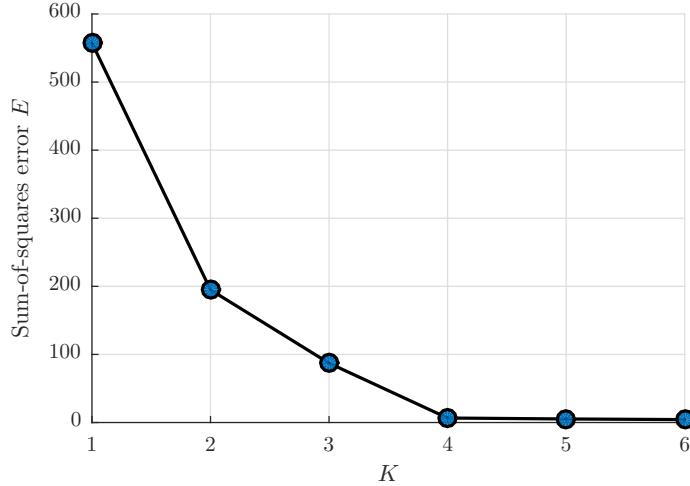


Fig. 18.5. Value of the sum-of-squares error function E in the converged state for the 2D dataset shown in fig. 18.3 for different values of K . The error decreases when K is increased, however, a suitable choice of K can potentially be found as where the drop in error levels off. In this case $K = 4$, which visually also seems to be an appropriate choice.

18.2.2 Practical issues with the K -means algorithm

The K -means algorithm is a very simple and efficient clustering algorithm, however, it has some drawbacks. Firstly, since we rely on the Euclidian distance, it prefers clusters that are “round” and of roughly equal size. For this reason, it may be affected both by outliers but also by simple scaling of one coordinate while keeping the others fixed so when applying the K -means algorithm it is recommended to consider standardizing the data. Secondly, while the K -means algorithm converges quickly, what clustering it converges to depends on how it was initialized. For this reason, it is often useful to consider a particular strategy when initializing the K -means algorithm and consider several restarts with different initialization. One popular (and simple) choice of initialization is the farthest-first procedure Gonzalez [1985] according to which for $k = 1, \dots, K$ we initialize μ_k by:

- Randomly assign one of the observations to be the location of the first cluster center, i.e. μ_1 .
- Initialize each subsequent μ_k as the observation x_i which is the *farthest* away from the cluster it is currently assigned as being closest to of μ_1, \dots, μ_{k-1} .

This initialization ensures the locations μ_k are well spread-out over the dataset and often gives much faster convergence and better final positions.

Thirdly, during the K -means algorithm, it is possible that a cluster μ_k has *no* observations assigned to it. In this case one can either remove the cluster, let it stay at its current location or assign μ_k to the observation which is the furthest away from its closest mean cluster location.

Finally, K -means requires us to choose a suitable K . This is a difficult problem and there is no single agreed-upon solution. One heuristic procedure is to run the K -means algorithm using different choices of K and consider the K where the drop in error *levels off*. This is done for the dataset in fig. 18.4 for $K = 1, \dots, 6$ and the sum-of-squares error can be seen in fig. 18.5, and the figure suggests $K = 4$ where the drop in error levels off.

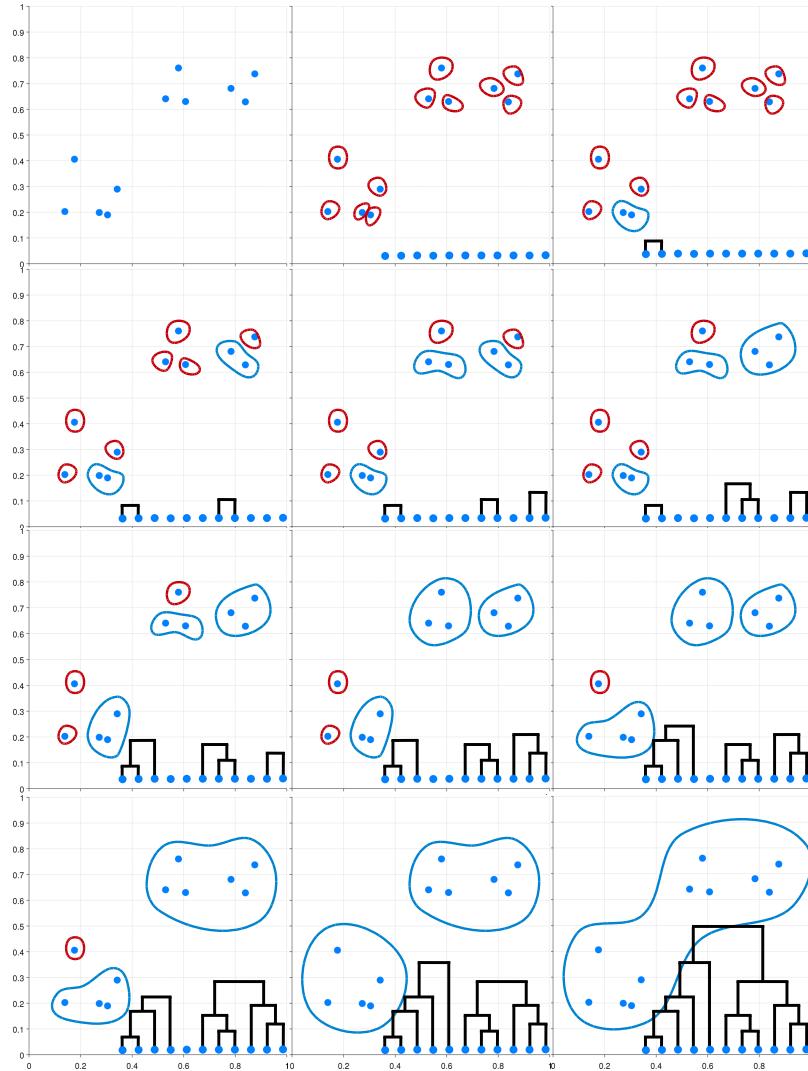


Fig. 18.6. Hierarchical agglomerative clustering applied to the 2D dataset shown in the top right. Each point is assigned to a singleton cluster (top middle), and the closest clusters are then merged until all clusters have been merged. The dendrogram illustrates which clusters are merged in each step and the height of each added clamp is the distance of the two merged clusters.

18.3 Hierarchical agglomerative clustering

A difficulty in K -means clustering was the requirement of finding a single agreed-upon value K . Hierarchical agglomerative clustering overcomes this limitation by instead of finding a *single* K arranging the data in a nested sequence of partitions organized as a hierarchy. The bottom of the hierarchy correspond to the finest partition (each observation is in a unique (singleton) cluster)

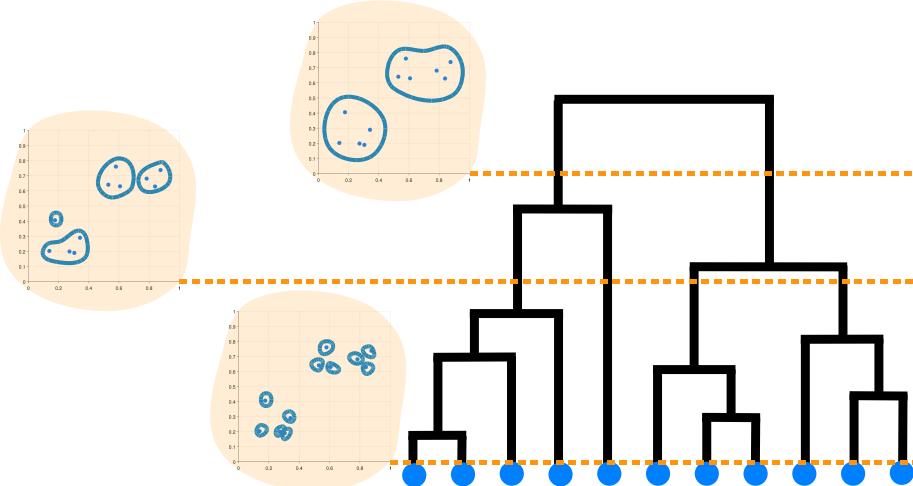


Fig. 18.7. By cutting the dendrogram at different heights, a different number of clusters can be obtained. As we will see, the shape of the dendrogram can be used as an indication of an appropriate cut-height.

whereas the top-level of the hierarchy corresponds to the coarsest possible partition corresponding to putting every observation in the same cluster.

Once again it is easier to show what hierarchical agglomerative clustering does than start with a mathematical definition. Recall *K*-means required a measure of distance between *observations* (the Euclidian distance). Hierarchical agglomerative clustering requires a measure of distance between *groups* of observations. We will later show natural examples, however, for now assume we are given such a measure. Hierarchical agglomerative clustering is then illustrated in fig. 18.6 when it is applied to the dataset shown in the upper-left pane consisting of $N = 11$ observations and proceeds by the following steps:

- Start by placing each observation in a separate group to provide the *coarsest* possible partition (top-middle pane). This correspond to the bottom-layer of the hierarchy shown as an insert.
- Iteratively *merge* the two closest clusters. In the hierarchy, this is indicated by drawing a “clamp” between the corresponding clusters. The *y*-location of the vertical bar in the clamp corresponds to the *distance* of the two clusters.
- Repeat until all observations are merged into a single cluster.

The hierarchy which is constructed is known as a *dendrogram*. The dendrogram is tree-structured and by construction corresponds to a nested sequence of partitions. Since the *y*-location of the vertical bars where clusters are merged indicate their location, the dendrogram can give a visual summary of both the algorithm and the data and is part of why hierarchical clustering is popular. Notice, the hierarchical agglomerative clustering algorithm is deterministic and converges in $N - 1$ steps; to obtain a particular clustering one can *cut* the dendrogram at a given height, see fig. 18.7, and often visual inspection of the dendrogram (in particular where vertical lines are long) can give a visual indication of what corresponds to a good cut, this will be indicated in a moment.

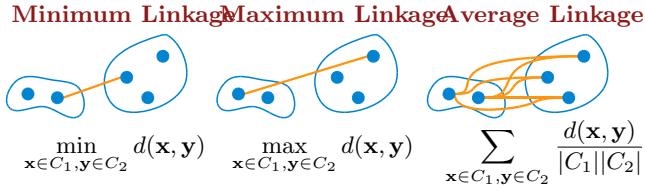


Fig. 18.8. Illustration of the three most popular linkage function, maximum (complete) linkage, minimum (single) linkage and average linkage.

18.3.1 Selecting linkage function

Recall in hierarchical agglomerative clustering, we merged the *closest* clusters in each step. This requires a distance function between *groups* of observations. If we assume we have a distance function between individual observations, for instance just the Euclidian distance $d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$, we can define such a distance function in three ways indicated in fig. 18.8. Consider two groups C_1 and C_2 of observations we can then define the three linkage functions as:

Minimum (or single) linkage Here the distance between the groups is the distance between the *closest* pair of observations

$$d(C_1, C_2) = \min_{\mathbf{x} \in C_1, \mathbf{y} \in C_2} d(\mathbf{x}, \mathbf{y}). \quad (18.5)$$

Maximum (or complete) linkage Here the distance between the groups is the distance between the *most distant* pair of observations

$$d(C_1, C_2) = \max_{\mathbf{x} \in C_1, \mathbf{y} \in C_2} d(\mathbf{x}, \mathbf{y}). \quad (18.6)$$

Average linkage Here the distance between the groups is the *average* distance between all pairs in the two groups

$$d(C_1, C_2) = \frac{\sum_{\mathbf{x} \in C_1, \mathbf{y} \in C_2} d(\mathbf{x}, \mathbf{y})}{|C_1||C_2|}, \quad (18.7)$$

where $|C_1|$ and $|C_2|$ is the number of observations in the two groups.

Ward's method

Another popular choice of linkage function is Ward's method (or simply Ward linkage) which is inspired by the K -means algorithm. Suppose at a given step of the clustering algorithm there are K clusters. We then compute the K centroid vectors $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$ as the mean of each cluster and compute the K -means error already introduced in eq. (18.1):

$$E = \sum_{i=1}^N \sum_{k=1}^K z_{ik} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|_2^2.$$

The two clusters whose merger provides the *smallest* increase in the above error are then merged, see also fig. 18.9.

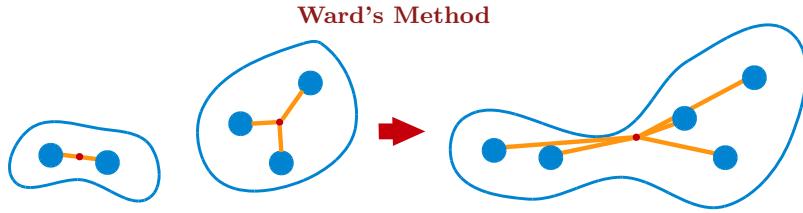


Fig. 18.9. Ward's method for linkage. At each step, the sum-of-squares error of the distance from each observation to its cluster center is computed, and the clusters, which provides the smallest increase in the error, is merged.

Thus, hierarchical agglomerative clustering requires that we select the linkage function from the outset. The choice of linkage function has an important effect on the type of clusters hierarchical agglomerative clustering is good at finding and, correspondingly, what type of clustering hierarchical agglomerative clustering is unsuited for identifying. We will illustrate this with three examples.

In fig. 18.10 we consider a dataset consisting of two half-moons. We apply hierarchical agglomerative clustering, cut the dendrogram at a height corresponding to two clusters, and color the dataset accordingly. Each row shows a linkage function, at the top *maximum* linkage, in the middle *average* linkage and at the bottom *minimum* linkage. As indicated, both average and maximum linkage cannot find the right clusters, whereas minimum linkage does. Why? Minimum linkage (bottom) only cares about the nearest set of observations, thus, it will *chain together* the two moons since each point in any of the moons is closer to another point in the same moon. On the other hand, maximum linkage (top) cares about the *furthest* distance. Thus, it favors clusters which are round and very compact. Average linkage is somewhere in between the two methods and produce clusters which are (roughly) comparable to K -means. If we notice the dendograms, we can see that (visually) the two clusters in the single-linkage (bottom) case stands out, whereas for complete linkage (top) four clusters might be more appropriate. Notice also the relative scale of the dendrogram y -axis. As expected, single linkage merge everything at a much lower height than complete linkage.

This also gives an indication of where minimum linkage may get into problems. Since minimum linkage only cares about the *closest* pairs of observations, if there is a chain of observations between two clusters minimum linkage will use these to *chain together* the two clusters. This is indicated in fig. 18.11. In the top-row, complete linkage (which is focused on compactness) finds the four clusters, whereas in the bottom-row, single linkage fails as there is a slight “chain” of points merging the two right-most clusters. Furthermore, a small group in the bottom-right is slightly further away from the other clusters and is assigned by single linkage its own cluster at this level of the dendrogram. Notice in addition, that the dendrogram for the complete linkage function quite clearly indicates there are four clusters in the dataset (the large vertical gap) whereas for the single linkage function the picture is not so clear.

Finally, consider the dataset comprised of two differently-sized clusters shown in fig. 18.12. For clusters of different size, complete linkage fails because complete linkage, when for instance determining where a point in the middle belongs, cares about the distance to the *edges* of the two point-clouds. Thus, it will try to roughly divide the point-clouds along the middle which in this case is wrong. Single-linkage on the other hand is ideally suited because there are no outliers and a clear gap between the two point-clouds, this is also indicated by the dendograms.

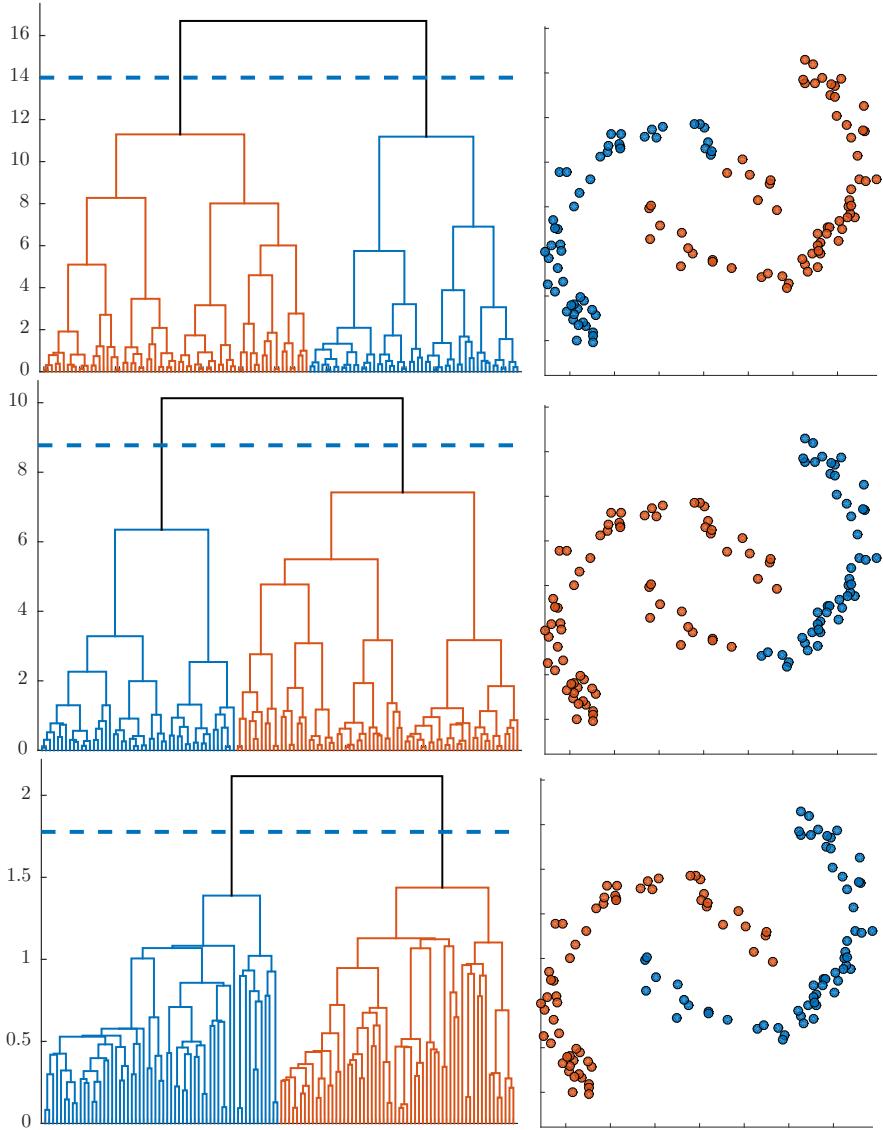


Fig. 18.10. Each row corresponds to hierarchical agglomerative clustering applied to the 2D dataset with different linkage functions. The choices are maximum linkage, average linkage, and minimum linkage. The colors indicate a cut-off corresponding to two clusters. Notice, only minimum linkage solves the problem due to its ability to chain together nearby clusters favoring connected components. Notice also the qualitative difference of the three dendograms.

In conclusion, complete linkage works well when all clusters are roughly round, of equal size or there are outliers in the dataset. It fails when clusters have very different size, are shaped oddly or they are defined by being connected.

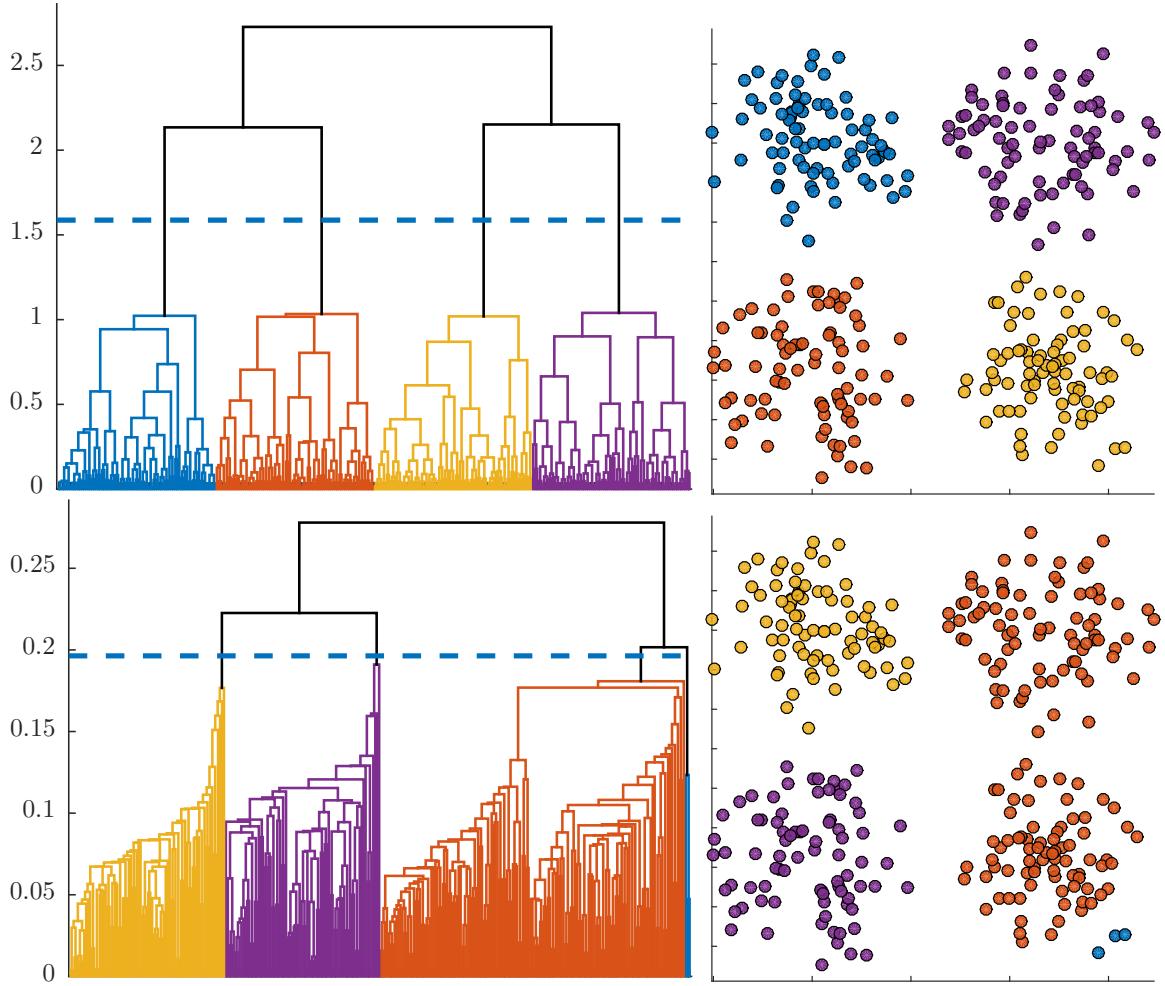


Fig. 18.11. Hierarchical agglomerative clustering applied to 2D dataset with complete/maximum (top) and single/minimum (bottom) linkage. Notice, single linkage is confused by the observations lying between the two right-most clusters, and the outliers, complete linkage is more robust and produce more compact clusters.

Single linkage on the other hand works well for the case where the clusters are internally connected and separated by gaps. It fails when there is outliers in the data or the dataset is otherwise very noisy.

18.4 Comparing partitions

How do we evaluate a partition-based model? We have previously considered this question in a loose fashion when we discussed how to select K in the K -means algorithm, however, without a definite

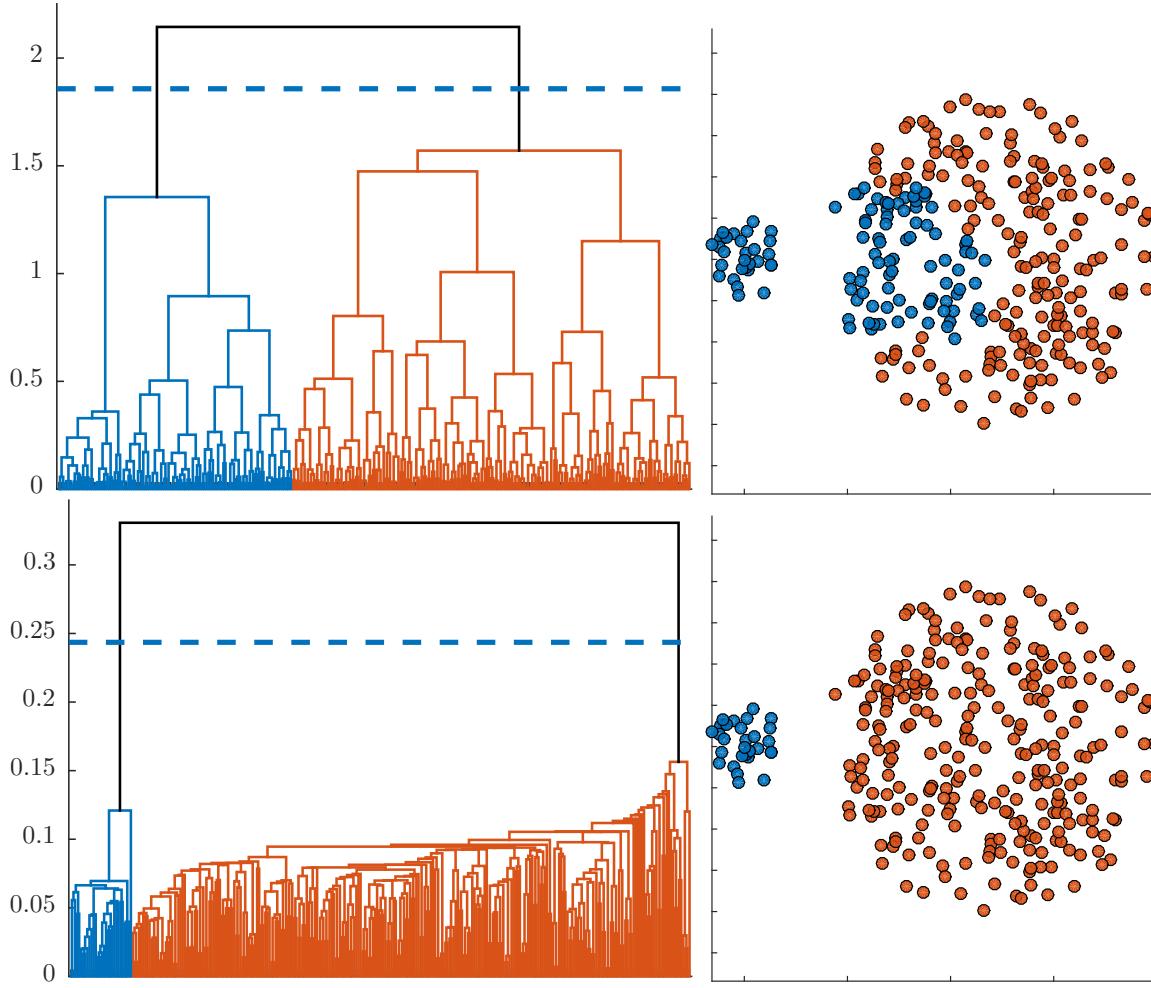


Fig. 18.12. Hierarchical agglomerative clustering applied to 2D dataset with maximum (top) and minimum (bottom) linkage. In this case maximum linkage (top row) tries to produce compact clusters of roughly equal size and thereby incorrectly mixes up the two blobs. Minimum linkage easily solves the problem since the two clusters are spatially separated.

recommendation. It is plausibly the case there *is* no definite way to evaluate a clustering. After all, different people might have different preferences. Consider for instance how different people might group the set of all animals, some may group them according to their utility (pets, domestic animals, dangerous animals, etc.) whereas others might cluster them based on their species and others again based on their behaviour (flying, swimming, crawling, burrowing). In this section, we will not attempt to cut this Gordian knot, but rather suppose we have access to side information (i.e. a true clustering) and determine how we might use this to evaluate our clustering method. The first step in any such procedure is to consider how *different* two clusterings are. This is a necessary

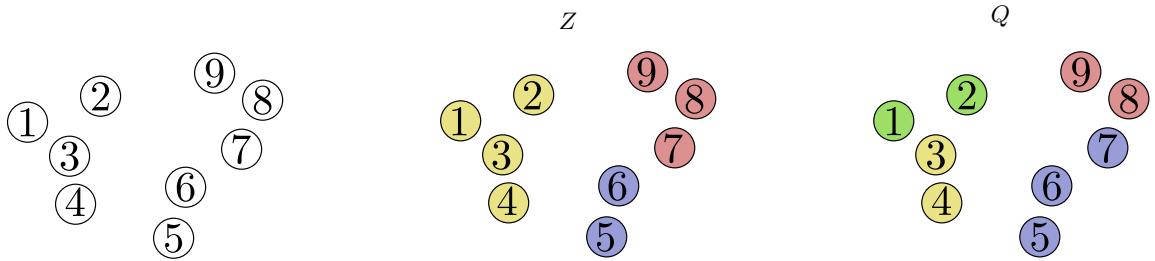


Fig. 18.13. A dataset of $N = 9$ observations are clustered into two partitions Z and Q indicated by the colors. In the case of Z there are $K = 3$ clusters and for Q there are $M = 4$ clusters.

component to any supervised evaluation of a clustering method and so this will be our focus of this section: To produce a proper measure of the difference between a clustering Z and Q . We will use the running example in fig. 18.13 where the $N = 9$ observations are clustered into two partitions Z and Q indicated by the colors.

The example illustrates two problems when comparing partitions. Firstly, that the number of clusters in the two clusterings may be different (consider for instance one clustering obtained by the K -means algorithm with $K = 4$ clusters compared to a true clustering with 3 clusters) and secondly that we have to figure out which cluster in one partition corresponds to which cluster in the other partition. Suppose the observations are labeled by $i = 1, \dots, N$ and the cluster assignments for partition Z is z_1, \dots, z_N such that $z_i = k$ means observation i is in cluster k . Similarly we denote q_1, \dots, q_N as the cluster assignments for Q . We will denote the total number of clusters in Z and Q as K and M respectively.

Example 18.4.1: Encoding partitions

To completely de-mystify the notation, consider the two partition-example in fig. 18.13. Suppose we label the colors 1 (yellow), 2 (blue), 3 (red) and 4 (green), we then have:

$$\begin{aligned} Z &= [1 \ 1 \ 1 \ 1 \ 2 \ 2 \ 3 \ 3 \ 3] \\ Q &= [4 \ 4 \ 1 \ 1 \ 2 \ 2 \ 2 \ 3 \ 3] \end{aligned}$$

An in this case, $K = 3$ and $M = 4$. The particular numbers would refer to the ground-truth class or partition number as obtained by a clustering method. Note whatever method we use to compare partitions should be *invariant to labeling*, meaning that comparing Z and Q should yield the same result as comparing Z and Q' defined as

$$Q' = [10 \ 10 \ 3 \ 3 \ 8 \ 8 \ 8 \ 1 \ 1]$$

It is, however, convenient to assume the clusters are labeled successively 1, 2, 3, ..., that is, the highest value in Z is K and the largest value in Q is M .

Before continuing, we will introduce a few results which can be defined purely from Z and Q . First, recall the delta function is defined as

$$\delta_{hk} = \begin{cases} 1 & \text{if } h = k \\ 0 & \text{otherwise} \end{cases}$$

Therefore, the number of observations in Z which belongs to cluster k can be computed as

$$\{\text{Number of observations in cluster } k \text{ in } Z\} = \sum_{i=1}^N \delta_{z_i, k}$$

More fundamentally, we will define the *joint count matrix* \mathbf{n} as the $K \times M$ matrix defined as:

$$n_{km} = \{\text{Number of observations assigned to cluster } k \text{ in } Z \text{ and } m \text{ in } Q\} \quad (18.8)$$

$$= \sum_{i=1}^N \sum_{j=1}^M \delta_{z_i, k} \delta_{z_j, m} \quad (18.9)$$

Based on this matrix, we can count the number of observations assigned to cluster k in Z (and similarly, m in Q) as:

$$\mathbf{n}^Z = \{\text{Number of observations assigned to cluster } k \text{ in } Z\} = \sum_{m=1}^M n_{km} \quad (18.10)$$

$$\mathbf{n}^Q = \{\text{Number of observations assigned to cluster } m \text{ in } Q\} = \sum_{k=1}^K n_{km} \quad (18.11)$$

In the following, all measures we introduce will be expressed using the joint count matrix \mathbf{n} . That is, if we wish to compute two partitions, this matrix should be what we compute first.

Finally, a surprise counting exercise which will prove very useful. Suppose we wish to count the possible (distinct) pairs we can make out of n observations. To count this, the first observation can be paired to all $n - 1$ other observations, the second to all $n - 2$ (but excluding the first, as we have counted this pair), the third can be paired to $n - 3$ and so on. All in all:

$$\begin{aligned} \{\text{Distinct pairs of } n \text{ observations}\} &= (n - 1) + (n - 2) + \cdots + 2 + 1 + 0 \\ &= \frac{n(n - 1)}{2} \end{aligned} \quad (18.12)$$

Example 18.4.2: Counting matrix, continued

To continue the example in fig. 18.13, for instance $n_{14} = 2$ as observations 1 and 2 are assigned to cluster 1 (yellow) and 4 (green) in Z and Q respectively. Generally, the reader is encouraged to verify:

$$\mathbf{n} = \begin{bmatrix} 2 & 0 & 0 & 2 \\ 0 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Note the horizontal/vertical sums of \mathbf{n} :

$$\mathbf{n}^Z = \begin{bmatrix} 4 \\ 2 \\ 3 \end{bmatrix}, \quad \mathbf{n}^Q = [2 \ 3 \ 2 \ 2]$$

Agree with the number of observations assigned to each cluster.

Finally, we can test our counting result. The number of distinct pairs of *blue* balls in Z are $2 \times (2 - 1) \times \frac{1}{2} = 1$ (which is true, because one unique pair can be made between two balls) and for the yellow balls: $4 \times (4 - 1) \times \frac{1}{2} = 6$, which the reader can verify by counting. A very patient reader can verify the total number of pairs is $\frac{N(N-1)}{2} = 9 \times (9 - 1) \times \frac{1}{2} = 36$.

18.4.1 Rand index

Consider two distinct observations i, j . To say that partition Z is similar to Q is to say that when i and j are placed in the same cluster in partition Z , they will also most often be together in partition Q and vice versa.

To make this more concrete, i, j are both in the same cluster in Z and Q if and only if $\delta_{z_i z_j} = 1$ and $\delta_{q_i q_j} = 1$. Therefore, *both* partition Z and Q agree that i, j are in the same cluster only if

$$1 = \delta_{z_i z_j} \delta_{q_i q_j} = S_{ij},$$

(which is otherwise 0). Similarly, *both* partition Z and Q agree that i, j are *not* in the same cluster only if

$$1 = (1 - \delta_{z_i z_j})(1 - \delta_{q_i q_j}) = D_{ij}.$$

We can then count the total number of times Z and Q agrees two observations are or are not in the same cluster by taking the sum over all *distinct* observations i, j :

$$D = \sum_{i=1}^{N-1} \sum_{j=i+1}^N D_{ij} \quad \text{and} \quad S = \sum_{i=1}^{N-1} \sum_{j=i+1}^N S_{ij}$$

There are a total of $\frac{1}{2}N(N - 1)$ distinct pairs of observations to compare, and so we can define the *Rand similarity* between Z and Q as the relative number of times Z and Q agree on the assignment of observations:

$$R(Q, P) = \frac{S + D}{\frac{1}{2}N(N - 1)}. \tag{18.13}$$

Notice that the way the Rand index both counts matches and non-matches (S and D) makes it somewhat comparable to the SMC.

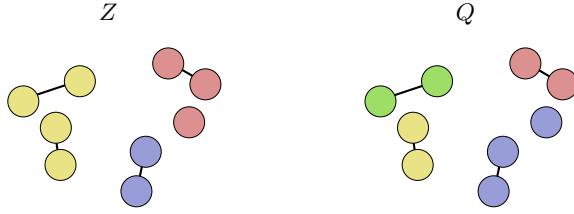


Fig. 18.14. Counting the pairs of observations contributing to S , i.e., pairs of observations the two partitions agree are in the same clusters.

Expressing the Rand index using the counting matrix

We can re-express the Rand index using the counting matrix. First, focus on S , pairs of observations assigned to the same cluster in both partitions. Each number n_{km} represent observations assigned to cluster k in Z and m in Q . We can form $\frac{n_{km}(n_{km}-1)}{2}$ distinct pairs of these and therefore we can conclude:

$$S = \sum_{k=1}^K \sum_{m=1}^M \frac{n_{km}(n_{km}-1)}{2} \cdot \sum_{k=1}^K \sum_{m=1}^M \frac{n_{km}(n_{km}-1)}{2}.$$

To find D , we compute:

$$D = \sum_{i=1}^{N-1} \sum_{j=i+1}^N (1 - \delta_{z_i z_j})(1 - \delta_{q_i q_j}) \quad (18.14)$$

$$= \left[\sum_{i=1}^{N-1} \sum_{j=i+1}^N 1 \right] - \left[\sum_{i=1}^{N-1} \sum_{j=i+1}^N \delta_{z_i z_j} \right] - \left[\sum_{i=1}^{N-1} \sum_{j=i+1}^N \delta_{q_i q_j} \right] + \left[\sum_{i=1}^{N-1} \sum_{j=i+1}^N \delta_{z_i z_j} \delta_{q_i q_j} \right] \quad (18.15)$$

$$= \left[\frac{N(N-1)}{2} \right] - \left[\begin{array}{l} \text{Pairs of observations} \\ \text{in same cluster in } Z \end{array} \right] - \left[\begin{array}{l} \text{Pairs of observations} \\ \text{in same cluster in } Q \end{array} \right] + [S] \quad (18.16)$$

$$= \frac{N(N-1)}{2} - \sum_{k=1}^K \frac{n_k^Z(n_k^Z-1)}{2} - \sum_{m=1}^M \frac{n_m^Q(n_m^Q-1)}{2} + S \quad (18.17)$$

Notice, $R(Q, Q) = R(Z, Z) = 1$ and in general $0 \leq R(Q, P) \leq 1$. In fig. 18.13, if we focus on the Q -partition, the green, yellow and red observations are all in the same blocks in the Z -partition as is one pair of blue observations (see fig. 18.14). Similarly we can count the pairs of observation both partitions agree are not in the same cluster which is illustrated in fig. 18.15. For more details see Example 18.4.3

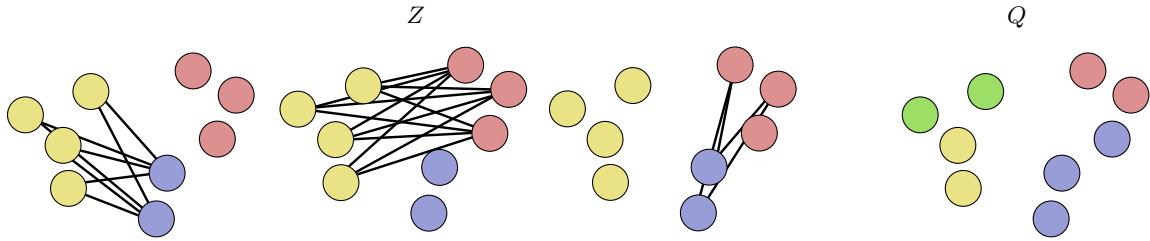


Fig. 18.15. Counting the pairs of observations contributing to D , pairs of observations the two partitions agree are in different clusters (here only shown for partition Z for simplicity).

Example 18.4.3: Rand index, continued

To get the Rand index from Z and Q and the counting matrix, we first compute S to be

$$S = \frac{2(2-1)}{2} + \frac{2(2-1)}{2} + \frac{2(2-1)}{2} + \frac{1(1-1)}{2} + \frac{2(2-1)}{2} = 4$$

Next, to compute D , we compute the two terms involving n^Z and n^Q to be:

$$\sum_{k=1}^K \frac{n_k^Z(n_k^Z - 1)}{2} = 6 + 1 + 3 = 10 \quad (18.18)$$

$$\sum_{m=1}^M \frac{n_m^Q(n_m^Q - 1)}{2} = 1 + 3 + 1 + 1 = 6 \quad (18.19)$$

This allow us to compute

$$D = 36 - 10 - 6 + 4 = 24$$

Finally, we obtain a Rand index of:

$$R(Z, Q) = \frac{4 + 24}{\frac{1}{2}8 \cdot 9} = \frac{7}{9}.$$

18.4.2 Jaccard similarity

A problem with the Rand index is that if there are many clusters, there will typically be many more pairs of observations in different clusters than in the same cluster and so in general we can expect $D \gg S$ which means the Rand index is often close to 1. The reader might notice this problem, and indeed the definition of the Rand index, is very similar to the definition of the *simple matching coefficient* where we also counted the number of times two vectors agreed on the negative and positive matches. We can therefore considered the Jaccard similarity where we disregard the trivial 00 matches:

$$J(Q, P) = \frac{S}{\frac{1}{2}N(N-1) - D} \quad (18.20)$$

Notice it is still the case that $0 \leq J(Q, P) \leq 1$.

Example 18.4.4: Jaccard similarity

We can easily compute the Jaccard similarity as all quantities are known. We get:

$$J(Q, P) = \frac{4}{\frac{1}{2}9 \cdot 8 - 24} = \frac{1}{3}.$$

18.4.3 Comparing partitions using normalized mutual information

Our third measure of cluster similarity is based on the *normalized mutual information*. It is similar to Jaccard similarity and Rand index but theoretically better motivated. Normalized mutual information is based on the idea of quantifying how much *information* one partition provides about the other partition. Recall from our earlier discussion of information theory in section 5.5 all we have to specify to compute the mutual information is a joint distribution $p_{km}(k, m)$ of two variables k and m , and then we can compute the mutual information mechanically (see Box 5.5.1). The two events we are interested in is simply that an observation is assigned to a given cluster, and the joint density corresponds to the event a particular observation is assigned to one cluster in Z and at the same time m in Q . In other words, we simply *define*

$$p_{km}(k, m) = \frac{n_{km}}{N}$$

where n_{km} is the familiar counting matrix. From here, it is all a matter of standard definitions, which have been re-produced in Box 18.4.1 for ease

Method 18.4.1: Information theory

We wish to compare the mutual information of two clusters assignments Z and Q . To do this, we define the probability an observation is assigned in k and m as:

$$p_{km}(k, m) = \frac{n_{km}}{N}, \quad \text{for } k = 1, \dots, K \text{ and } m = 1, \dots, M$$

Based on this matrix, we can define the marginal distributions as the K and M -dimensional vectors:

$$p_k(k) = \sum_{m=1}^M p_{km}(k, m), \quad p_m(m) = \sum_{k=1}^K p_{km}(k, m)$$

The *Entropy* in the 1 and 2d-case is then defines as:

$$H[Z] \equiv H[p_k] = - \sum_{k=1}^K p_k(k) \log p_k(k). \quad H[ZQ] \equiv H[p_{km}] = - \sum_{k=1}^K \sum_{m=1}^M p_{km}(k, m) \log p_{km}(k, m).$$

In both cases, it measures the *complexity* of p_k and p_{km} in *bits*. In addition, the *mutual information* and *normalized mutual information* is defined as:

$$\begin{aligned} \text{MI}[Z, Q] &= \text{MI}[p_{km}] = H[Z] + H[Q] - H[ZQ] \\ \text{NMI}[Z, Q] &= \text{NMI}[p_{km}] = \frac{\text{MI}[Z, Q]}{\sqrt{H[Z]}\sqrt{H[Q]}}. \end{aligned}$$

Where the $\text{NMI}[Z, Q]$ is understood as measuring the overlap of the two partitions.

Example 18.4.5: Mutual information, example

To continue our example fig. 18.13 we can compute the entropy of each partition as:

$$\begin{aligned}\text{Entropy of } Z: H[Z] &= -\frac{4}{9} \log \frac{4}{9} - \frac{1}{3} \log \frac{1}{3} - \frac{2}{9} \log \frac{2}{9} \approx 1.06 \\ \text{Entropy of } Q: H[Q] &= -\frac{2}{9} \log \frac{2}{9} - \frac{2}{9} \log \frac{2}{9} - \frac{2}{9} \log \frac{2}{9} - \frac{1}{3} \log \frac{1}{3} \approx 1.37.\end{aligned}$$

Similarly, the entropy of both partitions is:

$$\begin{aligned}H[p_{ZQ}] = H[ZQ] &= - \sum_{k=1}^K \sum_{m=1}^M p_{ZQ}(k, m) \log p_{ZQ}(k, m) \\ &= -4 \times \frac{2}{9} \log \frac{2}{9} - \frac{1}{9} \log \frac{1}{9} = 1.58.\end{aligned}$$

From this, we can easily compute the Mutual information and Normalized mutual information:

$$\text{MI}[Z, Q] = H[Z] + H[Q] - H[Z, Q] \approx 1.06 + 1.37 - 1.58 \approx 0.85.$$

and

$$\text{NMI}[Z, Q] = \frac{\text{MI}[Z, Q]}{\sqrt{H[Z]}\sqrt{H[Q]}} \approx \frac{0.85}{\sqrt{1.06}\sqrt{1.37}} \approx 0.70.$$

Problems

18.1. Question 1: In Table 18.1 is given the pairwise distances between the four smallest and four largest islands in the Galápagos data. A hierarchical clustering is used to cluster these eight observations using single (i.e., minimum) linkage. Which one of the dendograms given in Figure 18.16 corresponds to the clustering?

	O1	O2	O3	O4	O5	O6	O7	O8
O1	0	2.39	1.73	0.96	3.46	4.07	4.27	5.11
O2	2.39	0	1.15	1.76	2.66	5.36	3.54	4.79
O3	1.73	1.15	0	1.52	3.01	4.66	3.77	4.90
O4	0.96	1.76	1.52	0	2.84	4.25	3.80	4.74
O5	3.46	2.66	3.01	2.84	0	4.88	4.14	2.96
O6	4.07	5.36	4.66	4.25	4.88	0	5.47	5.16
O7	4.27	3.54	3.77	3.80	1.41	5.47	0	2.88
O8	5.11	4.79	4.90	4.74	2.96	5.16	2.88	0

Table 18.1. Pairwise Euclidean distance, i.e $d(Oa, Ob) = \|\mathbf{x}_a - \mathbf{x}_b\|_2 = \sqrt{\sum_m (x_{am} - x_{bm})^2}$, between eight observations of the Galápagos data. Red observations (i.e., O1, O2, O3, and O4) correspond to the four smallest islands whereas blue observations (i.e., O5, O6, O7, and O8) correspond to the four largest islands.

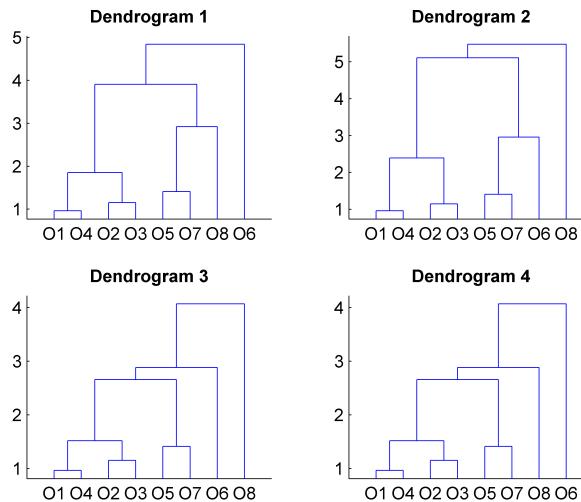


Fig. 18.16. Hierarchical clustering of the eight observations considered in Table 18.1.

- A Dendrogram 1.
- B Dendrogram 2.
- C Dendrogram 3.
- D Dendrogram 4.
- E Don't know.

18.2. Question 2: Consider the simple 1-dimensional data set comprised of $N = 7$ observations as shown in

table 18.2. Suppose we wish to apply K-means clustering to the dataset and the $K = 3$ one-dimensional cluster centers are initialized in $\mu_1 = 4$, $\mu_2 = 7$ and $\mu_3 = 14$. After terminating of the K -means clustering algorithm, what are the final (rounded) cluster centers μ_1, μ_2, μ_3 ?

X	3	6	7	9	10	11	14

Table 18.2. Simple 1-dimensional dataset comprised of $N = 7$ observations.

- A $\mu_1 = 3.00, \mu_2 = 8.00, \mu_3 = 12.50$
- B $\mu_1 = 3.00, \mu_2 = 7.33, \mu_3 = 11.67$
- C $\mu_1 = 4.50, \mu_2 = 9.25, \mu_3 = 14.00$
- D $\mu_1 = 5.33, \mu_2 = 10.00, \mu_3 = 14.00$
- E Don't know.

18.3. Question 3: In table 18.3 is given the pairwise cityblock distances between 8 observations. A hierarchical clustering is used to cluster these nine observations using *group average* linkage. Which of the dendograms shown in fig. 18.17 corresponds to the clustering?

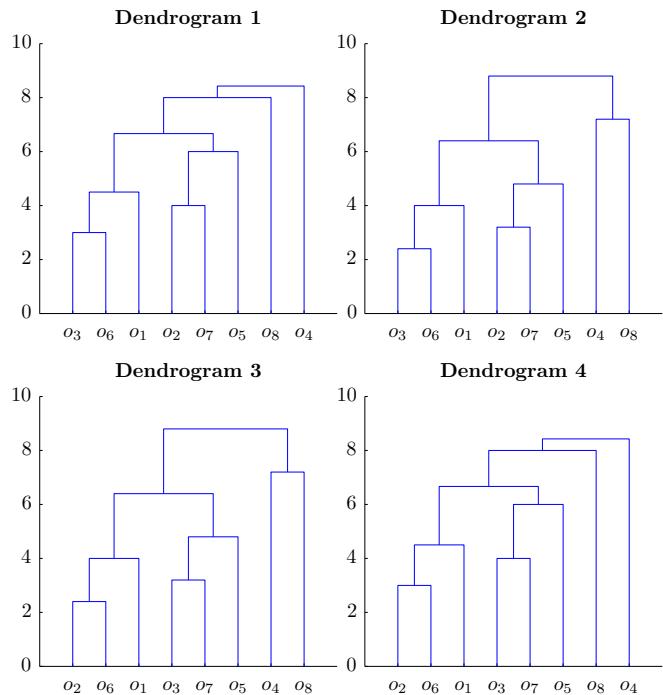


Fig. 18.17. Hierarchical clustering of the 8 observations considered in table 18.3

	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8
o_1	0	4	7	9	5	5	5	6
o_2	4	0	7	7	7	3	7	8
o_3	7	7	0	10	6	6	4	9
o_4	9	7	10	0	8	6	10	9
o_5	5	7	6	8	0	8	6	7
o_6	5	3	6	6	8	0	8	11
o_7	5	7	4	10	6	8	0	7
o_8	6	8	9	9	7	11	7	0

Table 18.3. Pairwise Cityblock distance, i.e. $d(o_i, o_i) = \|\mathbf{x}_i - \mathbf{x}_j\|_1 = \sum_{k=1}^M |x_{ik} - x_{jk}|$, between 8 observations. Each observation o_i corresponds to a $M = 15$ dimensional binary vector, $x_{ik} \in \{0, 1\}$. The blue observations $\{o_1, o_2, o_3, o_4\}$ belong to class C_1 and the black observations $\{o_5, o_6, o_7, o_8\}$ belong to class C_2 .

- A Dendrogram 1.
- B Dendrogram 2.
- C Dendrogram 3.
- D Dendrogram 4.
- E Don't know.

18.4. Question 4: Figure 18.18 contains four dendograms generated according to the distance matrix given in Table 18.4. By thresholding dendrogram 2 we obtain the following clusters

- Cluster 1: A4, B1, B2, B3
- Cluster 2: A1, B4
- Cluster 3: A2
- Cluster 4: A3

Let m_{ij} denote the number of observations of class j in cluster i , $m_i = \sum_j m_{ij}$ denote the number of observations in cluster i , and $m = \sum_i m_i$ denote the total number of observations. Let further $p_{ij} = \frac{m_{ij}}{m_i}$ denote the probability that a member of cluster i belongs to class j . The purity of cluster i is given as $p_i = \max_j p_{ij}$ and the overall purity of a clustering is given as $purity = \sum_{i=1}^K \frac{m_i}{m} p_i$. Let the class an observation belongs to be defined in terms of whether the person considered has a liver disease (i.e., B1, B2, B3 and B4) or not (i.e., A1, A2, A3, and A4). What is the purity of the above clustering?

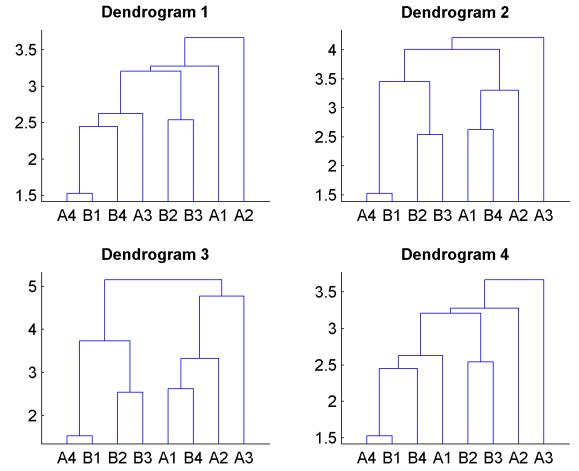


Fig. 18.18. Four dendograms generated according to the distance matrix given in Table 18.4.

	A1	A2	A3	A4	B1	B2	B3	B4
A1	0	3.33	3.73	5.06	4.05	3.76	4.79	2.63
A2	3.33	0	4.77	4.68	3.89	3.72	3.59	3.28
A3	3.73	4.77	0	3.67	3.93	3.86	5.15	4.35
A4	5.06	4.68	3.67	0	1.52	3.64	3.73	3.73
B1	4.05	3.89	3.93	1.52	0	3.21	3.21	2.45
B2	3.76	3.72	3.86	3.64	3.21	0	2.54	3.94
B3	4.79	3.59	5.15	3.73	3.21	2.54	0	4.44
B4	2.63	3.28	4.35	3.73	2.45	3.94	4.44	0

Table 18.4. Euclidean distances between four selected subjects without a liver disease (denoted A1, A2, A3, and A4) and four selected subjects with a liver disease (denoted B1, B2, B3 and B4).

- A purity = $\frac{1}{8}$
- B purity = $\frac{1}{2}$
- C purity = $\frac{2}{3}$
- D purity = $\frac{3}{4}$
- E Don't know.

Mixture models for unsupervised clustering

The goal of density estimation is to describe the probability distribution a given set of observation \mathbf{X} have originated from. Learning probability distributions is relevant in a number of contexts. Consider for instance a standard application of Bayes' theorem

$$p(y = c|\mathbf{x}) = \frac{p(\mathbf{x}|y = c)p(y = c)}{\sum_{c'=1}^C p(\mathbf{x}|y = c')p(y = c')}$$

Applying this to a practical problem involves estimating the C densities $p(\mathbf{x}|y)$. However representing the density can be useful in many other contexts, for instance if we estimate the density of all credit card transactions, a credit card transaction \mathbf{x} having low value $p(\mathbf{x})$ is then equivalent to an *unusual* credit card transaction which may warrant further investigation. In this chapter we will focus on probabilistic estimation of densities using the *Gaussian mixture-model* and a particular simple way to train the Gaussian mixture-model known as the *Expectation maximization* (EM) algorithm.

Mixture models were first considered around the middle of the 19th century and their explicit statement is usually attributed to the biostatistician Karl Pearson who used mixture models to analyse the length of crabs [Pearson, 1894]. The EM algorithm was first named by Dempster et al. [1977], however, ideas reminiscent of the EM algorithm has been used in different contexts before this.

19.1 The Gaussian mixture model

The goal of the Gaussian mixture-model (GMM) is to derive a distribution for an M -dimensional vector $\mathbf{x} \in \mathbb{R}^M$ which we will write as $p(\mathbf{x})$. We wish this distribution to be potentially very flexible and a common strategy for obtaining this in a tractable manner is to make $p(\mathbf{x})$ be a combination of simpler, more tractable elements. First recall the definition of the multivariate normal distribution introduced earlier in chapter 13. The multivariate normal distribution for an M -dimensional vector is defined as the density:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})},$$

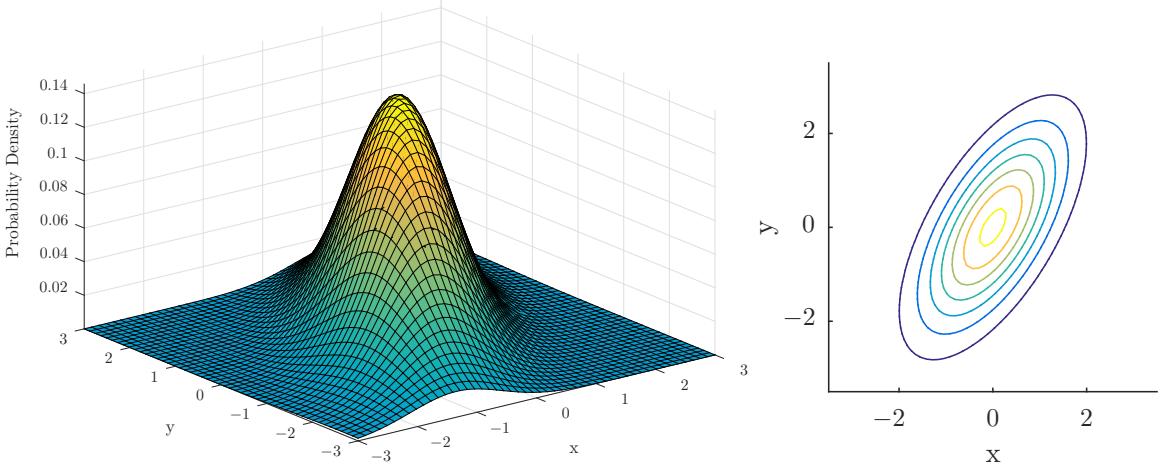


Fig. 19.1. Example of the probability density function of a 2-dimensional multivariate normal distribution. In left it is plotted as a function of $\mathbf{x} = [x \ y]^T$, i.e. $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, whereas on the right the same distribution is shown as a contour plot.

where $\boldsymbol{\mu} \in \mathbb{R}^M$ is the mean and $\boldsymbol{\Sigma}$ is the $M \times M$ covariance matrix. An example is given in fig. 19.1 corresponding to the multivariate normal distribution

$$\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix} \quad \text{and} \quad \boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

In the Gaussian mixture-model (GMM) we want to use the multivariate normal distribution as a building block to create a more flexible distribution. Suppose K is an integer for instance $K = 4$. Let's imagine we select an integer $1, \dots, K$ at random and we denote the event we selected k with the binary variable $z_k = 1$ and the event we did not select k as $z_k = 0$. For instance

$$\mathbf{z} = [0 \ 1 \ 0 \ 0]^T,$$

is the event we selected $k = 2$. Then \mathbf{z} is a binary vector where only one entry can be non-zero at a time. Suppose the probability we select k is π_k :

$$P(\text{We select option } k) = \pi_k,$$

then a little thought reveals that

$$p(\mathbf{z}) = \prod_{k=1}^K \pi_k^{z_k}. \tag{19.1}$$

Why? Well if we take the above example, we get:

$$p(\mathbf{z} = [0 \ 1 \ 0 \ 0]^T) = \prod_{k=1}^K \pi_k^{z_k} = \pi_1^0 \pi_2^1 \pi_3^0 \pi_4^0 = \pi_2,$$

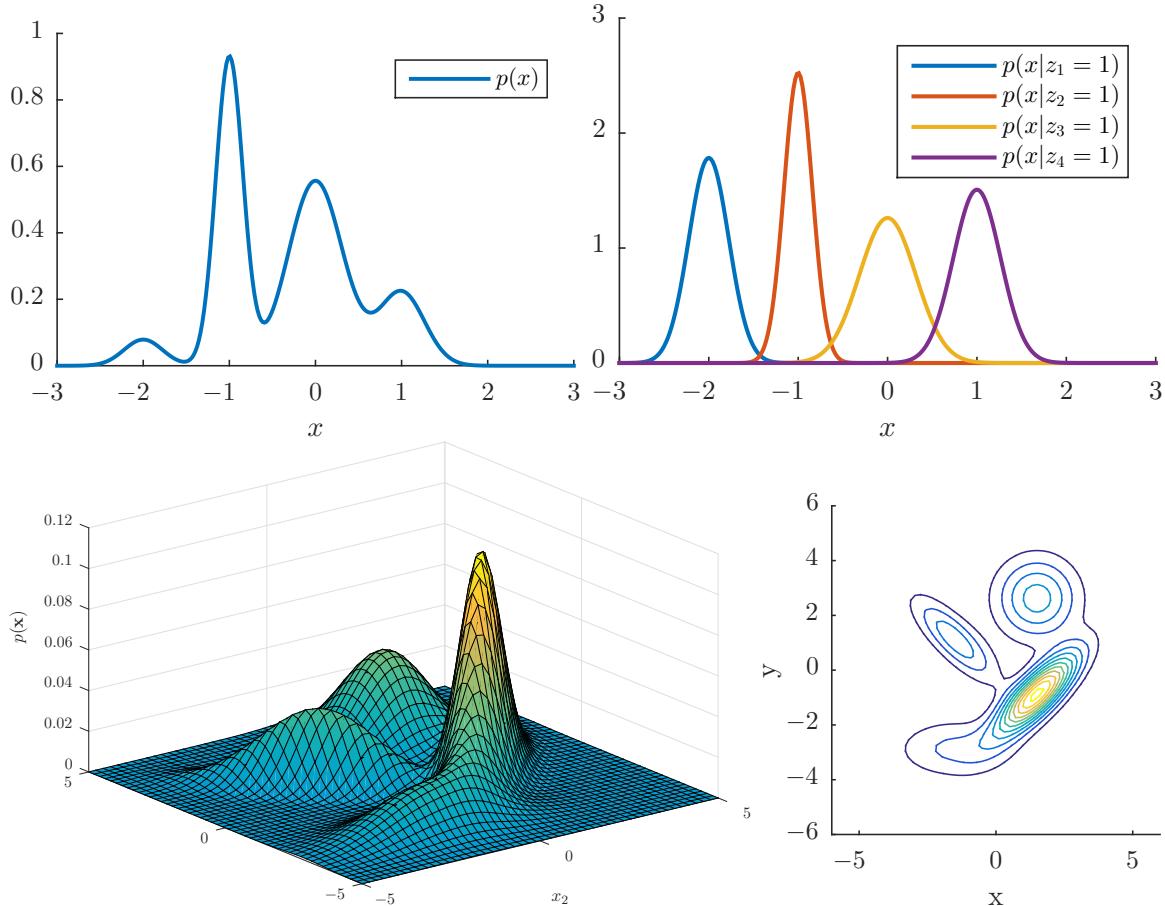


Fig. 19.2. Top row: One-dimensional Gaussian mixture model example with $K = 4$ mixture components. In the left column is shown the density, $p(x)$, and in the right pane the $K = 4$ individual mixture components. Notice the weights scale the mixture components in the density. In the lower pane is shown a 2D Gaussian mixture model, also with $K = 4$, both as a 3D surface plot and as a contour plot.

as we should expect. We now imagine that when we know what k is, for instance $k = 2$, then we know what distribution \mathbf{x} has, specifically we assume it is a multivariate normal distribution with parameters $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$. To put this in symbols

$$p(\mathbf{x}|z_k = 1) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

We can once again write this in the simpler form

$$p(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^K N(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}, \quad (19.2)$$

since for instance,

Algorithm 8: Expectation-Maximization algorithm

-
- 1: Initialize $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ and π_k for $k = 1, \dots, K$
 - 2: **while** The likelihood \mathcal{L} changes **do**
 - 3: Update $\gamma_{ik} = \frac{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\pi_k}{\sum_{k'=1}^K \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})\pi_{k'}} \text{ (E-step)}$
 - 4: Update the parameter values in this order (where $N_k = \sum_{i=1}^N \gamma_{ik}$): (M-step)
 - 5: $\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{ik} \mathbf{x}_i$
 - 6: $\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T$
 - 7: $\pi_k = \frac{N_k}{N}$
 - 8: Compute the likelihood $\mathcal{L} = \sum_{i=1}^N \log \left[\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right]$
 - 9: **end while**
-

$$\begin{aligned} p(\mathbf{z} = [0 \ 1 \ 0 \ 0]^T) &= \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)^0 \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)^1 \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)^0 \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_4, \boldsymbol{\Sigma}_4)^0 \\ &= \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2). \end{aligned}$$

We are actually done! Distribution $p(\mathbf{x})$ can then be found by the sum and product rule of probability theory. In particular, using eq. (19.1) and eq. (19.2), it must be the case

$$\begin{aligned} p(\mathbf{x}) &= \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z}} p(\mathbf{x} | \mathbf{z}) p(\mathbf{z}) \\ &= \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \end{aligned} \tag{19.3}$$

The normal distributions in the GMM is known as the *mixture components* and the values π_k are known as the *weights*. In fig. 19.2 is shown two examples of a Gaussian mixture model. The top row is an $M = 4$ mixture component example used to represent the density of a single real number x . In the right-pane the individual mixture components are plotted; notice the height does not correspond to their height in the GMM since they are scaled with π_k . In the bottom row is shown the same $M = 4$ GMM as a surface and contour plot.

19.2 The EM algorithm

The GMM is a general and flexible way to represent continuous densities, but without a useful way to train the GMM it is not very useful. The *Expectation Maximization* (EM) algorithm provides an elegant method for finding the parameters of a GMM which approximates a dataset \mathbf{X} of N observations. To state what the EM algorithm tries to accomplish it is convenient to introduce the following symbols for the parameters:

$$\begin{aligned} \boldsymbol{\pi} &= [\pi_1 \cdots \pi_K] \\ \boldsymbol{\mu} &= \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K\} \\ \boldsymbol{\Sigma} &= \{\boldsymbol{\Sigma}_1, \dots, \boldsymbol{\Sigma}_K\} \end{aligned}$$

the *objective* of the EM algorithm is then to find the values of the parameters $\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}$ which *maximizes* the log of the likelihood of the data

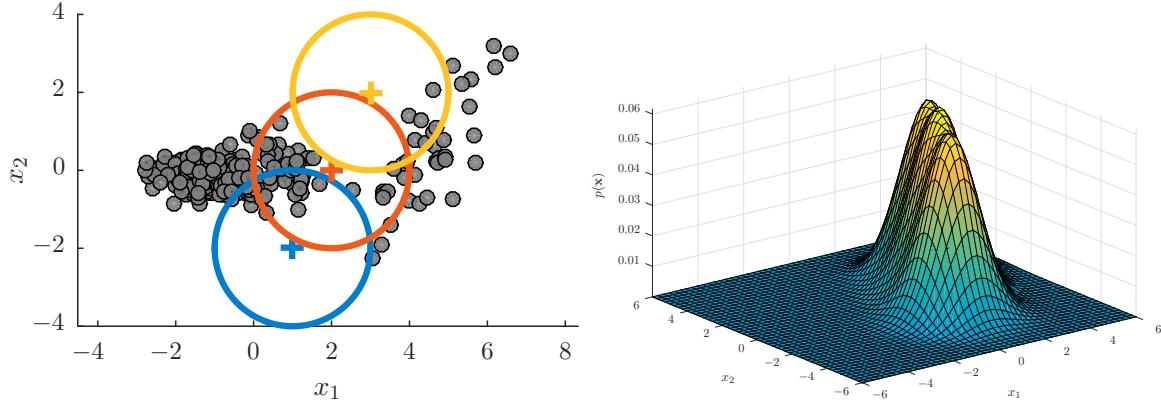


Fig. 19.3. The initialization step of the EM algorithm when applied to the 2D dataset shown as the gray points. The $K = 3$ mixture components are shown as a contour plot in the left pane, and in the right pane as a surface plot. The colored circles represent the area capturing twice the standard deviation of each mixture component.

$$\mathcal{L}(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \log p(\mathbf{X} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{i=1}^N \log p(\mathbf{x}_i | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{i=1}^N \log \left[\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right]. \quad (19.4)$$

This could be accomplished using gradient descent, which we encountered earlier in chapter 15, however, the EM algorithm takes advantage of the particular form of the problem to provide a much more effective method. We will first state what the EM algorithm does and later provide an argument for why the EM algorithm works.

First some notation: For a given data point \mathbf{x}_i , the probability \mathbf{x}_i belongs to component k can be computed with (as usual) Bayes theorem:

$$\begin{aligned} p(z_k = 1 | \mathbf{x}_i) &= \frac{p(\mathbf{x}_i | z_k = 1)p(z_k = 1)}{\sum_{k'=1}^K p(\mathbf{x}_i | z_{k'} = 1)p(z_{k'} = 1)} \\ &= \frac{\mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\pi_k}{\sum_{k'=1}^K \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})\pi_{k'}} = \gamma_{ik} \end{aligned} \quad (19.5)$$

We can then define the “total mass” of a component k as $N_k = \sum_{i=1}^N \gamma_{ik}$. Notice $N = \sum_{k=1}^K N_k$ because

$$\sum_{k=1}^K N_k = \sum_{k=1}^K \sum_{i=1}^N p(z_k = 1 | \mathbf{x}_i) = \sum_{i=1}^N \left[\sum_{k=1}^K p(z_k = 1 | \mathbf{x}_i) \right] = \sum_{i=1}^N 1 = N.$$

Since γ_{ik} denotes the probability observation i belongs to cluster k , we can define the *empirical mean*, the *empirical covariance* and the *empirical mass* of the clusters as

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{ik} \mathbf{x}_i, \quad \boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T, \quad \text{and} \quad \pi_k = \frac{N_k}{N}. \quad (19.6)$$

The idea behind the EM algorithm can thus be summarized as:

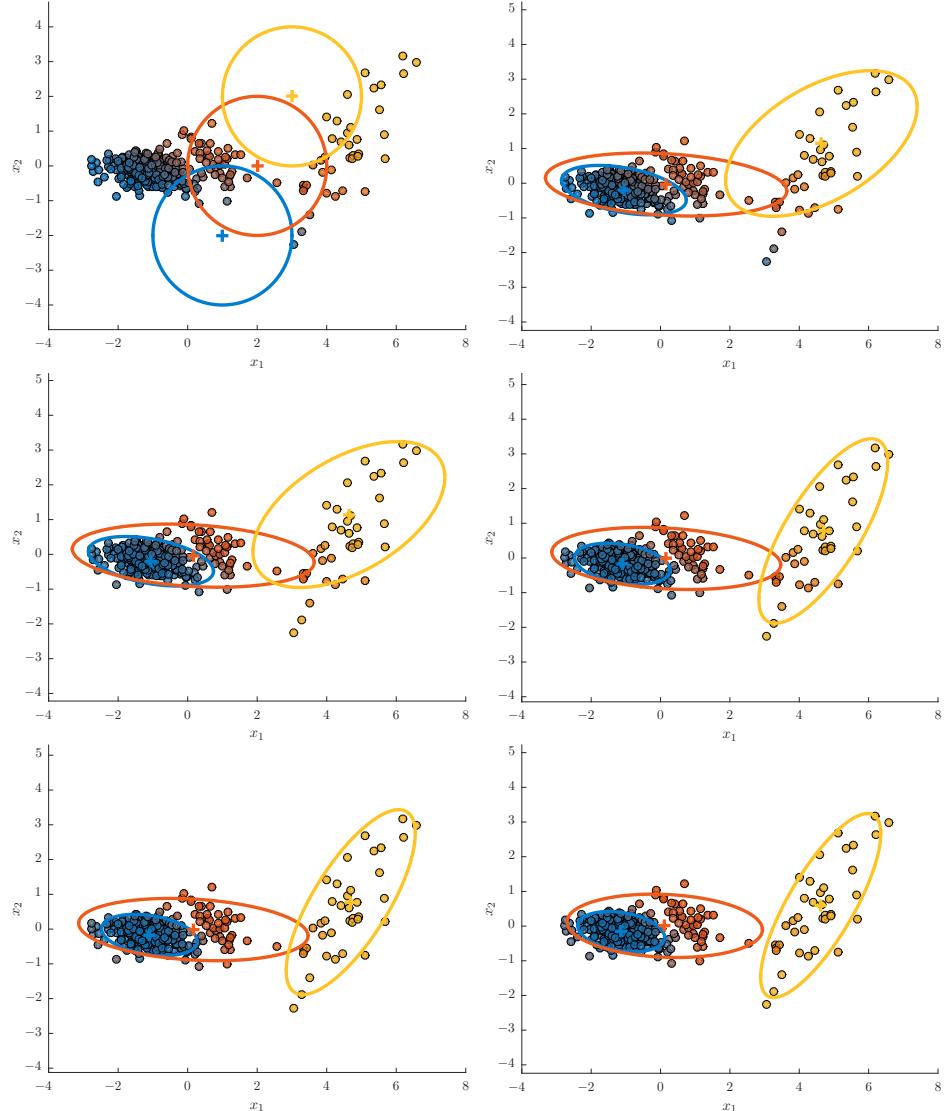


Fig. 19.4. Each row corresponds to the first three steps of the EM algorithm and each column to the E-step and M-step. In the top-left pane, the observations are assigned to the three clusters in the E-step. The top-right pane shows the M-step where the parameters π_k , Σ_k and μ_k are updated based on the assignments. This continues for two additional steps.

Initialize: First we initialize μ , Σ and π

Expectation step: Compute γ_{ik} for all i, k

Maximization step: Update μ_k , Σ_k and π_k

Iterate: Repeat the two previous steps until the likelihood eq. (19.4) does not change.

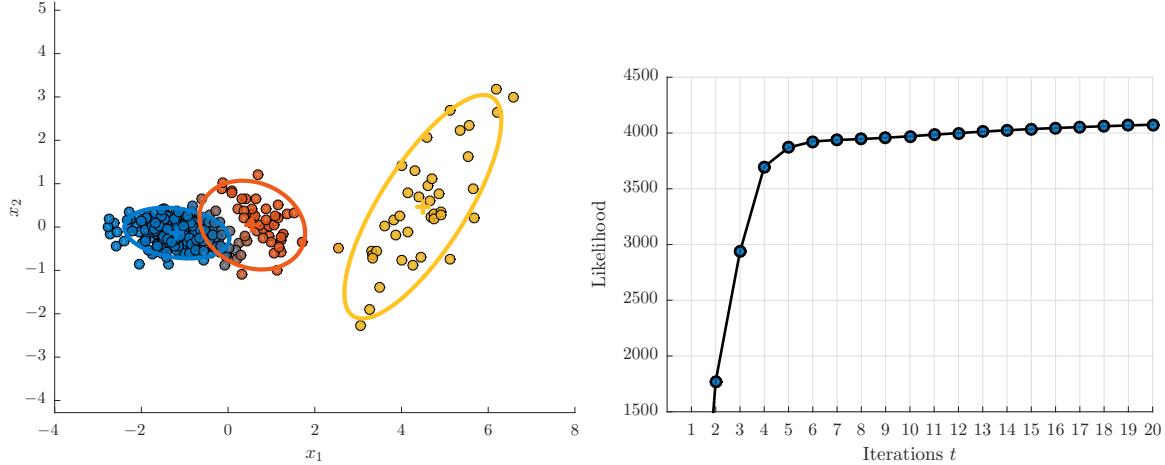


Fig. 19.5. Running the EM algorithm for 20 iterations produces the above partitioning. The likelihood is plotted in the right-pane. As can be seen, the likelihood continues to increase but at a diminishing rate.

Putting these steps together we obtain the EM algorithm as given in algorithm 8. To illustrate the EM algorithm, consider the 2d dataset shown in fig. 19.3 with the given initialization of clusters. In fig. 19.4 is shown the first three steps of the EM algorithm. The left-most column corresponds to the E-step and the right-most column to the M-step with the assignments to clusters γ_{ik} being indicated by the colors. Furthermore, the EM algorithm maximizes the likelihood and in fig. 19.5 is shown the 20th step of the EM algorithm and the log likelihood.

19.2.1 Why the EM algorithm works*

The above presentation leaves two important questions unanswered. Firstly, the steps of the EM algorithm might appear as arising from nothing and secondly, why should we believe the EM algorithm works? To begin with the later question, what the EM algorithm tries to do is to maximize the log of the likelihood of the data $\mathcal{L}(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$, and the way we will show this is simply by showing that both the *M*-step and the *E*-step increases the log-likelihood. Secondly, we will see the EM algorithm can be derived from more general considerations which also applies to other models.

To begin, suppose we collect the parameters $\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}$ into the symbol $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$. Recall according to eq. (19.3) each observation \mathbf{x}_i comes with a latent (binary) vector \mathbf{z}_i that indicates which mixture component x_i belongs to. That is, if $\mathbf{z}_{ik} = 1$ then \mathbf{x}_i belongs to component k and we write:

$$p(\mathbf{x}_i | \boldsymbol{\theta}) = \sum_{\mathbf{z}_i} p(\mathbf{x}_i, \mathbf{z}_i | \boldsymbol{\theta}) = \sum_{\mathbf{z}_i} p(\mathbf{x}_i | \mathbf{z}_i, \boldsymbol{\theta}) p(\mathbf{z}_i) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \quad (19.7)$$

If we collect all the \mathbf{z}_i 's in an $N \times K$ matrix \mathbf{Z} we can therefore write:

$$p(\mathbf{X} | \boldsymbol{\theta}) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z} | \boldsymbol{\theta}),$$

where $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = \prod_{i=1}^N p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta})$. We can now proceed with a little algebra. Recall that the log of the likelihood, which we wish to maximize, is simply $\mathcal{L}(\mathbf{X}|\boldsymbol{\theta}) = \log P(\mathbf{X}|\boldsymbol{\theta})$ and the later probability can be re-written using the basic rules of probability:

$$\log p(\mathbf{X}|\boldsymbol{\theta}) = \log \frac{p(\mathbf{X}|\boldsymbol{\theta})P(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})}{P(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})} = \log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) - \log p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}) \quad (19.8)$$

Suppose we consider any other setting of the parameters $\boldsymbol{\theta}^{\text{old}}$. We can then (at least symbolically) write up the distribution $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$ and taking the expectation of both sides of eq. (19.8) gives (the left-hand side is not affected by the expectation because it is independent of \mathbf{Z}):

$$\log p(\mathbf{X}|\boldsymbol{\theta}) = \mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})] - \mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})]. \quad (19.9)$$

Now to the quite amazing thing. First, it follows from Jensen's inequality¹ that

$$\mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})] \leq \mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})]. \quad (19.10)$$

In other words, considering only the last term the log-likelihood becomes as small as possible if $\boldsymbol{\theta} = \boldsymbol{\theta}^{\text{old}}$. Let's connect this to the EM algorithm. Suppose $\boldsymbol{\theta}^{\text{old}}$ is the value of $\boldsymbol{\theta}$ at a given step of the algorithm. Suppose then we select $\boldsymbol{\theta}$ as the value that maximize the *first* term in eq. (19.9):

$$\boldsymbol{\theta} = \arg \max_{\boldsymbol{\theta}} \mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})] \quad (19.11)$$

Using eq. (19.9) we then have that for *this* $\boldsymbol{\theta}$:

$$\text{By eq. (19.11)} : \mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})] \geq \mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}^{\text{old}})] \quad (19.12)$$

$$\text{By eq. (19.10)} : -\mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})] \geq -\mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})] \quad (19.13)$$

Using these two expression on each term in eq. (19.9) we have now shown

$$\log p(\mathbf{X}|\boldsymbol{\theta}) = \mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})] - \mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})] \quad (19.14)$$

$$\geq \mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}^{\text{old}})] - \mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})] \quad (19.15)$$

$$= \log p(\mathbf{X}|\boldsymbol{\theta}^{\text{old}}) \quad (19.16)$$

¹ Recall that Jensen's inequality says that for any concave function ϕ (and the logarithm is a concave function) and densities p it holds that: $\mathbb{E}_{p(x)} [\phi(f(x))] \leq \phi (\mathbb{E}_{p(x)} [f(x)])$. Then, for any density $r(x)$ it holds

$$E_{p(x)} [\log p(x)] = E_{p(x)} \left[\log \frac{p(x)}{r(x)} \right] + E_{p(x)} [\log r(x)] = -E_{p(x)} \left[\log \frac{r(x)}{p(x)} \right] + E_{p(x)} [\log r(x)].$$

By Jensen's inequality the first term is always less than 0 because $-E_{p(x)} \left[\log \frac{r(x)}{p(x)} \right] \geq -\log E_{p(x)} \left[\frac{r(x)}{p(x)} \right] = -\log \sum_x r(x) = -\log 1 = 0$. Applying this to the right-hand side of the above equation we get:

$$E_{p(x)} [\log p(x)] = -E_{p(x)} \left[\log \frac{r(x)}{p(x)} \right] + E_{p(x)} [\log r(x)] \geq E_{p(x)} [\log r(x)].$$

The result now follows by replacing x with \mathbf{Z} , p with $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$ and r with $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})$.

In other words, choosing $\boldsymbol{\theta}$ to maximize $\mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})]$ in eq. (19.11) also maximize the log-likelihood $\mathcal{L}(\mathbf{X}|\boldsymbol{\theta})$. How is this connected with the EM algorithm? Firstly, the posterior $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$ exactly corresponds to γ_{ik} computed in the E-step using eq. (19.5). We can then examine what happens in the maximization-step eq. (19.11) more closely by noticing:

$$\begin{aligned}\mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} [\log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})] &= \mathbb{E}_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})} \left[\sum_{i=1}^N \log p(\mathbf{x}_i, z_i|\boldsymbol{\theta}) \right] \\ &= \sum_{i=1}^N \sum_{k=1}^K \gamma_{ik} \log p(\mathbf{x}_i, z_i = 1|\boldsymbol{\theta}) \\ &= \sum_{i=1}^N \sum_{k=1}^K \gamma_{ik} \log [\pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)] \\ &= \sum_{k=1}^K N_k \log \pi_k + \sum_{i=1}^N \sum_{k=1}^K \gamma_{ik} \log \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\end{aligned}$$

We leave it to the reader to show that differentiating these expressions by the parameter we are interested in maximizing and setting the derivative equal to zero results in exactly the M -step updates given in section 19.2.

19.2.2 Some problems with the EM algorithm

The EM algorithm is guaranteed to always increase the log likelihood $\mathcal{L}(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$, however, this does not mean the EM algorithm is guaranteed to be well-behaved. Firstly, what values of $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ the EM algorithm converges to depends upon the initialization; this is similar to K-means clustering but in general the increased flexibility of the EM algorithm for GMMs increases this problem. Secondly, the EM algorithm may exhibit divergent behaviour. If a mixture component k is centered upon a single observation, and this is the only observation for which γ_{ik} is large, the EM algorithm may diverge in the sense the cluster becomes more and more peaked around this observation; in other words, the EM algorithm diverges. To compensate for this one can add a regularization term to $\boldsymbol{\Sigma}_k$ as

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T + \lambda \mathbf{I}$$

where $\lambda > 0$ is the regularization term. This difficulty increases with poor initialization and it is therefore recommended to initialize the EM algorithm to the output of the K-means clustering algorithm. Third, the EM algorithm in its present form requires

$$\underbrace{(K-1)}_{\boldsymbol{\pi}} + \underbrace{KM}_{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K} + \underbrace{K(M+1)M/2}_{\boldsymbol{\Sigma}_1, \dots, \boldsymbol{\Sigma}_K}$$

parameters; for high-dimensional datasets the number $K(M+1)M/2$ can be brought down by considering a diagonal covariance matrix to KM . There is however also goods news with regards to the EM algorithm for GMMs. Asides accomplishing the primary objective, a general density estimator which can be fitted efficiently, an advantage of the EM algorithm over K -means is that one can select K using cross-validation.

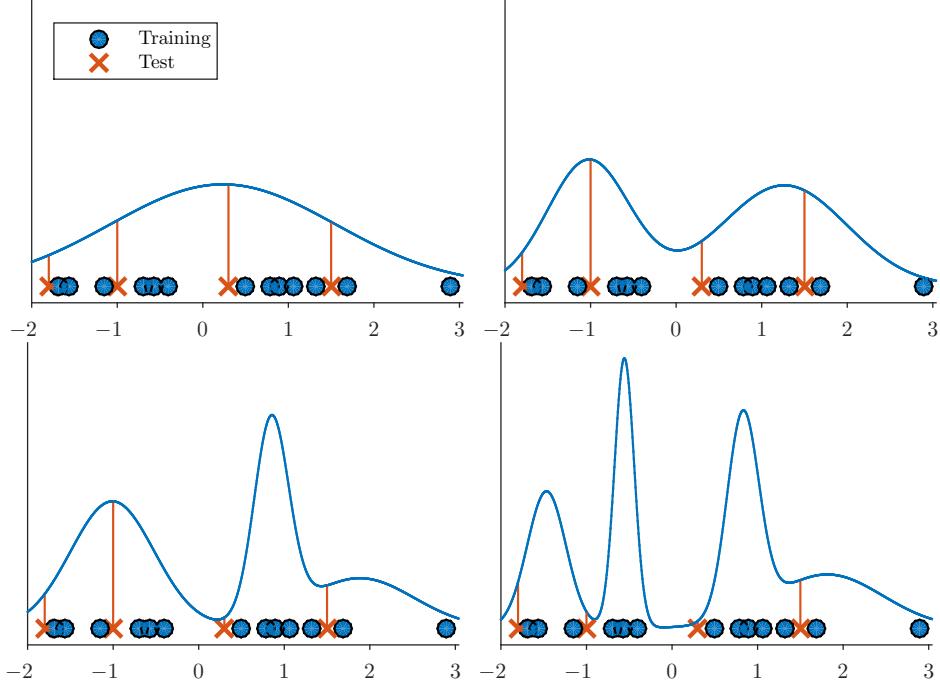


Fig. 19.6. Example of four GMMs fitted to a small dataset comprised of $N = 13$ training observations and $N^{\text{test}} = 4$ test-observations indicated by the red crosses. The figure illustrates how the GMM begins to overfit the data as the number of mixture components is increased. The test log-likelihood is shown in fig. 19.7.

19.2.3 Selecting K for the GMM using Cross-validation

As opposed to the K -means algorithm, the GMM provides a natural way to select K . Since the goal of fitting a GMM using for instance the EM algorithm is to maximize the log-likelihood, it is natural to quantify the predictive performance in terms of the log-likelihood measured on a test set \mathbf{X}^{test} :

$$\mathcal{L}^{\text{test}}(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \log p(\mathbf{X}^{\text{test}} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) \quad (19.17)$$

$$= \sum_{i=1}^{N^{\text{test}}} \log \left[\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_i^{\text{test}} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right]. \quad (19.18)$$

One can then apply cross-validation using $-\mathcal{L}^{\text{test}}(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ as an error measure to select the number of mixture components K . This procedure is illustrated in fig. 19.6 where four different GMMs corresponding to $K = 1, 2, 3, 4$ is fitted until convergence on a small 1d dataset comprised of $N = 13$ training observations and $N^{\text{test}} = 4$ test-observations indicated by the red crosses. As seen, the GMM begins to overfit as K becomes large leading to reduced test log-likelihood ($\mathcal{L}^{\text{test}}$) plotted in fig. 19.7. In a similar fashion, cross-validation could also be used to select the regularization parameter λ .

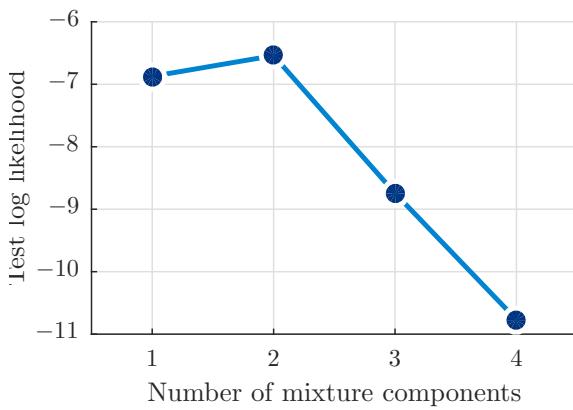


Fig. 19.7. Test log-likelihood as evaluated on the four test observations and four values of K , $K = 1, 2, 3, 4$ shown in fig. 19.6.

Problems

19.1. Question 1:

Let

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{2\pi|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$

define the multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. In Figure 19.8 is given 5000 observations drawn from a density defined by a Gaussian Mixture Model (GMM) with three clusters.

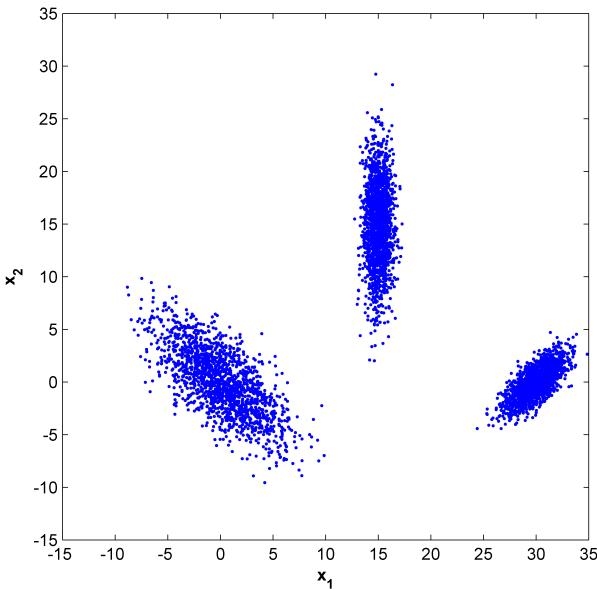


Fig. 19.8. 5000 data observations drawn from a Gaussian Mixture Model (GMM) with three clusters.

Which one of the following GMM densities was used to generate the data?

A

$$\begin{aligned} p(x) = & \frac{1}{3} \cdot \mathcal{N}(\mathbf{x} | \begin{bmatrix} 15 \\ 15 \end{bmatrix}, \begin{bmatrix} 0.5 & 0 \\ 0 & 15 \end{bmatrix}) \\ & + \frac{1}{3} \cdot \mathcal{N}(\mathbf{x} | \begin{bmatrix} 30 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 & -1.4 \\ -1.4 & 2 \end{bmatrix}) \\ & + \frac{1}{3} \cdot \mathcal{N}(\mathbf{x} | \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 10 & 7 \\ 7 & 10 \end{bmatrix}) \end{aligned}$$

B

$$\begin{aligned} p(x) = & \frac{1}{3} \cdot \mathcal{N}(\mathbf{x} | \begin{bmatrix} 15 \\ 15 \end{bmatrix}, \begin{bmatrix} 0.5 & 0 \\ 0 & 15 \end{bmatrix}) \\ & + \frac{1}{3} \cdot \mathcal{N}(\mathbf{x} | \begin{bmatrix} 30 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 & 1.4 \\ 1.4 & 2 \end{bmatrix}) \\ & + \frac{1}{3} \cdot \mathcal{N}(\mathbf{x} | \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 10 & -7 \\ -7 & 10 \end{bmatrix}) \end{aligned}$$

C

$$\begin{aligned} p(x) = & \frac{1}{3} \cdot \mathcal{N}(\mathbf{x} | \begin{bmatrix} 15 \\ 15 \end{bmatrix}, \begin{bmatrix} 0.5 & 0 \\ 0 & 15 \end{bmatrix}) \\ & + \frac{1}{3} \cdot \mathcal{N}(\mathbf{x} | \begin{bmatrix} 30 \\ 0 \end{bmatrix}, \begin{bmatrix} 10 & -7 \\ -7 & 10 \end{bmatrix}) \\ & + \frac{1}{3} \cdot \mathcal{N}(\mathbf{x} | \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 & 1.4 \\ 1.4 & 2 \end{bmatrix}) \end{aligned}$$

D

$$\begin{aligned} p(x) = & \frac{1}{3} \cdot \mathcal{N}(\mathbf{x} | \begin{bmatrix} 15 \\ 15 \end{bmatrix}, \begin{bmatrix} 15 & 0 \\ 0 & 0.5 \end{bmatrix}) \\ & + \frac{1}{3} \cdot \mathcal{N}(\mathbf{x} | \begin{bmatrix} 30 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 & 1.4 \\ 1.4 & 2 \end{bmatrix}) \\ & + \frac{1}{3} \cdot \mathcal{N}(\mathbf{x} | \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 10 & -7 \\ -7 & 10 \end{bmatrix}) \end{aligned}$$

E Don't know.

19.2. Question 2:

Which one of the following statements pertaining to clustering is *correct*?

A k-means and Gaussian Mixture Models are guaranteed to find the same solutions regardless of initialization.

B The level at which clusters merge in the dendrogram in hierarchical clustering using minimum/single-, maximum/complete- or group average linkage can be determined by the proximities between all the observations.

C In k-means the cluster centers are updated as the average of the observations belonging to the cluster regardless of the distance measure used.

D A Gaussian Mixture Model with diagonal covariance matrix has the same number of free parameters as k-means.

E Don't know.

19.3. Question 3:

Suppose the points in the scatter plot fig. 19.9 was generated from a Gaussian mixture-model (GMM) of the form

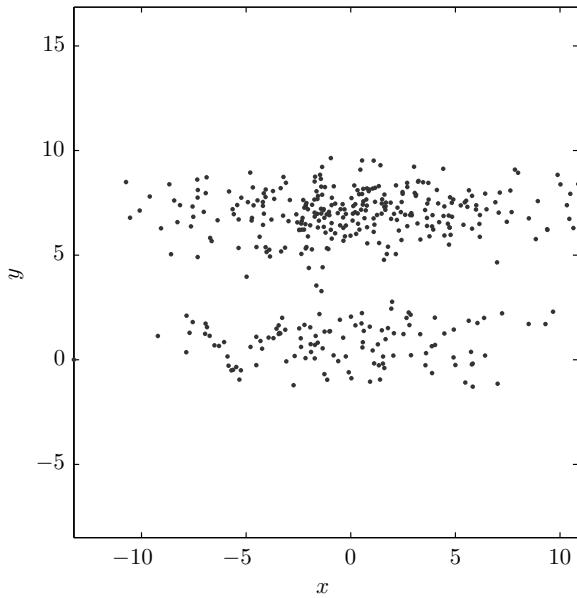


Fig. 19.9. Scatter plot of observations

$$p(x, y) = \sum_{i=1}^2 w_i \mathcal{N} \left(\begin{bmatrix} x \\ y \end{bmatrix} ; \begin{bmatrix} 0 \\ \mu_i \end{bmatrix}, \begin{bmatrix} \sigma_i^2 & 0 \\ 0 & \delta_i^2 \end{bmatrix} \right).$$

and suppose $\mu_1 = 7, \mu_2 = 1$. Which of the following is most likely to be true?

- A $w_1 = 0.5, \sigma_1^2 = 2\sigma_2^2, \delta_1^2 = 2\delta_2^2$
- B $w_1 = 0.7, \delta_1^2 > \sigma_2^2$
- C $w_1 = 0.7, \sigma_1^2 = 20, \delta_2^2 = 1$
- D $w_1 = 0.5, p(0, 0) < p(0, 7)$
- E Don't know.

Density estimation

Anomaly detection attempts to find observations that can be regarded as different from the other observations. A tempting way to put this is we should consider an observation anomalous when it lies in a *low density* region of the data, i.e. a region where we would consider it unexpected to find an observation. We will therefore mainly regard anomaly detection as a problem of estimating the density of a dataset and then obtaining the (candidate) outliers is simply a matter of finding the lowest-density observations.

The GMM can be regarded as the primary density-estimation tool for our disposal, however, for very large datasets the GMM might be too expensive to fit. An additional problem with the GMM is that it is affected by initialization and, as we will see in a moment, can have difficulties treating regions of different density leading to potentially spurious results. It is therefore useful to consider approximate, deterministic methods for density estimation which are more robust. In this section, we will consider two such approaches: *kernel density estimation*, which is an approximation to the GMM, and *Average relative density* which is an entirely separate method not based on probabilities.

Kernel density estimation was separately discovered by Murray Rosenblatt and Emanuel Parzen [Rosenblatt et al., 1956, Parzen, 1962], meanwhile the section on the average relative density is based on Tan et al. [2013].

20.1 The kernel density estimator

A problem with the Gaussian mixture-model is that it simply contains many parameters to be fitted and selecting different values of these parameters (or re-starting the EM algorithm from different initial configurations) will lead to different assignment of density. A *kernel density estimator* (KDE) is best seen as a deterministic approximation to the Gaussian mixture model which tries to overcome some of these limitations. Recall the density of a GMM with K components can be written as:

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \quad (20.1)$$

where $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are all parameters to be tuned. When we apply a KDE to a dataset \mathbf{X} of N observations we simply assume the GMM consists of $K = N$ components centered on top of each data point and with diagonal covariance matrix. In other words, we select

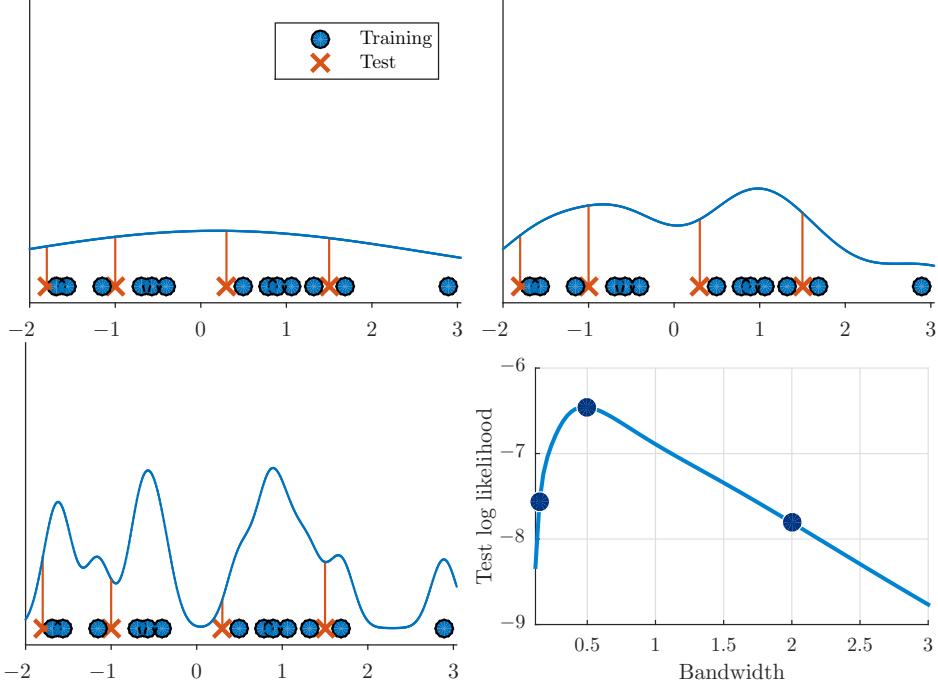


Fig. 20.1. Example of a KDE fitted to a small dataset comprised of $N = 13$ training observations and $N^{\text{test}} = 4$ test-observations indicated by the red crosses. The figure illustrates how the KDE overfits as the kernel parameter λ is varied as $\lambda = 2, 0.5, 0.15$. The last pane shows the test log-likelihood.

$$\pi_k = \frac{1}{N}, \quad \mu_k = \mathbf{x}_k \quad \text{and} \quad \Sigma_k = \lambda^2 \mathbf{I}$$

where λ is known as the *kernel width*. This gives a density of the form:

$$p_\lambda(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N \mathcal{N}(\mathbf{x} | \mathbf{x}_i, \lambda^2 \mathbf{I}). \quad (20.2)$$

20.1.1 Selecting the kernel width λ

We can select λ similar to how we selected K for the GMM namely by using cross-validation. Consider a test set \mathbf{X}^{test} , we can then similar to the GMM consider the log of the likelihood of the test data:

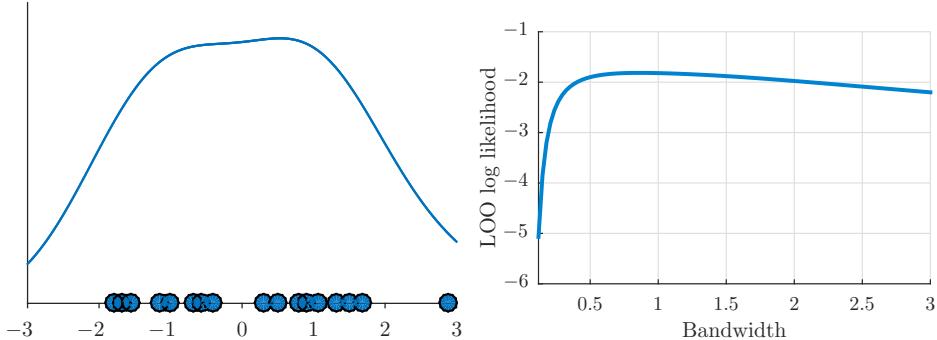


Fig. 20.2. LOO estimation of the kernel width parameter λ for the full dataset comprised $N = 17$ observations along with the KDE corresponding to the best bandwidth.

$$\begin{aligned}\mathcal{L}(\lambda) &= \log p(\mathbf{X}^{\text{test}} | \lambda) \\ &= \sum_{j=1}^{N^{\text{test}}} \log p(\mathbf{x}_j^{\text{test}} | \lambda) \\ &= \sum_{j=1}^{N^{\text{test}}} \log \left[\frac{1}{N} \sum_{i=1}^N \mathcal{N}(\mathbf{x}_j^{\text{test}} | \mathbf{x}_i, \lambda^2 \mathbf{I}) \right]\end{aligned}$$

A simple example using a dataset of $N = 13$ observations and a test set of $N^{\text{test}} = 4$ observations can be found in fig. 20.1. Another advantage of the KDE over the GMM is that leave-one-out cross-validation can be carried out very quickly: For each pair of observations we can pre-compute $M_{ij} = \mathcal{N}(\mathbf{x}_i | \mathbf{x}_j, \lambda^2 \mathbf{I})$ once and re-use them in the computation of the leave-one-out estimate of the log of the likelihood as:

$$\mathcal{L}(\lambda) = \frac{1}{N} \sum_{i=1}^N \log \left[\sum_{j \neq i} \frac{1}{N-1} M_{ij} \right].$$

The leave-one-out estimate of the log of the likelihood for the full dataset comprised of $N = 17$ observations can be found in fig. 20.2 along with KDE corresponding to the highest log likelihood according to the LOO estimator. LOO estimation is only one of several ways of selecting the kernel widths, an interested reader can consult Raykar and Duraiswami [2006] for other approaches.

Uneven densities and the GMM

Let us turn to a problem for which the KDE or GMM may not be suitable. In fig. 20.3 we have shown a simple 2d dataset comprised of two clusters of data of very uneven density and two candidate outliers indicated by the red circles. Comparing the two outliers, the right-most candidate outlier is clearly further away from its nearest neighbours, however, it also lies in a region of relatively lower density. The KDE is unable to make use of this difference in density as it uses the same kernel width for the entire dataset and therefore tends to consider the right-most point a better candidate outlier as shown in fig. 20.4 (top row) for different choices of the kernel width ($\lambda = 0.2, 1, 4$). In the bottom row we have shown the density obtained by applying the GMM for $K = 1, 2, 4$. As shown,

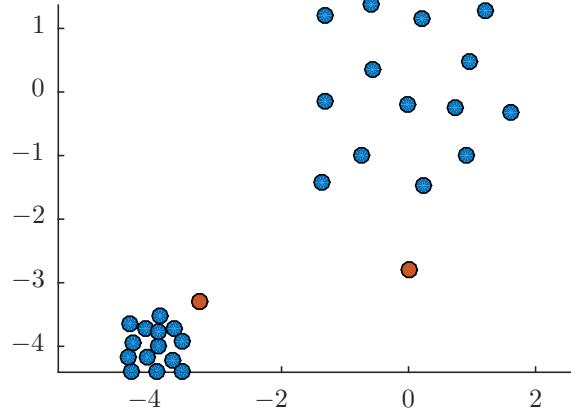


Fig. 20.3. A simple 2d dataset comprised of two clusters of data of very uneven density and two candidate outliers indicated by the red circles. The right-most candidate outlier is clearly further away from its nearest neighbours, however, it also lies in a region of relatively lower density. Which of the two should we suspect are outliers?

the GMM rapidly begins to overfit the data and we should therefore cross-validate to select K . In addition, the use of specific cluster centers may lead to artificially high-density regions as shown by the elongated oval shape in the plot for $K = 4$. As a rule, this makes the GMM more flexible for fitting densities and good at handling densities which are elongated along one or more directions (i.e. elliptical densities), but also somewhat prone to spurious behaviour due to the particulars of how the EM algorithm decided to place the cluster centers during a particular run. Thus, when using GMMs for outlier detection it is important the number of components be carefully determined (i.e., using cross-validation) and naturally, the observations which we wish to examine as being potential outliers should not be part of the data used for training the GMM density.

20.2 Average relative density

The *average relative density* (ARD) tries to overcome the difficulty we saw in the earlier section where the KDE or GMM was unable to handle clusters of different densities well. However, the ARD is also different from the KDE or GMM in that it does not rely on probabilities.

Recall the definition of the K nearest neighbourhood of a point \mathbf{x} given in eq. (12.1) from chapter 12, i.e. the K observations in the dataset \mathbf{X} which are *closest* to \mathbf{x} :

$$N_{\mathbf{X}}(\mathbf{x}, K) = \{\text{The } K \text{ observations in } \mathbf{X} \text{ which are nearest to } \mathbf{x}\}. \quad (20.3)$$

The average distance to the K nearest neighbours is given by

$$\frac{1}{K} \sum_{\mathbf{x}' \in N_{\mathbf{X}}(\mathbf{x}, K)} d(\mathbf{x}, \mathbf{x}'),$$

where d is the relevant distance measure for our dataset, for instance the Euclidian distance $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$. Intuitively, if the average distance to the nearest neighbours is low, that

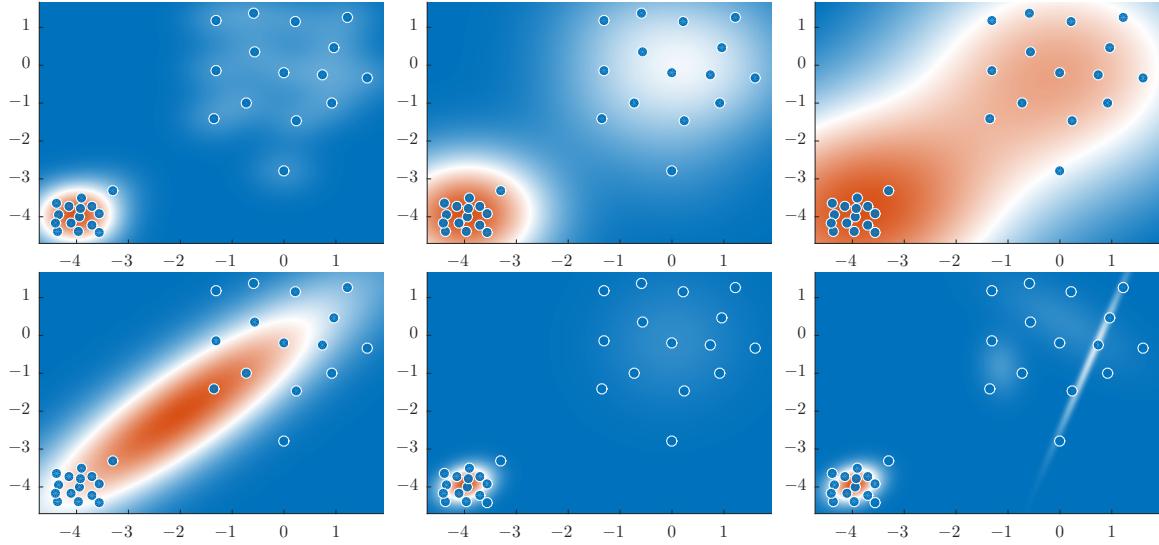


Fig. 20.4. (top row:) The kernel density estimator applied to a 2d dataset for different settings of the kernel width. From left to right we have plotted $\lambda = 0.2, 1, 4$. (bottom row:) Density as estimated by the GMM for $K = 1, 2, 4$ components and initialized using the K -means algorithm. The second component for $K = 2$ is quite faint and the GMM rapidly begins to overfit for $K > 2$

means there are many observations close to \mathbf{x} and so the *density* at \mathbf{x} is high and contrary, if the average distance is high, the density is low. It thus makes sense to define the density around \mathbf{x} computed by K neighbours as the inverse of the average distance

$$\text{density}_{\mathbf{X}}(\mathbf{x}, K) = \frac{1}{\frac{1}{K} \sum_{\mathbf{x}' \in N_{\mathbf{X}}(\mathbf{x}, K)} d(\mathbf{x}, \mathbf{x}')}. \quad (20.4)$$

Suppose for a dataset \mathbf{X} we wish to evaluate the density of observation i , \mathbf{x}_i , of the dataset. Obviously, we don't want to include \mathbf{x}_i as a member of the K -neighbourhood because that would bias the density upwards. Imagine if $K = 1$, then obviously $N_{\mathbf{X}}(\mathbf{x}_i, K) = \{\mathbf{x}_i\}$ because \mathbf{x}_i is in \mathbf{X} and so $\text{density}_{\mathbf{X}}(\mathbf{x}_i, K) = \frac{1}{\frac{1}{1} d(\mathbf{x}_i, \mathbf{x}_i)} = \frac{1}{0} = \infty$. Rather in the case where we wish to compute the density of an observation \mathbf{x}_i from \mathbf{X} we therefore use:

$$\text{density}_{\mathbf{X}_{\setminus i}}(\mathbf{x}_i, K) = \frac{1}{\frac{1}{K} \sum_{\mathbf{x}' \in N_{\mathbf{X}_{\setminus i}}(\mathbf{x}_i, K)} d(\mathbf{x}_i, \mathbf{x}')}, \quad (20.5)$$

where, as in eq. (12.6), $\mathbf{X}_{\setminus i}$ is simply \mathbf{X} with observation i removed:

$$\mathbf{X}_{\setminus i}^T = [\mathbf{x}_1 \ \mathbf{x}_2 \cdots \mathbf{x}_{i-2} \ \mathbf{x}_{i-1} \ \mathbf{x}_{i+1} \ \mathbf{x}_{i+2} \cdots \ \mathbf{x}_N].$$

One could use the (estimated) density directly, however, if we are looking for outliers it is perhaps more relevant still to look for those points where the density is lower than what it typically is for surrounding points. This is exactly what the average relative density (ard) attempts to accomplish

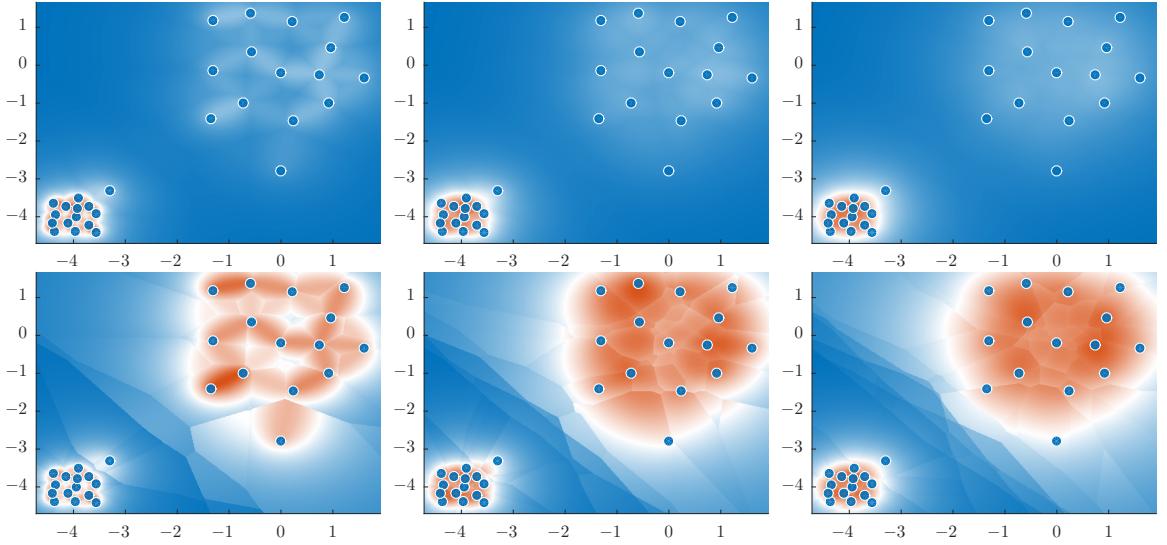


Fig. 20.5. (top row:) Density plotted for the 2d dataset shown in fig. 20.3 for $K = 2, 4, 6$. The density relies only on the average distances and so considers all the top-right points to be anomalous. (bottom row:) The ard takes the relative density into account and therefore do not consider the low-density cluster to be anomalous but rather considers the left-most candidate outlier to be far more suspicious.

by considering the density of a given point \mathbf{x} *relative* to the *average* of the density of the K nearest neighbours $x_j \in N_{\mathbf{X}}(\mathbf{x}, K)$ of \mathbf{x} where we use eq. (20.5) to estimate the density of each \mathbf{x}_j :

$$\text{ard}_{\mathbf{X}}(\mathbf{x}, K) = \frac{\text{density}_{\mathbf{X}}(\mathbf{x}, K)}{\frac{1}{K} \sum_{\mathbf{x}_j \in N_{\mathbf{X}}(\mathbf{x}, K)} \text{density}_{\mathbf{X}_{\setminus j}}(\mathbf{x}_j, K)}. \quad (20.6)$$

It is instructive to consider what this definition means for $K = 1$. In this case we first find the one observation in \mathbf{X} closest to \mathbf{x} namely $N_{\mathbf{X}}(\mathbf{x}, K) = \{\mathbf{x}_j\}$ and then we simply compute the relative density:

$$\text{ard}_{\mathbf{X}}(\mathbf{x}, 1) = \frac{\text{density}_{\mathbf{X}}(\mathbf{x}, 1)}{\text{density}_{\mathbf{X}_{\setminus j}}(\mathbf{x}_j, 1)}.$$

Suppose further that the observation in \mathbf{X} closest to \mathbf{x}_j (but which is not \mathbf{x}_j itself) is $N_{\mathbf{X}_{\setminus j}}(\mathbf{x}_j, 1) = \{\mathbf{x}_k\}$. In this case the above becomes:

$$\text{ard}_{\mathbf{X}}(\mathbf{x}, 1) = \frac{\frac{1}{d(\mathbf{x}, \mathbf{x}_j)}}{\frac{1}{d(\mathbf{x}_j, \mathbf{x}_k)}} = \frac{d(\mathbf{x}_j, \mathbf{x}_k)}{d(\mathbf{x}, \mathbf{x}_j)}$$

That is, if \mathbf{x} is closer to its nearest neighbour \mathbf{x}_j than it's nearest neighbour \mathbf{x}_j is to its nearest neighbour \mathbf{x}_k then the ard is high and vice versa.

Finally, if we wish to compute the ard for an already existing observation \mathbf{x}_i in \mathbf{X} then similar to eq. (20.5) we have to exclude that observation from \mathbf{X} to not bias the ard upwards. That is, we should use:

$$\text{ard}_{\mathbf{X}}(\mathbf{x}_i, K) = \frac{\text{density}_{\mathbf{X}_{\setminus i}}(\mathbf{x}_i, K)}{\frac{1}{K} \sum_{\mathbf{x}_j \in N_{\mathbf{X}_{\setminus i}}(\mathbf{x}_i, K)} \text{density}_{\mathbf{X}_{\setminus j}}(\mathbf{x}_j, K)}. \quad (20.7)$$

The result of plotting the density and the ard can be seen in the top and bottom rows of fig. 20.5. The top row illustrates the density for $K = 2, 4, 6$ and the bottom row the ard for the same choices of K . We see how the density marks all the points in the low-density region as outliers, however, the ard is able to take into account they are in a low-density region and consider the left-most candidate outlier far more likely to be anomalous.

Problems

20.1. Question 1:

	O1	O2	O3	O4	O5	O6	O7
O8	5.11	4.79	4.90	4.74	2.96	5.16	2.88

Table 20.1. Pairwise Euclidean distance, i.e $d(Oa, Ob) = \|\mathbf{x}_a - \mathbf{x}_b\|_2 = \sqrt{\sum_m (x_{am} - x_{bm})^2}$, between observation O8 and observation O1–O7 given in Table 20.2.

We would like to quantify if O8 is an outlier using a Gaussian kernel density estimator where we use the seven observations O1, O2, ..., O7 to estimate the density at observation O8 based on the Euclidean distances given in Table 20.2 and reproduced in terms of observation O8 in Table 20.1. The Gaussian kernel density estimator is given by

$$p(x) = \frac{1}{N} \sum_{n=1}^N \frac{1}{(2\pi)^{M/2} \sqrt{|\sigma^2 \mathbf{I}|}} \exp \left(-\frac{1}{2} (\mathbf{x} - \mathbf{x}_n)^\top \left(\frac{1}{\sigma^2} \mathbf{I} \right) (\mathbf{x} - \mathbf{x}_n) \right)$$

Thus, $N = 7$ and in our analysis we will use $\sigma = 1$ and as the dataset is 7-dimensional we have that $M=7$. We note that $|\sigma^2 \mathbf{I}|$ is the determinant of the diagonal matrix with σ^2 in the diagonal. For $\sigma = 1$ we have $|\sigma^2 \mathbf{I}| = 1$. What is the density at observation O8 using only observations O1–O7 in the above Gaussian kernel density estimator

	O1	O2	O3	O4	O5	O6	O7	O8
O1	0	2.39	1.73	0.96	3.46	4.07	4.27	5.11
O2	2.39	0	1.15	1.76	2.66	5.36	3.54	4.79
O3	1.73	1.15	0	1.52	3.01	4.66	3.77	4.90
O4	0.96	1.76	1.52	0	2.84	4.25	3.80	4.74
O5	3.46	2.66	3.01	2.84	0	4.88	1.41	2.96
O6	4.07	5.36	4.66	4.25	4.88	0	5.47	5.16
O7	4.27	3.54	3.77	3.80	1.41	5.47	0	2.88
O8	5.11	4.79	4.90	4.74	2.96	5.16	2.88	0

Table 20.2. Pairwise Euclidean distance, i.e $d(Oa, Ob) = \|\mathbf{x}_a - \mathbf{x}_b\|_2 = \sqrt{\sum_m (x_{am} - x_{bm})^2}$, between eight observations of the Galápagos data. Red observations (i.e., O1, O2, O3, and O4) correspond to the four smallest islands whereas blue observations (i.e., O5, O6, O7, and O8) correspond to the four largest islands.

- A $3.4 \cdot 10^{-32}$
- B $1.3 \cdot 10^{-8}$
- C $6.5 \cdot 10^{-6}$
- D $5.1 \cdot 10^{-2}$
- E Don't know.

20.2. Question 2: We suspect that observation O1 may be an outlier. In order to assess if this is the case we would like to calculate the average relative

KNN density based on the observations given in Table 20.3 only. We recall that the KNN density and average relative density for the observation \mathbf{x} are given by: $\text{density}(\mathbf{x}, K) = \left(\frac{1}{K} \sum_{y \in N(\mathbf{x}, K)} \text{distance}(\mathbf{x}, \mathbf{y}) \right)^{-1}$,

$$\text{a.r.d.}(\mathbf{x}, K) = \frac{\text{density}(\mathbf{x}, K)}{\frac{1}{K} \sum_{y \in N(\mathbf{x}, K)} \text{density}(\mathbf{y}, K)},$$

where $N(\mathbf{x}, K)$ is the set of K nearest neighbors of observation \mathbf{x} and $\text{a.r.d.}(\mathbf{x}, K)$ is the average relative density of \mathbf{x} using K nearest neighbors. Based on the data in Table 20.3, what is the average relative density for observation O1 for $K = 2$ nearest neighbors?

	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10
O1	0	393.5	68.1	165.4	271.8	200.6	210.9	206.1	166.3	365.0
O2	393.5	0	411.3	361.8	478.6	490.9	409.2	382.3	391.1	37.4
O3	68.1	411.3	0	119.8	208.4	136.6	152.8	154.3	111.1	387.1
O4	165.4	361.8	119.8	0	137.5	130.8	62.1	44.7	32.5	346.2
O5	271.8	478.6	208.4	137.5	0	99.0	76.8	101.0	116.4	468.5
O6	200.6	490.9	136.6	130.8	99.0	0	100.1	124.0	100.5	473.8
O7	210.9	409.2	152.8	62.1	76.8	100.1	0	29.5	45.2	396.8
O8	206.1	382.3	154.3	44.7	101.0	124.0	29.5	0	44.6	370.1
O9	166.3	391.1	111.1	32.5	116.4	100.5	45.2	44.6	0	375.1
O10	365.0	37.4	387.1	346.2	468.5	473.8	396.8	370.1	375.1	0

Table 20.3. Pairwise Euclidean distance between the 10 first observations in the PM10 data. Red observations (i.e., O1, O3, O5, O6, O7 and O8) are observations where the pollution levels are above the median value and dark green observations (i.e., O2, O4, O9 and O10) correspond to observations where the pollution level is below the median value.

- A 0.01
- B 0.02
- C 0.23
- D 0.46
- E Don't know.

20.3. Question 3 One definition of an outlier is the following:

An outlier is an object that has a low probability with respect to a probability distribution model of the data.

Consider the data set in Table 20.4. To detect outliers, we standardize the data

$$z_n = \frac{x_n - \text{mean}(x)}{\text{std}(x)}, \quad \text{mean}(x) = 6, \quad \text{std}(x) = 28,$$

and model them with a standard univariate Normal distribution $N(0, 1)$. We define an outlier as a data object with an attribute

$$|z_n| > c,$$

where c is a constant such that the probability that $|z_n| > c$ is equal to α , i.e., $p(|z_n| > c) = \alpha$. We choose $\alpha = 0.0027$ corresponding to $c = 3.00$. According to this definition, which data objects are judged to be outliers?

n	1	2	3	4	5	6	7	8
x_n	-3	-5	-8	0	75	-4	-1	-6

Table 20.4. A simple data set with eight data objects each with one attribute.

- A There are no outliers.
 B $x_5 = 75$ is an outlier.
 C $x_3 = -8$ and $x_5 = 75$ are outliers.
 D $x_4 = 0$, $x_3 = -8$, $x_5 = 75$ are outliers.
 E Don't know.

20.4. Question 4: We suspect that observation O8 may be an outlier. In order to assess if this is the case we would like to calculate the average relative KNN density based on the observations given in Table 20.2 only. We recall that the KNN density and average relative density for the observation \mathbf{x} are given by:
 $\text{density}(\mathbf{x}, K) = \left(\frac{1}{K} \sum_{\mathbf{y} \in N(\mathbf{x}, K)} \text{distance}(\mathbf{x}, \mathbf{y}) \right)^{-1}$,

$$\text{a.r.d.}(\mathbf{x}, K) = \frac{1}{K} \sum_{\mathbf{y} \in N(\mathbf{x}, K)} \text{density}(\mathbf{y}, K),$$

where $N(\mathbf{x}, K)$ is the set of K nearest neighbors of observation \mathbf{x} and $\text{a.r.d.}(\mathbf{x}, K)$ is the average relative density of \mathbf{x} using K nearest neighbors. Based on the data in Table 20.2, what is the average relative density for observation O8 for $K = 3$ nearest neighbors?

- A 0.19
 B 0.28
 C 0.56
 D 1.79
 E Don't know.

20.5. Question 5: Consider again the distances in table 20.5. We wish to compute the *average relative KNN density* (a.r.d) of the observations in table 20.5 using the cityblock distance indicated by the table. Letting $d(\mathbf{x}, \mathbf{y})$ denote the cityblock distance metric this is defined as

$$\text{density}(\mathbf{x}, K) = \frac{1}{\frac{1}{K} \sum_{\mathbf{y} \in N(\mathbf{x}, K)} d(\mathbf{x}, \mathbf{y})}$$

$$\text{a.r.d.}(\mathbf{x}, K) = \frac{\text{density}(\mathbf{x}, K)}{\frac{1}{K} \sum_{\mathbf{z} \in N(\mathbf{x}, K)} \text{density}(\mathbf{z}, K)},$$

$N(\mathbf{x}, K)$: Set of K -nearest neighbours of \mathbf{x} .

What is the a.r.d. of observation o_1 using $K = 1$ nearest neighbours?

	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8
o_1	0	4	7	9	5	5	5	6
o_2	4	0	7	7	7	3	7	8
o_3	7	7	0	10	6	6	4	9
o_4	9	7	10	0	8	6	10	9
o_5	5	7	6	8	0	8	6	7
o_6	5	3	6	6	8	0	8	11
o_7	5	7	4	10	6	8	0	7
o_8	6	8	9	9	7	11	7	0

Table 20.5. Pairwise Cityblock distance, i.e $d(o_i, o_i) = \|\mathbf{x}_i - \mathbf{x}_j\|_1 = \sum_{k=1}^M |x_{ik} - x_{jk}|$, between 8 observations. Each observation o_i corresponds to a $M = 15$ dimensional binary vector, $x_{ik} \in \{0, 1\}$. The blue observations $\{o_1, o_2, o_3, o_4\}$ belong to class C_1 and the black observations $\{o_5, o_6, o_7, o_8\}$ belong to class C_2 .

- A a.r.d.($\mathbf{x} = o_1, K = 1$) = $\frac{1}{4}$
 B a.r.d.($\mathbf{x} = o_1, K = 1$) = $\frac{1}{3}$
 C a.r.d.($\mathbf{x} = o_1, K = 1$) = $\frac{1}{2}$
 D a.r.d.($\mathbf{x} = o_1, K = 1$) = $\frac{3}{4}$
 E Don't know.

20.6. Question 6: Consider again the distances in table 20.5. Suppose we wish to perform mixture modelling (see chapter 9.2 of “Introduction to data mining”) and we consider mixture distributions of the form ($\lambda > 0$):

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{2\lambda} \exp(-d(\mathbf{x}, \boldsymbol{\theta})/\lambda)$$

Suppose we consider $K = 8$ mixture components, the parameter $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_8$ of each mixture component is taken to be the position of the observations o_1, \dots, o_8 and each mixture component is weighted equally in the full mixture distribution. Suppose we set $\lambda = 4$ and let d denote the cityblock distance metric. What is the probability density at observation o_1 ?

- A Probability density at $o_1 \approx 0.043$
 B Probability density at $o_1 \approx 0.031$
 C Probability density at $o_1 \approx 0.013$
 D Probability density at $o_1 \approx 0.341$
 E Don't know.

Association rule learning

Association rule learning is the discovery of interesting relations between features in a dataset. Suppose for instance we record the items that a large number of customers buy in a database, then we might discover that people who often buy onions, tomatoes, and burger dressing are also likely to buy hamburger meat. Automatic discovery of such rules is known as *association mining* and is of large commercial interest since the invention of the efficient Apriori method for discovering association rules in the early 90s [Agrawal et al., 1993, 1994] that, at the time of writing, has more than 38 000 citations. In this chapter, we will discuss basic concepts relating to association rule learning and introduce this popular method, the Apriori algorithm, for automatic rule discovery.

21.1 Basic concepts

We will still denote our dataset \mathbf{X} as an $N \times M$ matrix and assume \mathbf{X} is binary. Each row (observation) of the matrix corresponds to a transaction and each column (attribute) to an item. An example is shown in table 21.1 where we consider $N = 5$ transactions corresponding to $M = 4$ items. The canonical interpretation of \mathbf{X} in association rule learning is that each column corresponds to a set of things people can buy and each of the N observations corresponds to a “shopping cart” defining which items a given customer has bought. For instance if $X_{ij} = 1$ then person i bought item j . Because of this interpretation, it is common to use set notation. Suppose $\mathbf{x}_i = [0 \ 1 \ 1 \ 0]^T$. This will be written as the set:

$$t_i = \{I_2, I_3\}$$

to denote that transaction i corresponded to a person buying item I_2 and I_3 (butter and beer) and we will write t_1, \dots, t_N for all transactions. Since we will use set notation somewhat often it is perhaps a good idea to review some basic concepts. Suppose $r = \{I_1, I_2, I_4\}$ and $s = \{I_2, I_3, I_4, I_5\}$. Then we denote the size by vertical bars such that $|r| = 3$ and $|s| = 4$. The intersection, i.e. the elements in both sets, and the union, i.e. the elements in either of the sets are written as

$$r \cap s = \{I_2, I_4\}, \quad \text{and} \quad r \cup s = \{I_1, I_2, I_3, I_4, I_5\}.$$

A special set is the empty set $\emptyset = \{\}$ and two sets r, u are disjoint if $r \cap u = \emptyset$. For instance $\{I_1, I_2, I_4\} \cap \{I_3, I_5\} = \emptyset$. Set membership (i.e. if an element is in the set) is written as $x \in r$. In

Table 21.1. Small example dataset for association mining.

	milk	butter	beer	diapers
1	0	1	1	
0	1	0	1	
0	1	1	1	
0	0	1	0	
1	0	1	1	

our case $I_3 \in r$ is false but $I_3 \in s$ is true. If all elements in a set c is contained in a set r this will be written as $c \subseteq t$, for instance:

$$\{I_2, I_4\} \subseteq r.$$

Finally, set difference, that is the operation where we remove the elements in one set from another, is written as:

$$r \setminus s = \{I_1\}, \quad \text{and} \quad s \setminus r = \{I_3, I_5\}.$$

21.1.1 Itemsets and association rules

Returning to association mining, the set of all items will be written as

$$I = \{I_1, I_2, \dots, I_M\}. \quad (21.1)$$

Each transaction is then a subset of I . We will write $T = \{t_1, \dots, t_N\}$ for the set of all transactions. Notice, in set-notation, $t_i \subseteq I$. With this notation in place, we can make our first real definition: an *itemset* is simply a subset of I . Each transaction is an itemset, however, also $c = \{I_3, I_5\}$ is an itemset even though it is not a member of all transactions T . An *association rule* is written as

$$X \rightarrow Y, \quad (21.2)$$

where $X, Y \subseteq I$ and $X \cap Y = \emptyset$. The rule says that people who buy X will also tend to buy Y . For instance, we can consider the rule

$$\{\text{butter, beer}\} \rightarrow \{\text{milk}\},$$

which is saying that people who buy butter and beer will also tend to buy milk. What is a useful rule? One definition is to say it should satisfy two criteria:

High support It should be invoked fairly often, i.e. $X \cup Y$ should form a set of items many buy.

High confidence When the rule is triggered, i.e. someone buys X , then the person should be very likely to also buy Y .

These quantities are measured by the support and confidence which, as we will see, are nothing more than dressed-up probabilities.

21.1.2 Support

For an itemset X we can define a number, the support, which is the fraction of transactions which contains the itemset X . Formally:

$$\begin{aligned}\text{supp}(X) &= \frac{\{\text{Number of transactions containing } X\}}{N} \\ &= \frac{|\{t \in T | X \subseteq t\}|}{N}.\end{aligned}\tag{21.3}$$

Notice the use of set-notation: $\{t \in T | X \subseteq t\}$ is simply the set of transactions t in T such that X is contained as a subset of t and provides a more formal way of defining sets which we will return to later.

When we talk about the support of an association rule $X \rightarrow Y$, we mean the support of both X and Y . In other words:

$$\text{supp}(X \rightarrow Y) = \text{supp}(X \cup Y).\tag{21.4}$$

Let's try a few examples based on the market basket data \mathbf{X} having five transactions of four items given in Table table 21.1. For instance:

$$\text{supp}(\{I_2, I_3\}) = \frac{1}{5}, \quad \text{supp}(\{I_1, I_3, I_4\}) = \frac{2}{5}, \quad \text{supp}(\{I_1, I_2, I_3, I_4\}) = 0, \quad \text{supp}(\{I_3, I_4\} \rightarrow \{I_2\}) = \frac{1}{5}.$$

Support and probabilities

It is important to stress the support is nothing but the empirical probability of the itemset. If for instance $X = \{I_3, I_5\}$ we can just as easily write:

$$\text{supp}(X) = p(X) = p(I_3 = 1, I_5 = 1),$$

which is just the probability that the third and fifth coordinate is 1.

21.1.3 Confidence

To quantify confidence we introduce the *confidence* of an association rule $X \rightarrow Y$ as the fraction of transactions which contain X that also contains Y . Formally:

$$\text{conf}(X \rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)}.\tag{21.5}$$

Once again, this is closely linked to probabilities, namely the conditional probability of Y given X :

$$\text{conf}(X \rightarrow Y) = \frac{p(X \cup Y)}{p(X)} = p(Y|X).$$

To give a single example:

$$\text{conf}(\{I_3, I_4\} \rightarrow \{I_2\}) = \frac{\frac{1}{5}}{\frac{3}{5}} = \frac{1}{3}.$$

Returning to our original problem, we are interested in association rules $X \rightarrow Y$ where the support of the rule is higher than some pre-specified value and where the confidence too is higher than another pre-specified value (or, equivalently, find sets $Z = \{X \cup Y\}$ which occur with high probability and pairs of sets, X, Y , such that $p(Y|X)$ is large). This ensures the rules are relevant (high support) and that they are mostly true (confidence). However, finding rules with a high support pose a difficult challenge. If for instance we suppose $M = 1000$ (which is a fairly low number considering the number of items in an ordinary supermarket) and we are interested in all itemsets involving $k = 5$ items there are

$$\frac{M!}{(M-k)!k!} \approx 8.25 \times 10^{12}$$

potential itemsets to check. In the next section, we will look at a method, the Apriori algorithm, which makes this search much faster.

21.2 The Apriori algorithm

The Apriori algorithm is a way to discover association rules with high support. We assume we are given N transactions of M items and a minimum support value $0 < \epsilon \leq 1$. When a particular itemset X has support higher than ϵ , $\text{supp}(X) \geq \epsilon$, we say X is *frequent*. A word of warning: the Apriori algorithm may seem quite complicated especially on a first glance; however, it builds on a very simple principles which we will discuss first.

Itemsets have what is known as the *downwards closure property*. Downwards closure simply says that if X is frequent, then so is any of its subsets. This is very easy to prove: If a transaction contains X , then it also contains any subset $A \subset X$, and so

$$\begin{aligned} A \subset X \text{ implies } \text{supp}(A) &= \frac{|\{t \in T | A \subset t\}|}{N} \\ &\geq \frac{|\{t \in T | A \subset X \subset t\}|}{N} = \text{supp}(X). \end{aligned} \quad (21.6)$$

This implies that if A is not frequent (*infrequent*), then so is any set containing X . So how can this principle allow us to find all frequent itemsets? Suppose we first compute the support of all itemsets which only contain a single item $\{I_i\}$ for $i = 1, \dots, M$. If any of these are not frequent, then we can discard any itemset which contains that element since (by the downwards closure principle) it cannot be frequent. The idea is then to start with all (frequent) itemsets and iteratively find frequent itemsets with $k = 1, 2, \dots$ items. We let L_k denote the set of all frequent itemsets with k elements. For instance

$$L_1 = \{\{i\} | \text{supp}(\{i\}) \geq \epsilon\} \quad (21.7)$$

The algorithm then at step k compute L_k from L_{k-1} as follows:

- First generate a set of *candidate* itemsets of size k , C_k , by adding all items to the itemsets in L_{k-1} . Formally first define

$$C'_k = \{s \cup \{j\} | s \in L_{k-1}, j \notin s\} \quad (21.8)$$

Algorithm 9: Apriori algorithm

```

1: Given  $N$  transactions and let  $\epsilon > 0$  be the minimum support count
2:  $L_1 = \{\{j\} | \text{supp}(\{j\}) \geq \epsilon\}$ 
3: for  $k = 2, \dots, M$  and  $L_k \neq \emptyset$  do
4:    $C'_k = \{s \cup \{j\} | s \in L_{k-1}, j \notin s\}$ 
5:   Set  $C_k = C'_k$ 
6:   for each  $c \in C'_k$  do
7:     for each  $s \subset c$  such that  $|s| = k - 1$  do
8:       if  $s$  is not frequent, i.e.  $s \notin L_{k-1}$  then
9:          $C_k = C_k \setminus \{c\}$  (Remove  $c$  from  $C_k$ )
10:      end if
11:    end for
12:  end for
13:   $L_k = \{c | c \in C_k, \text{supp}(c) \geq \epsilon\}$  (compute support)
14: end for
15:  $L_1 \cup L_2 \cup \dots \cup L_k$  are then all frequent itemsets

```

Some of these itemsets can be known to be infrequent since they contain a subset of size $k - 1$ which is infrequent, i.e. is not in L_{k-1} . That an itemset c *contains* a subset *not* in L_{k-1} can be written as:

$$\{s | s \subset c, |s| = k - 1\} \cap L_{k-1} = \emptyset. \quad (21.9)$$

Removing all these itemsets from C_k can therefore be written as:

$$C_k = C'_k \setminus \{c | \{s | s \subset c, |s| = k - 1\} \cap L_{k-1} = \emptyset\}. \quad (21.10)$$

This is quite a daunting expression and it is somewhat easier to comprehend what it does by an example (which we will provide in a moment) or by breaking it into smaller steps. In line 5 to line 12 of algorithm 9 the expression eq. (21.10) is computed in a sequence of simpler steps which the reader is invited to consult.

- We then compute the support for each of the itemsets $c \in C_k$ and let L_k be those candidates in C_k with support greater than ϵ .
- The method terminates when $L_k = \emptyset$. All frequent itemsets are then $L_1 \cup L_2 \cup \dots \cup L_k$

More explicitly stated the Apriori method is given in algorithm 9

21.2.1 An example of the Apriori algorithm

The Apriori algorithm may appear quite daunting, but it is much easier to understand the steps by considering a concrete example. Suppose we set $\epsilon = 0.15$ and apply the Apriori algorithm to the problem in table 21.1 to find all frequent itemsets. To be frequent if $\epsilon = 0.15$ means the itemset must be found in *at least* 1 transactions (why? because $\frac{1}{5} = 0.2 > \epsilon$).

Initialization

We first observe that L_1 must contain all items because all items occur in at least one transaction. We write this as

$$L_1 = \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 \end{bmatrix}.$$

First iteration, k = 2:

The next step is to form C'_2 , which is done by taking each element in L_1 (for instance $\{I_3\}$) and then add each item which is *not* I_3 to this element to get $\{I_1, I_3\}$, $\{I_2, I_3\}$ and $\{I_3, I_4\}$. Doing this we obtain:

$$C'_2 = \begin{bmatrix} 1 & 1 & \cdot & \cdot \\ 1 & \cdot & 1 & \cdot \\ 1 & \cdot & \cdot & 1 \\ \cdot & 1 & 1 & \cdot \\ \cdot & 1 & \cdot & 1 \\ \cdot & \cdot & 1 & 1 \end{bmatrix}.$$

So far so good. Now, to form C_2 from C'_2 involve the following steps: Start with the first element in C'_2 , $\{I_1, I_2\}$. Then take each subset which has size $k - 1 = 1$, namely $\{I_1\}$ and $\{I_2\}$. We next check that these are in L_1 (which they are) and therefore, because both $\{I_1\}$ and $\{I_2\}$ are in L_1 , $c = \{I_1, I_2\}$ remains in C_k . Proceeding this way we see that in fact $C_2 = C'_2$ as shown above.

Finally, we go over all itemsets in C_2 and compute their support. We see that $\text{supp}(\{I_1, I_2\}) = 0$ and so this itemset is not included in L_2 , however, all other itemsets occur in at least 1 transaction and are therefore accepted, therefore

$$L_2 = \begin{bmatrix} 1 & \cdot & 1 & \cdot \\ 1 & \cdot & \cdot & 1 \\ \cdot & 1 & 1 & \cdot \\ \cdot & 1 & \cdot & 1 \\ \cdot & \cdot & 1 & 1 \end{bmatrix}.$$

Second iteration, k = 3:

In the second iteration, we first add all singleton sets to L_2 to get the candidate sets C'_3 . For instance, starting with $\{I_2, I_3\} \in L_2$ we obtain the candidate transactions $\{I_1, I_2, I_3\}, \{I_2, I_3, I_4\} \in C'_3$. Ignoring duplicates this list is:

$$C'_3 = \begin{bmatrix} 1 & 1 & 1 & \cdot \\ 1 & \cdot & 1 & 1 \\ 1 & 1 & \cdot & 1 \\ \cdot & 1 & 1 & 1 \end{bmatrix}.$$

Next we proceed by the difficult rule in eq. (21.10). We first set $C_3 = C'_3$ and start with the first transaction $c = \{I_1, I_2, I_3\}$. We consider all subsets of c with one element removed, $s = \{I_1, I_2\}, \{I_2, I_3\}, \{I_1, I_3\}$ and notice that $\{I_1, I_2\}$ is *not* in L_2 . Since this itemset is infrequent, we know by the downwards closure property that c cannot be frequent and can therefore be dismissed – which is why $c = \{I_1, I_2, I_3\}$ does not occur in C_3 . Doing this for all itemsets leave us with:

$$C_3 = \begin{bmatrix} 1 & \cdot & 1 & 1 \\ \cdot & 1 & 1 & 1 \end{bmatrix}.$$

Computing their support we notice:

$$\text{supp}(\{I_1, I_3, I_4\}) = \frac{2}{5}, \quad \text{and} \quad \text{supp}(\{I_2, I_3, I_4\}) = \frac{1}{5}$$

and therefore $L_3 = \begin{bmatrix} 1 & 1 & 1 \\ \cdot & 1 & 1 \\ \cdot & 1 & 1 \end{bmatrix}$.

Third iteration, k = 4:

Since $L_3 = \begin{bmatrix} 1 & 1 & 1 \\ \cdot & 1 & 1 \\ \cdot & 1 & 1 \end{bmatrix}$ the third iteration is very simple. We form the itemset $C'_4 = [1 1 1 1]$ and obtain in the next step that $s = \{I_1, I_2, I_3\}, \{I_1, I_3, I_4\}, \{I_1, I_2, I_4\}, \{I_2, I_3, I_4\}$ needs to be checked. Since only $\{I_1, I_3, I_4\}, \{I_2, I_3, I_4\} \in L_3$ we get $C_4 = \emptyset$ and therefore $L_4 = \emptyset$. The algorithm therefore terminates and we finally have:

$$L = \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 \\ 1 & \cdot & 1 & \cdot \\ 1 & \cdot & \cdot & 1 \\ \cdot & 1 & 1 & \cdot \\ \cdot & 1 & \cdot & 1 \\ \cdot & \cdot & 1 & 1 \\ 1 & \cdot & 1 & 1 \\ \cdot & 1 & 1 & 1 \end{bmatrix}$$

21.3 Using the Apriori algorithm to find itemsets with high confidence

Finding association rules $X \rightarrow Y$ with high confidence can easily be solved using the Apriori algorithm. In general, we want to find rules $X \rightarrow Y$ such that

$$\begin{aligned} \text{supp}(X \rightarrow Y) &= \text{supp}(X \cup Y) \geq \epsilon \\ \text{conf}(X \rightarrow Y) &= \frac{\text{supp}(X \cup Y)}{\text{supp}(X)} \geq \delta \end{aligned}$$

From the last equation we obtain: $\frac{1}{\delta} \text{supp}(X \cup Y) \geq \text{supp}(X)$. This suggests the following strategy: We first use the Apriori method to generate all itemsets Z with support greater than ϵ : $\text{supp}(Z) \geq \epsilon$. If ϵ is chosen reasonably large, these itemsets will typically be fairly small, perhaps on the order of about 5 – 10 items.

Given these itemsets, we can then search over all *subsets* X of Z and find those subsets where $\text{supp}(X) \leq \frac{1}{\delta} \text{supp}(Z)$. Then defining $Y = Z \setminus X$ we are guaranteed the association rule $X \rightarrow Y$ satisfy both the above requirements.

This procedure can be speed up by noticing that if $W \subset X$, then $\text{supp}(W) \geq \text{supp}(X)$. Thus, if we have already seen that a particular set X has $\text{supp}(X) > \frac{1}{\delta} \text{supp}(Z)$, there is no need to check the subsets X' of X because they are guaranteed to have low confidence too. Accordingly, we should consider an algorithm that starts by considering $X = Z \setminus \{i\}$ for each $i \in Z$ and only proceed to remove further elements from each X if the corresponding association rule $X \rightarrow Y$ has confidence greater than δ .

21.4 Some limitations

The Apriori algorithm is very efficient at identifying joint occurrences, i.e. joint probabilities that are frequent by having high support. However, it is only possible to find associations with high confidence within these itemsets identified to have high support, whereas there may be many other association rules of high confidence. As such, if you were to buy lobster you are likely to also buy lemon and white wine, however, as lobster is bought very infrequently the itemset $\{\text{lobster}, \text{lemon}, \text{white wine}\}$ will in general be below the support threshold ϵ and such high-confidence rule missed by the algorithm.

Customers may also have different preferences and can typically be segmented into customer types. It can therefore be useful to consider preferences within segments of consumers rather than in terms of all transactions. The basic association mining framework presented here only considers all transactions and therefore mix different costumer segments thereby potentially missing important segment specific associations if these segments are not somehow identified prior to the analysis (for instance using some clustering approach).

While our initial motivation for association mining was the analysis of market baskets, i.e. the transactions of customers and their items purchased, association mining has very general applicability. For instance association mining can be used within medicine (i.e., identifying patterns such as if you have these and these symptoms it is likely you also have these symptoms), bioinformatics (i.e., identifying association between genes), and general questionnaire data (i.e., if you answered this and this it is likely you will also answer this) to mention but a few potential domains of application. The Apriori algorithm assumes binary features whereas in many real applications, datasets may not be binary and therefore some type of binarization must be invoked prior to the analyses. How to binarize the data can be unclear whereas the binarization may influence results. For instance if we binarize an attribute such as age we could simply threshold by the median value (i.e. 50th percentile) or use one-out-of-K coding to include multiple binary features each denoting different age interval. In this case, the binarization will heavily influence the support, i.e. using the median we cannot have support $\epsilon > 50\%$ and if we for instance split according to 10th percentiles using 1-out-of-10 coding we will obtain 10 new binary attributes from the original attribute age and in general not be able to get support $\epsilon > 10\%$.

Finally, care should be taken interpreting association rules as support and confidence are nothing but joint and conditional probabilities respectively. Thus, similar to our discussion of Bayesian networks in chapter 13 the arrows in association mining does not imply causality. As such, we have for the example given in Table 21.1 that the confidence of the rule $\{\text{beer}\} \rightarrow \{\text{diapers}\}$ is 75% whereas the confidence of the rule $\{\text{diapers}\} \rightarrow \{\text{beer}\}$ is also 75%.

Problems

21.1. Question 1: We consider the twelve costumers given in Table 21.2. We will consider this data set a market basket problem in which the twelve customers have various combinations of the six items denoted M_H , M_L , P_H , P_L , D_H , D_L . Which one of the proposed solutions below includes *all* the frequent itemsets with support of more than 40 %?

	M_H	M_L	P_H	P_L	D_H	D_L
LIS1	1	0	0	1	1	0
LIS2	1	0	1	0	1	0
LIS3	0	1	0	1	0	1
LIS4	0	1	0	1	0	1
LIS5	1	0	1	0	0	1
LIS6	1	0	1	0	1	0
OPO1	1	0	1	0	0	1
OPO2	0	1	0	1	1	0
OPO3	0	1	1	0	0	1
OPO4	0	1	0	1	0	1
OPO5	0	1	1	0	0	1
OPO6	1	0	1	0	1	0

Table 21.2. Given are the six first costumers of Lisbon and Oporto including whether these costumers spent more or less than the median consumption of MILK (M_H , M_L), PAPER (P_H , P_L), and DELI (D_H , D_L). Subscript H and L are thus used to respectively denote a relatively high and low level of consumption (i.e., above or below the median consumption).

- A $\{M_L\}$, $\{M_H\}$, $\{P_H\}$, $\{P_L\}$, $\{D_H\}$, $\{D_L\}$.
- B $\{M_L\}$, $\{M_H\}$, $\{P_H\}$, $\{P_L\}$, $\{D_H\}$, $\{D_L\}$, $\{M_H, P_H\}$.
- C $\{M_L\}$, $\{M_H\}$, $\{P_H\}$, $\{P_L\}$, $\{D_H\}$, $\{D_L\}$, $\{M_H, P_H\}$, $\{M_L, D_L\}$.
- D $\{M_L\}$, $\{M_H\}$, $\{P_H\}$, $\{P_L\}$, $\{D_H\}$, $\{D_L\}$, $\{M_H, P_H\}$, $\{M_H, D_H\}$, $\{M_L, P_L\}$, $\{M_L, D_L\}$, $\{P_L, D_L\}$.
- E Don't know.

21.2. Question 2: Consider again the dataset in Table 21.2. What is the confidence of the association rule $\{M_L, D_L\} \rightarrow \{P_L\}$?

- A 25%
- B 40%
- C 60%
- D 80%
- E Don't know.

21.3. Question 3: We again consider the ten subjects given in Table 21.3. We will consider this data set a market basket problem where the customers are the ten subjects and they have various combinations of the six items denoted YAY , YAN , OAY , OAN , PAY , PAN . Which one of the proposed solutions below includes *all* the frequent itemsets with support of more than 52 %?

	YAY	YAN	OAY	OAN	PAY	PAN
S1	1	0	1	0	1	0
S2	1	0	1	0	0	1
S3	0	1	0	1	1	0
S4	0	1	1	0	1	0
S5	0	1	1	0	1	0
NS1	0	1	1	0	1	0
NS2	0	1	0	1	1	0
NS3	1	0	0	1	0	1
NS4	0	1	1	0	1	0
NS5	0	1	1	0	1	0

Table 21.3. Given are five subjects that survived in Haberman's study (denoted S1, S2, ..., S5) as well as the five subjects that did not survive in Haberman's study (denoted NS1, NS2, ..., NS5) including whether these subjects are young or old (YAY , YAN), were operated after 1960 or not (OAY , OAN), and had positive axillary nodes or not (PAY , PAN).

- A $\{YAN\}$, $\{OAY\}$, $\{PAY\}$.
- B $\{YAN\}$, $\{OAY\}$, $\{PAY\}$, $\{YAN, PAY\}$, $\{OAY, PAY\}$.
- C $\{YAN\}$, $\{OAY\}$, $\{PAY\}$, $\{YAN, OAY\}$, $\{YAN, PAY\}$, $\{OAY, PAY\}$.
- D $\{YAN\}$, $\{OAY\}$, $\{PAY\}$, $\{YAN, OAY\}$, $\{YAN, PAY\}$, $\{OAY, PAY\}$, $\{YAN, OAY, PAY\}$.
- E Don't know.

Solutions

Problems of Chapter 2

2.1 The correct answer is B: For both Age and PV there are a natural zero and we can apply all the operators $<, >, =, \neq, *, /$ thus these attributes are ratio. As Race, HT and UI are binary categorical, i.e. $=, \neq$ can be applied to them but not $<, >$ these attributes are not ordinal. Age is ratio but not continuous as the attribute is measured in whole years. Furthermore, MW is continuous and PV is discrete.

2.2 The correct answer is C: All the attributes are ratio since 0 means absence of what is being measured. As the Plants and E-Plants both are based on counts they are discrete whereas all remaining attributes are continuous and greater than zero as they quantify distances and areas (which are non-negative quantities).

2.3 The correct answer is D: x_1, x_2 are ratio, x_3 is ordinal, x_4 is nominal and x_5 is interval. Accordingly only option four is correct.

Problems of Chapter 3

3.1 The correct answer is A: CAL, PROT, FAT, FIB, SUG, POT, VIT, SHELF WEIGHT have negative coefficients of PCA1 whereas TYPE, SOD, CARB, CUPS have positive coefficients resulting in a negative projection onto the first principal component. The magnitude of the coefficients for PROT and SHELF are very small hence PCA2 does not primarily discriminate between low values of PROT and high values of SHELF and high values of PROT and low values of SHELF even though PROT has a small negative coefficient and SHELF a small positive coefficient. From Figure 3.15 it can be seen that the projection of the data onto PCA2 is negatively correlated with RAT as there is a strong tendency that small values of the projection onto PCA2 corresponds to relatively high values of RAT whereas large values in the projection onto PCA2 results in relatively low value of RAT. Finally, PCA1 and PCA2 will by definition always be orthogonal to each other despite the preprocessing of the data.

3.2 The correct answer is A: The first three principal components account for $\frac{\sigma_1^2 + \sigma_2^2 + \sigma_3^2}{\sigma_1^2 + \sigma_2^2 + \sigma_3^2 + \sigma_4^2 + \sigma_5^2 + \sigma_6^2} = 61.3\%$ of the variation. Thus, A is incorrect.

3.3 The correct answer is D: The variation explained by each principal component is given by $\frac{\sigma_i^2}{\sum_{i'} \sigma_{i'}^2}$. As such we find:

$$\begin{aligned} VarExpPC1 &= \frac{9.7^2}{9.7^2 + 6.7^2 + 5.7^2 + 3.7^2 + 3.0^2 + 1.3^2 + 0.7^2} = 0.4792 \\ VarExpPC1 - 3 &= \frac{9.7^2 + 6.7^2 + 5.7^2}{39.7^2 + 6.7^2 + 5.7^2 + 3.7^2 + 3.0^2 + 1.3^2 + 0.7^2} = 0.8733 \\ VarExpPC7 &= \frac{0.7}{9.7^2 + 6.7^2 + 5.7^2 + 3.7^2 + 3.0^2 + 1.3^2 + 0.7^2} = 0.0025 \\ VarExpPC1 - 4 &= \frac{9.7^2 + 6.7^2 + 5.7^2 + 3.7^2}{9.7^2 + 6.7^2 + 5.7^2 + 3.7^2 + 3.0^2 + 1.3^2 + 0.7^2} = 0.9431 \\ VarExpPC1 - 5 &= \frac{9.7^2 + 6.7^2 + 5.7^2 + 3.7^2 + 3.0^2}{9.7^2 + 6.7^2 + 5.7^2 + 3.7^2 + 3.0^2 + 1.3^2 + 0.7^2} = 0.9889 \end{aligned}$$

As such the first PC accounts for less than 50% of the variance, the first three principal components accounts for less than 90% whereas the last component accounts for less than 1%. As four components are insufficient but five components sufficient to account for 95% of the variance the correct answer is that five principal components are required in order to account for more than 95% of the variation in the data.

3.4 The correct answer is D: The terminology is taken from the appendix of Tan et al. The principal directions must be normalized and orthogonal leaving only C and D . It is however visually clear the shape is both elongated along and symmetrical about the diagonal direction leaving option D as the correct choice.

3.5 The correct answer is B: The projection can be found by subtracting the mean from \mathbf{X} and projecting onto the first two columns of \mathbf{V} (see appendix B of Tan et al). The first point with the mean subtracted has coordinates

$$[3 - 7/3 \ 2 - 4/3 \ 1 - 5/3]$$

This should be (left) multiplied with the first two columns of \mathbf{V} :

$$\begin{bmatrix} 3 - 7/3 \\ 2 - 4/3 \\ 1 - 5/3 \end{bmatrix}^\top \begin{bmatrix} -0.99 & -0.13 \\ -0.09 & 0.7 \\ 0.09 & -0.7 \end{bmatrix} = [-0.78 \ 0.85]$$

corresponding to option B .

3.6 The correct answer is C: The variance explained by each principal component is given by $\frac{\sigma_i^2}{\sum_{i'} \sigma_{i'}^2}$. As such we find:

$$\begin{aligned} VarExpPC1 &= \frac{2.69^2}{2.69^2 + 2.53^2 + 1.05^2 + 0.83^2 + 0.49^2 + 0.31^2} = 0.459 \\ VarExpPC1 - 3 &= \frac{2.69^2 + 2.53^2 + 1.05^2}{2.69^2 + 2.53^2 + 1.05^2 + 0.83^2 + 0.49^2 + 0.31^2} = 0.934 \\ VarExpPC5 - 6 &= \frac{0.49^2 + 0.31^2}{2.69^2 + 2.53^2 + 1.05^2 + 0.83^2 + 0.49^2 + 0.31^2} = 0.021 \\ VarExpPC4 &= \frac{0.83^2}{2.69^2 + 2.53^2 + 1.05^2 + 0.83^2 + 0.49^2 + 0.31^2} = 0.0436 \end{aligned}$$

As such the first PC accounts for more than 40% of the variance, the first three principal components accounts for less than 95% whereas the last two component accounts for more than 2%, thus, this is correct. As the fourth component accounts for 4.36% of the variance the last statement is incorrect.

Problems of Chapter 4

4.1 The correct answer is C: Notice the definition of the cityblock distance corresponds to the number of 01 and 10 matches between the binary vectors:

$$f_{01} + f_{10} = \sum_{k=1}^M |x_{ik} - x_{jk}|$$

Then since $f_{01} + f_{10} + f_{11} + f_{00} = M$ we can compute

$$SMC(o_1, o_3) = \frac{f_{11} + f_{00}}{M} = \frac{M - f_{01} - f_{10}}{M} = \frac{8}{15} = 0.53$$

The other options are not true, on the simulated dataset the correct options are: $\text{COS}(o_1, o_3) = 0.565685424949$, $J(o_1, o_3) = 0.363636363636$

4.2 The correct answer is D: The three measures of similarity is given by

$$\begin{aligned} J(\mathbf{x}, \mathbf{y}) &= \frac{f_{11}}{f_{01} + f_{10} + f_{11}} \\ SMC(\mathbf{x}, \mathbf{y}) &= \frac{f_{00} + f_{11}}{f_{01} + f_{10} + f_{11} + f_{00}} \\ \cos(\mathbf{x}, \mathbf{y}) &= \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \end{aligned}$$

We now have

$$\begin{aligned} J(\text{NS1}, \text{NS2}) &= \frac{1}{7} \\ SMC(\text{NS1}, \text{NS2}) &= \frac{2}{8} = \frac{1}{4} \\ \cos(\text{NS4}, \text{NS5}) &= \frac{2}{2 \cdot 2} = \frac{1}{2} \\ SMC(\text{NS5}, \text{AS5}) &= \frac{6}{8} = \frac{3}{4} \\ J(\text{NS5}, \text{AS5}) &= \frac{3}{5} \\ \cos(\text{NS5}, \text{AS5}) &= \frac{3}{4} \end{aligned}$$

Hence the correct answer is $\cos(\text{NS5}, \text{AS5}) = \frac{3}{4}$.

4.3 The correct answer is D: The three measures of similarity is given by

$$\begin{aligned} J(\mathbf{x}, \mathbf{y}) &= \frac{f_{11}}{f_{01} + f_{10} + f_{11}} \\ SMC(\mathbf{x}, \mathbf{y}) &= \frac{f + f_{11}}{f_{01} + f_{10} + f_{11} + f} \\ \cos(\mathbf{x}, \mathbf{y}) &= \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \end{aligned}$$

We now have

$$\begin{aligned} J(S1, S2) &= \frac{2}{4} = \frac{1}{2}, \\ J(S1, NS1) &= \frac{2}{4} = \frac{1}{2}, \\ SMC(S1, S2) &= \frac{4}{6} = \frac{2}{3}, \\ SMC(S1, NS1) &= \frac{4}{6} = \frac{2}{3}, \\ \cos(S1, S2) &= \frac{2}{\sqrt{3}\sqrt{3}} = \frac{2}{3}, \\ \cos(S1, NS1) &= \frac{2}{\sqrt{3}\sqrt{3}} = \frac{2}{3}. \end{aligned}$$

Hence, the correct answer is $SMC(S1, S2) = \cos(S1, S2)$.

Problems of Chapter ??

6.1 The correct answer is C: The trick to solving the problem is to *not* use Bayes theorem. If we introducing appropriate shorthand the following relations hold from basic probability theory:

$$\begin{aligned} p(B|H) &= 1 - p(R|H) \\ p(R|H) &= \frac{p(H, R)}{p(H)} \\ p(H) &= p(H|4)p(4) + p(H|2)(1 - p(4)) \end{aligned}$$

Plugging in the information we then have

$$\begin{aligned} p(4) &= p(2) = \frac{1}{2} \\ p(H) &= \frac{1}{2}(0.8 + 0.2) = 0.5 \\ p(R|H) &= \frac{0.1}{0.5} = 0.2 \end{aligned}$$

From which it follows $p(B|H) = 1 - 0.2 = 0.8$

6.2 The correct answer is C: We will let NS denote normal semen and AS denote abnormal semen whereas CD_Y and CD_N will indicate having had a childhood disease and not having had a childhood disease respectively. We now find using Bayes theorem:

$$\begin{aligned} P(NS|CD_Y) &= \frac{P(CD_Y|NS)P(NS)}{P(CD_Y)} \\ &= \frac{P(CD_Y|NS)P(NS)}{P(CD_Y|NS)P(NS) + P(CD_Y|AS)P(AS)} \\ &= \frac{0.125 \cdot 0.88}{0.125 \cdot 0.88 + 0.167 \cdot 0.12} \\ &= 0.8459 \end{aligned}$$

6.3 The correct answer is D: We will let PA denote positive axillary nodes detected and NA denote no positive axillary nodes detected whereas SY and SN will indicate having survived and not having survived respectively. We now find using Bayes theorem:

$$\begin{aligned} P(PA|SY) &= \frac{P(SY|PA)P(PA)}{P(SY)} \\ &= \frac{P(SY|PA)P(PA)}{P(SY|PA)P(PA)+P(SY|NA)P(NA)} \\ &= \frac{0.36 \cdot 0.56}{0.36 \cdot 0.56 + 0.14 \cdot 0.44} \\ &= 0.766 \approx 76.6\% \end{aligned}$$

Problems of Chapter 7

7.1 The correct answer is B: The 25th and 50th percentile but not the 50th and 75th percentiles of the attribute DB coincides. A1A and AsA will not necessarily be highly correlated even though their distributions may have a similar shape (hence, this is correct). For attributes to be correlated it is important they take on high or low values systematically, however, this can not be inspected in a boxplot. TB is not likely to be normal distribution as this attribute does not have a symmetric but highly right skewed distribution. The attribute GDR does not have a clear outlier, in fact the outlier corresponds to the females in the dataset and all we can deduce from the plot is that more than 75 % of the observations are males.

7.2 The correct answer is A: The solution can be obtained by first observing the median of the dataset is 1, leaving option A and B, then noticing the 10 observations taking the value 3 is $10/60 \approx 17\%$ of the dataset and since the top-most whisker must be at the 90th percentile according to Tan fig. 3.12 this leave option A.

7.3 The correct answer is D: Even though there are observations marked as outliers these should not necessarily be removed. None of the attributes appear to be normal distribution but have a skewed distribution. As a result, for all the attribute the mean value will be larger and somewhat different from the median value of the data. Only if we standardize the data each attribute is expected to be given equal importance in the PCA.

Problems of Chapter 8

8.1 The correct answer is C: WINDDIR and HOUR are not part of the model and thus irrelevant. As logCAR has a positive coefficient of 0.36 and WIND a negative coefficient of -0.19 fewer cars and more wind will decrease the pollution level. As x_7 has a positive coefficient pollution is according to the model increasing over time and not decreasing. As x_2 has a negative coefficient of -0.01 higher temperatures will result in lower pollution levels according to the model.

8.2 The correct answer is A: Since both x_1 , x_2 and x_6 have negative coefficients large values of these attributes will in general make the model predict the consumer is from Lisbon. Since the offset is negative this implies that if a costumer after the standardization has $x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 0$ the customer is more likely to come from Lisbon than Oporto as the offset $x_0 = -0.51$. As $y = 1$ denotes the consumer is from Oporto the logit function will return the probability a person

is from Oporto. Even though the coefficients of FRESH and PAPER are the smallest they may still contribute in the predictions and we can not from this analysis conclude that they should be removed from the modeling. This would require comparing the test performance including FRESH and PAPER to not including these two attributes.

8.3 The correct answer is B: It is not reasonable to say AreaNI is irrelevant as it is included in the model with a non-zero coefficient and as seen from the boxplot in Figure 8.10 the attribute has a large range compared to the other attributes. However, as the coefficient of x_5 pertaining to DistNI is negative this implies that short distances to neighboring island would result in a prediction of larger area than large distances to the neighboring island would and this is hence correct. Even though the coefficient for x_2 number of endemic plants has the largest magnitude coefficient the range of this attribute is rather limited as seen from Figure 8.10 and it is thus not reasonable to say this is the most important attribute for predicting the Area. As the coefficients in front of x_4 is positive and x_6 negative an island that is highly elevated and close to Santa Cruz Island will in general be predicted to be relatively large.

8.4 The correct answer is A: First observe the value of the linear regression function at $(x_1, x_2) = (0, 0)$ is

$$\frac{1}{1 + e^{-w_0}}$$

This gives for option *A, D* and *B, C* respectively:

$$\begin{aligned} A, D : \frac{1}{1 + e^{-w_0}} &= (1 + e^{-2})^{-1} = 0.88 \\ B, C : \frac{1}{1 + e^{-w_0}} &= (1 + e^{-(2)})^{-1} = 0.12 \end{aligned}$$

Inspecting any of the two figures clearly indicate we can rule out *B, C* leaving *A, D*. However consider option *D* and the point $(1, 1)$. Then we have a density estimate of

$$\frac{1}{1 + e^{-(2+1+1-10)}} = \frac{1}{1 + e^6} \approx 0.0025.$$

Since this should correspond to the “high-density” corner we are left with option *A*.

Problems of Chapter 9

9.1 The correct answer is C: The impurity gain is given by

$$\Delta = I(\text{parent}) - \sum_{j=1}^2 \frac{N(v_j)}{N} I(v_j),$$

where

$$I(t) = 1 - \sum_{i=0}^{C-1} p(i|t)^2.$$

Inserting for the split defined by the PAN attribute we obtain:

$$\begin{aligned}\Delta = & (1 - ((\frac{81}{306})^2 + (\frac{225}{306})^2)) \\ & - [\frac{170}{306}(1 - ((\frac{62}{170})^2 + (\frac{108}{170})^2)) \\ & + \frac{136}{306}(1 - ((\frac{19}{136})^2 + (\frac{117}{136})^2))] = 0.025.\end{aligned}$$

9.2 The correct answer is A: The two classes would be well separated by the decisions

$$A = \left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix} \right\|_{\infty} < 0.25$$

$$B = \left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \right\|_2 < 0.25.$$

Decision A would capture the red crosses close to (0.25, 0.5). Decision B will separate the circular shape of red crosses in the upper middle from all the remaining observations that are black circles.

9.3 The correct answer is B: The relevant definitions can be found in section 4.3 of the textbook. The split at 2.5 divide the set X into two parts, the lower and upper part. We also need the frequencies for all the data. For each of these we can compute the frequencies of the three classes:

$$\begin{aligned}\text{all : } & p(0|a) = 3/7, p(1|a) = 1/7, p(2|a) = 3/7 \\ \text{lower : } & p(0|l) = 3/5, p(1|l) = 0/5, p(2|l) = 2/5 \\ \text{upper : } & p(0|u) = 1/2, p(1|u) = 0/2, p(2|u) = 1/2\end{aligned}$$

From this we can compute the impurity: $I(x) = 1 - \max_i p(i|x)$

$$\begin{aligned}\text{all : } & I(a) = 1 - 3/7 = 4/7. \\ \text{lower : } & I(l) = 1 - 3/5 = 2/5. \\ \text{upper : } & I(u) = 1 - 1/2 = 1/2.\end{aligned}$$

Then combining these we have

$$\begin{aligned}\Delta = & I(a) - (5/7)I(l) - (2/7)I(u) \\ = & 4/7 - (5/7)(2/5) - (2/7)(1/2) \\ = & \frac{1}{7}(4 - 2 - 1) = \frac{1}{7}\end{aligned}$$

Which is roughly $\Delta \approx 0.143$.

9.4 The correct answer is A: Trying to evaluate the trees in the corners $(-1, 1)$ and $(1, -1)$ rule out all but option A and C. Then considering the corner of the upper-left square must be “cut off” by the circle allow one to rule out C leaving the correct choice A; alternatively evaluate the classifiers at $(-0.2, 0.2)$ also show C cannot be correct.

Problems of Chapter 10

10.1 The correct answer is D: The dataset is first split into two datasets of size

$$D_{cv} = 800, D_{val} = 200.$$

Focusing on the cross validation, for each of the $L = 6$ values of λ we need to evaluate $K = 10$ cross-validation splits into datasets of size

$$D_{cv-train} = 720, D_{cv-test} = 80$$

The time taken for this is

$$T_{cv} = LK(D_{cv-train}^2 + 1/2D_{cv-test}^2)$$

This gives us the optimal value of λ . To estimate the performance we need to test and train one final model. This takes:

$$T_{val} = (D_{cv}^2 + 1/2D_{val}^2)$$

The total time elapsed is then

$$T = T_{cv} + T_{val} = 31.956 \cdot 10^6.$$

10.2 The correct answer is B: Using forward and backward selection we would like to minimize the test error. Thus, the forward selection would first select x_3 having lowest test error and subsequently x_1 decreasing the test error the most in combination with x_3 . Subsequently, x_2 is selected as this has minimal test error in combination with x_1 and x_3 . Selecting also x_4 does not improve the test error hence the selection procedure will terminate when x_1 , x_2 , and x_3 are selected. The backward feature selection will remove feature x_3 to minimize the test error but removing additional features will increase the test error hence the backward selection strategy will terminate with the features x_1 , x_2 and x_4 being selected which is also the optimal feature combination for this data. Thus, backward selection will indeed result in a better model being selected than using forward selection.

10.3 The correct answer is D: For least squares linear regression the test error will not always decrease as we include more attributes in the model. We may indeed overfit to the data. However the training error will monotonically decrease.

10.4 The correct answer is D: Using forward and backward selection we would like to minimize the test error. Thus, the forward selection would first select x_5 having lowest test error and subsequently x_2 decreasing the test error the most in combination with x_5 . Further selecting features in combination with x_2 and x_5 does not improve the test error thus the model will terminate. The backward feature selection will remove feature x_5 to minimize the test error and subsequently x_1 which constitutes the optimal feature configuration for the problem. Thus, removing additional features will not increase the test error hence the backward selection strategy will terminate with the features x_2 and x_6 .

10.5 The correct answer is B: The error rate and accuracy for each of the classifiers is given by:

Logistic regression:

$$\text{error rate} = \frac{10+69}{306} = \frac{79}{306},$$

$$\text{accuracy} = \frac{215+12}{306} = \frac{227}{306}$$

Decision tree:

$$\text{error rate} = \frac{34+55}{306} = \frac{89}{306},$$

$$\text{accuracy} = \frac{191+26}{306} = \frac{217}{306}.$$

Thus, the error rate for logistic regression is smaller and the accuracy higher than the decision tree

classifier. As there are 225 subjects that died and only 81 subjects that survived the classes are imbalanced. Thus, predicting everything to be in the largest class (died) would give an accuracy of $\frac{225}{306}$ which is larger than the accuracy obtained by the decision tree classifier.

10.6 The correct answer is B: Recall the accuracy is defined as (Tan 4.2)

$$acc = \frac{f_{00} + f_{11}}{N}$$

i.e. the sum of true negatives and positives divided by the total number of observations. Since We can count the number of true negatives and positives as a function of θ to get

$$\begin{aligned}\theta &= 0.35 : f_{00} = 1, f_{11} = 2 \\ \theta &= 0.45 : f_{00} = 2, f_{11} = 2 \\ \theta &= 0.55 : f_{00} = 2, f_{11} = 1 \\ \theta &= 0.55 : f_{00} = 2, f_{11} = 0\end{aligned}$$

So the highest accuracy of The highest accuracy is $(2 + 2)/5 = 4/5 = 0.8$ is obtained at $\theta = 0.45$

10.7 The correct answer is B: 2-fold cross-validation is not the same as the hold method where 50% is hold out as two models are trained and evaluated on all the data by two-fold cross-validation whereas hold-out 50% only trains one model and evaluate the performance of this model on half the data. For a very small dataset it is better to use leave-one-out cross validation as this will keep as much data for training as possible. Only one level of cross-validation is needed for tuning model parameters. Two levels are used when quantifying performance of the model with parameters selected. Leave-one-out cross-validation is computationally expensive since as many models as observations needs to be trained.

10.8 The correct answer is B: Firstly notice the training error column can be disregarded. Forward selection then first selects x_2 , then x_2, x_4 , then x_2, x_3, x_4 and finally x_1, x_2, x_3, x_4 . Backward selection however start with x_1, x_2, x_3, x_4 , then disregards x_2 to form x_1, x_3, x_4 and terminates.

Since the test error for forward selection is 25 and for backward selection 15 only option B is correct.

Problems of Chapter 12

12.1 The correct answer is B: O1-O4 are all closest to each other than to O5-O8 and will thus be correctly classified. O5 is closest to O7, O2 and O4 and will thus be misclassified. O6 is closest to O1, O4 and O3 and will thus be misclassified. O7 is closest to O5, O8 and O2 and will thus be correctly classified and O8 is closest to O7, O5 and O4 and is thereby also correctly classified. As two observations will be misclassified the error rate will be $2/8=1/4$.

12.2 The correct answer is D: Since there are 7 observations, $K = 7$ must classify everything to the largest class and so $k_2 = 7$. Next, for $K = 1$ the ticks must be colored correctly and so $k_3 = 1$, however by checking the left-most part of the k_4 -pane it is easy to see $k_4 = 5$. This leaves option 4.

12.3 The correct answer is A: The true accuracy is 0.125 or 1/8. This is easy to see by going through table 12.2 and observing all observations except o_1 is misclassified

Problems of Chapter 13

13.1 The correct answer is B: According to the Naïve Bayes classifier we have

$$\begin{aligned}
 P(CKD = 1|RBC = 1, PC = 1, DM = 1, CAD = 1) &= \\
 &\frac{\left(\begin{array}{c} P(RBC = 1|CKD = 1) \times \\ P(PC = 1|CKD = 1) \times \\ P(DM = 1|CKD = 1) \times \\ P(CAD = 1|CKD = 1) \times \\ P(CKD = 1) \end{array} \right)}{\left(\begin{array}{c} P(RBC = 1|CKD = 1) \times \\ P(PC = 1|CKD = 1) \times \\ P(DM = 1|CKD = 1) \times \\ P(CAD = 1|CKD = 1) \times \\ P(CKD = 1) \end{array} \right) + \left(\begin{array}{c} P(RBC = 1|CKD = 0) \times \\ P(PC = 1|CKD = 0) \times \\ P(DM = 1|CKD = 0) \times \\ P(CAD = 1|CKD = 0) \times \\ P(CKD = 0) \end{array} \right)} \\
 &= \frac{2/9 \cdot 7/9 \cdot 6/9 \cdot 1/9 \cdot 9/15}{2/9 \cdot 7/9 \cdot 6/9 \cdot 1/9 \cdot 9/15 + 1/6 \cdot 1/6 \cdot 1/6 \cdot 1/6 \cdot 6/15} = 0.9614.
 \end{aligned}$$

13.2 The correct answer is D: According to the Bayes classifier we have

$$\begin{aligned}
 P(CKD = 1|RBC = 1, PC = 1, DM = 1) &= \\
 &\frac{\left(\begin{array}{c} P(RBC = 1, PC = 1, DM = 1|CKD = 1) \times \\ P(CKD = 1) \end{array} \right)}{\left(\begin{array}{c} P(RBC = 1, PC = 1, DM = 1|CKD = 1) \times \\ P(CKD = 1) \end{array} \right) + \left(\begin{array}{c} P(RBC = 1, PC = 1, DM = 1|CKD = 0) \times \\ P(CKD = 0) \end{array} \right)} \\
 &= \frac{2/9 \cdot 9/15}{2/9 \cdot 9/15 + 0/6 \cdot 6/15} = 1
 \end{aligned}$$

13.3 The correct answer is D: According to the Naïve Bayes classifier we have

$$\begin{aligned}
 P(S|YAY_Y = 1, OAY_Y = 1, PA_Y = 1) &= \\
 &\frac{\left(\begin{array}{c} P(YAY_Y = 1 = 1|S) \times \\ P(OAY_Y = 1|S) \times \\ P(PA_Y = 1|S) \times \\ P(S) \end{array} \right)}{\left(\begin{array}{c} P(YAY_Y = 1 = 1|S) \times \\ P(OAY_Y = 1|S) \times \\ P(PA_Y = 1|S) \times \\ P(S) \end{array} \right) + \left(\begin{array}{c} P(YAY_Y = 1 = 1|NS) \times \\ P(OAY_Y = 1|NS) \times \\ P(PA_Y = 1|NS) \times \\ P(NS) \end{array} \right)} \\
 &= \frac{2/5 \cdot 4/5 \cdot 4/5 \cdot 5/10}{2/5 \cdot 4/5 \cdot 4/5 \cdot 5/10 + 1/5 \cdot 3/5 \cdot 4/5 \cdot 5/10} = \frac{8}{11}.
 \end{aligned}$$

13.4 The correct answer is A: True answer is: 0.83. This can be found by computing the per-class probabilities

$$\begin{aligned} p(f_1 = 0|C_1) &= 3/5, \quad p(f_2 = 1|C_1) = 1 \\ p(f_1 = 0|C_2) &= 1/5, \quad p(f_2 = 1|C_2) = 3/5 \end{aligned}$$

The class label priors is $p(C_1) = p(C_2) = \frac{1}{2}$ and so the Naive-Bayes estimate is

$$\begin{aligned} p_{NB}(C_1|f_1 = 0, f_2 = 1) &= \\ \frac{p(f_1 = 0|C_1)p(f_2 = 1|C_1)p(C_1)}{p(f_1 = 0|C_1)p(f_2 = 1|C_1)p(C_1) + p(f_1 = 0|C_2)p(f_2 = 1|C_2)p(C_2)} \\ &= \frac{3/5}{3/5 + (1/5)(3/5)} = \frac{5}{6} \end{aligned}$$

Problems of Chapter 14

14.1 The correct answer is B: The aim of regularized least squares regression is to reduce the model's variance without introducing too much bias and not the reverse. Linear regression with transformed inputs can indeed model nonlinear relations, as the inputs may by the transformation be non-linearly transformed. It is useful to plot attributes vs. residuals to investigate non-linear relationships between each attribute and the output that is not presently accounted for by the model and forward selection can both be used for regression and classification problems.

Problems of Chapter 15

15.1 The correct answer is C: The aim of regularized least squares regression is to reduce the model's variance without introducing too much bias and not the reverse. The regularization strength is normally chosen to be the value that minimizes the error on the test set using cross-validation. Artificial neural networks with linear transfer functions can indeed be written in terms of a linear regression model. However, forward and backward selection uses the test-error and not the training error to determine which attributes to remove or select.

15.2 The correct answer is A: To compute the output, initialize $n_1 = f(1) = 1, n_2 = f(2) = 2$. Then we can compute:

$$\begin{aligned} n_3 &= f(1 * 0.5 + 2 * (-0.4)) = f(-3/10) = 0 \\ n_4 &= f(1 * 0.4 + 2 * 0) = f(0.4) = 0.4 \end{aligned}$$

Then for the output we have

$$\hat{y} = n_5 = f(n_3 * (-0.4) + n_4 * 0.1) = f(0.04) = 0.04.$$

And so the correct output is $\hat{y} = 0.04$.

15.3 The correct answer is D: Classifier 1 has a decision boundary defined by one linear boundary and is thus based on logistic regression whereas classifier 2 has a very complicated decision boundary that is defined in terms of the observation that is the closest and is therefore based on the 1-nearest neighbor classifier. The decision boundary of classifier 3 is based on horizontal and vertical lines

and thus is based on the decision tree classifier. Classifier four has smooth but non-linear decision boundaries and is thus based on the artificial neural network.

15.4 The correct answer is C: It is apparent the decision boundary which best match a 1NN classifier is P_3 ; this rule out all but option C.

Problems of Chapter 16

16.1 The correct answer is C: By coarse inspection, the point $(0.1, 0.95)$ lies on the ROC curve. This corresponds to a FPR of 0.1 and a TPR of 0.95. Automatically this rules out the instances where the red curve is higher than the black, and by inspection it can be seen to roughly correspond to $\theta = 0.1$ in figure C (see Tan et. al. chapter 5.7).

16.2 The correct answer is B: The false positive rate is

$$FPR = \frac{FP}{TN + FP}$$

Thus we can find the number of false positives as

$$FP = \frac{FPR \times TN}{1 - FPR} \approx 96.0$$

Notice that

$$\begin{aligned} TPR &= \frac{TP}{TP + FN} \\ N &= (TP + FN) + (TN + FP) \end{aligned}$$

Then

$$TPR = \frac{TP}{N - (TN + FP)}$$

which implies $TP = TPR \times (N - TN - FP) \approx 171.0$

16.3 The correct answer is A: The precision (p) and recall (r) as well as true negative rate (TNR) and false positive rate (FPR) is given by: Logistic regression:

$$p = \frac{12}{12 + 10} = \frac{6}{11}, \quad (21.11)$$

$$r = \frac{12}{12 + 69} = \frac{12}{81}, \quad (21.12)$$

$$TNR = \frac{215}{215 + 10} = \frac{43}{45}, \quad (21.13)$$

$$FPR = \frac{10}{215 + 10} = \frac{2}{45}. \quad (21.14)$$

Decision tree:

$$p = \frac{26}{26 + 34} = \frac{13}{30}, \quad (21.15)$$

$$r = \frac{26}{26 + 55} = \frac{26}{81}, \quad (21.16)$$

$$TNR = \frac{191}{191 + 34} = \frac{191}{225}, \quad (21.17)$$

$$FPR = \frac{34}{191 + 34} = \frac{34}{225}. \quad (21.18)$$

Thus, the precision of the logistic regression classifier is indeed higher than the precision of the decision tree classifier whereas the remaining statements are incorrect.

Problems of Chapter 17

17.1 The correct answer is B: There is one misclassified observation and so $\varepsilon_1 = 1/4$ and $\alpha_1 = \frac{1}{2} \log \frac{1-\varepsilon_1}{\varepsilon_1} = 0.5493$. Then the un-normalized weights become:

$$w = [e^{-\alpha_1} \ e^{\alpha_1} \ e^{-\alpha_1} \ e^{-\alpha_1}] = \left[\frac{1}{\sqrt{3}} \ \sqrt{3} \ \frac{1}{\sqrt{3}} \ \frac{1}{\sqrt{3}} \right]$$

normalizing gives:

$$w = \frac{1}{6} [1 \ 3 \ 1 \ 1]$$

and so option B is the correct.

17.2 The correct answer is B: In boosting miss-classified observations are indeed given more importance in the next round. Bagging sample with replacement such that the same observation can be included multiple times within a round and hence some observations are not included. Boosting does not use leave-one-out cross-validation to learn which observation to sample in the next round. When combining classifiers in bagging this is attained by majority voting.

Problems of Chapter 18

18.1 The correct answer is D: In single linkage the observations that is the closest between the clusters define the level in which they merge. As such O1 and O4 are the closest to each other with a distance of 0.96. Then O2 and O3 merge at 1.15 and subsequently 05 and 07 at 1.41. 01 and 04 next merge with 02 and 03 at 1.52. As the minimum distance between clusters now become the distance between 05 and 07 the cluster O1, O2, O3, O4 merge with 05, 07 at 2.66. Thus dendrogram 1 and 2 are incorrect. As 08 merge having a distance to O7 of 2.88 whereas 06 has a minimal distance to O1 of 4.07 dendrogram 4 is correct and dendrogram 3 incorrect.

18.2 The correct answer is A: Running the K -means method gives that at first iteration we have clusters

$$(3) \quad (6, 7, 9, 10) \quad (11, 14)$$

Cluster centers are then

$$\mu_1 = 3, \quad \mu_2 = 8, \quad \mu_3 = 12.5$$

New clusters are then

$$(3) \quad (6, 7, 9, 10) \quad (11, 14)$$

and so the method has converged.

18.3 The correct answer is D: The true answer is D, dendrogram 4. This is easy to see by for instance comparing the height at which observations 2 and 6 are linked to the true answer in the table.

18.4 The correct answer is D: $purity = \frac{4}{8} \cdot \frac{3}{4} + \frac{2}{8} \cdot \frac{1}{2} + \frac{1}{8} \cdot 1 + \frac{1}{8} \cdot 1 = \frac{3}{4}$

Problems of Chapter 19

19.1 The correct answer is B: Answer option B is the correct answer as: Cluster located with center at (15,15) has much more spread in the x_2 direction (i.e., variance 15) than the x_1 direction (i.e., variance 0.5). Furthermore, the cluster located at (30,0) has positive covariance and the cluster located at (0,0) negative covariance. These properties only holds for answer option B.

19.2 The correct answer is B: k-means and Gaussian Mixture Models (GMM) are dependent on initialization. The dendrogram height is indeed determined by proximity and linkage function. The update of centers in k-means depends on distance measure, i.e. Euclidean distance results in updating centers to be the average observation whereas the 1-norm results in centers updated to be the median value. A Gaussian Mixture Model with diagonal covariance matrix does not have the same number of free parameters as k-means as each center has M parameters and the GMM also includes M parameters to define the covariance of each cluster as well as a parameter defining the size, thus resulting in $2KM + K - 1$ parameters in total (-1 due to the sum to one constraint of the parameter defining the relative size) whereas k-means would have KM parameters.

19.3 The correct answer is C: The two mixture components have similar shape but differ in their weight. This leaves options B and C. Since they are elongated in the horizontal direction then $\sigma_1^2 = \sigma_2^2 > \delta_1^2 = \delta_2^2$, ruling out option B. This leaves the last option. The correct values are in fact: $w_1 = 0.7, \sigma_i^2 = 20, \delta_i^2 = 1$

Problems of Chapter 20

20.1 The correct answer is C: According to the Gaussian kernel density estimator:

$$p(x) = \frac{1}{N} \sum_{n=1}^N \frac{1}{(2\pi)^{M/2} \sqrt{|\sigma^2 \mathbf{I}|}} \exp\left(-\frac{1}{2} (\mathbf{x} - \mathbf{x}_n)^\top \left(\frac{1}{\sigma^2} \mathbf{I}\right) (\mathbf{x} - \mathbf{x}_n)\right)$$

We find for $\sigma^2 = 1$

$$p(x) = \frac{1}{7} \sum_{n=1}^7 \frac{1}{(2\pi)^{M/2} \sqrt{|\sigma^2 \mathbf{I}|}} \exp\left(-\frac{1}{2} \|\mathbf{x} - \mathbf{x}_n\|_2^2\right)$$

We thus find

$$\begin{aligned}
p(O8) &= \frac{1}{7} \sum_{n=1}^7 \frac{1}{(2\pi)^{M/2} \sqrt{\sigma^{2M}}} \exp(-\frac{1}{2} \|\mathbf{x}_{O8} - \mathbf{x}_{On}\|_2^2) \\
&= \frac{1}{7 * (2\pi)^{M/2}} (\exp(-\frac{5.11^2}{2}) + \exp(-\frac{4.79^2}{2}) \\
&\quad + \exp(-\frac{4.90^2}{2}) + \exp(-\frac{4.74^2}{2}) + \exp(-\frac{2.96^2}{2}) \\
&\quad + \exp(-\frac{5.16^2}{2}) + \exp(-\frac{2.88^2}{2})) = 6.5 \cdot 10^{-6}
\end{aligned}$$

20.2 The correct answer is D:

$$\begin{aligned}
\text{density}(\mathbf{x}_{O1}, 2) &= \left(\frac{1}{2} \cdot (68.1 + 165.4)\right)^{-1} = 0.0086 \\
\text{density}(\mathbf{x}_{O3}, 2) &= \left(\frac{1}{2} \cdot (68.1 + 111.1)\right)^{-1} = 0.0112 \\
\text{density}(\mathbf{x}_{O4}, 2) &= \left(\frac{1}{2} \cdot (32.5 + 44.7)\right)^{-1} = 0.0259 \\
\text{a.r.d.}(\mathbf{x}, K) &= \frac{\text{density}(\mathbf{x}_{O1}, 2)}{\frac{1}{2}(\text{density}(\mathbf{x}_{O3}, 2) + \text{density}(\mathbf{x}_{O4}, 2))} \\
&= \frac{0.0086}{\frac{1}{2} \cdot (0.0112 + 0.0259)} = 0.46
\end{aligned}$$

20.3 The correct answer is A: The z-score is defined as

$$z_n = \frac{x_n - \text{mean}(x)}{\text{std}(x)} = \frac{x_n - 6}{28}.$$

The z-scores for all data objects are given by (rounded to one decimal)

n	1	2	3	4	5	6	7	8
$ z_n $	0.3	0.4	0.5	0.2	2.5	0.4	0.3	0.4

Thus, according to the definition, no objects are judged to be outliers, since no z-score has an absolute value greater than 3.00.

20.4 The correct answer is C:

$$\begin{aligned}
\text{density}(\mathbf{x}_{O8}, 3) &= \left(\frac{1}{3} \cdot (2.88 + 2.96 + 4.74)\right)^{-1} = 0.2836 \\
\text{density}(\mathbf{x}_{O7}, 3) &= \left(\frac{1}{3} \cdot (1.41 + 2.88 + 3.54)\right)^{-1} = 0.3831 \\
\text{density}(\mathbf{x}_{O5}, 3) &= \left(\frac{1}{3} \cdot (1.41 + 2.66 + 2.84)\right)^{-1} = 0.4342 \\
\text{density}(\mathbf{x}_{O4}, 3) &= \left(\frac{1}{3} \cdot (0.96 + 1.76 + 1.52)\right)^{-1} = 0.7075 \\
\text{a.r.d.}(\mathbf{x}, K) &= \frac{\text{density}(\mathbf{x}_{O8}, 3)}{\frac{1}{3}(\text{density}(\mathbf{x}_{O7}, 3) + \text{density}(\mathbf{x}_{O6}, 3) + \text{density}(\mathbf{x}_{O4}, 3))} \\
&= \frac{0.2836}{\frac{1}{3} \cdot (0.3831 + 0.4342 + 0.7075)} = 0.56
\end{aligned}$$

20.5 The correct answer is D: The true ARD is 0.75. The nearest neighbour of o_1 is o_2 and the density at each of these points is respectively $\frac{1}{4}$ and $\frac{1}{3}$ from which it follows the a.a.d is

$$\text{a.r.d.}(\mathbf{x}, K) = \frac{\text{density}(\mathbf{x}, K)}{\frac{1}{K} \sum_{z \in N(\mathbf{x}, K)} \text{density}(\mathbf{z}, K)} = \frac{1/4}{1/3} = \frac{3}{4}$$

20.6 The correct answer is A: The true density is 0.043. This is easiest to compute as

$$p(\mathbf{x} = o_1) = \sum_{i=1}^8 \frac{1}{8} \frac{1}{2\lambda} \exp(-d(o_1, o_i)/\lambda)$$

Problems of Chapter 21

21.1 The correct answer is C: For a set to have support more than 40% the set must occur at least 5 out of the 12 times. All the itemsets that have this property are $\{M_L\}$, $\{M_H\}$, $\{P_H\}$, $\{P_L\}$, $\{D_H\}$, $\{D_L\}$, $\{M_H, P_H\}$, $\{M_L, D_L\}$.

21.2 The correct answer is C: The confidence is given as

$$\begin{aligned} P(P_L = 1 | M_L = 1, D_L = 1) &= \\ \frac{P(P_L = 1, M_L = 1, D_L = 1)}{P(M_L = 1, D_L = 1)} & \\ = \frac{3/12}{5/12} = 3/5 &= \frac{\sigma_{P_L, M_L, D_L}}{\sigma_{M_L, D_L}} \end{aligned}$$

21.3 The correct answer is B: For a set to have support more than 52% the set must occur at least 6 out of the 10 times. All the itemsets that have this property are $\{YA_N\}$, $\{OA_Y\}$, $\{PA_Y\}$, $\{YA_N, PA_Y\}$, $\{OA_Y, PA_Y\}$.

A

Mathematical Notation

We have tried to keep the mathematical content of the book to the minimum necessary to achieve a proper understanding. However, this minimum level is nonzero, and the reader should have a basic grasp of linear algebra, calculus, analysis and probability theory.

Elementary notation

Numbers are denoted by variables such as x and the set of all real numbers $(0, 10, \frac{1}{3}, -4, \sqrt{2}, \pi, \text{etc.})$ will be denoted as \mathbb{R} . The notation \mathbb{R}^d will denote the product space, and so if we consider a vector:

$$\boldsymbol{x} = \begin{bmatrix} -1 \\ 4 \\ 1 \end{bmatrix}$$

we will write this as $\boldsymbol{x} \in \mathbb{R}^3$ and the notation: \mathbb{R}^3 is used to indicate all three-dimensional vectors. Notice vectors are bolded and are typically lower-case roman letters $\boldsymbol{x}, \boldsymbol{y}, \dots$. We use $\mathbb{R}_+ = [0, \infty[$ for the set of all non-negative numbers. The symbol T is used to indicate the transpose of a vector or matrix. For instance

$$\boldsymbol{x}^T = [-1 \ 4 \ 1].$$

Uppercase bold roman letters $\boldsymbol{W}, \boldsymbol{A}, \boldsymbol{B}, \dots$ indicate matrices, for instance

$$\boldsymbol{A} = \begin{bmatrix} -1 & 0 & 2 \\ 1 & 1 & -2 \end{bmatrix},$$

in which case we say \boldsymbol{A} is a 2×3 matrix and we write $\boldsymbol{A} \in \mathbb{R}^{2 \times 3}$. The i th element of a vector is written as x_i and the i, j 'th element of a matrix as A_{ij} (or $A_{i,j}$). For instance $x_2 = 4$ and $A_{2,3} = -2$.

If we have N observations $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$ of a M -dimensional vector

$$\boldsymbol{x} = [x_1 \dots, x_M]^T,$$

we can combine the observations into an $N \times M$ data matrix \boldsymbol{X}

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix}, \quad (\text{A.1})$$

in which the i th row of \boldsymbol{X} corresponds to the row vector \boldsymbol{x}_i^T . We will use this notation for our data matrix and the *rows* of \boldsymbol{X} will correspond to N *observations* and the M *columns* of \boldsymbol{X} will correspond to M *attributes*.

Finally the expectation of a function f , for instance $f(x, y) = \sin(x)e^{-y-x}$, with respect to a random variable x having the density function $p(x)$ will be denoted

$$\mathbb{E}_x[f] = \int_{-\infty}^{\infty} f(x, y)p(x) dx.$$

In cases where it is clear from the context which variable will be averaged over we will omit the suffix and simply write $\mathbb{E}[x^2]$.

Linear Algebra

The matrix product is written as \mathbf{Ax} . Recall for two matrices \mathbf{A}, \mathbf{B} : $(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$ and that the identity matrix I_M is the $M \times M$ matrix such that for instance

$$\mathbf{I}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We may suppress M and simply write \mathbf{I} . Remember $\mathbf{AI} = \mathbf{IA} = \mathbf{A}$ assuming the dimensions match.

An $N \times M$ matrix \mathbf{A} is said to be symmetric if $\mathbf{A}^T = \mathbf{A}$ and quadratic if $N = M$. For a quadratic matrix \mathbf{A} , if there exists a matrix \mathbf{A}^{-1} such that

$$\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I},$$

then \mathbf{A} is said to be invertible and \mathbf{A}^{-1} is the inverse. A necessary and sufficient requirement is that the determinant of \mathbf{A} , $\det(\mathbf{A})$, is non-zero.

Subspaces and Eigenvalues

If we are given vectors $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$ and numbers $a_1, \dots, a_n \in \mathbb{R}$ then the vector

$$\mathbf{x} = a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_n\mathbf{x}_n,$$

is said to be a *linear combination* of $\mathbf{x}_1, \dots, \mathbf{x}_n$. A *subspace* V of a vector space \mathbb{R}^d is a set of vectors V which is closed under linear combination. That is if $\mathbf{x}_1, \dots, \mathbf{x}_n \in V$ then

$$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_n\mathbf{x}_n \in V.$$

All vectors that can be written as a linear combination of a set of vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$ is called the *span* of $\mathbf{x}_1, \dots, \mathbf{x}_n$. Notice the span is a subspace of \mathbb{R}^d .

A set of vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$ is said to be linearly independent if

$$\mathbf{0} = a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_n\mathbf{x}_n,$$

implies $a_1 = a_2 = \cdots = a_n = 0$. Otherwise, they are said to be linearly dependent. For each subspace V of \mathbb{R}^d it is possible to find a set of vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$, $n \leq d$ such that $\mathbf{x}_1, \dots, \mathbf{x}_d$ are linearly independent and such that the span of $\mathbf{x}_1, \dots, \mathbf{x}_n$ is V . Such a set is known as a *basis* of V and n is the dimension of the subspace V . The basis is further said to be orthonormal if

$$\mathbf{x}_i^T \mathbf{x}_j = \delta_{ij},$$

where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. Finally, recall that if \mathbf{A} is quadratic and there exists a vector $\mathbf{x} \neq \mathbf{0}$ and a number λ such that

$$\mathbf{Ax} = \lambda\mathbf{x},$$

then \mathbf{x} is said to be an *eigenvector* of \mathbf{A} and λ the corresponding *eigenvalue*. Suppose $\mathbf{x}_1, \mathbf{x}_2$ are eigenvectors of \mathbf{A} with eigenvalues λ_1, λ_2 and suppose $\lambda_1 \neq \lambda_2$ then $\mathbf{x}_1, \mathbf{x}_2$ are orthogonal:

$$\mathbf{x}_1^T \mathbf{x}_2 = 0.$$

In particular, if \mathbf{A} is a $d \times d$ symmetric matrix, $\mathbf{A}^T = \mathbf{A}$, then there exists an orthonormal basis of eigenvectors for \mathbb{R}^d .

Analysis

A function f that maps from a D dimensional space to a single real number will be written as

$$f : \mathbb{R}^D \rightarrow \mathbb{R},$$

to explicitly specify the dimensions of the spaces f map between. This notation, as well as the notation $\mathbf{x} \in \mathbb{R}^d$, may appear cumbersome at a first glance however when considering functions between high-dimensional spaces it will often be a helpful guide to keep track of what the functions do. For the same reason we will only use this notation when it benefits the readability and not for formal exactness.

High-dimensional functions will play a special role in the following and so if we consider a function which maps from a D dimensional space to a M dimensional space ($M > 1$) we will write \mathbf{f} with a boldface:

$$\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^M.$$

To give two quick examples, if

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix},$$

then we can define $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ where:

$$f(\mathbf{x}) = x_1 + x_2 x_3,$$

or another example $\mathbf{g} : \mathbb{R}^3 \rightarrow \mathbb{R}^2$

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} x_1 + \sin(x_2) \\ x_3 + 2e^{-x_1} \end{bmatrix}.$$

Notice, we may also sometimes write $f(\mathbf{x})$ as $f(x_1, x_2, x_3)$. We assume the reader is familiar with derivative evaluated at a point x or the partial derivatives evaluated in \mathbf{x}

$$\frac{df}{dx}(x) \quad \text{and} \quad \frac{\partial f}{\partial x_2}(\mathbf{x}), \tag{A.2}$$

as well as integrals over some or all variables of a function:

$$I = \int_{\mathbb{R}^D} f(\mathbf{x}) d\mathbf{x}, \quad I(x_1) = \int_{\mathbb{R}^2} f(x_1, x_2, x_3) dx_2 dx_3 \tag{A.3}$$

However, knowledge of integrating actual function will not be used and integrals are mostly used to state theoretical results. Finally we will use $\nabla f(\mathbf{x})$ as the divergence of a function $f : \mathbb{R}^D \rightarrow \mathbb{R}$ evaluated at \mathbf{x} :

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \vdots \\ \frac{\partial f}{\partial x_D}(\mathbf{x}) \end{bmatrix}$$

Slightly more advanced concepts

Recall the multivariate Taylor expansion of a function $f : \mathbb{R}^D \rightarrow \mathbb{R}$ around a point \mathbf{x} can be written as

$$f(\mathbf{x} + \boldsymbol{\delta}) = f(\mathbf{x}) + \boldsymbol{\delta}^T \nabla f(\mathbf{x}) + \text{higher order terms}.$$

For one dimension this is the familiar result

$$f(x + \delta) = f(x) + \delta \frac{df}{dx}(x) + \text{higher order terms}.$$

We will use the notation

$$\mathbf{x}^* = \arg \min_{\mathbf{x}} f(\mathbf{x}),$$

as the value \mathbf{x}^* that minimizes f and similar $\arg \max$ to find the maximum. Recall also that if we wish to find the minimum or maximum of a function $f : \mathbb{R}^D$ under a constraint that another function $h : \mathbb{R}^D \rightarrow \mathbb{R}$ is zero, written as:

$$\mathbf{x}^* = \arg \min_{\mathbf{x}: h(\mathbf{x})=0} f(\mathbf{x}).$$

This can be done by introducing a Lagrange multiplier $\lambda \in \mathbb{R}$ and solve the problem

$$f(\mathbf{x}) + \lambda h(\mathbf{x}) = 0,$$

by taking derivatives with respect to \mathbf{x} and λ and set these equal to zero. That is, by simultaneous solving the $D + 1$ equations:

$$\nabla f(\mathbf{x}) + \lambda \nabla h(\mathbf{x}) = 0 \text{ and } h(\mathbf{x}) = 0.$$

We will only use this technique once and a reader not familiar with the method of Lagrange multipliers should consult the many excellent guides available online as texts or videos.

Probability Theory

In probability theory, we always consider the probability of an *event*, i.e. something which either does or does not occur. The probability of an event is written with an upper-case P and is a number between 0 and 1. For instance if we consider the outcome of a coin-flip, the outcome that the coin is heads (or tails) are events which either do or do not occur and so we can let $B = 0$ denote the event the coin is heads and $B = 1$ if the coin is tails we will write

$$P(B = 0) = 0.4, \quad P(B = 1) = 0.6,$$

to indicate there is a 40% chance the coin is heads. B is called a random or stochastic variable, i.e. something which outcome is the result of a random process or is otherwise unknown. If we suppose b corresponds to the actual result of the coin-flip (i.e. a person writes down $b = 0$ if the coin came out heads and $b = 1$ if the coin came out tails) we can write the probability of the outcome as simply:

$$P(B = b)$$

Sometimes this will be abbreviated as

$$P(b).$$

We stress probabilities are computed of *events* (boolean occurrences) and it is therefore not possible to talk about the probability of a continuous number, for instance *the probability Napoleon was exactly 1.731 meters tall*. For continuous numbers we use the *probability density function* (sometimes simply the probability density) which, for a random variable $(x_1, x_2, \dots, x_d) \in \mathbb{R}^d$ is a non-negative function

$$p : \mathbb{R}^d \rightarrow \mathbb{R}_+,$$

which integrates to one:

$$\int_{\mathbb{R}^d} p(x_1, \dots, x_d) dx_1, \dots, dx_d = 1,$$

where \mathbb{R}_+ is the interval $[0, \infty[$. Since the integration limits are often redundant we may also write the above as simply

$$\int_{\mathbb{R}^d} p(\mathbf{x}) d\mathbf{x} = 1.$$

The probability density function is not the same as a probability, however you can obtain probabilities by integrating the probability density function. For $d = 1$ we can for instance consider the probability density function

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}},$$

for which $\int p(x) dx = 1$. In this case the probability that the random variable x falls within the interval $[2, 3]$ is simply:

$$P(x \in [2, 3]) = \int_2^3 p(x) dx.$$

Notice, this is a proper probability. In the Napoleon example too we are allowed to say consider *the probability Napoleon was between 1.73 and 1.75 meters tall* which is a proper event. Confusion of the probability density function and probabilities is common and may lead to difficulties later. Probability theory is a rich and interesting mathematical discipline and this introduction does not do it service; however, in this book we will take a “naive approach” and not dwell on the details.

Finally the expectation of a function $f(x, y)$ with respect to a random variable x governed by probability density p is written as

$$\mathbb{E}_x[f] = \int p(x) f(x, y) dx.$$

References

- Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.
- Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.
- Patricia ME Altham. The analysis of matched proportions. *Biometrika*, 58(3):561–576, 1971.
- RE Barlow. Introduction to de finetti (1937) foresight: its logical laws, its subjective sources. In *Breakthroughs in statistics*, pages 127–133. Springer, 1992.
- Mr. Bayes and Mr. Price. An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, f. r. s. communicated by mr. price, in a letter to john canton, a. m. f. r. s. *Philosophical Transactions*, 53:370–418, 1763. doi: 10.1098/rstl.1763.0053. URL <http://rstl.royalsocietypublishing.org/content/53/370.short>.
- Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *The Journal of Machine Learning Research*, 18(1):2653–2688, 2017.
- Yoshua Bengio and Yves Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of machine learning research*, 5(Sep):1089–1105, 2004.
- Christoph Bergmeir, Rob J Hyndman, and Bonsoo Koo. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120:70–83, 2018.
- C.M. Bishop. *Pattern Recognition and Machine Learning*. Information science and statistics. Springer, 2013. ISBN 9788132209065. URL <https://books.google.dk/books?id=HL4HrgEACAAJ>.
- Daniel A Bloch, Geoffrey S Watson, et al. A bayesian study of the multinomial distribution. *The Annals of Mathematical Statistics*, 38(5):1423–1435, 1967.
- L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984. ISBN 9780412048418. URL <https://books.google.co.in/books?id=JwQx-W0mSyQC>.
- Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- David A Burn. 22 designing effective statistical graphs. 1993.
- Nitesh V Chawla. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 853–867. Springer, 2005.
- P Collinson. Of bombers, radiologists, and cardiologists: time to roc. *Heart*, 80(3):215–217, 1998.

- Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4): 547–553, 2009.
- Richard T Cox. Probability, frequency and reasonable expectation. *American journal of physics*, 14(1):1–13, 1946.
- Belur V Dasarathy and Belur V Sheela. A composite classifier system design: concepts and methodology. *Proceedings of the IEEE*, 67(5):708–713, 1979.
- Bruno De Finetti. La prévision: ses lois logiques, ses sources subjectives. In *Annales de l'institut Henri Poincaré*, volume 7, pages 1–68, 1937.
- Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.
- Thomas G Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.
- Evelyn Fix and Joseph L Hodges. Discriminatory analysis-nonparametric discrimination: consistency properties. Technical report, DTIC Document, 1951.
- Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.
- J.G. Garnier and A. Quetelet. *Correspondance mathématique et physique*. Number v. 10. Impr. d'H. Vandekerckhove, 1838. URL <https://books.google.dk/books?id=8GsEAAAAYAAJ>.
- C.F. Gauß. *Theoria Motus Corporum Coelestium In Sectionibus Conicis Solem Ambientium*. Frid. Perthes et J. H. Besser, 1809. URL <https://books.google.dk/books?id=7jJbAAAAcAAJ>.
- Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- Alan Hájek. “mises redux”—redux: Fifteen arguments against finite frequentism. In *Probability, Dynamics and Causality*, pages 69–87. Springer, 1997.
- Alan Hájek. Fifteen arguments against hypothetical frequentism. *Erkenntnis*, 70(2):211–235, 2009.
- Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12:993–1001, 1990.
- T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer New York, 2009. ISBN 9780387848587. URL <https://books.google.dk/books?id=tVIjmNS30b8C>.
- Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.
- Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.
- Earl B Hunt, Janet Marin, and Philip J Stone. Experiments in induction. 1966.
- G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer New York, 2014. ISBN 9781461471370. URL <https://books.google.dk/books?id=at1bmAEACAAJ>.
- Edwin T Jaynes. Prior probabilities. *IEEE Transactions on systems science and cybernetics*, 4(3): 227–241, 1968.
- Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.
- H. Jeffreys. *Theory of Probability*. The International series of monographs on physics. At The Clarendon Press, 1939. URL https://books.google.dk/books?id=6_ogAAAAMAAJ.

- Harold Jeffreys. An invariant form for the prior probability in estimation problems. *Proc. R. Soc. Lond. A*, 186(1007):453–461, 1946.
- Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.
- D. Kahneman, P. Slovic, and A. Tversky. *Judgment Under Uncertainty: Heuristics and Biases*. Cambridge University Press, 1982. ISBN 9780521284141. URL https://books.google.co.uk/books?id=_0H8gwj4a1MC.
- Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143. Morgan Kaufmann, 1995.
- DD Kosambi. Statistics in function space. *J. Indian Math. Soc.*, 7(1):76–88, 1943.
- A.M. Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes*. F. Didot, 1805. URL <https://books.google.dk/books?id=FRc0AAAAQAAJ>.
- David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.
- James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.
- Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- Claude Nadeau and Yoshua Bengio. Inference for the generalization error. In *Advances in neural information processing systems*, pages 307–313, 2000.
- Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.
- Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.
- Judea Pearl, Madelyn Glymour, and Nicholas P Jewell. *Causal Inference in Statistics: A Primer*. John Wiley & Sons, 2016.
- Karl Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, 185:71–110, 1894.
- Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2(11):559–572, 1901. doi: 10.1080/14786440109462720. URL <http://dx.doi.org/10.1080/14786440109462720>.
- Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7(15):510, 2008.
- Vikas C Raykar and Ramani Duraiswami. Fast optimal bandwidth selection for kernel density estimation. In *SDM*, pages 524–528. SIAM, 2006.
- Murray Rosenblatt et al. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.
- Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.
- Steve Selvin, M. Bloxham, A. I. Khuri, Michael Moore, Rodney Coleman, G. Rex Bryce, James A. Hagans, Thomas C. Chalmers, E. A. Maxwell, and Gary N. Smith. Letters to the editor. *The American Statistician*, 29(1):67–71, 1975. ISSN 00031305. URL <http://www.jstor.org/stable/2683689>.
- C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1948.tb01338.x.
- H. Steinhaus. Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci.*, 1:801–804, 1956.

- S. S. Stevens. On the theory of scales of measurement. *Science*, 103(2684):677–680, 1946. ISSN 0036-8075. doi: 10.1126/science.103.2684.677. URL <http://science.sciencemag.org/content/103/2684/677>.
- Gilbert W Stewart. On the early history of the singular value decomposition. *SIAM review*, 35(4):551–566, 1993.
- Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617, 2002.
- P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2013. ISBN 9780133128901. URL https://books.google.dk/books?id=_ZQ4MQEACAAJ.
- Andrey Nikolayevich Tikhonov. On the stability of inverse problems. In *Dokl. Akad. Nauk SSSR*, volume 39, pages 195–198, 1943.
- Edward R Tufte, Nora Hillman Goeler, and Richard Benson. *Envisioning information*, volume 126. Graphics press Cheshire, CT, 1990.
- Alan M Turing. Computing machinery and intelligence. *Mind*, pages 433–460, 1950.